TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Olari Pipenberg 242742IVCM

DESIGN AND IMPLEMENTATION OF E-ITS COMPLIANT KUBERNETES CLUSTER WITH AUTOMATED AUDITING

Master's Thesis

Supervisor: Risto Vaarandi
PhD

Co-supervisor: Thomas Lepik
MBA

Tallinn 2026

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Olari Pipenberg 242742IVCM

E-ITS-IGA VASTAVUSES OLEVA KUBERNETESE KLASTRI KAVANDAMINE JA RAKENDAMINE AUTOMATISEERITUD AUDITEERIMISEGA

Magistritöö

Juhendaja: Risto Vaarandi
PhD

Kaasjuhendaja: Thomas Lepik
MBA

Tallinn 2026

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Olari Pipenberg

18.05.2026

Abstract

The variety of Kubernetes distributions and the complexity of configuration options make security compliance management challenging. Organisations following the Estonian Information Security Standard (E-ITS) lack a standardised, automated approach to assessing the compliance of Kubernetes clusters. As a result, security assessment and enforcement become more resource-intensive and more prone to human error when performed manually.

This thesis presents the design and implementation of a Kubernetes cluster intended to support E-ITS compliance while integrating automated audit mechanisms. The study followed the Design Science Research Methodology (DSRM), combining literature review, prototype design, implementation, and evaluation. The solution was evaluated through laboratory testing, analytical compliance assessment, and examination in two organisational contexts. The results show that the proposed approach can support automated verification of E-ITS security measures and provide a practical basis for building more secure, auditable, and E-ITS-aligned Kubernetes environments.

The thesis is written in English and is 75 pages long, including 7 chapters, 14 figures and 13 tables.

Annotatsioon

E-ITS-iga vastavuses oleva Kubernetese klastri kavandamine ja rakendamine automatiseeritud auditeerimisega

Erinevate Kubernetese distributsioonide paljusus ja seadistusvõimaluste keerukus muudavad turbemeetmetele vastavuse haldamise keeruliseks. Eesti infoturbestandardit (E-ITS) rakendavatel asutustel puudub ühtne standardiseeritud ja automatiseeritud viis Kubernetese klastrite nõuetele vastavuse hindamiseks. Tulenevalt sellest on käsitsi turbemeetmete hindamine ja rakendamine ressursimahukas ja selle käigus võib tekkida vigu.

Käesolev magistritöö käsitleb E-ITS-i meetmetele vastavust toetava ning automatiseeritud auditeerimist võimaldava Kubernetese klastri kavandamist ja rakendamist. Töös kasutati metoodikana DSRM-i (*Design Science Research Methodology*). Selle käigus viidi läbi kirjanduse ülevaade, prototüübi kavandamine, rakendamine ja hindamine. Lahendust hinnati laboratoorsete katsete ja analüüsi abil ning selle rakendatavust vaadeldi kahe asutuse näitel. Tulemused näitavad, et pakutud lahendus toetab E-ITS-i turbemeetmete automatiseeritud kontrolli ning pakub võimalust turvalisemate, auditeeritavate ja E-ITS-iga vastavuses olevate Kubernetese klastrite loomiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 75 leheküljel, 7 peatükki, 14 joonist, 13 tabelit.

List of Abbreviations and Terms

AI	Artificial intelligence
API	Application Programming Interface
AZ	Availability Zone
BGP	Border Gateway Protocol
CaC	Compliance as Code
CIS	Center for Internet Security
CSI	Container Storage Interface
CISA	Cybersecurity and Infrastructure Security Agency
CMDB	Configuration Management Database
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
CPU	Central Processing Unit
CRD	Custom Resource Definition
CRIU	Checkpoint/Restore in Userspace
CVE	Common Vulnerabilities and Exposures
DSR	Direct Server Return
DR	Disaster Recovery
DSRM	Design Science Research Methodology
ECMP	Equal-Cost Multi-Path routing
E-ITS	Estonian Information Security Standard
EKS	Elastic Kubernetes Service
IaC	Infrastructure-as-Code
ICT	Information and communication technology
eBPF	Extended Berkeley Packet Filter
Intel SGX	Intel Software Guard Extensions
KMS	Key Management Service
K8S	Kubernetes
LSM	Linux Security Module
MTU	Maximum Transmission Unit
mTLS	Mutual Transport Layer Security
NAT	Network Address Translation
NSA	National Security Agency
OPA	Open Policy Agent

OS	Operating System
Proxmox VE	Proxmox Virtual Environment
PSS	Pod Security Standard
PaC	Policy-As-Code
RBAC	Role-based access control
RKE1	Rancher Kubernetes Engine 1
RKE2	RKE Government
SIEM	Security Information and Event Management
saBPF	secure audit BPF
SFF	Small Form Factor
SD	SIGHUP Distribution
SSD	Solid-state drive
SSH	Secure Shell
SSL	Secure Sockets Layer
SOPS	Mozilla Secrets OPerationS
TLS	Transport Layer Security
VM	Virtual Machine

Table of Contents

List of Figures

List of Tables

1. Introduction

Microservices have become foundational to modern IT systems. This has led to the adoption of containers to ensure scalability and portability across environments. Kubernetes has become the leading container orchestration platform in the industry. Following Linux, it is one of the largest open-source projects in the world and is used by more than 70% of Fortune 100 companies [1].

In addition, Kubernetes is increasingly used as a platform for emerging workloads, including artificial intelligence (AI). According to the Cloud Native Computing Foundation (CNCF) Annual Cloud Native Survey, 66% of organisations running generative AI workloads use Kubernetes to manage them [2].

However, the variety of Kubernetes distributions and configuration possibilities makes security compliance management challenging. As a result, security enforcement becomes resource-intensive and more prone to human error when done manually.

Estonian Information Security Standard (E-ITS) implementation guidance is needed to support its adoption in cloud-native computing environments. Furthermore, the NIS2 directive was implemented in Estonia and entered into force on 1 January 2026 [3]. It introduces stricter cybersecurity requirements and expands the scope of regulated organisations. As a result, organisations are required to follow security standards and frameworks. This increases the need for effective and automated compliance mechanisms.

1.1 Research Problem

The E-ITS defines a comprehensive framework to manage information security risks in public and private sector organisations [4]. Although E-ITS outlines what needs to be achieved, it lacks practical implementation examples or verifiable audit mechanisms that guide organisations in translating requirements into practice. This is especially relevant in Kubernetes environments, where practical implementation guidance and automated compliance verification are lacking. Beyond compliance, ensuring actual security alignment is critical, as gaps may exist between E-ITS requirements and current Kubernetes security best practices.

1.2 Research Questions

This study is based on one primary research question (PRQ), supported by three secondary research questions (SRQs).

- **[PRQ]**: How can a Kubernetes cluster be designed and implemented to comply with the E-ITS and automatically assess its security level?

To address this question, the following secondary research questions are formulated:

- **[SRQ1]**: What are the main challenges and approaches to securing and ensuring compliance in Kubernetes environments?
- **[SRQ2]**: To what extent can existing compliance and auditing frameworks be used to implement and verify E-ITS security measures in Kubernetes environments?
- **[SRQ3]**: What design and implementation principles are used to build Kubernetes clusters that are intended to be secure, auditable, and compliant?

Secondary research questions (SRQ1–SRQ3) aim to identify existing approaches, assess compatibility with E-ITS, and explore the foundational principles for building secure and compliant Kubernetes clusters. Collectively, these insights support the answer to the primary research question (PRQ), which focuses on the design and implementation of a solution. Secondary questions are answered through a literature review, supported by an analysis to select the Kubernetes distribution and its components for the prototype. The primary research question (PRQ) is then answered through practical design and prototyping, guided by the findings of the secondary questions.

1.3 Scope and Goal

This study is based on the E-ITS as defined in the 2024 Estonian version and the 2023 English version available at the time of writing. The E-ITS includes a dedicated APP.4.4 (Kubernetes) module that describes security measures specifically for Kubernetes. During the writing of this thesis, a draft version of E-ITS 2026 was published, in which the APP.4.4 module was renamed to SYS.1.7. This study is based on the APP.4.4 module. The underlying security measures and principles remain applicable to the updated structure of the standard.

Although Kubernetes is a container orchestration platform, its security depends on the underlying operating system (OS) as well as the containers that it manages. For this reason,

security for both the OS and containers was also considered when designing a secure and compliant cluster. Therefore, the E-ITS modules SYS.1.3 (Linux and Unix server) and SYS.1.6 (Containerisation) were also considered. It should also be noted that certain design and process-related measures cannot be automated and must still be reviewed manually.

According to Cloud Native Landscape, over 60 Kubernetes distributions are certified by CNCF [5]. The diversity of distributions highlights the need for systematic comparisons regarding security and compliance. CNCF certification does not necessarily mean that a distribution is open source or publicly available. Therefore, this study only considered CNCF-certified distributions that were open source.

This research was limited to on-premises deployments rather than cloud-based Kubernetes services. Public cloud implementations vary and are often closed for detailed technical analysis. Therefore, this study focused on self-managed, locally deployed clusters, where configuration control and compliance verification remain entirely under the organisation's control. However, the developed solution could also be deployed in cloud environments, such as Amazon EC2 instances or Azure Virtual Machines.

Certain supporting services, such as logging and monitoring, may rely on external systems that are not specific to Kubernetes environments. While important for operational security and observability, their deployment and configuration are often organisation-specific and were therefore considered outside the scope of this study.

A reference secure design for Kubernetes clusters that practitioners can adopt can help reduce configuration errors. Automated compliance auditing can offer organisations transparent and repeatable mechanisms to verify compliance. This is especially valuable in scenarios where manual checks are time-consuming, error-prone, and require significant resources. To demonstrate the feasibility of the proposed approach, the study included the development of a prototype implementation.

1.4 Structure of the Thesis

The structure of this thesis is as follows:

- Chapter 2 presents the literature review, identifying existing approaches and research gaps related to Kubernetes security and E-ITS compliance.
- Chapter 3 describes the research methodology.
- Chapter 4 presents the design of the proposed solution.
- Chapter 5 describes the implementation of the prototype.

- Chapter 6 evaluates the prototype through laboratory testing, E-ITS compliance assessment, and examination in two organisational settings. The results are then discussed to answer the research questions.
- Chapter 7 concludes the thesis and provides recommendations for future work.

2. Literature Review

This literature review explores approaches to Kubernetes compliance. The focus is on automation, alignment with security standards, including the E-ITS, and the principles of secure cluster design. The purpose is to assess how previous work addresses the thesis research questions and to identify gaps.

2.1 Search Strategy

The first stable release of Kubernetes (v1.0) was launched in July 2015. Based on that, the time frame for publications from 2015 to 2026 was selected for the literature review. Limiting the scope to this period ensures the inclusion of both foundational and contemporary works relevant to Kubernetes. Therefore, studies published before 2015 were excluded. E-ITS is a relatively new national standard and became effective in January 2023, replacing the previous IT baseline security system (ISKE). To ensure that the review remains comprehensive, the scope of the search was expanded to include both international frameworks and national standards. In terms of national standards, the review considered the German BSI and the U.S. NIST security framework. For international standards, the search expanded to ISO 27001, which is widely recognised in industry.

The search strategy included the following databases:

- ACM Digital Library
- ESTER
- Google Scholar
- IEEE Xplore
- SpringerLink
- TalTech library
- University of Tartu Digital Archive ADA

The following keywords and search combinations with boolean operators were used:

- Keywords:
  - "audit"
  - "auditing"
  - "BSI"

  - "compliance"
  - "Estonian Information Security Standard"
  - "E-ITS"
  - "ISO 27001"
  - "Kubernetes"
  - "K8S"
  - "NIST"
  - "secure"
  - "security"
- Search Combinations:
  - **Search query 1:** *("Kubernetes" OR "K8S") AND ("secure" OR "security" OR "compliance" OR "audit" OR "auditing")*
  - **Search query 2:** *("Kubernetes" OR "K8S") AND ("Estonian Information Security Standard" OR "BSI" OR "NIST" OR "ISO 27001")*

Initially, the search strategy included combinations of the terms "Estonian Information Security Standard" and "E-ITS". Unfortunately, this resulted in irrelevant results as the abbreviation "E-ITS" was not consistently used in the literature to refer to the Estonian standard. To address this, the search strategy was refined using only the full phrase "Estonian Information Security Standard". A third query was also tested, which combined the security and compliance terms with "design", "implementation", and "guideline". However, this produced almost duplicate results for query 1 and did not contribute significant new findings.

Table 1. Search results from various databases

Database	**Search query 1**	**Search query 2**
ACM Digital Library	1840	254
ESTER	3851	16
Google Scholar	1820	320
IEEE Xplore	473	4
SpringerLink	3409	270
TalTech library	1409	11
University of Tartu Digital Archive ADA	34	5

2.2 Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were applied to select the literature:

Inclusion criteria:

1. Publications from 2015 to 2026.

2. Relevant to one or more of the research questions.
3. Articles published in peer-reviewed journals and conference proceedings.
4. Books published by recognised publishers (e.g., O'Reilly Media, Springer).
5. Includes real-world case studies, tools, or implementation guidance.

Exclusion criteria:

1. Articles that are not published in English or Estonian.
2. Articles that are not directly related to any of the research questions or objectives.
3. Duplicate articles in multiple databases.
4. Studies that focus entirely on unrelated technologies without any focus on compliance or security.

2.3 Selection Process

The initial database searches produced 13,716 results in total in the two search queries (see Table 1). The titles and abstracts were first selected to eliminate irrelevant papers. In addition to database searches, a combination of snowball and reverse snowball techniques was applied. A total of 31 publications were selected for the final analysis.

2.4 Selected Papers

The reviewed papers were categorised into the following groups:

- Challenges in Securing Kubernetes
- Guidelines and Standards
- Application of E-ITS
- Compliance and Automation
- Comparative Studies of Kubernetes Distributions
- Components for Secure Kubernetes Design

Challenges in Securing Kubernetes

A survey by Randal revisits the historical development of virtual machines (VMs) and containers [6]. The article also highlights that vulnerabilities such as Spectre, Meltdown, Foreshadow, and L1TF can bypass software-level isolation, exposing weaknesses in modern container and virtualisation environments. The lack of strong isolation has also motivated research into alternative approaches. For example, Arnautov et al. proposed

SCONE, which uses Intel SGX enclaves to harden containers and protect applications even if the host kernel is compromised [7]. Although Randal's survey does not focus directly on Kubernetes, it shows that, historically, security was often secondary to other design priorities [6]. This historical trend also continues to shape the security and compliance challenges of today. For example, by default, Kubernetes does not enforce any Pod Security Standard (PSS) level [8]. This means that workloads in Kubernetes can use privileged features unless explicitly configured. Similarly, while Kubernetes network policies provide network-level filtering between pods, they are not enabled by default and must be explicitly defined [9].

The book by Martin and Hausenblas provides a hands-on, threat-driven guide to securing Kubernetes clusters with real-world scenarios referring to real CVEs [10]. For the demonstration, Kubernetes with default settings (vanilla Kubernetes) is used with examples. The authors introduce threat modelling as a foundation for Kubernetes security, supported by detailed walk-throughs of known attack paths and vulnerabilities. This approach is complemented by the Microsoft Kubernetes Threat Matrix, which classifies Kubernetes attack techniques [11]. The book covers various technical layers in terms of security [10]. It is a highly valuable resource when Kubernetes clusters need to be designed or improved in terms of security. However, the book does not provide a structured methodology for systematic and repeatable security evaluation. As a result, its applicability for standardised compliance auditing is limited, although it can still serve as a valuable foundation.

Chowdhury and Freund analysed vulnerabilities and exploits in on-premises Kubernetes clusters by deploying a deliberately insecure cluster and simulating real-world attack scenarios [12]. They correlated Kubernetes audit logs with OS logs using Graylog and defined rules for detecting suspicious activities. The study highlights the importance of detection and monitoring in Kubernetes security, providing guidance on which logs should be collected and analysed [12]. These findings highlight the need for multi-layer visibility in Kubernetes security. However, while their work focuses on post-event detection through log correlation, it does not address how such multi-layer insights can be systematically integrated to support continuous and automated security verification.

Misconfiguration is widely recognised as a major challenge in Kubernetes environments, as the platform exposes a large number of configuration options that can directly impact security. Zhang et al. analysed 719 defects extracted from 2,260 configuration scripts across 185 open-source repositories [13]. Their findings show that configuration defects are common and span 15 distinct categories, including issues related to pod scheduling, probing, and security. These results indicate that Kubernetes configuration is inherently error-prone and highlight the need for systematic approaches and improved tool support to

detect and prevent configuration-related issues.

The study by Sphan et al. discovered 18,467 Internet hosts exposing internal Kubernetes components that should not be publicly accessible [14]. Using large language models, Curtis and Eisty analysed Stack Overflow posts on Kubernetes [15]. Security-related discussions were the fourth most prevalent topic in all posts. The results of this study also show that there is still a lack of clear practical guidance and the need to develop tools to address Kubernetes security challenges [15]. Together, these results confirm the need for continued research in Kubernetes security.

Based on a survey of 600 DevOps, engineering, and security professionals, the 2024 Red Hat Report found that misconfiguration is the second-highest concern [16]. Similarly, Rahman et al. did an empirical study on Kubernetes manifests, which were taken from 92 open-source software repositories [17]. They uncovered 1,051 security misconfigurations out of 2,039 manifests. Based on this study, only 60% of the reported misconfiguration issues were fixed by the practitioners. The authors of this study also developed their own tool and compared existing ones [17]. However, performance was not compared to the more widely adopted Trivy [18] which is explicitly designed to be used in CI/CD pipelines. Overall, it can be observed that misconfiguration represents a significant challenge in Kubernetes.

Guidelines and Standards

Shamim et al. provide an overview of widely accepted Kubernetes security practices [19]. The paper focuses on practical insights grounded in real-world use. The authors conducted a review of 104 Internet artifacts. This included blog posts, videos, and technical presentations. From this, they derived 11 key practices known as the "XI Commandments", which include [19]:

- Authentication and Authorisation
- Implementing Kubernetes-specific Security Policies
- Vulnerability scanning
- Logging
- Namespace separation
- Encrypt and restrict access to etcd
- Continuous update
- Limit resources quotas
- Enable SSL/TLS support
- Separate sensitive workload

- Secure metadata access

Nerella and Puvvada extended this research by performing a gap analysis on the data set of Shamim et al. and on existing practices [20]. Their work proposed the following categories: access management, tighten default permissions, and restrict privileges through policies [20]. Unlike Shamim et al., their study is more practical and suggests specific tools to support implementation. In addition, Santos conducted a literature review and proposed a roadmap for Zero Trust implementation in Kubernetes [21]. Together, these papers outline the security practices currently used in Kubernetes clusters in production and identify foundational principles such as least privilege and layered isolation. However, the practices described remain generic and do not provide direct guidance on how to verify them. Furthermore, the works of Shamim et al. and Nerella and Puvvada are based primarily on a review of the grey literature.

The NIST SP 800-190 provides a guideline for securing containerised environments [22]. It describes security issues in container environments and offers recommendations for their mitigation [22]. One notable recommendation from NIST SP 800-190 is to use container-specific host OS instead of general-purpose ones to reduce attack surfaces [22]. This recommendation is especially relevant for Kubernetes, where several distributions are designed with minimal components to reduce possible vulnerabilities. Although the guide was published in 2017, it still provides a valuable foundation. Similarly, the National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA) have released the hardening guide, which provides security recommendations specifically for Kubernetes [23]. This guide offers a good starting point for securing clusters. However, its recommendations remain generic, which limits their direct usefulness for verifying Kubernetes configuration. The guide also refers to the Center for Internet Security (CIS) Kubernetes Benchmarks as additional security hardening guidance [23].

CIS is a community-driven nonprofit organisation that develops globally recognised best practices for securing IT systems and data [24]. The CIS Kubernetes Benchmark provides highly practical guidance, as each recommendation includes commands to verify the compliance of the configuration and remediation steps to address detected problems [25]. The benchmark covers recommendations for control plane components, etcd (distributed key-value store), control plane configuration, worker nodes, and policies [25]. In addition, CIS provides specific benchmarks for container-optimised OSes such as Bottlerocket and Talos Linux [26]. The CIS Benchmarks can serve as a useful reference for mapping E-ITS measures to specific controls.

Application of E-ITS

Existing research addressing the application of E-ITS to Kubernetes is limited. Koit's thesis focuses on providing methodological support for public sector organisations in formulating precise, E-ITS-compliant security requirements for ICT service procurements [27]. Her work highlights that a considerable number of organisations no longer maintain their own IT infrastructure and instead rely on external service providers. This reinforces the principle of "trust but verify", emphasising the need for systems provided by the service provider to comply with E-ITS.

The thesis of Lepik focuses on the possibilities of automation of E-ITS controls and proposes principles for automated compliance checks. It also addresses the resource-intensive nature of compliance processes. The work provides a foundational framework for data-driven compliance evaluation. It relies on structured asset and configuration data, including configuration management databases (CMDB), as a basis for evaluating compliance [28]. However, such an approach depends on whether the stored configuration data accurately reflects the actual system state and remains up to date. In dynamic environments, inconsistencies may occur. This can affect the accuracy of compliance assessments based solely on CMDB data.

Lindeberg's thesis contributes by analysing and implementing an E-ITS-compliant cluster with a focus on the APP.4.4 Kubernetes module [29]. Although highly valuable in practice, the work is limited to a single E-ITS module. It does not explicitly address container and operating system security. Additionally, it is not clear whether PSS is enforced in the implementation. This may leave certain security aspects unaddressed despite compliance with E-ITS requirements.

Compliance and Automation

As manual checks are slow, resource-intensive and prone to errors, Allam conducted research on policy-driven engineering to support compliance automation [30]. His article suggests embedding compliance checks into pipelines through Policy-As-Code (PaC), where rules are defined declaratively and are enforced automatically at the deployment time. The article also references tools such as OPA (Open Policy Agent), Kyverno, Conftest, and Sentiel to enable automated compliance enforcement in practice [30]. Additionally, Yanagawa et al. propose a framework that integrates Compliance as Code (CaC) with PaC to enable continuous compliance between heterogeneous policy validation points [31]. As validating admission controllers are a key Kubernetes concept [32], the findings of both

studies are relevant and align with the goals of this thesis. Moreover, leveraging OPA or Kyverno policies can provide an effective mechanism for verifying and enforcing security controls in Kubernetes clusters.

Kapetanidou et al. evaluated six widely used Kubernetes security tools [33]. Based on their findings, KubeLinter and Terrascan were more effective for static code analysis, while Trivy and Kubescape provided stronger runtime threat detection [33]. The insights from this study are relevant for compliance automation, as it outlines the strengths and limitations of tools suitable for continuous audit checks. These tools could possibly be combined with OPA or Kyverno for automated verification and enforcement of security measures.

Comparative Studies of Kubernetes Distributions

A comparative analysis by Malviya and Dwivedi suggests that Kubernetes does not provide authentication and authorisation by default, while OpenShift enforces stricter policies [34]. Similarly, a comparative analysis of lightweight Kubernetes distributions for edge computing demonstrates that k3s and k0s provide advantages in maintainability and simplicity [35]. However, k3s and k0s, compared to vanilla Kubernetes, KubeEdge, and OpenYurt, exhibit significantly lower security compliance [35]. In addition, the study by Ascensão et al. confirms that lightweight distributions perform poorly on security benchmarks [36]. These studies provide valuable information, as they rely on tools such as kube-bench [37], which evaluates security compliance against CIS benchmarks [26]. However, a research gap remains, as most existing work is limited to comparing the security of lightweight Kubernetes distributions with vanilla Kubernetes. At the same time, vanilla Kubernetes with default configurations is already vulnerable to known threats and requires additional hardening for use in production systems.

Components for Secure Kubernetes Design

Lim et al. demonstrate secure audit BPF (saBPF), which is an extension of the eBPF framework capable of deploying secure system-level audit mechanisms in container granularity [38]. They present the practicality of saBPF in Kubernetes by designing an audit framework, an intrusion detection system, and a lightweight access control mechanism. The system extends the eBPF and Linux Security Modules (LSM) frameworks to support container-level audit policies without requiring extensive kernel modifications, making it practical for real-world deployment [38]. This paper presents an automated approach to auditing container workloads within Kubernetes. The ability to embed audit rules at the

pod level allows saBPF to align closely with the principles of enforceable, context-aware security. The framework could support automatic audit mechanisms through custom eBPF programs and policy templates at the container level. However, as also discussed in the paper, the eBPF itself is not without risks since it can be exploited through weaknesses in the verifier. Furthermore, it appears that saBPF was developed primarily for research and demonstration purposes, and the project is not actively maintained. It may be more practical to adopt or extend the existing and more widely supported eBPF-based framework.

To enable networking within a cluster, Kubernetes makes use of the Container Network Interface (CNI) [39]. Choosing the appropriate CNI is critical when designing a cluster, because CNIs differ in their network models and security features. Kim et al. benchmarked several CNIs under high policy load and analysed their security features and performance implications [40]. Based on their findings, the most widely deployed CNIs for secure deployments are Antrea, Calico, and Cilium. Cilium, based on eBPF, offers a rich set of security features and a trade-off between the scalability of policies and the processing performance [40]. Similarly, the research by Siracusa et al. further demonstrates the flexibility of eBPF to implement modular and programmable network functions in Kubernetes clusters [41]. These insights highlight the importance of selecting an appropriate CNI based on security and performance requirements.

Focusing on CNIs, Budigiri et al. have also evaluated network policies as a lightweight security mechanism for Kubernetes [42]. Their study confirms that network policies impose negligible performance overhead, but also highlight risks such as misconfiguration. [42]. Although Calico and Cilium provide strong capabilities, a correct configuration is essential for secure deployments.

Settanni, Lisena, and Basile systematised runtime security enforcement in Kubernetes by proposing a capability model [43]. The model evaluates enforcement capabilities across different solutions, including CNIs such as Calico and Cilium, and runtime security tools such as KubeArmor and Tetragon. The authors also present an Analyzer to extract cluster security requirements and a Translator to generate tool-specific configurations from high-level policies [43]. Although the study provides theoretical grounds, its real-world applicability remains limited due to the unavailability of the proof-of-concept implementation.

2.5 Discussion and Research Gap

The reviewed literature shows that Kubernetes security and compliance is a complex challenge. Vanilla Kubernetes presents security risks when deployed with default settings.

Although many works explain what should be done, they lack specific technical steps to verify cluster configurations. Existing studies highlight that misconfiguration continues to be a significant concern in the industry. These findings relate to **SRQ1** on challenges and approaches to securing and ensuring compliance in Kubernetes environments. This ongoing problem creates an opportunity for this thesis to explore an approach that reduces misconfiguration and provides a more reliable way to verify Kubernetes compliance.

To the best of current knowledge, contributions for E-ITS and Kubernetes compliance studies are limited to master's theses, with no comprehensive peer-reviewed studies available. This gap highlights the relevance of **SRQ2**, which investigates the compatibility of existing compliance and auditing approaches with E-ITS.

Existing comparative studies of Kubernetes distributions provide useful insight into differences in security posture as measured by benchmarks. However, these studies are mostly limited to lightweight distributions, and no work has systematically compared CNCF-certified distributions in terms of security. Following NIST guidance on reducing the attack surface could provide a useful starting point for such comparisons. This aligns with **SRQ3**, which focuses on design and implementation principles for secure, auditable, and compliant Kubernetes clusters.

CIS Controls may serve as a useful foundation for mapping E-ITS measures, since existing studies and tools often use the CIS Kubernetes Benchmarks. However, their compatibility with the E-ITS needs to be analysed. Moreover, to implement PaC, controls could be integrated with the policy engine to enable automated enforcement and checks. This creates an opportunity to move beyond manual checks toward standardised and auditable verification. Studies show that the use of eBPF-based tools may help to enhance Kubernetes security and observability. However, correct configuration is essential for effectiveness. These insights provide guidance in this thesis, while the gaps motivate **PRQ** to design and implement an E-ITS–compliant Kubernetes cluster.

In conclusion, the literature lacks practical, implementation-oriented best-practice guidance for achieving and verifying E-ITS compliance in Kubernetes environments. Existing studies address individual aspects, but none analyse security, compliance, and automation holistically. This gap defines the focus of the present study.

2.6 Novelty

Prior research has addressed Kubernetes security and compliance automation, yet no study has combined these dimensions into an integrated approach. Most existing studies either

provide general security recommendations or focus narrowly on specific aspects. As E-ITS is a relatively new standard, practical implementation guidance is needed to support its adoption in cloud-native computing environments. Beyond compliance, ensuring actual security alignment is critical, as gaps may exist between E-ITS requirements and current Kubernetes security best practices.

This thesis presents a holistic implementation and a reference secure design for Kubernetes clusters, guiding practitioners in achieving and auditing E-ITS compliance. Through a practical prototype, it demonstrates how the standard's measures can be translated into Kubernetes configurations and continuously verified through automated auditing. The study also identifies and justifies suitable components, such as distribution, CNI, and tools that support implementing secure and E-ITS-compliant clusters.

3. Research Methodology

This study applied Design Science Research Methodology (DSRM), which offered a structured process to solve practical problems through artifact creation [44]. The DSRM process follows six iterative phases. The approach is qualitative in nature and is particularly suitable for information technology research [44]. In this context, the artifact was a Kubernetes cluster designed to comply with E-ITS and support automated auditing.

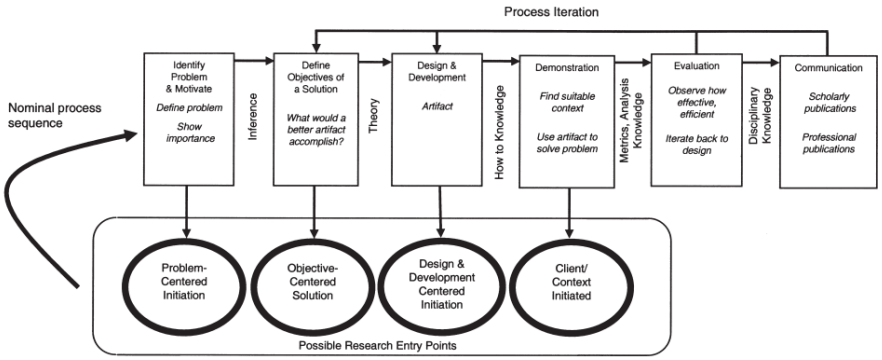

Figure 1. DSRM Process Model [44]

The DSRM process follows six iterative phases: problem identification, definition of solution objectives, design and development, demonstration, evaluation, and communication [45].

The literature review supported the first two phases of DSRM by addressing SRQ1–SRQ3. It identified Kubernetes security challenges, examined how E-ITS aligns with existing standards, and outlined principles for secure and auditable cluster design. It also revealed gaps and defined the problem space and objectives of the proposed artifact.

The third phase, *Design and Development*, selected components for the artifact. Based on the findings of the literature review, the study analysed CNCF-certified Kubernetes distributions and relevant platform components to identify configurations that best support established security best practices. Based on this analysis, a suitable Kubernetes distribution and supporting components were selected and justified, and a secure cluster architecture was designed. The resulting design was then systematically mapped to E-ITS security measures to ensure alignment with defined requirements and to support structured evaluation.

The fourth phase, *Demonstration*, demonstrated the functionality of the artifact through

deployment in a laboratory environment. The Kubernetes cluster was implemented together with its supporting components, including networking, storage, policy enforcement, and auditing mechanisms. To demonstrate practical applicability, mock workloads were deployed within the cluster to simulate real-world application deployments. These workloads provided a realistic context for testing policy enforcement, network segmentation, resource isolation, and auditing capabilities.

The fifth phase, *Evaluation*, measured the effectiveness of the artifact through three complementary approaches. The first approach was laboratory testing, which was conducted in a controlled environment to validate the technical and security behaviour of the solution. This included tests for availability and fault tolerance, backup and restore procedures, secure communication, and misconfiguration and post-compromise scenarios. The second approach was an analytical compliance assessment. The implemented solution was compared against selected E-ITS security measures. The assessment also examined how far these measures could be supported through automated evidence generation. The third approach focused on practical applicability and examined the solution in two organisational settings. Together, these evaluation approaches were used to assess the effectiveness of the artifact, discuss the evaluation results, and answer the research questions.

Finally, the *Communication* phase summarised the overall research process and its results. This phase presented the developed artifact and the key findings derived from its design, demonstration, and evaluation. It also outlined the main contributions of the research. These include the proposed design, the developed prototype, recommendations for E-ITS, and directions for future research.

4. Solution Design and Development

This chapter follows the Design and Development stage of the DSRM where the architecture of the artifact is defined. The objective of the artifact is to establish a secure and auditable Kubernetes cluster that reflects the commonly accepted security and operational best practices identified in the literature.

The literature review highlighted key challenges in Kubernetes environments, including weak default settings, misconfiguration risks, and the need for automated and repeatable compliance verification mechanisms. It also emphasised the role of the CIS Benchmarks as a practical reference for security hardening and technical control verification.

Based on these findings, the design and development process is structured into three stages. First, a Kubernetes distribution is selected as the foundation for the prototype, guided by criteria derived from the literature review and the defined research scope. Second, the components of the Kubernetes cluster are selected and integrated into the proposed architecture. Third, the designed solution is systematically mapped to the E-ITS measures to relate the architecture to defined security requirements and support structured compliance assessment.

4.1 Selection of a Kubernetes Distribution for the Prototype

The CNCF runs the Certified Kubernetes Conformance Program, which ensures that a Kubernetes distribution supports the required Application Programming Interfaces (API) [46]. This ensures that applications can run on different platforms but does not address security and compliance requirements. Therefore, choosing a distribution requires further evaluation of its security posture.

4.1.1 Inclusion and exclusion criteria

The following inclusion criteria were applied to select an appropriate Kubernetes distribution for the prototype.

Inclusion criteria:

1. Must belong to the Certified Kubernetes - Distribution category.

2. Must be open-source, with publicly available source code.
3. Must be deployable on-premises.

Exclusion criteria:

1. Lightweight distros such as k3s and k0s due to weaker benchmark scores, as identified in the literature review. These distributions target resource-constrained environments, where achieving lightweight performance may involve compromises in security.
2. Feature-oriented tools (e.g., kURL, k3k, vCluster) that enhance Kubernetes functionality rather than providing a complete distribution.

4.1.2 Comparative analysis of candidate distributions

After applying the inclusion and exclusion criteria, the following note on the selection process must be made. Rancher Kubernetes Engine 1 (RKE1) was excluded from the comparison because it reached end-of-life on 31 July 2025, as stated in the official documentation [47]. Its successor, RKE Government (RKE2), was selected for further analysis.

The author also examined Constellation, a security-focused distribution marketed as *"the world's most secure Kubernetes"* through its use of confidential computing [48]. While reviewing the distribution, the author identified a security-related configuration issue and reported it to the vendor via GitHub Issue [49]. A fix for this issue was merged into the main branch on 16 October 2025. However, on 5 January 2026, the Constellation project was archived on GitHub. Due to its archival status and the corresponding lack of long-term viability, Constellation was excluded from the selection.

As a result, the distributions included in the comparison were:

- AWS EKS-D
- Charmed Kubernetes
- Elastisys Welkin
- Flatcar Container Linux
- RKE Government (RKE2)
- Red Hat OpenShift
- Talos Linux
- SIGHUP Distribution (SD)

Building on NIST SP 800-190 [22], the selection was refined by identifying Kubernetes

distributions that rely on container-specific OS.

Red Hat OpenShift is tightly integrated with Red Hat CoreOS, an immutable OS derived from Red Hat Enterprise Linux (RHEL) [50]. Similarly, Flatcar Container Linux [51] and Talos Linux [52] are designed as single-purpose systems, providing Kubernetes on top of a container-optimised OS.

By comparison, other Kubernetes distributions support a broader range of deployment models or remain dependent on generic Linux distributions. AWS EKS-D supports container-specific system Bottlerocket along with generic Linux distributions [53]. Elastisys Welkin supports Flatcar Container Linux in addition to several general-purpose Linux distributions [54]. Distributions such as Charmed Kubernetes [55], RKE2 [56] and SIGHUP Distribution [57] rely exclusively on general-purpose OS.

The comparison of Kubernetes distributions in terms of support for container-specific operating systems is summarised in Table 2.

Table 2. Support for Container-Specific Operating Systems

Distribution	**Supports Container-Specific OS**
AWS EKS-D	Yes
Charmed Kubernetes	No
Elastisys Welkin	Yes
Flatcar Container Linux	Yes
RKE Government (RKE2)	No
Red Hat OpenShift	Yes
Talos Linux	Yes
SIGHUP Distribution (SD)	No

4.1.3 Justification of the Selected Kubernetes Distribution

Elastisys Welkin is implemented on top of Kubespray [58], which means that its Kubernetes cluster provisioning and lifecycle management are fundamentally dependent on this project. As a result, Welkin inherits both the architectural assumptions and the operational constraints of Kubespray. Although Elastisys provides user level documentation for the Welkin platform [59], limited public information is available on distribution technical implementation. This suggests that deploying and operating outside the managed context may require direct vendor involvement.

By comparison, Amazon Elastic Kubernetes Service (EKS) is a mature and widely adopted managed Kubernetes service that has proven its reliability and scalability in the public cloud. It is built on EKS-D, the upstream Kubernetes distribution on which EKS is based.

For production use outside AWS, organisations generally rely on EKS Anywhere, which builds on EKS-D and provides the necessary operational features [60]. Using EKS-D directly for production workloads without a provider is therefore questionable.

Flatcar Container Linux, OpenShift, and Talos Linux can all be considered suitable options for building a secure on-premises Kubernetes cluster, as they rely on container-specific operating systems. At the time of writing, there was no official CIS benchmark available for Flatcar Container Linux while it exists for Talos Linux and OpenShift.

From an E-ITS module SYS.1.3 Linux and Unix server perspective, Talos Linux does not provide SSH (Secure Shell) access and instead relies exclusively on an API-driven management model [61]. In addition, it does not include a package manager. All OS settings are managed declaratively and applied via the Talos API. Extensions to the system image are added declaratively at build time rather than installed dynamically, which allows the resulting system image to remain immutable and verifiable. This design reduces the attack surface and supports automated auditing of the system.

In terms of E-ITS modules SYS.1.6 Containerisation and APP.4.4 Kubernetes, Talos Linux also differs from OpenShift and Flatcar-based deployments in its handling of the etcd key-value store. In Talos, etcd is managed as a dedicated system service running outside the Kubernetes cluster rather than as a static pod. This design limits the attack surface inside the Kubernetes cluster. Additionally, Talos Linux enables etcd encryption at rest by default, which is also suggested by the E-ITS to encrypt the etcd datastore.

Based on the comparative analysis presented above, Talos Linux was selected for the prototype implementation [62].

4.2 Selection of Kubernetes Cluster Components

Since Talos Linux was selected as the Kubernetes distribution for the prototype, component selection was guided primarily by its official documentation and its published reference architecture [63]. These materials provided a baseline for identifying suitable options for networking, storage and security features. However, their recommendations were not applied directly. Instead, the proposed solutions were independently evaluated, and alternative components were selected when they offered clearer advantages.

In addition to the vendor recommendations, the design includes components that address security, resilience, and auditability requirements. These include solutions for backup, disaster recovery, secret management, and automated configuration verification, which

improve the overall reliability of the cluster and support evaluation against E-ITS requirements.

4.2.1 Cluster Networking

The CNI plugin provides connectivity in the cluster [39]. In other words, it is primarily responsible for facilitating east–west traffic between pods. Choosing an appropriate CNI is an important design choice, as migrating an existing Kubernetes cluster to a different CNI may require service downtime or, in some cases, a full cluster redeployment.

From an E-ITS perspective, the CNI shall support network isolation and policy enforcement. The E-ITS Kubernetes module also highlights the risks associated with weak default configurations. By default, pods are allowed to communicate freely with each other, cluster nodes, and external systems. Without proper restrictions, this communication can be exploited by malicious workloads to target cluster components [4].

Talos Linux installs Flannel as the default CNI [63]. However, Flannel does not support network policies and therefore cannot enforce traffic isolation. For this reason, it was not considered suitable for the prototype.

Based on the literature review, eBPF-based CNIs such as Calico and Cilium are well suited for the prototype, as they support Kubernetes Network Policies. Both CNIs are also supported by Talos Linux [64] [65]. Additionally they provide improved observability and better performance compared to traditional netfilter-based implementations (iptables and nftables). This is illustrated in Figure 2 which compares the packet-processing pipeline of standard iptables-based container networking with the eBPF-based datapath.

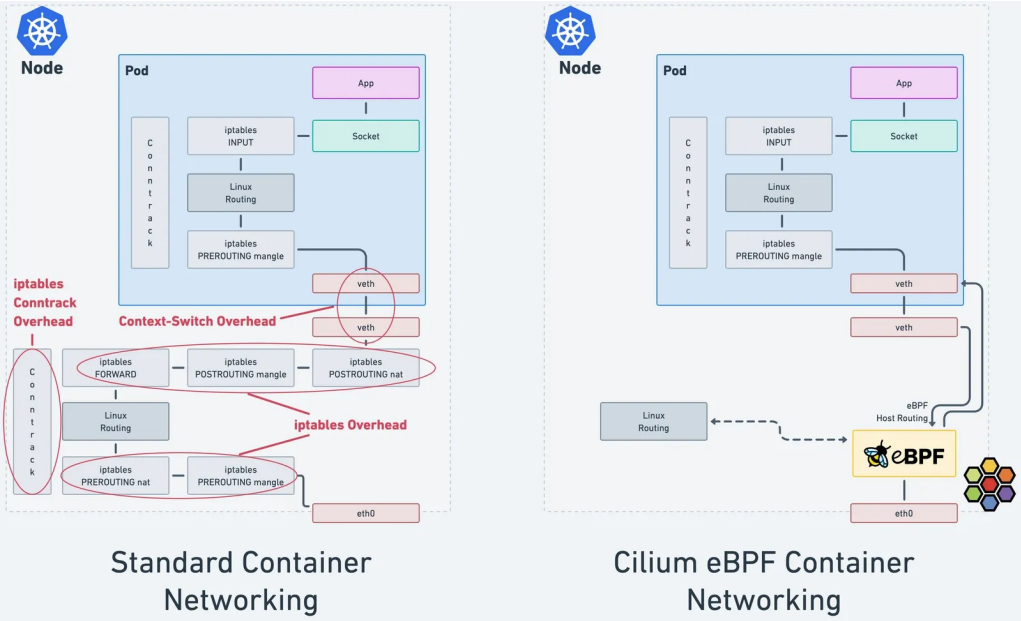

Figure 2. Comparison of iptables-based container networking and Cilium's eBPF datapath [66]

Cilium was selected as the CNI for the prototype, because it leverages eBPF technology. In addition to providing CNI functionality, it offers a broader ecosystem of complementary components, such as Hubble [67] for network observability and Tetragon [68] for runtime security monitoring. From an industry perspective, Cilium is also adopted as the default CNI in Amazon EKS Anywhere for enterprise on-premises Kubernetes deployments [69]. This further supports the suitability of Cilium as a CNI for the proposed prototype.

External Service Exposure

Kubernetes supports several types to provide access to services: ClusterIP, NodePort, LoadBalancer and ExternalName [70]. While ClusterIP and ExternalName are intended for internal or DNS-based access, NodePort and LoadBalancer are used to expose services to clients outside the cluster.

The limitations of NodePort services are that ports are exposed on all cluster nodes, which increases the attack surface by making every node accessible from outside the cluster. In addition, NodePort relies on a fixed port range, making routing, access control, and firewall rule management complex.

In cloud environments, the LoadBalancer service type is typically used by a managed cloud load balancer. However, for on-premises deployments, Kubernetes does not provide a native implementation for LoadBalancer services. Furthermore, external load balancer solutions commonly rely on network address translation (NAT). Although NAT simplifies

service exposure, it breaks end-to-end network transparency by changing the original client source address. This loss of end-to-end connectivity reduces the effectiveness of network-level security mechanisms, which are often essential.

To address these limitations, a Border Gateway Protocol (BGP)-based service exposure approach was selected for the prototype. Externally reachable service IP addresses can be advertised to the upstream network using BGP. This allows external traffic to be routed to the cluster using standard Layer 3 routing, preserving end-to-end connectivity and maintaining visibility of the original client source address. In addition, the use of BGP makes it possible to explicitly control which cluster nodes are reachable for ingress traffic by configuring BGP peering only on selected nodes.

Although the Talos reference architecture recommends MetalLB or Kube-VIP [63], Cilium provides native BGP functionality directly within the CNI datapath [71]. Using Cilium avoids introducing an additional component and enables routing control to be managed within a single integrated networking layer. For this reason, Cilium was selected for the prototype.

The use of BGP further enables equal-cost multipath routing (ECMP), allowing multiple nodes to advertise the same service address prefix. This allows ingress traffic to be distributed across several nodes at the routing layer, improving scalability and resilience.

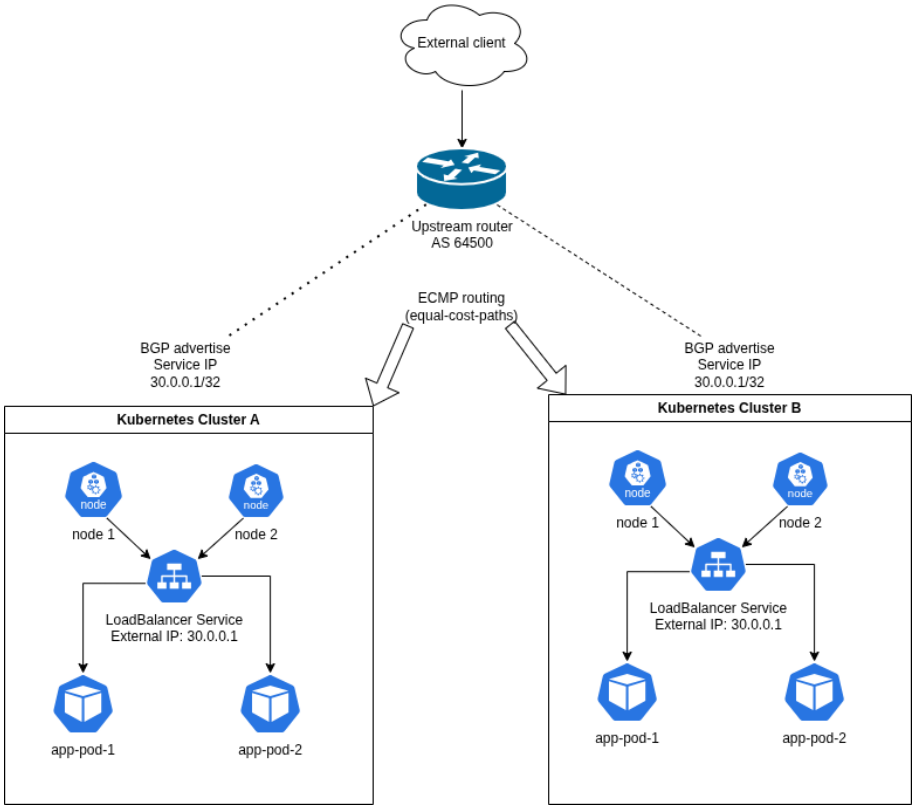

Figure 3. BGP-Based Multi-Cluster Service IP Exposure

As illustrated in Figure 3, each Kubernetes cluster node advertises the same external service IP to the upstream router. The router treats these advertisements as equal-cost paths and distributes ingress traffic across them using ECMP. A BGP-based service exposure design can support different deployment strategies and disaster-recovery testing. The same external service IP addresses can be advertised from multiple Kubernetes clusters. Traffic can then be shifted between environments by selectively controlling route advertisement. This can be done without changing DNS records or client configurations.

Ingress and Gateway API-Based Traffic Management

For north–south traffic, Kubernetes provides the Ingress resource, which enables HTTP and HTTPS routing from external clients to services within the cluster [72]. The Ingress resource defines routing rules, while the actual traffic handling is performed by an Ingress controller deployed in the cluster. However, Ingress design limitations, such as limited extensibility and limited configuration flexibility, restrict its suitability for more advanced traffic management scenarios. As of recent Kubernetes releases, the Ingress API is considered frozen by the Kubernetes project. The official Kubernetes documentation recommends considering migration to the Gateway API for new deployments. Furthermore, as of March 2026, the widely used ingress-nginx controller has reached the end-of-life [73].

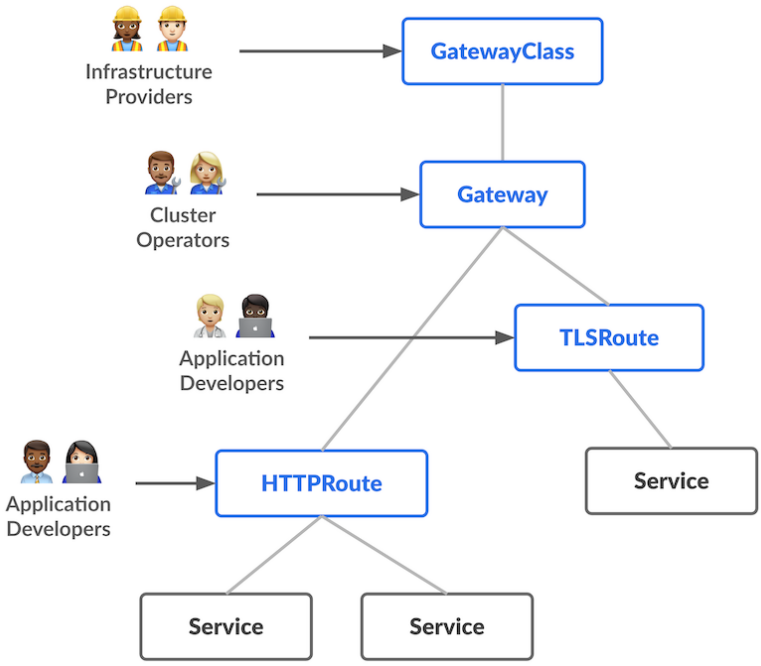

Figure 4. The Gateway API resource model [74]

The Gateway API was introduced as the next iteration of the Ingress API, providing a more extensible and role-oriented model for managing external traffic. As illustrated in Figure 4, the Gateway API defines three target personas with clearly separated responsibilities [74]. This model separates infrastructure providers, cluster operators, and application developers. Based on these characteristics, the Gateway API was selected to manage north–south traffic in the prototype implementation.

As with Ingress, the Gateway API defines only the resource model and requires a compatible controller to implement traffic handling functionality. For the prototype implementation, Traefik [75] was selected as the Gateway API controller. Traefik Proxy is widely adopted within the Kubernetes ecosystem, as reflected by its active community and more than 63,000 stars on GitHub [76]. Traefik provides middleware features that complement the Gateway API when additional request processing capabilities are required, such as IP-based allow and deny rules and rate limiting. Furthermore, the Talos Linux documentation includes configuration examples for Traefik, demonstrating its compatibility with the selected platform [77].

Service Mesh Consideration

Kubernetes as a platform does not ensure the confidentiality or integrity of communications between workloads by default. Traffic exchanged inside the cluster is neither encrypted nor authenticated unless additional mechanisms are implemented. Service meshes are

used to provide secure communication features such as mutual Transport Layer Security (mTLS), service identity, and Layer 7 traffic control. Common open-source platforms such as Istio, Linkerd, and Consul can provide such features to manage east-west traffic between workloads [78].

However, service meshes also introduce additional operational complexity, resource overhead, and management challenges that must be carefully considered [78]. Research by Bremler et al. shows that service meshes can introduce measurable performance overhead, with latency increases ranging from 8% to 166% along with substantial CPU and memory overhead [79].

As an alternative to traditional service meshes, the Cilium CNI selected in the prototype also supports identity-based mTLS implemented using eBPF [80]. At the time of writing, this functionality was classified as beta. Additionally, as discussed by Posta, the implementation of Cilium uses a node-local authentication cache that is only eventually consistent to decide whether a flow is authenticated [81]. This lack of consistency can result in unstable behaviour and potential failures in production implementations.

Recent developments in Cilium also introduce a ztunnel-based mechanism for transparent encryption and authentication between workloads [82]. This approach is conceptually similar to Istio Ambient mode, as both rely on per-node proxies to intercept and secure workload traffic. However, at the time of the prototype design, the Cilium functionality was still classified as beta and was therefore not considered sufficiently mature for production use.

Talos Linux also provides KubeSpan, which enables WireGuard-based node-to-node encryption [63]. However, KubeSpan operates at the transport layer and does not provide workload-level identity or authorisation controls. Furthermore, the Talos reference architecture notes that KubeSpan introduces encryption overhead and is not recommended for high-throughput workloads such as storage replication [63].

Taking all these factors into account, a service mesh using Istio Ambient mode was chosen for the prototype. Ambient mode provides mTLS without requiring sidecar proxies, which reduces resource consumption and latency overhead compared to traditional service meshes [83]. These characteristics align with the findings of Bremler et al., who identified Ambient mode as having the least performance impact among the evaluated solutions. Furthermore, Istio is a mature and widely adopted service mesh platform, making it suitable for production environments.

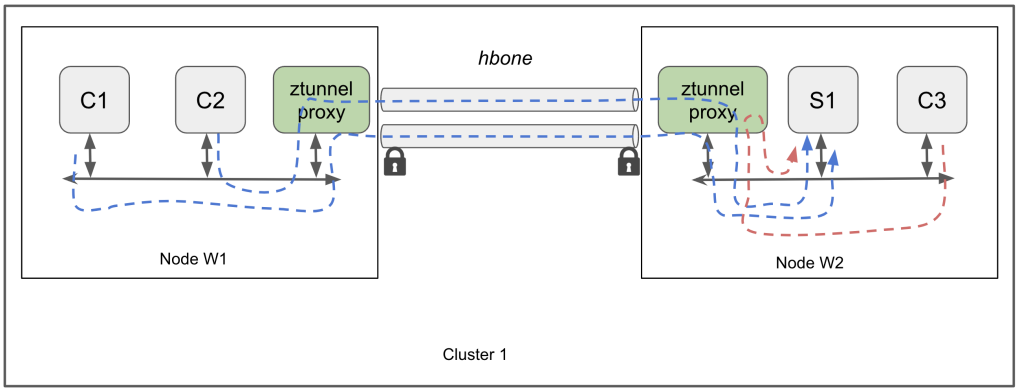

Figure 5. Istio ztunnel datapath [84]

Figure 5 presents an example deployment of Istio Ambient mode in a Kubernetes cluster, where workloads are distributed across two worker nodes. Each node (`W1` and `W2`) runs a single instance of the ztunnel proxy, which is responsible for securing traffic between workloads participating in the mesh.

In the illustrated scenario, client pods `C1`, `C2`, and `C3` communicate with a service pod `S1` through the ztunnel proxies. Traffic between workloads is protected using mTLS, and communication is handled entirely by the node-level ztunnel components without the use of sidecar containers.

4.2.2 Persistent Storage

In its early days, Kubernetes primary use-case was to run stateless applications. As demand for stateful workloads grew, the Kubernetes project introduced persistent volumes and later adopted the Container Storage Interface (CSI) [85]. The official Kubernetes CSI driver catalogue lists over 130 drivers [86]. However, the catalogue explicitly notes that the driver information listed is not validated and users must confirm the capabilities of each driver with its maintainers.

Security Considerations in CSI Driver Selection

Selecting an appropriate CSI driver for an on-premises Kubernetes cluster can be challenging. Many vendors provide CSI drivers with security or architectural limitations that require careful evaluation.

The Synology CSI driver requires privileged permissions on the storage array [87]. Similarly, the HPE CSI driver installation documentation provides example configurations that rely on administrative backend accounts (e.g., admin, 3paradm) for accessing the storage system [88].

The vSphere CSI driver requires read-only access to all ancestor objects, often extending up to the datacentre level [89]. This means that the Kubernetes cluster service account gains visibility into infrastructure resources beyond its own scope, including other VMs and datacentre components. A similar issue exists in the Proxmox CSI implementation, where it is required to have visibility for service account across all VMs and storage resources managed by the environment [90].

Taken together, these cases illustrate that CSI driver maturity and security practices vary considerably between vendors, reinforcing the need for a thorough assessment before deploying any storage provider in on-premises Kubernetes environments.

Storage Provider Selection

For the prototype, Longhorn was selected as the storage provider because it is one of the storage solutions recommended by Talos maintainers to use with Talos Linux [91]. Longhorn is open-source and does not rely on external storage, which may introduce security limitations or dependencies. Instead, it runs entirely within the Kubernetes cluster. Longhorn is comparatively simple to deploy and operate, while offering features such as volume replication, snapshots, backup, fault tolerance, and volume encryption to support data protection at rest [92]. Alternatively, Rook-Ceph could be used for the prototype, as it provides similar features. However, the Talos documentation notes that Ceph can perform poorly on small clusters and is difficult to troubleshoot without sufficient knowledge of its architecture [93].

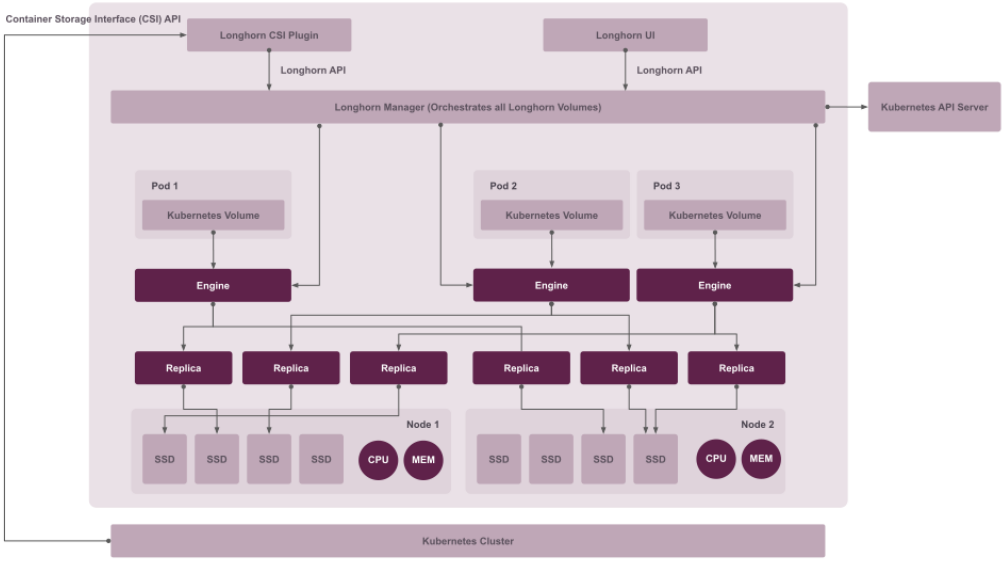

Figure 6. Concept of Longhorn distributed block storage architecture [92]

As shown in Figure 6, Longhorn avoids a single point of failure by replicating each persistent volume across multiple nodes. This design improves storage availability and

resilience in Kubernetes environments.

4.2.3 Cluster Backup

E-ITS APP.4.4 Kubernetes module suggests that Kubernetes clusters have regular backups, ensuring that all critical components necessary for restoring the cluster are included [4]. In Kubernetes, objects are stored as YAML manifests. However, not all cluster components are stateless. Among the control plane services, etcd is the stateful component. It stores the entire cluster state. For backing up and restoring, etcd snapshot can be used, as also suggested by Talos Linux documentation [94].

One limitation of the etcd snapshot–based backup approach is that it does not support granular restoration. For example, it is not possible to restore a specific Kubernetes namespace. Instead, the entire cluster state must be restored, which makes this method unsuitable for selective recovery scenarios. As a result, in practice it can be considered primarily for disaster recovery (DR) situations.

In addition, if persistent storage is used in the cluster, then etcd snapshots alone are not sufficient because they do not include the data stored on persistent volumes. Therefore, a dedicated backup tool is required to provide backups and support granular recovery of namespaces and persistent data.

Sameer et al. conducted a comparative study of Kubernetes backup and disaster recovery tools, evaluating widely used solutions [95]. Although the study evaluates tools at a high level, the authors performed implementations of Velero and Kasten K10 and highlighted these two as strong candidates based on their features, integration capabilities, and practical usability in their test environment.

As Velero is open-source, it is also used in commercial enterprise backup products such as Cohesity DataProtect [96] and NetBackup [97]. In the prototype developed for this work, Velero is used as a backup tool due to its open-source availability and broad industry adoption.

4.2.4 Monitoring and Logging Considerations

The Talos Linux reference architecture recommends Prometheus and VictoriaMetrics for collecting and storing metrics, while Grafana provides a visualisation layer for these systems [63].

For logging, Talos Linux supports forwarding system logs to external logging servers [63]. Kubernetes workloads can be integrated with established log aggregation solutions such as Loki or VictoriaLogs, enabling centralised log collection and analysis [63].

In the proposed design, monitoring and log aggregation components are integrated with external systems in order to minimise the risk of unauthorised modification and to preserve the integrity and availability of operational and audit data. This architectural separation supports the requirements of the E-ITS logging module OPS.1.1.5. Since data is transmitted to an external infrastructure, secure transport mechanisms (such as TLS) are required to prevent interception or manipulation in transit.

According to the DSRM adopted in this study, the artifact is defined as a Kubernetes cluster. Consequently, the monitoring and logging stack is treated as an external supporting component and is positioned outside the defined artifact boundary. Therefore, its detailed deployment and operational configuration are not considered within the scope of the prototype implementation.

4.2.5 Declarative Git-Based Change Management and Access Control

Kubernetes supports both declarative and imperative configuration management [98]. Although imperative access via tools such as `kubectl` is operationally convenient and sometimes necessary, uncontrolled direct modifications may introduce configuration drift, reduce traceability, and complicate the verification of compliance with defined security requirements.

To mitigate these risks, the proposed architecture implements a declarative, version-controlled configuration management model based on GitOps principles [99]. In a GitOps approach, the desired state of the system is defined as code and stored in a version-controlled repository [100]. This serves as a single source of truth for the deployed applications in the cluster. From a compliance perspective, this improves traceability, enforces controlled change management, and supports configuration integrity. Instead of granting cluster users, such as developers or application administrators, direct access to modify cluster resources, changes are introduced through the version control system.

The Talos Linux documentation and reference architecture present both Argo CD and Flux as GitOps-based application management solutions [63] [101]. Both tools follow a pull-based reconciliation model in which a controller continuously retrieves the desired state from a Git repository and reconciles it with the cluster state. Pull-based mechanisms offer advantages over push-based deployment models, as they allow the cluster itself

to verify and enforce its desired configuration state independently of external pipeline executions [102]. Additionally, pull-based mechanisms do not require direct access from external systems to the Kubernetes API server. This reduces the attack surface, as no inbound connectivity to the cluster is necessary.

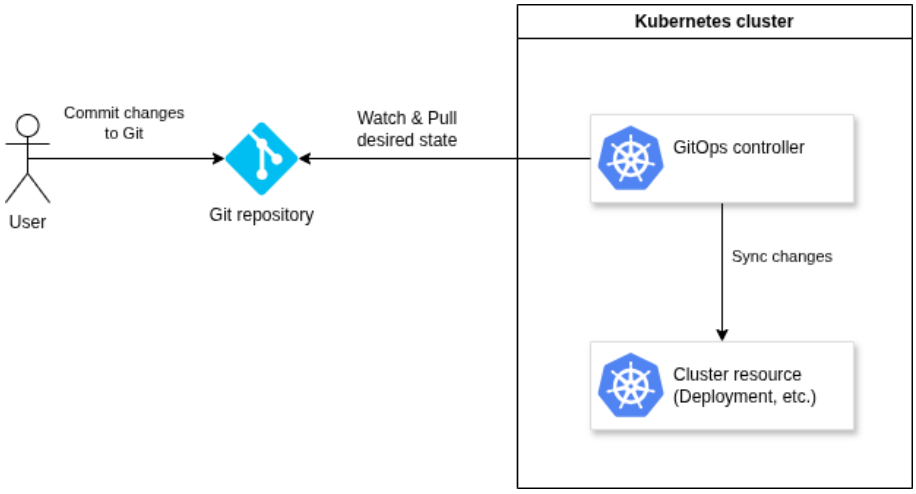

Figure 7. Pull-based GitOps reconciliation model for declarative change management in the Kubernetes cluster

In evaluating GitOps controllers for the prototype, both Argo CD and Flux were considered strong candidates. Argo CD provides a comprehensive web-based user interface for application management, whereas Flux follows a more lightweight, Kubernetes-native approach based primarily on custom resources and controllers. Based on AWS Prescriptive Guidance, Flux was selected due to its stronger multi-tenancy support, which provides improved isolation between different teams within the shared cluster environment [103].

4.2.6 Secrets Management

In Kubernetes, Secret objects are used to store sensitive data (e.g., passwords, tokens) in base64-encoded format [104]. Base64 encoding does not provide confidentiality, as the original Secret value can be trivially decoded. While Talos Linux enables encryption of secrets at rest by default, it protects only the stored cluster state and does not address the risks associated with secret management.

In a GitOps approach, the desired state of configuration is maintained in a version-controlled repository. Storing Kubernetes Secret manifests containing sensitive values in Git would expose confidential information to anyone with repository access. Therefore, an additional mechanism is required to ensure that sensitive information remains encrypted outside of the cluster. Several approaches exist to address this challenge, including the External Secrets Operator [105], Sealed Secrets [106], and file-level encryption tools

such as Mozilla Secrets OPerationS (SOPS) [107]. Each solution differs in terms of key management model, operational complexity, and integration with GitOps workflows.

The External Secrets Operator can integrate Kubernetes with many different external secret management systems and retrieve sensitive data dynamically at runtime [105]. In this model, secrets are not stored in the Git repository but instead are referenced from an external backend. While this approach reduces the risk of repository-based plaintext secret exposure, it introduces a dependency on an additional external secret management system that must be compatible with the External Secrets Operator. This dependency may increase operational complexity and, depending on the selected backend solution, can introduce additional infrastructure requirements and associated costs.

Sealed Secrets follow a different architecture and are implemented through a controller running inside the Kubernetes cluster [106]. Sensitive data is encrypted using the public key generated by the Sealed Secrets controller before being committed to the repository. The corresponding private key is stored within the cluster and is used by the controller to decrypt the sealed resource during reconciliation. The solution is Kubernetes-native and integrates seamlessly with declarative workflows, as decryption is handled transparently by the controller. However, the Sealed Secrets encryption model introduces limitations in terms of portability and resiliency. Since the key pair is generated within the cluster, encrypted secrets are bound to that specific cluster instance. In scenarios such as cluster migration, disaster recovery, or multi-cluster deployments, this tight coupling to a cluster-specific key may introduce additional operational challenges.

File-level encryption tools such as Mozilla SOPS provide a more flexible alternative. Compared with Sealed Secrets, encryption keys can be managed independently of the cluster, improving portability across environments. Additionally, SOPS supports integration with external key management services (KMS), allowing encryption keys to be stored and managed outside of the Kubernetes environment [107].

For the prototype in this study, SOPS was selected to manage secrets. This decision was primarily based on its native integration with Flux, which enables automatic decryption of SOPS encrypted manifests during reconciliation without requiring additional controllers or external dependencies.

4.2.7 Policy Enforcement and Compliance Automation

Ensuring that a Kubernetes cluster remains compliant requires mechanisms that both prevent non-compliant configurations from being deployed and continuously verify that

the actual cluster state aligns with defined security controls. The E-ITS module APP.4.4 Kubernetes emphasises the need for systematic verification of cluster configurations, and specifically requires automatic configuration audits in which the actual node, cluster, and workload settings are compared with approved reference values using compatible auditing tools [4].

Selection of a Policy Engine

Kubernetes does not enforce many security defaults such as network policies, privileged container restrictions, or resource limits. To address these gaps, one option is to use admission controllers to enforce defined security requirements before workloads are admitted into the cluster [32]. However, native Kubernetes admission controllers provide limited functionality and lack the flexibility and expressiveness required for advanced and customisable policy enforcement For this reason, dedicated policy engines such as OPA Gatekeeper, Kubewarden, and Kyverno can be used. They provide richer validation logic, policy-based mutation, reporting, and configuration audits that extend well beyond the capabilities of the built-in controllers.

Policy engines OPA Gatekeeper and Kubewarden require writing policies using the Rego policy language or policies compiled to WebAssembly. While Kyverno policies are expressed directly as standard Kubernetes YAML resources, they are intuitive for platform engineers who are already familiar with Kubernetes. Kyverno also benefits from an extensive community-maintained library of over 500 sample policies that can be reused and adapted [108].

For reporting and compliance visibility, Kyverno integrates with OpenReports. This enables policy results to be produced as Kubernetes Custom Resource Definitions (CRD), providing a standardised reporting format across tools [109]. These results can be collected and visualised using Policy Reporter, which provides a web-based user interface with an unified view of policy evaluation results [110]. Policy Reporter can additionally forward findings to external systems such as Slack, Elasticsearch, and Loki. It also supports generic webhook targets to forward the results to external systems [111]. This enables integration with systems, including Security Information and Event Management (SIEM) platforms. For example, Wazuh can collect Kubernetes-related security data via webhooks, as demonstrated for Kubernetes audit logs [112]. Taken together, these characteristics made Kyverno a practical and accessible choice for implementing policy enforcement and compliance automation in the prototype.

Continuous Scanning

To complement policy enforcement, the prototype makes use of continuous security scanning to detect vulnerabilities, and deviations from established security baselines. While Kyverno enforces and validates policies during resource admission and through background scans, it does not perform vulnerability analysis of container images or assess the security posture of cluster components at runtime.

For this purpose, the prototype integrates the Trivy Operator, which provides automated and continuous security scanning within Kubernetes clusters [113]. The operator periodically scans container images, Kubernetes workloads, and cluster configuration, producing vulnerability reports and compliance findings. This aligns with the literature review, which highlights the importance of integrating vulnerability scanning and compliance tools into Kubernetes workflows to support continuous compliance monitoring [33].

To centralise reporting, the Trivy Operator is integrated with a Policy Reporter adapter that translates scan results into the OpenReports specification [114]. This allows Trivy findings to be aggregated alongside Kyverno policy results.

In addition to Kubernetes-level findings, the proposed architecture includes an Infrastructure Audit Component responsible for evaluating node-level configuration relevant to security compliance. This component analyses the configuration of the underlying node infrastructure and generates compliance findings that are reported through the OpenReports CRDs alongside the results produced by Kubernetes-native security tools.

The overall architecture of the policy enforcement and compliance automation pipeline is illustrated in Figure 8.

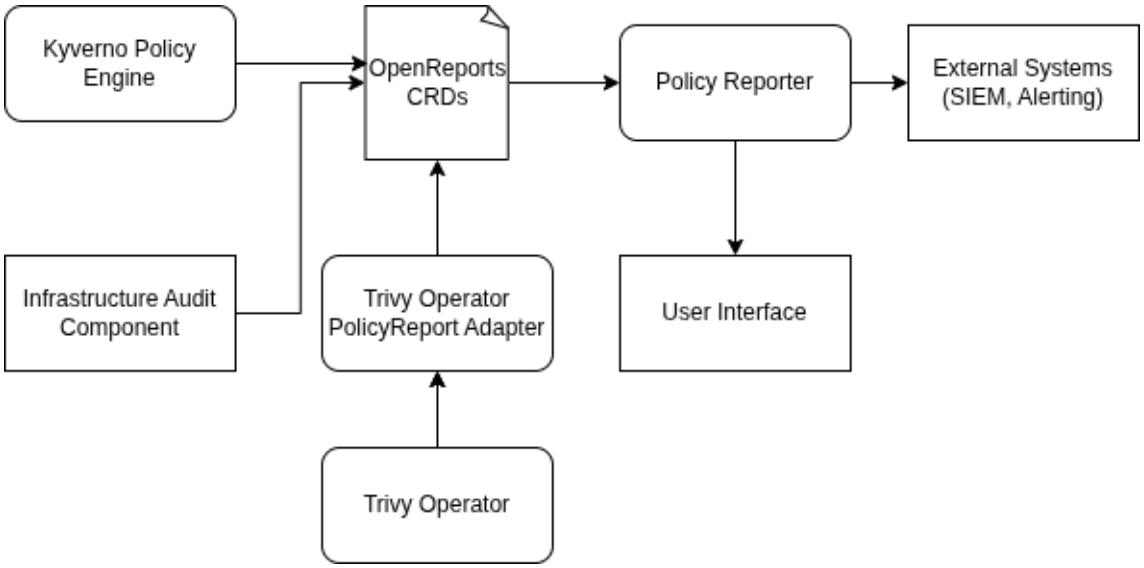

Figure 8. Conceptual architecture of the policy enforcement and compliance automation pipeline

4.3 E-ITS Security Measures in Platform Design

Once the Kubernetes distribution and supporting components were selected, the proposed platform design was systematically mapped to the E-ITS security measures defined in the SYS.1.3 (Linux and Unix servers), SYS.1.6 (Containerisation), and APP.4.4 (Kubernetes) modules. The mapping was based on both the E-ITS English version (2023) and the Estonian version (2024). The purpose of this mapping was to analyse how the architectural decisions and selected technologies address the security measures defined by the standard. Detailed mappings of individual measures and their implementation in the platform design are provided in Appendices 2, 3, and 4.

4.3.1 SYS.1.3 Linux and Unix servers module mapping

For the module SYS.1.3 (Linux and Unix servers), a substantial portion of the measures are not applicable in the proposed design. These measures primarily target traditional operating system administration practices, such as user and group management, software package installation, and interactive access mechanisms. By comparison, the proposed platform is based on Talos Linux, which is an immutable and minimal operating system designed specifically for Kubernetes. Talos Linux does not provide administrative interfaces such as SSH and does not support the direct installation of software packages on the host system. Instead, extensions are added to the operating system image before deployment. As a result, many operating system–level measures defined in SYS.1.3 are inherently eliminated. The applicable measures within this module are addressed through built-in operating system

hardening and container runtime security mechanisms.

4.3.2 SYS.1.6 Containerisation module mapping

In the module SYS.1.6 (Containerisation), the majority of measures are applicable and are designed to be implemented through Kubernetes-native and container runtime features.

Some advanced isolation and detection measures are not implemented in the initial design but could be included in future extensions of the platform. For example, host-based attack detection features are not included in the initial design. Tools such as Falco [115] and Tetragon [68] could be used to provide this functionality. However, their effective use requires carefully defined detection rules and may rely on an external logging and analysis platform to process and evaluate events. The integration of such systems is outside the scope of this work.

Hypervisor-based container isolation were not considered in the design. Container workloads are instead scheduled onto specific nodes based on their roles, providing isolation at the cluster level. More advanced isolation features, such as hypervisor-based container runtimes (e.g., Kata Containers), are not included in the initial design due to their additional resource requirements.

Certain features required for forensics, such as memory extraction from running containers, were not supported by the Talos Linux container runtime at the time of writing this work. Support for this functionality is planned for upcoming versions of the Talos Linux [116].

4.3.3 APP.4.4 Kubernetes module mapping

In the APP.4.4 Kubernetes module, the majority of measures are directly applicable and are addressed through Kubernetes-native features and policy-driven controls. The platform design uses role-based access control (RBAC), service accounts, and admission control.

Access control is implemented using least-privilege principles and a multi-tenant structure based on the Flux GitOps model. Separate service accounts are defined for workloads. The default service account is not used for application workloads, and privileged accounts are limited to components that require elevated access.

Security requirements are enforced through policies, which validate workloads during deployment. These policies ensure that workloads comply with predefined constraints,

such as restrictions on privileges, required security settings, and approved configurations.

The platform design also includes capabilities for continuous security and compliance monitoring. These capabilities enable the detection of vulnerabilities and misconfigurations in workloads and cluster resources, as well as the aggregation of policy evaluation results.

4.4 Conclusion

This chapter presented the design and development of a secure and auditable Kubernetes cluster following the Design and Development phase of the DSRM. A Kubernetes distribution and supporting components were selected to address identified security challenges, including misconfiguration risks, weak default settings, and the need for automated compliance verification. The resulting architecture was mapped to E-ITS security measures across the SYS.1.3, SYS.1.6, and APP.4.4 modules.

Talos Linux was selected as the distribution for the prototype because of its reduced attack surface as a Kubernetes-specific and container-optimised OS. Cluster components were selected to support core functionality and address security, resilience, and auditability requirements for the platform. This includes the use of Cilium CNI for networking. A BGP-based approach was selected for service exposure. Istio was introduced to provide service mesh functionality in ambient mode. The Kubernetes Gateway API was selected in combination with the Traefik Gateway Controller to manage ingress traffic. Longhorn was chosen for distributed storage, while Velero provides backup and recovery capabilities. A GitOps-based model was adopted using Flux for configuration management. SOPS was chosen for secrets management.

Policy enforcement, security scanning, and infrastructure auditing mechanisms were integrated into the design. These were supported by Kyverno and the Trivy Operator to enable automated verification of security requirements. A dedicated infrastructure audit component was included to evaluate node-level configuration and produce compliance findings.

A unified reporting model was established. OpenReports CRDs were selected as a central abstraction for aggregating compliance results across system layers. The aggregated results were exposed to the Policy Reporter for centralised reporting and visualisation.

5. Implementation

This chapter describes the implementation of the proposed Kubernetes architecture within a controlled laboratory environment, corresponding to the Demonstration phase of the DSRM. The deployment was carried out using declarative Infrastructure-as-Code (IaC) tooling and GitOps principles to ensure reproducibility, version control, and automated configuration management.

The implementation encompasses both the underlying infrastructure provisioning and the configuration of the Kubernetes platform required to operate the cluster. In addition, representative application workloads were deployed to validate the behaviour of the system and support subsequent evaluation. The complete source code for the implementation, including infrastructure definitions and cluster configuration manifests, is available in the associated public repository [117].

5.1 Underlying Infrastructure and Virtualisation Layer

The prototype Kubernetes cluster was deployed in the author's home laboratory environment. The laboratory infrastructure consisted of three small form factor (SFF) computers. Instead of installing Kubernetes directly on the underlying hardware, all physical computers were configured with Proxmox Virtual Environment (Proxmox VE) as the hypervisor platform [118]. The three computers were joined into a Proxmox cluster to enable centralised management.

The Proxmox VE was selected as the virtualisation platform because it is open source, ensuring free accessibility and the absence of proprietary licensing restrictions [118]. Furthermore, Proxmox VE has modest hardware requirements, which makes it well suited for deployment on resource-constrained SFF computers. To simulate a multi-zone deployment model, each physical host was treated as a separate availability zone (AZ). Kubernetes control plane and worker nodes were distributed across these simulated zones to model fault isolation and reduce a single point of failure. The resulting logical topology is illustrated in Figure 9.

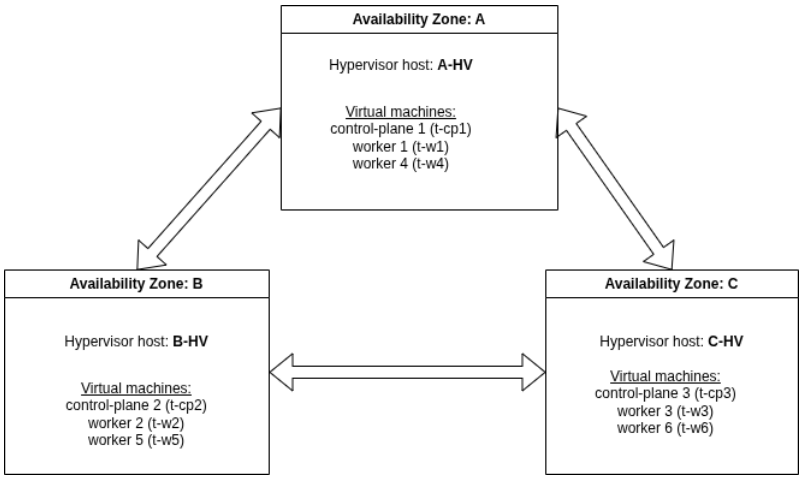

Figure 9. Simulated availability zone modelling in the laboratory environment.

By deploying Kubernetes within VMs, it was possible to simulate an enterprise environment where Kubernetes is installed on virtualised infrastructure. In addition, the use of virtualisation improved flexibility by enabling rapid provisioning and reconfiguration of cluster nodes.

The specifications of the VMs and their placement in the simulated AZs are presented in Table 3. Resource allocation was differentiated between the node roles and workload types to reflect their operational responsibilities. All VMs were provisioned with solid-state drive (SSD) storage to ensure consistent I/O performance, particularly for etcd operations and distributed storage workloads.

Table 3. VM specifications and placement of Kubernetes cluster nodes

Hostname	**Node Role**	**Workload**	**Specifications**	**AZ**
t-cp1	Control Plane	–	4 vCPU, 4 GB RAM, 100 GB SSD	A
t-cp2	Control Plane	–	4 vCPU, 4 GB RAM, 100 GB SSD	B
t-cp3	Control Plane	–	4 vCPU, 4 GB RAM, 100 GB SSD	C
t-w1	Worker	Infrastructure	4 vCPU, 8 GB RAM, 100 GB SSD (system disk), 150 GB SSD (data disk)	A
t-w2	Worker	Infrastructure	4 vCPU, 8 GB RAM, 100 GB SSD (system disk), 150 GB SSD (data disk)	B
t-w3	Worker	Infrastructure	4 vCPU, 8 GB RAM, 100 GB SSD (system disk), 150 GB SSD (data disk)	C
t-w4	Worker	Application	4 vCPU, 8 GB RAM, 100 GB SSD	A
t-w5	Worker	Application	4 vCPU, 8 GB RAM, 100 GB SSD	B
t-w6	Worker	Application	4 vCPU, 8 GB RAM, 100 GB SSD	C

The worker nodes were logically divided into two groups. Nodes `t-w1`, `t-w2`, and `t-w3`

were designated for cluster infrastructure components, while nodes `t-w4`, `t-w5`, and `t-w6` were provisioned for application workloads.

Due to hardware limitations in the laboratory environment, all cluster nodes were provisioned in accordance with the minimum resource recommendations defined in the Talos Linux documentation [119]. The allocated CPU and memory resources were sufficient to ensure stable operation of the control plane components as well as the deployed infrastructure and application workloads during testing.

To support distributed storage, infrastructure worker nodes were provisioned with two separate SSD disks: a 100 GB system disk and a 150 GB dedicated data disk. The dedicated data disk was used by Longhorn to store volume replicas and associated storage metadata. By comparison, application worker nodes were provisioned with a single 100 GB system disk, as they primarily consume persistent volumes provided by the infrastructure nodes.

5.2 Infrastructure Provisioning

Infrastructure provisioning was implemented using OpenTofu [120], following an IaC approach [121]. The overall infrastructure provisioning and deployment workflow of the laboratory environment is shown in Figure 10. The diagram illustrates the automated workflow used to provision the virtual infrastructure, bootstrap the Kubernetes cluster, and subsequently manage the cluster configuration through the Flux GitOps controller.

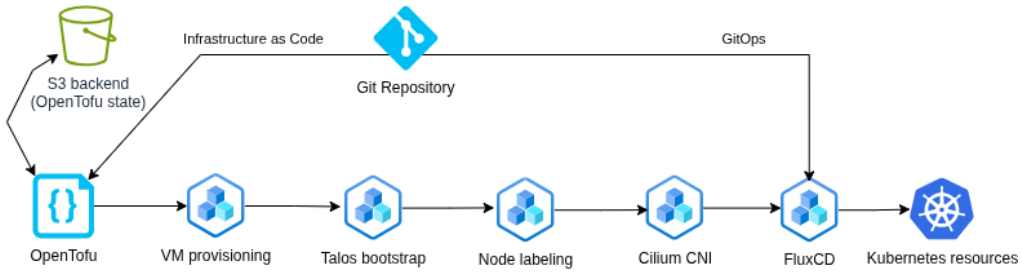

Figure 10. Automated provisioning and bootstrap workflow of the Kubernetes cluster using OpenTofu and Flux.

Proxmox VE provides a management API that enables programmatic interaction with the virtualisation platform [118]. To leverage this capability, infrastructure provisioning was automated using OpenTofu's Proxmox provider [122]. The VMs forming the Kubernetes cluster were declaratively defined and provisioned through this provider. In addition to provisioning the virtual infrastructure layer, Talos Linux nodes were provisioned, configured, and bootstrapped using the OpenTofu Talos provider [123]. This ensured that the underlying virtual infrastructure and OS configuration were reproducible and version-controlled. Because OpenTofu requires a state file to maintain the current state of managed

infrastructure resources, the state was securely stored in an S3-compatible object storage backend hosted within the author's home laboratory environment.

To enforce the separation between infrastructure and application workloads, Kubernetes node labels were applied during cluster provisioning using the OpenTofu Kubernetes provider [124]. Infrastructure worker nodes were labeled with the key `node-role.kubernetes.io/infra`, allowing cluster-level components such as storage, and controllers to be scheduled specifically on these nodes. Although Talos Linux supports defining node labels through `MachineConfig`, the Kubernetes `NodeRestriction` admission controller limits which labels can be set by worker nodes [125]. As also noted in the Talos documentation, this restriction improves cluster security by preventing worker nodes from assigning privileged labels to themselves.

The author's preferred approach was to deploy all cluster components through the GitOps controller (Flux). This provides a single source of truth for Kubernetes-level resources, while OpenTofu manages the underlying infrastructure. However, Kubernetes requires a functional networking plugin before controllers and workloads can operate. Therefore, the Cilium CNI plugin was deployed during the cluster bootstrap phase using the Helm provider for OpenTofu [126]. The initial installation provided basic cluster networking, while additional networking resources were later applied through Flux.

Talos Linux supports installing CNI manifests during the bootstrap process by referencing externally hosted manifests [127]. However, it is noted that such manifests may contain sensitive cryptographic key material. Additionally, embedding manifests directly within the bootstrap configuration can complicate future maintenance activities for networking components. For this reason, Cilium was deployed and managed declaratively as a Helm release using OpenTofu.

Once CNI plugin had been deployed and node labeling was completed, Flux was deployed using OpenTofu Flux provider [128]. This enabled infrastructure-related applications and cluster configuration manifests to be managed directly through the GitOps workflow from the initial deployment stage.

During the implementation phase, a limitation of the Talos OpenTofu provider was identified, as it does not currently support automated cluster upgrade operations [129]. Due to this limitation, cluster upgrades were performed in accordance with the official Talos documentation using the `talosctl` command-line tool.

Furthermore, because distributed storage was provisioned within the cluster, maintenance

operations required nodes to be upgraded sequentially rather than simultaneously. Updating nodes one by one ensured that storage replicas remained available and minimised the risk of service disruption or potential data loss during the upgrade process. Automated upgrade support may be added to the Talos OpenTofu provider in the future. Until such support is available, the upgrade procedure could be automated as part of future work.

5.3 Network Architecture and Segmentation

The laboratory environment was implemented using a segmented network architecture to enforce isolation between the Kubernetes node network, the administrative management network, and the servers network hosting infrastructure services. In addition, Kubernetes networking layers were also used within the cluster, including the pod network and the internal and external service networks. The overall network topology of the environment is illustrated in Figure 11.

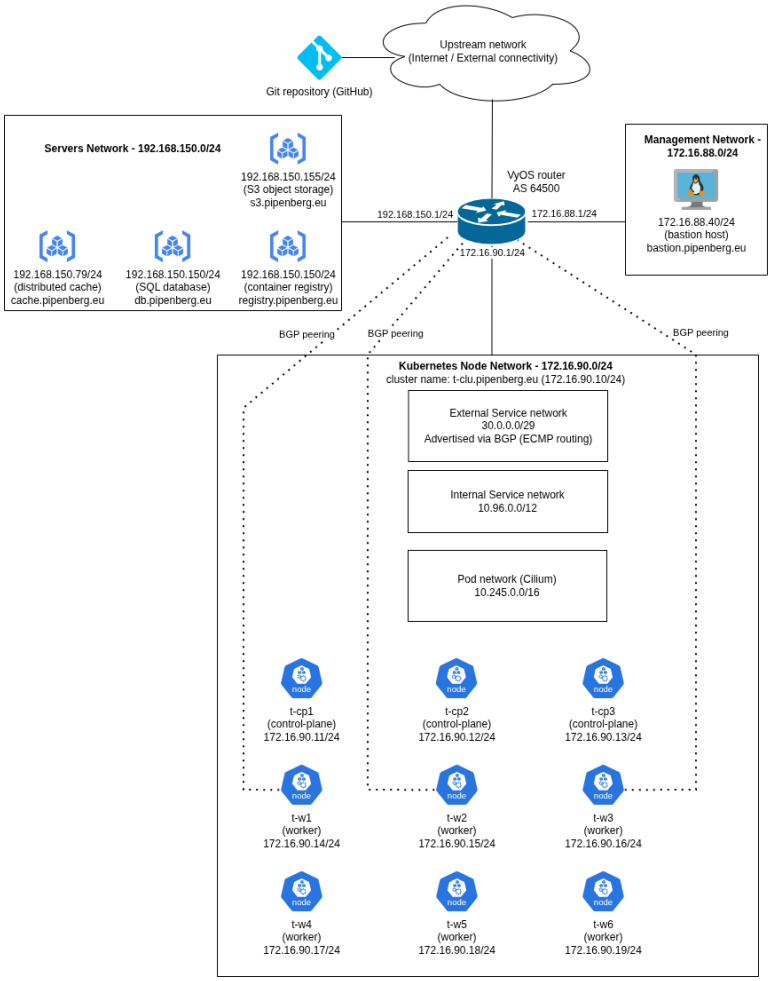

Figure 11. Laboratory network architecture

The VMs forming the laboratory environment were deployed on a Proxmox virtualisation cluster connected to the upstream network. The hypervisor hosts provide the underlying

compute infrastructure but are not explicitly shown in the network architecture figure.

To simulate a realistic network environment, a VyOS [130] router was deployed as a virtual machine acting as the edge router connecting the laboratory infrastructure to the Internet. VyOS was selected due to its open-source licensing and native support for dynamic routing protocols such as BGP. It also provides a configuration experience similar to traditional network operating systems such as Cisco IOS and Juniper Junos, including a commit-based configuration model.

The router provides routing between the segmented networks. The router also serves as the BGP peer for selected Kubernetes infrastructure nodes (`t-w1`, `t-w2`, `t-w3`). These nodes advertise externally accessible service prefixes from the 30.0.0.0/29 network. The network 30.0.0.0/29 does not belong to the private address space defined in RFC1918 [131]. It was used in the laboratory environment only for demonstration purposes to represent externally routable service addresses.

5.3.1 Supporting Infrastructure Services and External Dependencies

This thesis focused on the design and implementation of the Kubernetes platform and its compliance mechanisms. However, several supporting infrastructure services were required to enable the laboratory environment.

The supporting infrastructure services were placed in a dedicated servers network. Within this segment, a container registry server was provisioned to host container images used by the Kubernetes workloads. The registry was configured as a pull-through cache and integrated with Talos Linux, following the official documentation [132]. This allows for reduced dependency on external container registries and improves resilience in scenarios such as network disruptions or upstream rate limiting.

In addition, an S3-compatible object storage service (`s3.pipenberg.eu`), a caching service (`cache.pipenberg.eu`), and a SQL database service (`db.pipenberg.eu`) were deployed to support application workloads. These services provide auxiliary functionality but are not part of the Kubernetes platform itself.

Administrative access to the environment was provided through a bastion host (`bastion.pipenberg.eu`) located in the management network. This enabled secure management of infrastructure components while preventing direct exposure of internal services to external networks.

In accordance with the GitOps approach described earlier, the desired cluster configuration is stored in a Git repository that serves as the source of truth for the deployed infrastructure and Kubernetes resources. In production environments, the Git repository hosting the cluster configuration would typically be deployed within the organisation's internal infrastructure, for example within the servers network. However, due to the limited resources available in the laboratory environment and to allow public access to the implementation artifacts of this thesis, the repository was hosted on a public GitHub platform.

5.3.2 Cluster Network Configuration

The networking configuration implemented in the prototype used IPv4 addressing. IPv6 support was not included to reduce complexity of implementation in the laboratory environment. Extending the design to support dual-stack networking could be considered in future work.

Cluster networking for the Kubernetes cluster was implemented using the Cilium CNI. Cilium was configured to operate in native routing mode, where pod traffic was routed directly between nodes using the underlying network rather than encapsulating packets with tunnelling protocols. This approach avoided additional encapsulation overhead and preserved the maximum transmission unit (MTU) available for workload traffic [133].

The native routing networking configuration enabled Direct Server Return (DSR) for service traffic. With this configuration, response traffic is sent directly from backend pods to clients rather than through the node performing load balancing. This preserves the original client source IP address and reduces the load on infrastructure nodes [134].

Cilium was deployed in kube-proxy replacement mode, which is required for enabling DSR. In this configuration, Kubernetes service load balancing is handled directly within the eBPF datapath instead of relying on the traditional kube-proxy component. To support the service mesh functionality implemented in the cluster, the Cilium networking configuration followed the official Cilium integration guidance for Istio Ambient mode [135]. This included enabling kube-proxy replacement and applying the recommended networking configuration required for compatibility with the service mesh datapath.

An additional cluster-wide policy was implemented to allow kubelet health probes required by Istio Ambient mode [136]. Since the cluster networking configuration follows a default-deny approach, probe traffic from the reserved address used by the Cilium datapath would otherwise be blocked. This policy therefore allows the probe traffic to reach ambient workloads.

BGP Peering was configured between the Kubernetes infrastructure nodes and the VyOS router. To improve the security of routing sessions, authentication was configured for BGP peers using a shared secret. A dedicated LoadBalancer IP pool was defined for services in the `traefik` namespace, allowing the controller to advertise its service address through BGP while preventing other services from being externally exposed. The Cilium BGP configuration is available in the project repository [137].

5.3.3 Cluster-Wide Network Policy Enforcement

To restrict network traffic, cluster-wide Cilium network policies were implemented. Following a defence-in-depth approach, these policies provide traffic filtering at both the node level and the Kubernetes workload level.

Talos Linux provides a built-in firewall that allows administrators to define host-level network filtering rules directly within the operating system [138]. However, the Talos documentation notes that if another tool implements node-level network traffic filtering, it may take precedence in the `nftables` processing chain. As a result, operating system-level firewall rules may be bypassed.

To avoid maintaining multiple firewall configurations and to ensure consistent policy management, node-level traffic filtering was implemented centrally using Cilium's host firewall. This approach allowed policies to be defined and managed through Kubernetes resources rather than through operating system configuration.

The policies were stored in a Git repository and automatically reconciled with the cluster using Flux. This approach ensured that the desired network configuration remained version-controlled and reproducible. In addition, it provided a recovery mechanism in case a misconfigured network policy led to an administrative lockout. The intended configuration could be automatically restored through the reconciliation process.

Cluster-wide Cilium policies were defined to enforce host-level traffic restrictions on Kubernetes nodes and to permit only the communication required for cluster operation. Furthermore, a default-deny posture was applied to application workload traffic at the host firewall level. By default, workloads were permitted to access only the Kubernetes API server and the cluster DNS service, unless further communication flows were explicitly authorised through policy.

Hubble was used to inspect and validate network flows between workloads, supporting verification of policy behaviour and troubleshooting.

The complete configurations for the cluster-wide policies configuration is available in the project repository [139].

5.3.4 Service Mesh Configuration

Istio Ambient mode was enabled by defining the required namespace labels in the Git repository containing the cluster configuration manifests. These manifests were applied to the cluster through Flux reconciliation. Namespaces intended to participate in the service mesh were labeled with `istio.io/dataplane-mode=ambient`. When workloads were deployed in these namespaces, they automatically became part of the mesh. Communication between participating workloads was then secured using mTLS provided by the Istio Ambient data plane.

In Istio, `STRICT` mTLS mode enforces that all service-to-service communication must be authenticated and encrypted using mTLS. By default, Istio operates in `PERMISSIVE` mode, which allows both mTLS and plaintext traffic and enables gradual adoption of the service mesh. In the prototype environment, `STRICT` mode was not enforced, as the service mesh introduces additional overhead, as discussed in the design chapter. Furthermore, some workloads in the cluster may not require participation in the service mesh, as secure data exchange is already handled at the application level. In environments where service mesh participation must be enforced, policies requiring `STRICT` mTLS mode can be implemented using a Kyverno policy [140].

During testing of the prototype, it was observed that when Istio Ambient mode was enabled for namespace, traffic was redirected to destination port 15008. This behaviour occurs because the Istio Ambient data plane intercepts service-to-service communication and forwards it through the node-level ztunnel proxy. As a result, Kubernetes network policies evaluate traffic directed to the ztunnel port rather than the original application port. Consequently, the network policies needed to be updated to allow port 15008 in addition to the original application ports to ensure that service-to-service communication within the mesh was not unintentionally blocked. This behaviour is also documented in the Istio documentation [141]. Organisations enabling Istio Ambient mode should therefore take this behaviour into account when designing Kubernetes network policies.

5.4 Talos Linux Configuration

In the implementation, it was assumed that official Talos Linux images are used, and scenarios involving custom or modified images were not considered. The Talos Linux

node images were generated using the Talos Image Factory [142] with a predefined set of extensions to support the required hardware and platform functionality. These included kernel modules for hardware compatibility, storage integration, and virtualisation support.

The security configuration of the Kubernetes nodes was aligned with the recommendations of the CIS Talos Linux Benchmark [26]. Alignment with the benchmark also supported the implementation of the E-ITS security requirements defined for the prototype.

Verification of the implemented configuration against the CIS Talos Linux Benchmark was performed through inspection of the running node configuration using the `talosctl` command. The verification results were documented by comparing the running Talos node configuration with the CIS benchmark controls and recording the findings in a structured table. The results of the verification are summarised in Appendix 5. Out of 21 controls, 16 were implemented, while 5 were not implemented. Two controls were implemented using Cilium instead of native Talos functionality. These controls related to node-level firewall functionality implemented through cluster-wide Cilium network policies.

Secure Boot was not implemented in the prototype. Talos Linux nocloud images used for Proxmox do not automatically enroll Secure Boot keys. Talos documentation suggests that keys can be enrolled using an ISO [143]. However, this introduces additional complexity that conflicts with the automated provisioning approach used in this work. Additionally, the prototype runs on virtualised infrastructure. Therefore, Secure Boot was not considered essential for the first iteration, but remains a possible improvement for future work.

One of the benchmark recommendations is to enable KubeSpan to provide node-to-node traffic encryption. In the prototype environment this feature was not enabled. Instead, an Istio service mesh was deployed to support optional traffic encryption.

The benchmark also recommends forwarding kernel and service logs to a remote logging system to ensure the integrity and availability of audit data. In the prototype environment, remote log forwarding was not configured because the deployment of an external logging and monitoring infrastructure was outside the scope of the prototype implementation.

5.5 Infrastructure Applications Deployment

Infrastructure applications were deployed using the GitOps workflow implemented with Flux. The Flux multi-tenancy model was used to separate platform-level infrastructure components from tenant workloads [144]. In this model, platform administrators manage cluster infrastructure components through a dedicated platform repository, while appli-

cation workloads are deployed from separate tenant repositories. Flux reconciles these repositories independently, enabling a clear separation between platform-level configuration and tenant-managed workloads. This separation allows platform administrators to manage cluster infrastructure independently from application developers and supports controlled access management within the shared cluster environment.

Infrastructure applications were deployed using Helm [145] and managed through Flux. This approach simplified application deployment by enabling version management, configuration customisation, and controlled upgrades. The components deployed in the prototype using this approach are summarised in Table 4.

Table 4. Infrastructure applications deployed through the Flux GitOps workflow

Infrastructure Application	**Helm Releases**	**Purpose**
cert-manager	cert-manager	Automated TLS certificate management
Descheduler	descheduler	Restarts pods and optimises resource utilisation
Istio Service Mesh	base, istio-cni, istiod, ztunnel	Service mesh functionality
Kyverno	kyverno	Kubernetes policy engine
Longhorn	longhorn	Distributed persistent storage
Metrics Server	metrics-server	Cluster resource metrics collection
Policy Reporter	policy-reporter	Visualisation of policy evaluation results
Traefik	traefik	Gateway API controller for ingress traffic
Trivy Operator	trivy-operator, trivy-adapter	Security Scanning
Velero	velero	Backup and disaster recovery

To ensure correct deployment order, infrastructure applications were configured with explicit dependencies within the Flux. For instance, Policy Reporter was configured to be deployed only after Traefik using the `dependsOn` field in Flux Kustomization resources.

5.5.1 Secrets Management

Sensitive configuration values stored in the Git repository were protected using SOPS encryption. The tool `age` was used for key management, following Flux recommendations to prefer it over OpenPGP for its simplicity and improved usability [146]. Secrets were

encrypted prior to being committed to the repository and automatically decrypted by Flux during the reconciliation process.

To support multi-tenancy, separate encryption keys were generated for platform-level and tenant-level resources. Tenant-specific encryption keys were stored in encrypted form and decrypted within the cluster using a platform-level key. This ensured a secure distribution of tenant keys and their availability to Flux during reconciliation. As a result, tenant-specific secrets could be decrypted while maintaining separation between tenant environments. Using separate encryption keys also limits the blast radius of a potential compromise and prevents tenants from using the GitOps reconciliation process to access secrets outside of their own scope.

5.5.2 Longhorn Distributed Storage

Longhorn was deployed as the distributed storage solution for the prototype cluster.

Longhorn components were not included in the Istio service mesh due to the characteristics of distributed storage systems. Storage traffic operates at the block level and is sensitive to latency. Introducing service mesh proxies could therefore negatively affect performance. Instead, Longhorn was configured to use its built-in mTLS support to secure communication between components [147].

It is generally recommended to implement encryption at rest at a lower infrastructure layer to minimise performance overhead. However, for demonstration purposes, Longhorn volume encryption was enabled to simulate secure storage in environments where such lower-layer encryption is not available [148].

The configuration of Longhorn followed the recommendations outlined in the Longhorn best practices guide [149]. For example, Longhorn was configured with two replicas instead of the default three to improve I/O performance, while still providing sufficient fault tolerance for the laboratory environment.

5.5.3 Backup and Recovery

Velero was configured to protect and recover cluster resources, including Kubernetes objects and persistent volumes.

Backups were configured to be stored in an S3-compatible storage system in the laboratory

environment. This ensured that backup data was stored outside the cluster and could be used for recovery in case of failure.

Backups were scheduled to run periodically to ensure that up-to-date recovery points were available for both cluster state and application data.

To enable persistent volume backups for Longhorn, CSI snapshot support was configured in the cluster. This required the installation of the Kubernetes snapshot controller and its associated CRDs. Without this configuration, Velero would not be able to perform volume-level backups for CSI drivers such as Longhorn [150].

The Velero configuration, including backup schedules and backup target, was managed declaratively using Flux Kustomization resources as part of the GitOps workflow.

5.5.4 Policy Enforcement

Policy enforcement within the cluster was implemented using Kyverno, which was deployed as part of the infrastructure components through the GitOps workflow. Policies were derived from the community-maintained policy library and adapted to the prototype [108].

The complete set of Kyverno policies used in the prototype is available in the associated public repository [117]. All policies and resource management configurations were maintained in the Git repository and applied declaratively using Flux.

Kyverno cluster-wide policies were defined to enforce best practices and prevent misconfiguration. In particular, a policy was applied to require all containers to define CPU and memory resource requests as well as memory limits. This ensured that workloads explicitly declare their resource requirements and prevents uncontrolled resource consumption in the shared cluster environment.

Additional policies were implemented to further strengthen workload configuration and security. A mutation policy was used to automatically assign default resource requests and limits for init containers where these were not explicitly defined. To control the software supply chain, a policy was enforced to restrict container images to trusted registries.

Furthermore, a policy was applied to require the definition of liveness and readiness probes. This allowed the platform to monitor workloads and restart them if they fail. It also ensured that traffic is not routed to workloads that are not ready to handle requests.

Namespace Security and Resource Governance

In addition to policy enforcement, Kubernetes ResourceQuotas were defined for each tenant namespace to control resource consumption. These quotas limit the total amount of CPU, memory, storage, and the number of Kubernetes resources that can be used within a namespace. This ensures fair resource allocation and prevents individual tenants from exhausting shared cluster resources. Kubernetes documentation also notes that access to modifying or deleting ResourceQuota objects should be restricted so that these limits remain effective [151]. Since ResourceQuota is a namespace-scoped object, a tenant with full access within their namespace could otherwise modify or remove these limits, potentially bypassing enforced resource constraints.

To address it, an additional policy was implemented to prevent unauthorised modification or deletion of ResourceQuota objects. This policy restricts such operations exclusively to the Flux controller service account, ensuring that tenants cannot bypass enforced resource limits.

A similar approach was applied to namespace-level security controls. Since PSS profiles are enforced based on namespace labels, modification of namespace metadata introduces a potential security risk. A tenant user could alter these labels to reduce the enforced security level, for example by switching to a more permissive PSS profile, thereby allowing the deployment of privileged workloads.

To mitigate this risk, an additional Kyverno policy was implemented to restrict modifications to namespace resources. This policy denies update operations on namespaces unless they originate from the Flux controller service accounts.

5.6 Auditing Implementation

The prototype environment implements automated auditing mechanisms to verify whether the deployed cluster configuration aligns with defined security requirements and benchmark recommendations. The auditing approach combines Kubernetes-level compliance verification with analysis of the underlying infrastructure configuration. These mechanisms evaluate both Kubernetes resources and the node configuration, providing visibility into the security posture of the deployed environment.

5.6.1 Infrastructure-Level Auditing

To support automated verification of infrastructure-level security requirements, an additional OpenTofu module was developed. The module inspects the infrastructure configuration and performs rule checks based on the configuration values applied to the deployed components.

Implementing audit logic within OpenTofu ensures that configuration verification remains consistent with the infrastructure provisioning workflow. Since the module is executed during each OpenTofu apply operation, the evaluation is automatically performed whenever infrastructure changes are applied. This approach keeps the compliance results synchronised with the deployed configuration and eliminates the need for a separate scanning process.

In the prototype environment, the module was used to verify selected controls from the CIS Talos Linux Benchmark. The Talos `MachineConfig` definitions generated during cluster provisioning are passed to the auditing module, which inspects the configuration patches applied to each node. These configuration patches represent the effective operating system configuration used when provisioning the Talos Linux nodes.

Each benchmark rule is represented as a logical condition that checks whether the required configuration parameters are present in the machine configuration definitions. Based on these checks, the module derives a pass or fail result for each rule evaluated across the cluster nodes.

The evaluation results are exported by the module as a Kubernetes ConfigMap resource created during the infrastructure provisioning process. This ConfigMap stores the pass or fail status for each evaluated benchmark rule.

The results are then integrated into the cluster reporting pipeline using Flux. A Flux Kustomization references the generated ConfigMap and substitutes the rule evaluation results into an OpenReports ClusterReport custom resource. The generated OpenReports resources are collected and visualised using Policy Reporter, which provides a user interface for viewing compliance results within the cluster. The overall workflow of the infrastructure-level auditing and reporting pipeline is illustrated in Figure 12.

Figure 12. Infrastructure-level auditing and reporting workflow

The audit results at the node-level for the prototype environment, visualised in the Policy Reporter, are provided in the Appendix 6. Reverse path filtering was intentionally disabled as part of the network design described earlier, and remote logging was not implemented within the scope of the prototype. Consequently, the corresponding checks appear as failed in the generated compliance report.

Talos Linux also provides a feature to expose its API so that it can be accessed within the Kubernetes cluster [152]. This approach was initially considered as a potential method for retrieving node configuration data for the auditing process. However, the read-only role available for this interface does not provide access to the node configuration files required for verifying several security controls. Granting administrative access would expose privileged API credentials within the Kubernetes cluster, which could introduce additional security risks. Therefore, this approach was not used in the prototype implementation.

5.6.2 Kubernetes-Level Auditing

Kubernetes-level auditing was implemented using the same reporting model as infrastructure-level auditing. OpenReports CRDs served as the core abstraction for representing compliance results within the cluster. This ensured a unified method for collecting and visualising audit findings across different layers of the system.

Auditing of configuration compliance was achieved using Kyverno policy reports generated from the policies implemented in the prototype. These reports provided insight into both compliant and non-compliant resources, allowing misconfigurations to be identified.

In addition, the Trivy Operator was deployed to perform automated security scanning and benchmark evaluation. For instance, it was used to assess cluster configuration against CIS Kubernetes Benchmarks and to identify vulnerabilities in deployed workloads. The results were stored as Kubernetes custom resources and translated into the OpenReports format, enabling their integration into the unified reporting pipeline. All findings were aggregated and visualised through Policy Reporter. providing a consolidated view of policy compliance and overall security posture

It should be noted that the reported results also include findings related to infrastructure

applications deployed within the cluster. As a result, certain controls produce a higher number of failed checks, which do not necessarily indicate misconfigurations. Instead, they may reflect the functional requirements of these components. For example, controls such as "Minimize access to secrets" may report multiple failures due to the use of cluster-scoped roles by system components (e.g., cert-manager), which inherently require broader permissions to operate correctly.

5.7 Deployment of Workloads on the Prototype Platform

To demonstrate the practical operation of the platform, mock applications were deployed within the Kubernetes cluster using Helm charts managed through the GitOps workflow. Two separate GitHub repositories were created to simulate independent application teams deploying workloads to the shared cluster environment [153] [154]. Each repository contained the required Kubernetes manifests to deploy the applications.

The workloads were deployed into separate namespaces to represent distinct tenants within the cluster. Each tenant corresponded to an independent application team, with isolated resources and configuration managed through separate Git repositories.

5.7.1 Deployed Applications

The first repository deployed the Google Online Boutique demonstration application [155], which implements a microservices-based architecture composed of multiple independent services communicating through APIs. This workload represents a distributed application typical of modern cloud-native environments, as illustrated in Figure 13.

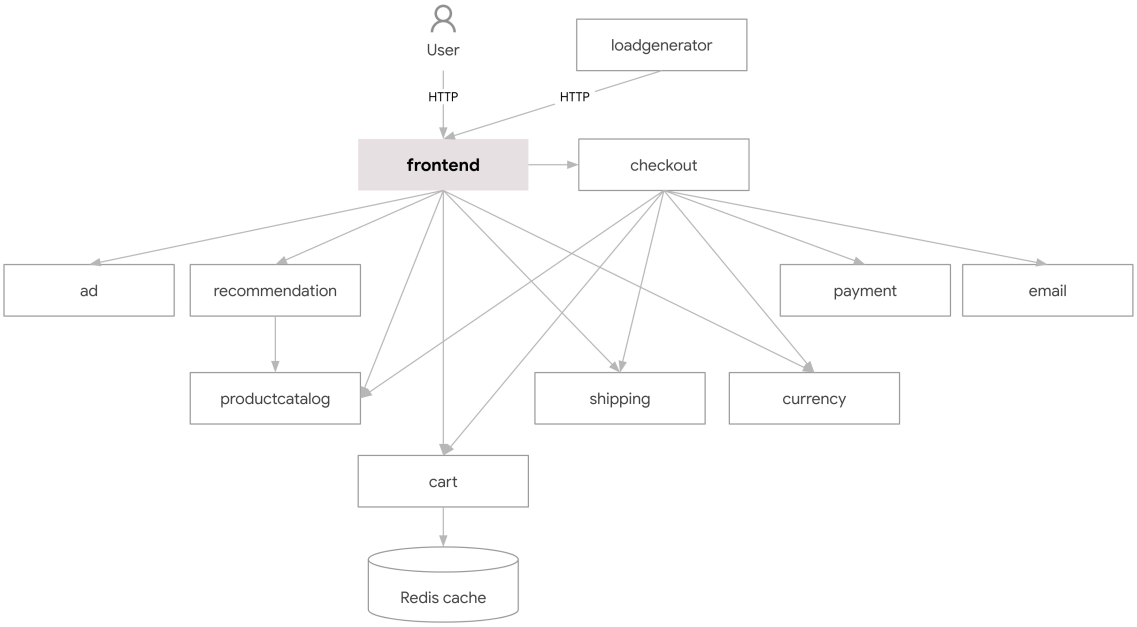

Figure 13. Architecture of the Google Online Boutique microservices demo application [155]

It is noted that the provided Helm chart did not include horizontal scaling configurations, and therefore the deployments were executed with a single replica. In production environments, such services would typically be deployed with multiple replicas to ensure high availability and resilience.

The second repository deployed the Nextcloud application using a community-maintained Helm chart [156]. In contrast to the microservices-based application, Nextcloud represented a stateful application deployed within the cluster. The application was configured to run with two replicas using shared distributed persistent storage, which enabled high availability of the application layer. To improve resilience, pod anti-affinity rules were defined to ensure that replicas were scheduled across nodes located in different availability zones. This reduced the risk of service disruption in the event of node or zone-level failures and supported the overall high availability objectives of the platform.

Due to limitations in the Helm chart configuration, the database and caching services were deployed as external services outside the Kubernetes cluster for simplification. It should be noted that distributed persistent storage may not be optimal for database workloads, which typically benefit from local storage in terms of performance. Additionally, database backup strategies require application-level consistency, which is not guaranteed by volume-level backups alone.

5.7.2 Networking and Security Configuration

Traefik was deployed as an infrastructure component to act as the Gateway API controller for ingress traffic. Both applications were exposed externally using the Kubernetes Gateway API with HTTPRoute resources, enabling controlled access. Istio Ambient Mode was enabled to provide transparent service-to-service communication. Furthermore, Istio `STRICT` mode was enabled to enforce mTLS for all service-to-service communication within the mesh.

As the platform was configured to deny network ingress and egress by default, explicit network policies were defined for each application to permit only the required communication paths. Although the Helm charts provided network policy templates, these templates were not enabled. This was because they did not account for the additional communication requirements introduced by Istio Ambient Mode.

5.8 Conclusion

The proposed Kubernetes architecture was successfully implemented in the laboratory environment using IaC and GitOps principles. The underlying infrastructure was provisioned using OpenTofu. Three availability zones were configured in the laboratory environment to support the simulation of fault isolation and high availability. Kubernetes platform configuration and application workloads were managed using Flux through a GitOps-based reconciliation process.

Key platform components were configured to support the defined architectural requirements. Cilium was configured to support host-level traffic filtering and a default-deny network policy model for workload communication, and a BGP-based approach was used for service exposure. Representative application workloads were deployed to simulate a multi-tenant environment, supported by separate Git repositories to support logical separation of responsibilities.

To support auditability and compliance verification, additional mechanisms were implemented. Kyverno policies were used to enforce security requirements and support auditing. OpenReports CRDs were used to represent structured policy evaluation results, while Policy Reporter provided a unified view for reviewing policy outcomes. In addition, a dedicated OpenTofu module was developed to audit Talos Linux configuration against the CIS benchmark, enabling structured verification of node-level security settings.

6. Evaluation

This chapter presents the evaluation phase of DSRM assessing the developed solution in relation to the research objectives. The evaluation focuses on assessing the solution's functionality, security, and its ability to support automated verification of E-ITS requirements.

The evaluation is structured in three parts. First, the solution is tested in a laboratory environment to assess its behaviour under controlled conditions using representative workloads. Second, its compliance with E-ITS measures is analysed, with emphasis on the applicability of the measures to the prototype and the extent to which they can be verified through automated auditing. Third, the applicability of the solution is examined in organisational contexts.

The results are then discussed to interpret the findings, identify limitations, and provide answers to the research questions.

6.1 Laboratory Evaluation

Laboratory testing was used to validate the proposed prototype and verify its technical correctness, fault tolerance, and security behaviour in a controlled and reproducible environment. This approach enabled systematic experimentation while avoiding operational risks and the potential exposure of sensitive organisational information. The evaluation was performed using two representative applications introduced in Chapter 5: the Google Online Boutique microservices application and Nextcloud. These were deployed within the same prototype cluster in separate namespaces to simulate independent teams operating within a shared environment.

To ensure that the evaluation covered relevant security threats, the laboratory scenarios were mapped to selected tactics from the Microsoft Kubernetes Threat Matrix [11]. These included initial access, execution, privilege escalation, lateral movement, discovery and impact. The matrix provided a structured classification of these tactics, enabling a systematic interpretation of the evaluation results.

While many security-relevant threats could be indirectly assessed through aggregated policy reporting results produced by the prototype, practical laboratory evaluation remained essential. It enabled the validation of system behaviour under realistic scenarios, including

misconfigurations, failure conditions, and attempted policy violations, which were not fully captured by automated compliance checks alone.

6.1.1 Deployment Observations

Both application namespaces were initially created by the platform to enforce the `restricted` PSS profile. However, it was observed that the Nextcloud deployment required root privileges during initialisation, making it incompatible with the `restricted` profile level. This was due to the initialisation process requiring file copy operations with elevated privileges. Consequently, the namespace was temporarily adjusted to the `baseline` level to allow successful deployment.

Following this, additional configuration changes were applied to make the Nextcloud deployment compatible with the `restricted` profile, allowing the stricter policy to be re-enabled. While modifying the application's Dockerfile to align with stricter security requirements would be the preferred approach, this is not always feasible when relying on third-party software. This highlighted the practical challenges of enforcing strict security policies in organisational environments, where platform operators must balance security best practices with application compatibility.

Furthermore, the platform enforces the definition of resource requests and limits for all workloads as part of its baseline configuration requirements. Pods that do not define these constraints are rejected during deployment. As a result, explicit resource requests and limits were defined for the application containers. This ensures that workloads operate within controlled resource boundaries and prevents uncontrolled resource consumption.

It was also observed that both Helm chart deployments did not support defining resource requests and limits for initContainers. While this would typically require manual modification, the platform mitigated this gap through a Kyverno mutation policy, which automatically applied default resource requests and limits to initContainers.

6.1.2 Availability and Fault Tolerance Test

To evaluate the resilience of the platform, a failure scenario was simulated by disconnecting the network connectivity with availability zone in the laboratory environment. To represent a worst-case scenario, at least one pod replica of each application was scheduled in the affected zone. A distributed storage replica was also placed in the same zone prior to the test. The initial placement of workloads across nodes is shown in Appendix 7.

During the test, workloads became temporarily unavailable due to the loss of the zone. However, the platform automatically recovered as Kubernetes scheduled replacement pods on the remaining nodes. As shown in Appendix 7, the original pods on the affected nodes remained in the `Terminating` state, while traffic was redirected to the healthy replicas. The Nextcloud application recovered in two minutes, while the Online Boutique application recovered in three minutes.

The observed difference in recovery time may be attributed to the Nextcloud deployment being configured with two replicas, allowing faster restoration of service availability. It was also observed that pods running on unreachable nodes remained in the *Terminating* state, which reflects expected Kubernetes behaviour. This condition can be resolved by applying node taint `node.kubernetes.io/out-of-service:NoExecute` [157].

6.1.3 Restore Test

A data loss scenario was simulated by deleting the namespace hosting the Nextcloud application. This action resulted in the removal of all associated Kubernetes resources, including persistent volumes.

The restore test was performed using Velero by restoring a previously created backup of the namespace. As a result, the application and its associated resources were successfully recreated within the cluster.

The success of the restore operation was validated by confirming that the Nextcloud application became operational after restoration. Data integrity was verified by comparing the SHA-512 checksum of a file uploaded prior to the test with the checksum obtained after restoration. As shown in Appendix 8, the checksum remained identical before and after the restore operation, confirming that application data was fully preserved.

6.1.4 Secure Communication Verification

Secure communication within the prototype was verified by initiating an HTTP request between a Nextcloud pod and the frontend pod of the Google Online Boutique application. The communication occurred within the Istio service mesh. The corresponding network traffic was then captured for analysis.

Packet capture was performed using Talos-provided `tcpdump` [158] via `talosctl`. The captured traffic was analysed using `tshark` [159] to identify TLS traffic. The packet trace

showed TLSv1.3 messages exchanged between the pods, confirming that service-to-service communication was encrypted. This suggested that the Istio service mesh was correctly configured to provide secure communication between services.

Detailed command outputs used in this verification are provided in Appendix 9.

6.1.5 Misconfiguration and Post-Compromise Isolation Scenario

To evaluate the robustness of the platform under adverse conditions, a simulated post-compromise scenario was conducted. It was assumed that a threat actor gained shell access to the Nextcloud container through a web application vulnerability.

For the purpose of this experiment, the Nextcloud namespace was intentionally reconfigured back to the `baseline` PSS enforcement level to simulate a less restrictive security configuration and to evaluate potential attack scenarios. This setup enabled the analysis of lateral movement and privilege escalation risks under weakened policy constraints. The application was executed with root privileges as defined by the default Helm chart configuration.

Furthermore, a service account used for automated configuration reconciliation was mounted to the Nextcloud pod, granting full access to resources within the namespace. This configuration deviated from the principle of least privilege, as the assigned permissions exceeded those required for the application's normal operation, thereby increasing the risk of privilege escalation.

Although the application was originally deployed with restrictive network policies, an additional policy was introduced to allow unrestricted egress traffic for the Nextcloud namespace. This change was permitted by the service account used for automated reconciliation, which had sufficient privileges to modify network policies within the namespace.

In addition, the service mesh functionality was disabled in the cluster during this experiment. This was necessary because the transparent proxying behaviour of the data plane affects port scanning results, resulting in misleading indications of port accessibility, as connections may be accepted by the proxy rather than the target workload.

Scheduling Restriction Test

As part of the post-compromise scenario, the compromised pod attempted to bypass node isolation controls by scheduling a workload to the infrastructure-designated nodes. This

was simulated by creating a pod with Kubernetes tolerations that allow scheduling onto nodes reserved for infrastructure components (see Appendix 10).

The scheduling request was denied by the policy enforcement mechanism. Specifically, the policy engine rejected the pod due to a violation of the policy restricting the use of tolerations that allow scheduling onto infrastructure nodes.

This demonstrated that, despite elevated privileges within the namespace, critical cluster-level isolation controls remained enforced. The platform successfully prevented workloads from being scheduled onto infrastructure nodes, thereby protecting sensitive system components from potentially compromised tenants.

Privilege Escalation Resistance Test

In this scenario, the compromised pod attempted to modify namespace-level security controls by updating the PSS enforcement level (see Appendix 11).

The request was denied by the admission control mechanism. The policy enforcement layer restricted modifications to namespace configuration, allowing only explicitly approved service accounts to perform such operations.

This control is critical because changing the PSS level to privileged would permit the deployment of workloads with elevated privileges. Such workloads could enable access to host resources, and this can lead to node-level compromise.

The results demonstrated that, despite the compromised workload having elevated permissions within the namespace, critical security configurations remained protected. The platform successfully preserved the enforced security posture.

Network scanning

A network scanning test was conducted from a compromised Nextcloud pod. The *nmap* [160] tool was installed within the pod and used to scan cluster nodes and the pod network. The scan targeted all TCP ports. This was based on the assumption that all services in the lab environment were communicating over TCP. Kubernetes DNS was the only service identified as using UDP and was therefore excluded from the scan scope. Due to the size of the Kubernetes pod network, the scope of the scan was limited to pods present in the cluster at the time of testing. To verify external access to the cluster, an additional network scan was performed from the laboratory router, targeting the node network and externally exposed services.

During the scans, it was observed that several infrastructure applications, including Longhorn and Policy Reporter, had overly permissive default network policy configurations that allowed unintended access. Consequently, additional network policy restrictions were introduced to limit such exposure.

Furthermore, a misconfiguration of the Traefik service was identified. The service was configured as a LoadBalancer. By default, this also exposes a NodePort, which makes the service accessible on all cluster nodes. While it was assumed that the host firewall would block traffic to NodePorts, it did not do so. It was observed that the current implementation of the Cilium CNI host firewall does not block Kubernetes NodePort traffic. This behaviour has also been reported as an issue in Cilium GitHub [161]. Although Traefik was intended to be publicly accessible, this configuration introduced a potential risk: future services using NodePort could become unintentionally exposed. Since NodePort was not intended to be used for service exposure in the prototype, the Traefik service was reconfigured. In addition, a Kyverno policy was introduced to restrict the use of NodePort services and prevent similar misconfigurations in the future.

Overall, the scan results indicate that the prototype follows the intended default-deny network policy model, while also highlighting specific components that required additional hardening. These results demonstrated that unintended exposure could occur due to default configurations and misconfigurations, even in a hardened environment. This highlighted the importance of a layered security approach, where multiple controls, including policy enforcement and network validation, are used to reduce risk.

6.2 E-ITS Compliance Evaluation

Based on the design mapping presented in Chapter 4, the implemented prototype was evaluated to assess how effectively the selected E-ITS security measures were satisfied in practice. The evaluation focused on analysing the deployed system and verifying whether the requirements defined by each measure were met. Detailed results for individual measures are provided in Appendices 12, 13, and 14.

Each measure was classified as achieved, partially achieved, not achieved, or not applicable, based on the extent to which its requirements were satisfied.

A measure was classified as achieved when its requirements were fully satisfied by the implemented solution. Measures were classified as partially achieved when only some aspects of the requirements were fulfilled or when additional improvements were identified. Measures were classified as not achieved when the required requirements were not

implemented in the current design. Measures were classified as not applicable when their requirements did not apply to the proposed architecture due to its design characteristics, such as the use of an immutable operating system.

The results summarised in Table 5 indicate clear differences across the evaluated modules. In the SYS.1.3 module, the majority of measures (8 out of 10) were classified as not applicable. This reflects the use of an immutable operating system, which eliminates several traditional system administration controls assumed by the standard. In contrast, the SYS.1.6 and APP.4.4 modules demonstrate a higher proportion of achieved and partially achieved measures. In SYS.1.6, many measures are partially achieved because they require organisational and process-level aspects outside the prototype scope. The three not achieved measures relate to advanced host-based detection, isolation mechanisms, and container evidence collection. The detection and isolation mechanisms were excluded from the initial design due to their complexity and scope limitations. At the time of implementation, Talos Linux did not support the required kernel configuration for Checkpoint/Restore in Userspace (CRIU) [116]. This limited the fulfilment of the evidence collection measure.

Table 5. Summary of evaluation results of E-ITS measures across modules

Module	**Achieved**	**Partially achieved**	**Not achieved**	**Not applicable**	**Total**
SYS.1.3	1	1	0	8	10
SYS.1.6	15	8	3	0	26
APP.4.4	18	3	0	0	21

The results in Table 6 show that only a subset of measures can be supported by aggregated policy reporting results. The percentages were calculated based on applicable measures, excluding those classified as not applicable. The proportion varies across modules, with complete coverage in SYS.1.3 and lower levels in SYS.1.6 and APP.4.4.

Table 6. Proportion of measures supported by aggregated policy reporting results

Module	**Measures validation supported by aggregated policy reporting results**
SYS.1.3	100%
SYS.1.6	54%
APP.4.4	24%

Measures supported by policy reporting are primarily associated with technical controls that can be directly observed from system configuration and runtime state. However, measures related to architectural design decisions or organisational processes require analytical assessment and cannot be fully supported by system-generated evidence alone.

The full coverage in SYS.1.3 should be interpreted with caution, as it is based on only two applicable measures due to the use of an immutable operating system. However, as

described in Chapter 5, the developed OpenTofu audit module made it relatively easy to implement automated checks for infrastructure-level controls. This indicates that infrastructure-level measures are technically easier to verify than measures in Containerisation and Kubernetes modules.

Although SYS.1.6 and APP.4.4 exhibit lower proportions, there is potential to increase the coverage of evidence-supported measures. This could be achieved by introducing additional policy definitions, for example through tools such as Kyverno, to extend verification capabilities. For instance, the measure APP.4.4.M5 (Backup of cluster information) could be addressed by defining a policy to verify the presence of backups.

The lower proportion of APP.4.4 measures supported through automated evidence generation also reflects the complexity and architectural flexibility of Kubernetes. As discussed in the literature review, Kubernetes exposes a large number of configuration options. Kubernetes security objectives can be achieved through multiple technically valid implementation approaches depending on the selected tooling and architectural design. For example, the current prototype supports multiple approaches for implementing network-level isolation, including Kubernetes NetworkPolicy, Cilium NetworkPolicy, and Cilium ClusterwideNetworkPolicy resources. In addition, the APP.4.4 module contains several conceptual, architectural, and process-oriented measures that are more difficult to verify through deterministic policy-based checks alone. For instance, a check can verify that network policies are applied to a pod. However, it cannot assess whether the policy rules match the pod's expected network communication pattern unless sufficient workload-specific context is provided.

6.3 Evaluation in Organisational Contexts

The proposed prototype was fully implemented in Organisation A and is currently in the implementation phase in Organisation B as part of an ongoing migration process. Both organisations are based in Estonia and employ over 250 people, representing medium-to-large enterprise environments with established operational and security requirements.

Prior to this work, both organisations operated vanilla Kubernetes clusters on general-purpose operating systems, reflecting a more traditional cluster management approach. The implementations relied on Ansible, used in an imperative manner. This approach, while flexible, does not inherently enforce a declarative desired state model and may introduce challenges related to configuration drift. Both environments were also hosted on virtualised infrastructure, consistent with the deployment model of the proposed prototype.

6.3.1 Organisation A

For Organisation A, the primary focus was on reducing operational overhead. In the existing setup, the use of general-purpose operating systems introduced unnecessary system components, increasing the system footprint and prolonging maintenance windows for updates and patching. In addition, the presence of unused components increased system complexity and, in at least one instance, contributed to an incident.

The implementation of the proposed prototype enabled shorter maintenance windows and more predictable and faster deployment processes. The use of the GitOps model for cluster administration was aligned with the organisation's primary objectives of improving operational efficiency and reducing manual effort.

A key consideration for Organisation A is the adoption of Talos Linux as a new technology. Although the organisation has strong expertise in Linux system administration, Talos Linux introduces a different operational model that does not provide shell or SSH access. Instead, all management operations are performed through an API-driven interface, requiring changes to established operational practices. This has also affected existing integration processes. For example, the CMDB used for asset inventory relied on SSH-based OS discovery. This is not supported in the Talos Linux environment and requires additional development.

The introduction of enforced policies required users to pay more attention to security and configuration requirements during development and deployment. The platform enforced predefined policies, meaning that workloads could not be deployed unless they met the required compliance and security criteria. While this introduced some additional effort for users, it encouraged correct practices and improved the overall security posture.

6.3.2 Organisation B

Organisation B represents an ongoing implementation scenario, where the proposed prototype is currently being introduced as part of a migration process.

In this environment, multiple partners are involved, which creates a strong requirement for workload and access isolation within Kubernetes clusters. To address this, the proposed GitOps-based approach, together with the Flux multi-tenancy model, provides a structured mechanism for separating responsibilities and enforcing isolation between teams.

Prior to this work, deployments were managed through a CI/CD platform, where Kubernetes access was configured by providing service account kubeconfig credentials. This approach required exposing the Kubernetes API to external systems, increasing operational and security considerations. In contrast, the use of Flux eliminates the need for external systems to directly access the Kubernetes API, as changes are reconciled from within the cluster.

Organisation B aims to implement a fully GitOps-based model using Flux. However, a key consideration is whether all partners have sufficient knowledge and experience to effectively work within this model.

Although Flux was selected as the controller in the proposed prototype, this does not strictly limit the implementation to a single tool. The GitOps approach can be applied more broadly, where infrastructure components may be managed using Flux, while other tools can be used for application-level deployments if required.

6.3.3 Cross-Organisational Observations

Across both organisations, the aggregated reporting functionality of the prototype provided value by improving visibility into compliance and configuration status. This was particularly beneficial for users with limited Kubernetes expertise, as it enabled them to assess the system state without requiring detailed technical knowledge.

In both organisations, a service mesh was not included in the implementation, as it was not considered to provide sufficient value relative to the additional operational complexity. However, the platform was designed and configured to allow the integration of a service mesh in the future if required. In the case of Organisation A, a service mesh could provide additional value by supporting integration between traditional virtual machine-based environments and Kubernetes workloads.

Both organisations currently utilise the Traefik ingress controller for exposing services and are migrating towards the Gateway API to achieve a more standardised and secure traffic management model.

6.4 Discussion and Limitations

The results show that the proposed architecture enables a high degree of automation and supports compliance verification through policy-based controls. Policies are used to both

enforce and audit security requirements, mitigating insecure default configurations. The use of an immutable, container-specific operating system minimises the attack surface. In combination with GitOps principles, it reduces the risk of configuration drift and improves auditability. This ensures that system states can be consistently reproduced, enhancing overall system resilience. These architectural decisions also introduce operational trade-offs that may require additional consideration in production environments. As observed in Organisation A, Talos Linux introduces a different operational model without direct shell or SSH access, requiring changes to established operational practices. Furthermore, Organisation B aims to implement a fully GitOps-based model, which requires sufficient experience with GitOps tools.

In the prototype, aggregated reports from both infrastructure and Kubernetes layers provide evidence for compliance validation. However, these results must be interpreted in context. For example, several infrastructure components require privileged permissions for correct operation. As a result, such configurations may be flagged as false positives by automated checks. This highlights a limitation of current tooling, where automated checks lack sufficient contextual awareness and therefore require manual validation.

A similar limitation applies to static analysis tools. As identified in the literature review, static analysis tools such as KubeLinter and Terrascan can support configuration verification. These tools were applied to the Git repository containing all declarative configuration manifests used in the prototype. However, no issues were reported. This may indicate that the configuration aligns with commonly enforced best practices, but also suggests that such tools may not fully capture compliance requirements specific to E-ITS.

The proposed solution implements the Flux multi-tenancy model and provides isolation through workload separation, default-deny network policies, and policy enforcement. However, it should be noted that Kubernetes natively provides weak isolation [162]. Core control plane components, such as the API server and CoreDNS, are inherently shared across all workloads within a cluster. For example, DNS reverse lookups may expose information about workloads across namespaces. The shared API and DNS server can be affected by resource-intensive or malicious workloads, potentially impacting other tenants.

Such limitations are not specific to the implemented solution but are related to the architectural design of Kubernetes, particularly its reliance on shared control plane components. Mitigating these limitations within a single cluster would require introducing additional complexity, such as custom DNS policies [163]. This highlights a trade-off between increased system complexity and the use of separate clusters for stronger isolation. E-ITS guidance also emphasises, under the APP.4.4 Kubernetes module, that workloads within a

single Kubernetes cluster should have similar security requirements [4].

It is also important to distinguish between compliance and actual security. Although the proposed solution enables automated verification of controls, compliance with defined policies does not guarantee the absence of vulnerabilities or misconfigurations outside the scope of those controls. The implemented architecture is hardened through the use of an immutable, container-specific operating system, reduced attack surface, and restrictive default configurations. However, security in Kubernetes environments still depends on continuous maintenance and operational practices that extend beyond that.

The prototype was evaluated in a controlled laboratory environment and was also found to be useful in two organisational contexts. Further improvements to the prototype are expected through continued feedback from these organisations. Since the evaluation included only two organisations, the results may not be generalisable to other organisational contexts with different requirements. However, as the source code developed in this work is publicly available, organisations can adopt and adapt the prototype to their own needs.

6.5 Answers to Research Questions

This section answers the primary and secondary research questions of the thesis based on the literature review, the design and implementation of the prototype, and the evaluation results.

[PRQ]: How can a Kubernetes cluster be designed and implemented to comply with the E-ITS and automatically assess its security level?

A Kubernetes cluster can be designed and implemented to support E-ITS compliance through a hardened platform design. Version-controlled declarative configuration provides a single source of truth, while policy enforcement and aggregated reporting enable automated verification within the cluster. In this thesis, this was achieved by selecting Talos Linux as a container-optimised OS with a reduced attack surface. Cilium was used for networking, with a BGP-based approach for service exposure and the Gateway API with Traefik for ingress control. Istio was introduced in ambient mode to provide service mesh functionality. Longhorn and Velero were used for storage and backup functionality. GitOps-based configuration management was implemented using Flux, with SOPS for secrets management. Technical controls were enforced and audited using the Kyverno policy engine. The Trivy Operator and a developed infrastructure auditing module supported automated verifications. A unified reporting model was implemented using OpenReports CRDs, while Policy Reporter was used to aggregate and visualise the results.

The evaluation demonstrated that the prototype implementation is feasible in practice. In the E-ITS APP.4.4 module, 18 out of 21 measures were achieved and 3 were partially achieved. In SYS.1.6, 15 out of 26 measures were achieved and 8 were partially achieved. Due to the use of a container-optimised operating system, most SYS.1.3 measures were classified as not applicable. This highlighted limitations of the E-ITS.

Automatic assessment of the security level was supported through aggregated reporting. These reports provided measurable evidence for a subset of E-ITS measures, particularly those that could be directly observed from configuration state and runtime system properties. However, the evaluation showed that automated assessment remains partial and must be complemented by expert judgement for measures depending on architectural context and organisational processes. Nevertheless, the developed architecture provides an extensible basis for increasing automated evidence generation through additional policies and data sources.

[SRQ1]: What are the main challenges and approaches to securing and ensuring compliance in Kubernetes environments?

The main challenges in securing and ensuring compliance in Kubernetes environments are weak default configurations, high system complexity, and the risk of misconfiguration. Kubernetes does not enforce many security controls by default. As a result, network isolation and workload restrictions must be explicitly configured, while access control must be carefully defined according to the principle of least privilege.

Another challenge is the lack of practical guidance for implementing and verifying compliance requirements. Existing standards define what should be achieved but do not provide methods for continuous and automated validation in dynamic environments.

These challenges can be addressed through a layered approach that combines hardened platform design, declarative configuration management, policy enforcement, and automated auditing. In this thesis, this was implemented using an immutable operating system, GitOps-based configuration management and policy-based enforcement mechanisms.

The results show that combining these approaches reduces misconfiguration risks, improves system visibility, and enables continuous compliance verification. However, effective security still depends on correct configuration and cannot rely on any single mechanism, as evaluation revealed misconfiguration risks even in a hardened environment.

[SRQ2]: To what extent can existing compliance and auditing frameworks be used to

implement and verify E-ITS security measures in Kubernetes environments?

Existing compliance and auditing frameworks, together with supporting tools, can be used to varying extents to implement and verify E-ITS security measures in Kubernetes environments. Their effectiveness is highest when measures can be expressed as technical and machine-verifiable controls. In this study, aggregated reporting from multiple sources provided a foundation for continuous and automated compliance assessment. However, applicability is partial rather than complete, as many E-ITS measures depend on architectural context, organisational processes, or expert interpretation.

In the evaluated prototype, automated evidence generation supported 100% of applicable SYS.1.3 measures, 54% of SYS.1.6 measures, and 24% of APP.4.4 measures, showing that existing tools are most effective for directly observable technical controls and less effective for architectural or process-oriented requirements.

Therefore, existing frameworks provide a foundation for implementing and verifying E-ITS measures, but must be complemented with standard-specific mappings and expert judgement.

[SRQ3]: What design and implementation principles are used to build Kubernetes clusters that are intended to be secure, auditable, and compliant?

Secure, auditable, and compliant Kubernetes clusters are built using a combination of architectural and operational design principles. A key principle is minimising the attack surface and applying systematic hardening across all layers of the platform. This is achieved by reducing unnecessary components, limiting exposed interfaces, and enforcing restrictive configurations.

Another principle is the use of declarative and version-controlled configuration as a single source of truth. This enables reproducibility, traceability, and resistance to configuration drift.

Security requirements should be enforced through policy engines rather than manual processes. Network isolation should follow a default-deny approach, with access explicitly allowed through defined policies.

Automated auditing and reporting should be integrated into the platform to provide continuous visibility and evidence of compliance. Aggregating results from multiple sources enables structured and repeatable validation.

Effective security depends on continuous validation, and these principles must be applied together, as no single approach alone is sufficient.

7. Conclusions and Recommendations

7.1 Summary

The main goal of this thesis was to design and implement a secure Kubernetes cluster that supports compliance with the E-ITS. The work addressed the lack of practical implementation guidance and standardised automated compliance verification for Kubernetes environments, especially in on-premises deployments. This goal was addressed through the development of a hardened prototype and its evaluation in both laboratory and organisational contexts. The proposed prototype was successfully implemented in one organisation and is currently in the implementation phase in a second organisation, demonstrating its practical applicability. The thesis produced a reference architecture for a Kubernetes platform designed to achieve E-ITS compliance, together with the corresponding implementation source code, enabling further adaptation [117]. In addition, two separate Git repositories were created for mock applications [153] [154]. These repositories simulated independent application teams deploying workloads to the shared cluster environment through the GitOps workflow.

The evaluation confirmed that the proposed platform supports a substantial proportion of E-ITS measures in a structured and verifiable way. In APP.4.4 (Kubernetes), 18 of 21 measures were achieved and 3 were partially achieved. In SYS.1.6 (Containerisation), 15 of 26 measures were achieved and 8 were partially achieved. In SYS.1.3 (Linux and Unix servers) due to the use of a container-optimised operating system 8 of 10 measures were classified as not applicable. Aggregated policy reporting supported evidence generation for 100% of applicable SYS.1.3 measures, 54% of SYS.1.6 measures, and 24% of APP.4.4 measures. These results show that automated verification can support compliance assessment. However, coverage remained uneven across modules, and some measures still required manual interpretation. Evidence coverage could be improved by including additional policies and data sources.

The novelty of this thesis lies in its holistic approach. Earlier work has addressed Kubernetes security, compliance, and automation separately, whereas this study integrated these aspects into a single solution. The implications of this work are relevant for organisations that must comply with E-ITS or similar information security standards. The proposed approach supports practitioners in building and managing secure Kubernetes platforms, while also assisting auditing activities through more repeatable, transparent, and evidence-

based compliance verification. This can help to reduce manual effort, lower the risk of misconfiguration, and improve the overall security posture of the environment.

7.2 Recommendations for Improving E-ITS

E-ITS provides directly defined and implementation-oriented measures. At the same time, it is very detailed and sometimes too technology specific. Some measures assume traditional system administration approaches, such as SSH access and the ability to modify system after deployment. In containerised environments with immutable systems, these assumptions often do not apply.

One possible suggestion would be to reformulate certain E-ITS measures in a more implementation-agnostic manner. This recommendation is also supported by Firesmith, who emphasises that security requirements should define the objective rather than prescribe specific mechanisms [164]. A similar limitation can be observed in the maintenance of the standard itself. For example, while not in scope of this work, the E-ITS includes a dedicated module for Windows Server 2012, but does not explicitly address more recent versions. This illustrates the challenge of keeping a highly technology-specific standard up to date over time.

Some measures contain inconsistent wording, which may be related to translation. For example, the APP.4.4.M9 measure states that pods without a service account should not be assigned a service account. At the same time, it suggests that such pods should use tokens to communicate with the Kubernetes control plane. In practice, pods obtain tokens through their associated service accounts, making the requirement contradictory. This has also been identified previously by Lindeberg [29].

Another limitation can be seen in the APP.4.4.M21 measure, which recommends periodic restarting of pods. This approach may be suitable for stateless workloads. However, it may not be directly applicable to stateful workloads. Although the measure states that availability should be ensured during restarts, this can be difficult to guarantee in practice. In such cases, restarts can impact availability and may lead to data loss in stateful systems with asynchronous replication.

7.3 Future Work

The proposed solution enables the use of static analysis techniques for validating configuration states. The desired state is maintained as a single source of truth, and the running state

is continuously reconciled against it. Future work could explore the use of large language models for analysing configurations and supporting the auditing process. In addition, static analysis can be integrated into CI/CD workflows, allowing configuration changes to be validated against security and compliance requirements as they are introduced.

From a networking perspective, future work may consider using dual-stack configurations. Although components in the prototype support IPv6, its impact on network policies, security controls, and compliance verification was not evaluated.

As service mesh technologies evolve, it would be valuable to evaluate whether emerging features, such as Cilium's ambient mesh, could replace Istio in a production environment. This may reduce the number of components and simplify the overall system architecture.

The integration of runtime security tools, such as Falco or Tetragon, would extend the prototype by enabling detection of runtime threats and supporting runtime auditing. For instance, Falco can act as an additional producer of reports for the prototype, as it supports integration with OpenReports [165].

Container-level evidence collection should also be revisited in relation to the prototype developed in this thesis. Talos Linux added support for the kernel configuration required by CRIU after the prototype implementation was completed. This may allow container snapshots to be evaluated as evidence in investigations.

Since User Namespaces reached General Availability after the prototype implementation, future work should evaluate enforcing them as an additional hardening measure [166]. User Namespaces can help reduce the impact of container escape scenarios by limiting the privileges granted to container processes on the host.

Alternative approaches to infrastructure provisioning, such as Crossplane or Cluster API, may also be explored. These solutions provide Kubernetes-native control planes for infrastructure management. However, they also introduce additional bootstrapping considerations.

References

[1] Cloud Native Computing Foundation. *Kubernetes Project Journey Report*. Accessed: 17 May 2026. URL: https://www.cncf.io/reports/kubernetes-project-journey-report/.

[2] Adrienn Lawson and Jeffrey Sica. *CNCF Annual Cloud Native Survey: The Infrastructure of AI's Future*. Tech. rep. Accessed: 17 May 2026. The Linux Foundation, Jan. 2026.

[3] Riigi Infosüsteemi Amet. *Uuest aastast laienes küberturvalisuse seadus*. Accessed: 17 May 2026. URL: https://www.ria.ee/uudised/uuest-aastast-laienes-kuberturvalisuse-seadus.

[4] Riigi Infosüsteemi Amet. *E-ITS põhidokumendid*. Accessed: 17 May 2026. URL: https://eits.ria.ee/et/versioon/2024/eits-poohidokumendid/etalonturbe-kataloog.

[5] Cloud Native Computing Foundation. *CNCF Landscape*. Accessed: 17 May 2026. URL: https://landscape.cncf.io/?group=certified-partners-and-providers&view-mode=grid.

[6] Allison Randal. "The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers". In: *ACM Comput. Surv.* 53.1 (Feb. 2020). ISSN: 0360-0300. DOI: 10.1145/3365199. URL: https://doi.org/10.1145/3365199.

[7] Sergei Arnautov et al. "SCONE: Secure Linux Containers with Intel SGX". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[8] Kubernetes Authors. *Pod Security Standards*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/security/pod-security-standards/.

[9] Kubernetes Authors. *Network Policies – The two sorts of pod isolation*. Accessed: 17 May 2026. Kubernetes. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/#the-two-sorts-of-pod-isolation.

[10] Andrew Martin and Michael Hausenblas. *Hacking Kubernetes: Threat-Driven Analysis and Defense*. O'Reilly Media, 2021. URL: https://learning.oreilly.com/library/view/hacking-kubernetes/9781492081722/.

[11] Microsoft. *Threat Matrix for Kubernetes*. Accessed: 17 May 2026. URL: https://microsoft.github.io/Threat-Matrix-for-Kubernetes/.

[12] Sunny Chowdhury and Florian Freund. "Identifying and Analyzing Vulnerabilities and Exploits in On-Premises Kubernetes". In: *ICT Systems Security and Privacy Protection*. Ed. by Lili Nemec Zlatolas et al. Cham: Springer Nature Switzerland, 2025, pp. 155–169. ISBN: 978-3-031-92886-4.

[13] Yue Zhang et al. "Configuration Defects in Kubernetes". In: *IEEE Transactions on Software Engineering* 52.3 (2026), pp. 1125–1144. DOI: 10.1109/TSE.2026.3659785.

[14] Noah Spahn et al. "Container Orchestration Honeypot: Observing Attacks in the Wild". In: *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. RAID '23. Hong Kong, China: Association for Computing Machinery, 2023, pp. 381–396. ISBN: 9798400707650. DOI: 10.1145/3607199.3607205. URL: https://doi.org/10.1145/3607199.3607205.

[15] J. Alexander Curtis and Nasir U. Eisty. "The Kubernetes Security Landscape: AI-Driven Insights from Developer Discussions". In: *2025 IEEE/ACIS 23rd International Conference on Software Engineering Research, Management and Applications (SERA)*. 2025, pp. 179–186. DOI: 10.1109/SERA65747.2025.11154503.

[16] Red Hat, Inc. *State of Kubernetes Security Report 2024*. Accessed: 17 May 2026. URL: https://www.redhat.com/en/resources/state-kubernetes-security-report-2024.

[17] Akond Rahman et al. "Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study". In: *ACM Trans. Softw. Eng. Methodol.* 32.4 (May 2023). ISSN: 1049-331X. DOI: 10.1145/3579639. URL: https://doi.org/10.1145/3579639.

[18] Aqua Security / aquasecurity. *Trivy scanner*. Accessed: 17 May 2026. URL: https://github.com/aquasecurity/trivy.

[19] Md. Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices". In: *2020 IEEE Secure Development (SecDev)*. 2020, pp. 58–64. DOI: 10.1109/SecDev45635.2020.00025.

[20] Harshavardhan Nerella and Prasanna Sai Puvvada. “Reinforcing Kubernetes Security: A Gap Analysis and Modern Security Practices for Enterprise”. In: *2024 First International Conference on Data, Computation and Communication (ICDCC)*. 2024, pp. 780–785. DOI: 10.1109/ICDCC62744.2024.10961639.

[21] Rui Filipe dos Santos. “Applying Zero Trust to Kubernetes Clusters”. In: *ARIS2 - Advanced Research on Information Systems Security* 5.1 (May 2025), pp. 57–71. DOI: 10.56394/aris2.v5i1.58. URL: https://aris-journal.com/aris/index.php/journal/article/view/58.

[22] Murugiah Souppaya, John Morello, and Karen Scarfone. *Application Container Security Guide*. NIST Special Publication 800-190 800-190. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 2017. DOI: 10.6028/NIST.SP.800-190. URL: https://doi.org/10.6028/NIST.SP.800-190.

[23] National Security Agency (NSA), Cybersecurity, and Infrastructure Security Agency (CISA). *Kubernetes Hardening Guidance*. Technical Report Version 1.2 U/OO/168286-21 | PP-22-0324. U.S. Department of Defense / Cybersecurity Technical Report, Aug. 2022. URL: https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF.

[24] Center for Internet Security (CIS). *About Us*. Accessed: 17 May 2026. URL: https://www.cisecurity.org/about-us.

[25] Center for Internet Security (CIS). *CIS Kubernetes Benchmarks*. Accessed: 17 May 2026. URL: https://www.cisecurity.org/benchmark/kubernetes.

[26] Center for Internet Security (CIS). *CIS Benchmarks*. website. Accessed: 17 May 2026. URL: https://www.cisecurity.org/cis-benchmarks.

[27] Kadri Koit. “Describing the Requirements of the Estonian Information Security Standard (E-ITS) in Public Procurements”. MA thesis. University of Tartu, 2024.

[28] Thomas Lepik. “Eesti Infoturbestandardi turvameetmete rakendatuse automaatkontrolli põhimõtted”. MA thesis. Tallinna Ülikool, Digitehnoloogiate instituut, 2023.

[29] Marko Lindeberg. “Analysis and Implementation of ‘APP.4.4: Kubernetes’ from the Estonian Information Security Standard (E-ITS)”. MA thesis. Tallinn, Estonia: Tallinn University of Technology, School of Information Technologies, 2025.

[30] Hitesh Allam. “Policy-Driven Engineering: Automating Compliance Across DevOps Pipelines”. In: *International Journal of Emerging Trends in Computer Science and Information Technology* 6.1 (Mar. 2025), pp. 89–100. DOI: 10.63282/3050-9246.IJETCSIT-V6I1P111. URL: https://www.ijetcsit.org/index.php/ijetcsit/article/view/259.

[31] Takumi Yanagawa et al. “A Secure Framework for Continuous Compliance across Heterogeneous Policy Validation Points”. In: *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. 2024, pp. 176–182. DOI: 10.1109/CLOUD62652.2024.00029.

[32] Kubernetes Authors. *Admission Control in Kubernetes*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/.

[33] Ioanna Angeliki Kapetanidou, Alexandros Nizamis, and Konstantinos Votis. “An evaluation of commonly used Kubernetes security scanning tools”. In: *Proceedings of the 2nd International Workshop on MetaOS for the Cloud-Edge-IoT Continuum*. MECC ’25. Rotterdam, Netherlands: Association for Computing Machinery, 2025, pp. 20–25. ISBN: 9798400715600. DOI: 10.1145/3721889.3721924. URL: https://doi.org/10.1145/3721889.3721924.

[34] Anshita Malviya and Rajendra Kumar Dwivedi. “A Comparative Analysis of Container Orchestration Tools in Cloud Computing”. In: *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*. 2022, pp. 698–703. DOI: 10.23919/INDIACom54597.2022.9763171.

[35] Diyaz Yakubov and David Hästbacka. “Comparative Analysis of Lightweight Kubernetes Distributions for Edge Computing: Security, Resilience and Maintainability”. In: *Service-Oriented and Cloud Computing*. Ed. by Claus Pahl et al. Cham: Springer Nature Switzerland, 2025, pp. 96–104.

[36] Pedro Ascensão et al. “Assessing Kubernetes Distributions: A Comparative Study”. In: *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*. 2024, pp. 832–837. DOI: 10.1109/MELECON56669.2024.10608706.

[37] Aqua Security. *kube-bench: Checks whether Kubernetes is deployed according to security best practices as defined in the CIS Kubernetes Benchmark*. GitHub repository. Accessed: 17 May 2026. URL: https://github.com/aquasecurity/kube-bench.

[38] Soo Yee Lim et al. “Secure Namespaced Kernel Audit for Containers”. In: *Proc. ACM Symp. Cloud Computing (SoCC ’21)*. Seattle, WA, USA: ACM, 2021, pp. 518–530. DOI: 10.1145/3472883.3486976. URL: https://doi.org/10.1145/3472883.3486976.

[39] Kubernetes Authors. *Network Plugins*. Accessed: 17 May 2026. Kubernetes. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/.

[40] Bom Kim, Jinwoo Kim, and Seungsoo Lee. "Exploring Security Enhancements in Kubernetes CNI: A Deep Dive Into Network Policies". In: *IEEE Access* 13 (2025), pp. 35322–35338. DOI: 10.1109/ACCESS.2025.3543841.

[41] Sebastiano Miano et al. "A Framework for eBPF-Based Network Functions in an Era of Microservices". In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 133–151. DOI: 10.1109/TNSM.2021.3055676.

[42] Gerald Budigiri et al. "Network Policies in Kubernetes: Performance Evaluation and Security Analysis". In: *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. 2021, pp. 407–412. DOI: 10.1109/EuCNC/6GSummit51104.2021.9482526.

[43] Francesco Settanni, Giuseppe Lisena, and Cataldo Basile. "Towards A Capability Model of Kubernetes Runtime Security Enforcement Mechanisms". In: *Availability, Reliability and Security*. Ed. by Bart Coppens et al. Cham: Springer Nature Switzerland, 2025, pp. 266–284. ISBN: 978-3-032-00630-1.

[44] Ken Peffers et al. "A design science research methodology for information systems research". In: *Journal of Management Information Systems* 24 (Jan. 2007), pp. 45–77.

[45] Jan vom Brocke, Alan Hevner, and Alexander Maedche. "Introduction to Design Science Research". In: *Design Science Research. Cases*. Ed. by Jan vom Brocke, Alan Hevner, and Alexander Maedche. Cham: Springer International Publishing, 2020, pp. 1–13. ISBN: 978-3-030-46781-4. DOI: 10.1007/978-3-030-46781-4_1. URL: https://doi.org/10.1007/978-3-030-46781-4_1.

[46] Cloud Native Computing Foundation. *Certified Kubernetes Software Conformance*. Accessed: 17 May 2026. URL: https://www.cncf.io/training/certification/software-conformance/.

[47] Rancher Labs. *Rancher Kubernetes Engine (RKE) Documentation*. Accessed: 17 May 2026. URL: https://rke.docs.rancher.com/.

[48] Edgeless Systems. *Constellation - The Kubernetes built for confidential computing*. Accessed: 11 Nov. 2025. URL: https://www.edgeless.systems/products/constellation.

[49] Edgeless Systems. *GitHub Issue #3973: Security Configuration Concern*. Accessed: 17 May 2026. URL: https://github.com/edgelesssys/constellation/issues/3973.

[50] Red Hat. *Article 6907891 — Red Hat Customer Portal*. Accessed: 17 May 2026. URL: https://access.redhat.com/articles/6907891.

[51] Flatcar Container Linux. *Getting Started with Kubernetes — Flatcar Docs*. Accessed: 17 May 2026. URL: https://www.flatcar.org/docs/latest/container-runtimes/getting-started-with-kubernetes/.

[52] Sidero Labs. *Philosophy — Talos Linux Minimal OS Design*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/talos/v1.12/learn-more/philosophy#minimal.

[53] Amazon Web Services. *Operating System Management Overview — EKS Anywhere*. Accessed: 17 May 2026. URL: https://anywhere.eks.amazonaws.com/docs/osmgmt/overview/.

[54] Elastisys AB. *compliantkubernetes-kubespray Repository on GitHub*. Accessed: 17 May 2026. URL: https://github.com/elastisys/kubespray/tree/5ed66dddb891fb42af00e4244df1d41d81673f78.

[55] Canonical. *Quick Start — Charmed Kubernetes*. Accessed: 17 May 2026. URL: https://ubuntu.com/kubernetes/charmed-k8s/docs/quickstart.

[56] SUSE. *RKE2 Support Matrix — SUSE*. Accessed: 17 May 2026. URL: https://www.suse.com/suse-rke2/support-matrix/all-supported-versions/rke2-v1-35/.

[57] SIGHUP. *Getting Started — Distro on VMs*. Accessed: 17 May 2026. URL: https://docs.sighup.io/docs/getting-started/distro-on-vms#prerequisites.

[58] Kubernetes SIGs. *Supported Linux Distributions — Kubespray*. Accessed: 17 May 2026. URL: https://github.com/kubernetes-sigs/kubespray#supported-linux-distributions.

[59] Elastisys AB. *Welkin Operator Manual — Getting Started*. Accessed: 17 May 2026. URL: https://elastisys.io/welkin/operator-manual/getting-started/.

[60] Amazon Web Services. *Amazon EKS Anywhere*. Accessed: 17 May 2026. Amazon Web Services. URL: https://aws.amazon.com/eks/eks-anywhere/.

[61] Sidero Labs. *Talos Linux FAQ*. Accessed: 17 May 2026. Sidero Labs. URL: https://docs.siderolabs.com/talos/v1.12/troubleshooting/faqs.

[62] Justin Garrison. *Which Linux distro is the most secure for Kubernetes?* Accessed: 17 May 2026. Sidero Labs. URL: https://www.siderolabs.com/blog/which-linux-distro-is-the-most-secure-for-kubernetes/.

[63] Sidero Labs. *Kubernetes Cluster Reference Architecture with Talos Linux*. Technical Reference Document. Version 2025-05/2.

[64] Sidero Labs. *Deploy Calico*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/cni/deploy-calico.

[65] Sidero Labs. *Deploying Cilium*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/cni/deploying-cilium.

[66] Isovalent. *Why Replace iptables with eBPF?* Accessed: 17 May 2026. URL: https://isovalent.com/blog/post/why-replace-iptables-with-ebpf/.

[67] Cilium Authors. *Hubble: Network Observability for Kubernetes*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/stable/observability/hubble/index.html.

[68] Cilium Project. *Tetragon: eBPF-based Security Observability and Runtime Enforcement*. Accessed: 17 May 2026. URL: https://tetragon.io/.

[69] Isovalent. *AWS EKS Anywhere Chooses Cilium*. Accessed: 17 May 2026. URL: https://isovalent.com/blog/post/2021-09-aws-eks-anywhere-chooses-cilium/.

[70] Kubernetes Authors. *Service*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/services-networking/service/.

[71] Cilium. *BGP Support*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/stable/network/bgp-toc/.

[72] Kubernetes Authors. *Ingress*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/.

[73] Tabitha Sable. *Ingress NGINX Retirement: What You Need to Know*. Accessed: 17 May 2026. URL: https://kubernetes.io/blog/2025/11/11/ingress-nginx-retirement/.

[74] Kubernetes. *Kubernetes Gateway API*. Accessed: 17 May 2026. URL: https://gateway-api.sigs.k8s.io/.

[75] Traefik Documentation. *Traefik Kubernetes with Gateway API*. Accessed: 17 May 2026. URL: https://doc.traefik.io/traefik/reference/install-configuration/providers/kubernetes/kubernetes-gateway/#traefik-kubernetes-with-gateway-api.

[76] Traefik Labs. *Traefik Proxy GitHub Repository*. Accessed: 17 May 2026. URL: https://github.com/traefik/traefik.

[77] Sidero Labs Documentation. *Deploy Traefik as a Gateway API controller on a Talos-managed Kubernetes cluster*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/advanced-guides/deploy-traefik.

[78] Amazon Web Services. *What is a Service Mesh?* Accessed: 17 May 2026. URL: https://aws.amazon.com/what-is/service-mesh/.

[79] Anat Barr et al. *Technical Report: Performance Comparison of Service Mesh Frameworks: the MTLS Test Case*. Nov. 2024. DOI: 10.48550/arXiv.2411.02267.

[80] Cilium Project. *Mutual Authentication (Beta)*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/latest/network/servicemesh/mutual-authentication/mutual-authentication/.

[81] Christian Posta. *How Cilium's Mutual Authentication Can Compromise Security*. The New Stack, Accessed: 12 Apr. 2026. URL: https://thenewstack.io/how-ciliums-mutual-authentication-can-compromise-security/.

[82] Cilium Authors. *Ztunnel Transparent*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/latest/security/network/encryption-ztunnel/.

[83] Istio. *Ambient Mode Overview*. Accessed: 17 May 2026. URL: https://istio.io/latest/docs/ambient/overview/.

[84] Istio Project. *Ambient data plane*. Accessed: 17 May 2026. URL: https://istio.io/latest/docs/ambient/architecture/data-plane/.

[85] Bob Killen et al. *10 Years of Kubernetes*. Accessed: 17 May 2026. URL: https://kubernetes.io/blog/2024/06/06/10-years-of-kubernetes/.

[86] Kubernetes CSI Contributors. *Drivers - Kubernetes CSI Developer Documentation*. Accessed: 17 May 2026. URL: https://kubernetes-csi.github.io/docs/drivers.html.

[87] Sidero Labs Documentation Contributors. *iSCSI Storage with Synology CSI — Setting up the Synology user account*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/csi/synology-csi#setting-up-the-synology-user-account.

[88] HPE SCOD Documentation. *HPE CSI Driver Deployment — Secret Parameters*. Accessed: 17 May 2026. URL: https://scod.hpedev.io/csi_driver/deployment.html#secret_parameters.

[89] VMware Documentation. *Preparing for Installation of vSphere Container Storage Plug-in*. Accessed: 17 May 2026. URL: https://techdocs.broadcom.com/us/en/vmware-cis/vsphere/container-storage-plugin/3-0/getting-started-with-vmware-vsphere-container-storage-plug-in-3-0/vsphere-container-storage-plug-in-deployment/preparing-for-installation-of-vsphere-container-storage-plug-in.html.

[90] sergelogvinov/proxmox-csi-plugin Contributors. *Proxmox CSI Plugin Installation Guide*. Accessed: 17 May 2026. URL: https://github.com/sergelogvinov/proxmox-csi-plugin/blob/main/docs/install.md.

[91] Longhorn Documentation. *Talos Linux Support*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/advanced-resources/os-distro-specific/talos-linux-support/.

[92] Longhorn Documentation. *Concepts*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/concepts/.

[93] Sidero Labs. *Storage — Kubernetes Guides*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/csi/storage.

[94] Sidero Labs, Inc. *Disaster Recovery - Talos v1.12 Documentation*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/talos/v1.12/build-and-extend-talos/cluster-operations-and-maintenance/disaster-recovery.

[95] Sameer, Suman De, and R Prashant Singh. "Selective Analogy of Mechanisms and Tools in Kubernetes Lifecycle for Disaster Recovery". In: *2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*. 2022, pp. 1–6. DOI: 10.1109/ICMNWC56175.2022.10031926.

[96] Inc. Cohesity. *Upgrade Velero and Datamover*. Accessed: 12 Apr. 2026. URL: https://docs.cohesity.com/baas/data-protect/kubernetes-upgradevelero.htm?tocpath=Kubernetes%7C_____3#UpgradeVeleroandDatamover.

[97] Veritas Technologies LLC. *Kubernetes Administrator's Guide*. Accessed: 17 May 2026. URL: https://www.veritas.com/support/en_US/doc/150157643-167856485-0/v152798094-167856485.

[98] Kubernetes Authors. *Kubernetes Object Management*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/.

[99] Alex Williams. *4 Core Principles of GitOps*. The New Stack, May 11, 2023. Accessed: 12 Apr. 2026. URL: https://thenewstack.io/4-core-principles-of-gitops/.

[100] Cloud Native Computing Foundation. *GitOps - Cloud Native Glossary*. Accessed: 17 May 2026. URL: https://glossary.cncf.io/gitops/.

[101] Sidero Labs Documentation. *inlineManifests and extraManifests — Example Use Case: Install a GitOps controller with extraManifests*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/advanced-guides/inlinemanifests#example-usecase-install-a-gitops-controller-with-extramanifests.

[102] Ramadoni, Ema Utami, and Hanif Al Fatta. "Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD". In: *2021 4th International Conference on Information and Communications Technology (ICOIACT)*. 2021, pp. 186–191. DOI: 10.1109/ICOIACT53268.2021.9563984.

[103] Amazon Web Services. *Use Cases for GitOps Tools on Amazon EKS*. Accessed: 17 May 2026. URL: https://docs.aws.amazon.com/prescriptive-guidance/latest/eks-gitops-tools/use-cases.html.

[104] Kubernetes Authors. *Secrets*. Accessed: 17 May 2026. Kubernetes. URL: https://kubernetes.io/docs/concepts/configuration/secret/.

[105] External Secrets Community. *External Secrets Operator Documentation*. Accessed: 17 May 2026. URL: https://external-secrets.io/latest/.

[106] Bitnami Labs. *Sealed Secrets*. Accessed: 17 May 2026. URL: https://github.com/bitnami-labs/sealed-secrets.

[107] Mozilla SOPS Contributors. *Mozilla SOPS (Secrets OPerationS)*. Accessed: 17 May 2026. URL: https://github.com/getsops/sops.

[108] The Kyverno Authors. *Policies | Kyverno*. Accessed: 17 May 2026. URL: https://kyverno.io/policies/.

[109] Cloud Native Computing Foundation. *Announcing OpenReports: Standardized Kubernetes Reporting*. Accessed: 17 May 2026. URL: https://www.cncf.io/blog/2025/05/06/announcing-openreports-standardized-kubernetes-reporting/.

[110] Kyverno Project. *Policy Reporter Documentation*. Accessed: 17 May 2026. URL: https://kyverno.github.io/policy-reporter-docs/.

[111] Kyverno Authors. *Targets · Policy Reporter*. Accessed: 17 May 2026. URL: https://kyverno.github.io/policy-reporter/core/targets/.

[112] Awwal Ishiaku. *Auditing Kubernetes with Wazuh*. Wazuh Blog. Accessed: 17 May 2026. URL: https://wazuh.com/blog/auditing-kubernetes-with-wazuh/.

[113] Aqua Security. *Trivy Operator Documentation*. Accessed: 17 May 2026. URL: https://aquasecurity.github.io/trivy-operator/latest/.

[114] Florian Jogeleit. *Trivy Operator Policy Reporter Adapter*. Accessed: 17 May 2026. URL: https://github.com/fjogeleit/trivy-operator-polr-adapter.

[115] The Falco Authors. *Falco Documentation*. Accessed: 17 May 2026. URL: https://falco.org/docs/.

[116] Sidero Labs. *Pull request #1493*. Accessed: 17 May 2026. URL: https://github.com/siderolabs/pkgs/pull/1493.

[117] O. Pipenberg. *Pilvelaevastik*. Accessed: 17 May 2026. URL: https://github.com/opipenbe/pilvelaevastik.

[118] Proxmox Server Solutions GmbH. *Proxmox VE Administration Guide*. Accessed: 17 May 2026. URL: https://pve.proxmox.com/pve-docs/pve-admin-guide.html.

[119] Sidero Labs. *Proxmox*. Talos Documentation, v1.12. Accessed: 12 Apr. 2026. URL: https://docs.siderolabs.com/talos/v1.12/platform-specific-installations/virtualized-platforms/proxmox.

[120] OpenTofu Project. *OpenTofu: Open-Source Infrastructure as Code Tool*. Accessed: 17 May 2026. URL: https://opentofu.org/.

[121] Amazon Web Services. *What is Infrastructure as Code?* Accessed: 17 May 2026. URL: https://aws.amazon.com/what-is/iac/.

[122] bpg. *OpenTofu Provider for Proxmox VE*. Accessed: 17 May 2026. URL: https://search.opentofu.org/provider/bpg/proxmox/latest.

[123] Sidero Labs. *siderolabs/talos Provider – OpenTofu Registry*. Accessed: 17 May 2026. URL: https://search.opentofu.org/provider/siderolabs/talos/latest.

[124] OpenTofu Project. *Kubernetes Provider*. Accessed: 17 May 2026. URL: https://search.opentofu.org/provider/opentofu/kubernetes/latest.

[125] Sidero Labs. *Node Labels and Node Taints*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/advanced-guides/node-labels.

[126] OpenTofu Project. *Helm Provider for OpenTofu*. Accessed: 17 May 2026. URL: https://search.opentofu.org/provider/opentofu/helm/latest.

[127] Sidero Labs. *Deploy Cilium CNI – Method 3: Helm Manifests Hosted Install*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/cni/deploying-cilium#method-3-helm-manifests-hosted-install.

[128] FluxCD. *Flux Provider (fluxcd/flux)*. Accessed: 17 May 2026. URL: https://search.opentofu.org/provider/fluxcd/flux/latest.

[129] Sidero Labs. *graceful upgrades through terraform (Issue #140)*. Accessed: 17 May 2026. URL: https://github.com/siderolabs/terraform-provider-talos/issues/140.

[130] VyOS maintainers and contributors. *About*. Accessed: 17 May 2026. URL: https://docs.vyos.io/en/latest/introducing/about.html.

[131] Robert Moskowitz et al. *Address Allocation for Private Internets*. RFC 1918. Feb. 1996. DOI: 10.17487/RFC1918. URL: https://www.rfc-editor.org/info/rfc1918.

[132] Sidero Labs. *Pull-Through Image Cache*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/talos/v1.12/configure-your-talos-cluster/images-container-runtime/pull-through-cache.

[133] Cilium Authors. *Routing Concepts: Native Routing*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/stable/network/concepts/routing/#native-routing.

[134] Cilium Authors. *Kubernetes Without kube-proxy: Direct Server Return (DSR)*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/stable/network/kubernetes/kubeproxy-free/#direct-server-return-dsr.

[135] Cilium Authors. *Integration with Istio*. Accessed: 17 May 2026. URL: https://docs.cilium.io/en/stable/network/servicemesh/istio/.

[136] Istio Authors. *Platform-Specific Prerequisites: Cilium*. Accessed: 17 May 2026. URL: https://istio.io/latest/docs/ambient/install/platform-prerequisites/#cilium.

[137] O. Pipenberg. *Cilium BGP Configuration for Kubernetes Cluster*. Accessed: 17 May 2026. URL: https://github.com/opipenbe/pilvelaevastik/blob/main/infrastructure/cilium/cilium-bgp.yaml.

[138] Sidero Labs. *Ingress Firewall*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/talos/v1.12/networking/ingress-firewall.

[139] O. Pipenberg. *Cilium Host Firewall Configuration (YAML)*. Accessed: 17 May 2026. URL: https://github.com/opipenbe/pilvelaevastik/blob/main/infrastructure/cilium/cilium-host-firewall.yaml.

[140] Kyverno Project. *Enforce Strict mTLS for Istio*. Accessed: 17 May 2026. URL: https://kyverno.io/policies/istio-cel/enforce-strict-mtls/enforce-strict-mtls/.

[141] Istio Authors. *Ambient mode and Kubernetes NetworkPolicy*. Accessed: 17 May 2026. URL: https://istio.io/latest/docs/ambient/usage/networkpolicy/.

[142] Sidero Labs. *Talos Linux Image Factory*. Accessed: 17 May 2026. URL: https://factory.talos.dev/.

[143] Sidero Labs. *SecureBoot*. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/talos/v1.12/platform-specific-installations/bare-metal-platforms/secureboot.

[144] FluxCD Authors. *Flux Multi-Tenancy Configuration*. Accessed: 17 May 2026. URL: https://fluxcd.io/flux/installation/configuration/multitenancy/.

[145] The Helm Authors. *Helm Documentation*. Accessed: 17 May 2026. URL: https://helm.sh/docs/.

[146] FluxCD. *Manage Kubernetes secrets with Mozilla SOPS*. Accessed: 17 May 2026. URL: https://fluxcd.io/flux/guides/mozilla-sops/#encrypting-secrets-using-age.

[147] Longhorn Authors. *mTLS Support*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/advanced-resources/security/mtls-support/.

[148] Longhorn Authors. *Volume Encryption*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/advanced-resources/security/volume-encryption/.

[149] Longhorn Authors. *Best Practices*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/best-practices/.

[150] Longhorn Authors. *Enable CSI Snapshot Support*. Accessed: 17 May 2026. URL: https://longhorn.io/docs/1.10.1/snapshots-and-backups/csi-snapshot-support/enable-csi-snapshot-support/.

[151] Kubernetes Authors. *Resource Quotas*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/policy/resource-quotas/#how-kubernetes-resourcequotas-work.

[152] Sidero Labs. *Talos API Access from Kubernetes*. Sidero Documentation. Accessed: 17 May 2026. URL: https://docs.siderolabs.com/kubernetes-guides/advanced-guides/talos-api-access-from-k8s.

[153] O. Pipenberg. *Market Team Project*. Accessed: 17 May 2026. URL: https://github.com/opipenbe/market-team-project.

[154] O. Pipenberg. *Cloud Team Project*. Accessed: 17 May 2026. URL: https://github.com/opipenbe/cloud-team-project.

[155] Google Cloud Platform. *Microservices Demo (Online Boutique)*. Accessed: 17 May 2026. URL: https://github.com/GoogleCloudPlatform/microservices-demo.

[156] Nextcloud Community. *Nextcloud Helm Chart for Kubernetes*. Accessed: 17 May 2026. URL: https://github.com/nextcloud/helm/tree/main/charts/nextcloud.

[157] Kubernetes Authors. *Well-Known Labels, Annotations and Taints*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/reference/labels-annotations-taints/#node-kubernetes-io-out-of-service.

[158] The Tcpdump Group. *Tcpdump & Libpcap Official Website*. Accessed: 17 May 2026. URL: https://www.tcpdump.org/.

[159] Wireshark Foundation. *TShark (1) Manual Page*. Accessed: 17 May 2026. URL: https://www.wireshark.org/docs/man-pages/tshark.html.

[160] Nmap Project. *Nmap: Network Mapper - Free Security Scanner*. Accessed: 17 May 2026. URL: https://nmap.org/.

[161] Cilium Authors. *Pull Request #44959: cilium/cilium*. Accessed: 17 May 2026. URL: https://github.com/cilium/cilium/pull/44959.

[162] Kubernetes Authors. *Multi-tenancy*. Accessed: 17 May 2026. URL: https://kubernetes.io/docs/concepts/security/multi-tenancy/.

[163] CoreDNS Authors. *Kubernetes Metadata Multi-Tenancy Policy*. Accessed: 17 May 2026. URL: https://github.com/coredns/policy#kubernetes-metadata-multi-tenancy-policy.

[164] Donald G. Firesmith. “Engineering Security Requirements”. In: *Journal of Object Technology* 2.1 (Jan. 2003), pp. 53–68.

[165] OpenReports. *reports-api*. Accessed: 17 May 2026. URL: https://github.com/openreports/reports-api.

[166] Rodrigo Campos Catelin and Giuseppe Scrivano. *Kubernetes v1.36: User Namespaces in Kubernetes are finally GA*. Accessed: 17 May 2026. URL: https://kubernetes.io/blog/2026/04/23/kubernetes-v1-36-userns-ga/.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Olari Pipenberg

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Design and Implementation of E-ITS Compliant Kubernetes Cluster with Automated Auditing", supervised by Risto Vaarandi and Thomas Lepik
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

18.05.2026

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Mapping of E-ITS SYS.1.3 Linux and Unix server Measures in the Platform Design

Table 7. SYS.1.3 module: design decisions

E-ITS measure	**Design decision**
M2 Proper definition of indicators	This measure is not applicable, as Talos Linux does not use traditional UNIX user and group management.
M3 Avoiding automatic connection of removable devices to the file system	This measure is not applicable, as Talos Linux does not support automatic mounting of removable devices.
M4 Protection of applications	This measure is not applicable, as Talos Linux uses a hardened kernel.
M5 Secure installation of software packages	This measure is not applicable, as Talos Linux does not allow the installation of software packages on the host system. All applications are deployed as container images rather than installed directly on the operating system.
M6 Administration of users and groups	This measure is not applicable, as Talos Linux does not use traditional UNIX user and group management.
M8 SSH-encrypted data exchange	This measure is not applicable, as Talos Linux does not use the SSH. API-based access secured with mTLS is used.
M10 Prevention of the unauthorised expansion of rights	Workloads are designed to operate with restricted privileges, following the principle of least privilege.
M14 Prevention of system and user information intelligence	This measure is not applicable, as Talos Linux does not provide interactive user access to the operating system. Access to system information is restricted to controlled API-based mechanisms and is not available to general users.
SYS.1.3.M16 Additional prevention of the unauthorised expansion of rights	System call restrictions are supported through container runtime security mechanisms and Kubernetes security policies. Additional restriction mechanisms such as seccomp profiles are supported but require application-specific configuration.

E-ITS measure	Design decision
SYS.1.3.M17 Additional kernel security	Implemented through Talos Linux, which uses a hardened Linux kernel with security-focused configuration.

Appendix 3 – Mapping of E-ITS SYS.1.6 Containerisation Measures in the Platform Design

Table 8. SYS.1.6 module: design decisions

E-ITS measure	**Design decision**
M1 Secure configuration of container platform	The platform is designed as a multi-tenant Kubernetes environment supporting isolated workloads, automated deployment via GitOps, and controlled resource usage. Security risks such as misconfiguration, privilege escalation, and unintended communication are mitigated through RBAC, network policies, and policy enforcement mechanisms.
M2 Lifecycle & regular security updates	Container lifecycle management is implemented using a GitOps workflow, where application configurations are version-controlled and continuously reconciled to the cluster. Updates and changes are applied through controlled commits, supported by access control and auditing mechanisms.
M3 Secure operation of containerised IT systems	Secure operation is ensured through workload isolation, namespace separation, and network segmentation. A hardened operating system underpins container execution, while availability is supported by Kubernetes scheduling, self-healing mechanisms, and workload health checks.
M4 Secure implementation of container images	Container images are deployed declaratively via GitOps using version-controlled manifests. Image build processes are external to the platform, while deployment focuses on controlled and traceable usage of predefined images.
M5 Partitioning the management network of containers	Network design separates management and application traffic. Default-deny policies enforce strict communication rules, allowing only explicitly defined connections. Access to management interfaces is restricted to authorised components.
M6 Use of secure container images	Container images are sourced from approved registries and referenced using versioned tags. Security controls include restricting image sources and performing vulnerability scanning.

E-ITS measure	Design decision
M7 Storage of container log data	Logging follows Kubernetes practices, where containers write to standard output streams. Logs are collected at the node level or forwarded externally, ensuring persistence independent of container lifecycle.
M8 Secure management of container access data	Sensitive data is managed using Kubernetes secrets, stored separately from application images. Secrets are encrypted (e.g., via SOPS) and access is restricted through RBAC policies.
M9 Assessing the suitability of applications	Applications are evaluated for compatibility with Kubernetes, including required privileges, behaviour in multi-tenant environments, and resilience to misconfiguration or failure.
M10 Guidelines for the use of containers	Operational rules are enforced through platform policies, including restrictions on image sources, security contexts, and network communication.
M11 Partitioning applications in containers	Applications are structured according to containerisation best practices, where each container provides a single service to support modular and maintainable deployments.
M12 Secure distribution of container images	Image distribution is controlled through approved registries and GitOps-managed deployments, ensuring traceability and controlled rollout of images.
M13 Approving container images for use	Images are evaluated before deployment using scanning and controlled workflows. Only validated images from trusted sources are intended for use.
M14 Updating container images	Updates are performed by modifying versioned image references in Git-managed manifests, following an immutable infrastructure approach where containers are redeployed rather than modified.
M15 Defining the resource limitations of a container	Resource usage is controlled through Kubernetes requests and limits, ensuring fair allocation and preventing resource exhaustion in multi-tenant environments.
M16 Secure remote maintenance of containers	Remote operations are performed exclusively via Kubernetes and Talos APIs, preventing direct host access and enforcing controlled management interfaces.
M17 Defining the minimum rights required for operating a container	Containers operate with minimal privileges using Pod Security Standards and security contexts, restricting unnecessary access and limiting potential attack surface.

E-ITS measure	Design decision
M18 Restricting the rights of containerised applications	Applications are isolated from the host system, ensuring that container-level identities do not have host-level privileges.
M19 Regulating access to data recorders	Storage access is controlled through Kubernetes volume management, ensuring containers access only required volumes. Longhorn is used as a distributed storage backend.
M20 Security of the configuration information of a container	Configuration is managed via GitOps, ensuring version control, traceability, and controlled changes to all deployment manifests.
M22 Using container images as evidence	Not implemented in the platform design.
M23 Ensuring the constancy of containers	Containers follow immutability principles, with restricted execution environments and externalised writable data.
M24 Host-based attack detection	Host-based detection mechanisms are not included in the current design.
M25 High availability of containerised applications	High availability is achieved through multi-zone node distribution and resilient scheduling strategies.
M21 Expanded security rules of containerisation	Security policies enforce restrictions on communication, privileges, and file system access using Kubernetes mechanisms.
M26 Further isolation and encapsulation of containers	Additional isolation mechanisms (e.g., hypervisor-based isolation) are not implemented in the current design.

Appendix 4 – Mapping of E-ITS APP.4.4 Kubernetes Measures in the Platform Design

Table 9. APP.4.4 module: design decisions

Measure	**Design decision**
M1 Designing the partition of applications	Logical separation is achieved through namespaces, default-deny network policies, and multi-tenancy with resource quotas. A single cluster is used under the assumption that applications have similar protection requirements.
M2 Automation of the development of applications with the help of CI/CD	A Flux-based GitOps model is adopted, with role separation and access control enforced via repository permissions and Kubernetes RBAC. The platform supports integration with external CI/CD pipelines for full lifecycle management.
M3 Planning the Kubernetes identity and rights management	Identity and access management is implemented through Flux multi-tenancy with separation of roles (infrastructure, platform, and workload tenants) and Kubernetes RBAC. Resource quotas are applied to enforce tenant boundaries.
M4 Partition of pods	Pod isolation is provided by Kubernetes and the container runtime through Linux namespaces and cgroups (v2), enforced by the Talos Linux operating system.
M5 Backup of cluster information	Backup of cluster information is implemented using Velero for both persistent volumes and Kubernetes resources. Infrastructure components and application configurations are maintained in version-controlled Git repositories through GitOps, ensuring recoverability of infrastructure and services.
M6 Secure resetting of the pods	Not implemented at the platform level. This requirement depends on application-specific init container logic and must be addressed during application development.
M7 Partition of Kubernetes networks	Network partitioning is implemented using Kubernetes network policies and Cilium ClusterWideNetworkPolicies. Administrative, control plane, and application traffic are logically separated. A default-deny model is enforced, with policies managed centrally by platform administrators and at namespace level by tenant users. Configuration is managed declaratively via GitOps.

Measure	Design decision
M8 Security of the Kubernetes configuration files	Configuration is managed using a GitOps approach with Flux, where all manifests are stored in version-controlled repositories. Access is restricted, and all changes are applied declaratively through the reconciliation process.
M9 Security of the Kubernetes service accounts	Access control is enforced through GitOps-based multi-tenancy and Kubernetes RBAC. Kyverno policies are used to reduce unnecessary service account token usage.
M10 Security of the automation process	Automation is implemented using Flux GitOps with a multi-tenancy model, enabling separation of responsibilities and controlled configuration management.
M11 Monitoring the use of containers	Container monitoring is implemented using Kubernetes liveness and readiness probes to ensure health checks and automatic recovery. Kyverno policies are used to enforce probe configuration where applicable.
M12 Security of infrastructure applications	Infrastructure applications are secured through RBAC and separation of responsibilities, mutual TLS (Istio) for encrypted communication, automated backups (Velero), and GitOps-based configuration management ensuring auditability.
M13 Automatic configuration audits	Automated configuration auditing is implemented using Kyverno policies, the Infrastructure Audit Component, and the Trivy Operator. Results are aggregated via Policy Reporter.
M14 Use of specialised nodes	The cluster is designed with role-based node separation, where control plane components run on dedicated management nodes, infrastructure services are assigned to specific nodes, and workloads run on worker nodes. Dedicated bastion nodes are not implemented.
M15 Partition of applications at the levels of nodes and clusters	A single Kubernetes cluster is used under the assumption of similar protection requirements. Node-level separation is enforced by assigning infrastructure components and application workloads to distinct node groups.
M16 Use of Kubernetes operators	Automated management of cluster resources is implemented through GitOps using Flux, with declarative configuration stored and reconciled from version-controlled repositories.
M17 Certification of nodes	Node authentication is implemented using mTLS with certificates managed by Talos Linux. TPM-based attestation is not supported in the current implementation.

Measure	Design decision
M18 Microsegmentation	Microsegmentation is implemented using Cilium ClusterWideNetworkPolicies enforcing a default-deny model, with additional workload-specific policies applied as needed.
M19 Guaranteeing the high availability of Kubernetes	High availability is achieved through a multi-node cluster distributed across availability zones, supported by GitOps-based configuration management and Velero backups.
M20 Encryption of the control plane storage space	Encryption at rest for etcd and control plane storage is enabled through Talos Linux.
M21 Periodic restarting of the pods	Periodic restarting of pods is supported using the Kubernetes Descheduler to trigger controlled rescheduling, primarily for stateless workloads.

Appendix 5 - Verification of the Prototype Talos Linux Configuration Against the CIS Benchmark

Table 10. Results of Verifying the Prototype Configuration Against the Talos Linux CIS Benchmark

Category	Control	Verification Method	Status
Initial Setup	Install Talos with Secure Boot enabled	`talosctl -n <node> get securitystate`	Not implemented
Initial Setup	Encrypt system disks	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Initial Setup	Ensure updated software is installed	`talosctl -n <node> get version`	Implemented
Time Synchronization	Ensure NTP is configured	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Kubernetes	Ensure control plane scheduling is disabled	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Kubernetes	Ensure Pod Security Standards policy is restricted	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Access Control	Ensure RBAC is enabled	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure packet redirect sending is disabled	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure source routed packets are not accepted	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure ICMP redirects are not accepted	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure secure ICMP redirects are not accepted	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure suspicious (martian) packets are logged	`talosctl -n <node> get machineconfig -o yaml`	Implemented

Category	Control	Verification Method	Status
Network Configuration	Ensure broadcast ICMP requests are ignored	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure bogus ICMP responses are ignored	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Ensure TCP SYN Cookies are enabled	`talosctl -n <node> get machineconfig -o yaml`	Implemented
Network Configuration	Enable reverse path filtering if there are only symmetrical routes	`talosctl -n <node> get machineconfig -o yaml`	Not implemented due to BGP-based routing
Network Configuration	Apply an appropriate control plane firewall policy	Inspection of Cilium host firewall policies	Implemented
Network Configuration	Apply an appropriate worker node firewall policy	Inspection of Cilium host firewall policies	Implemented
Network Configuration	Enable KubeSpan where network-level encryption is required	`talosctl -n <node> get machineconfig -o yaml`	Not implemented (service mesh used selectively)
Logging and Auditing	Ensure kernel logging is configured to send logs to a remote service	`talosctl -n <node> get machineconfig -o yaml`	Not implemented
Logging and Auditing	Ensure service logging is configured	`talosctl -n <node> get machineconfig -o yaml`	Not implemented

Appendix 6 - Prototype Talos Node Compliance Report

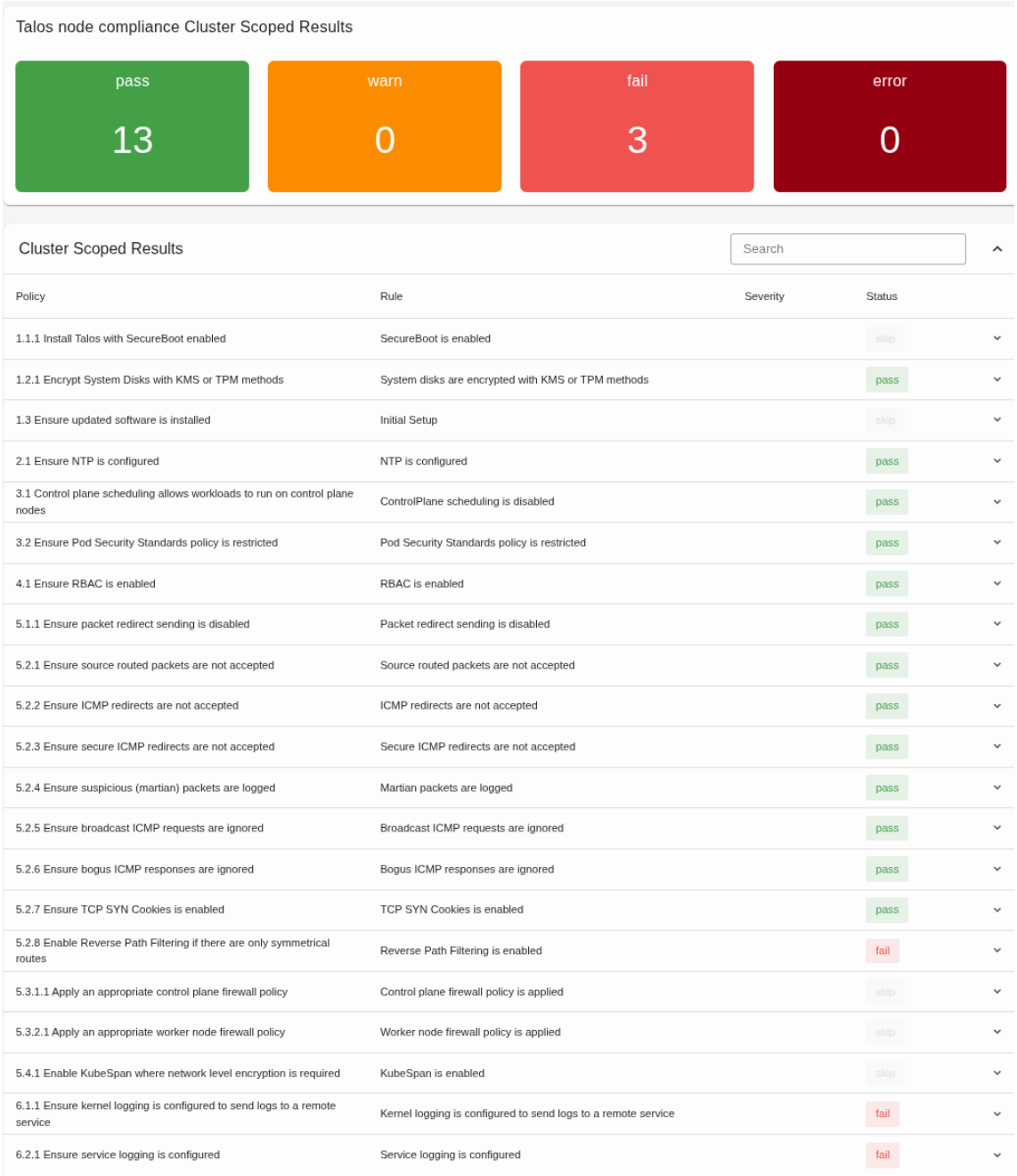

Figure 14. Talos Node Compliance report results

Appendix 7 - Availability and Fault Tolerance Test

The following output shows that at least one replica of each application was scheduled on node `t-w4`, while a distributed storage replica was located on node `t-w1`. Both nodes belong to the affected availability zone prior to the failure.

```
kubectl get nodes
NAME    STATUS     ROLES           AGE   VERSION
t-cp1   Ready      control-plane   23h   v1.34.5
t-cp2   Ready      control-plane   23h   v1.34.5
t-cp3   Ready      control-plane   23h   v1.34.5
t-w1    Ready      infra           23h   v1.34.5
t-w2    Ready      infra           23h   v1.34.5
t-w3    Ready      infra           23h   v1.34.5
t-w4    Ready      <none>          23h   v1.34.5
t-w5    Ready      <none>          23h   v1.34.5
t-w6    Ready      <none>          23h   v1.34.5

kubectl get pods -o wide -n shop
NAME                                 READY   STATUS    RESTARTS   AGE     IP
                NODE   NOMINATED NODE   READINESS GATES
adservice-5967684f6b-g92gp           1/1     Running   0          6m23s
    10.245.1.61    t-w4   <none>           <none>
cartservice-5997576d5b-ltdhc         1/1     Running   0          6m23s
    10.245.1.212   t-w4   <none>           <none>
checkoutservice-dd8f5b4bb-8lrhl      1/1     Running   0          6m23s
    10.245.1.235   t-w4   <none>           <none>
currencyservice-7bc4786db9-vvwqz     1/1     Running   0          6m22s
    10.245.1.21    t-w4   <none>           <none>
emailservice-f6b44d77b-2v6kw         1/1     Running   0          6m22s
    10.245.1.184   t-w4   <none>           <none>
frontend-697676dd64-8scmf            1/1     Running   0          6m22s
    10.245.1.180   t-w4   <none>           <none>
loadgenerator-6ff96b4c5f-hgs4f       1/1     Running   0          6m22s
    10.245.1.5     t-w4   <none>           <none>
paymentservice-75fdd996c5-gffcd      1/1     Running   0          6m22s
    10.245.1.74    t-w4   <none>           <none>
productcatalogservice-84d7b7fd4-cfx7p    1/1     Running   0          6m16s
    10.245.1.114   t-w4   <none>           <none>
recommendationservice-7976fc545b-gksxp   1/1     Running   0          6m16s
    10.245.1.192   t-w4   <none>           <none>
redis-cart-58d5cc968-96q8g           1/1     Running   0          6m16s
    10.245.1.250   t-w4   <none>           <none>
shippingservice-c8c6d4ff8-8n47w      1/1     Running   0          6m15s
    10.245.1.116   t-w4   <none>           <none>

kubectl get pods -o wide -n nextcloud
NAME                         READY   STATUS    RESTARTS   AGE     IP              NODE
    NOMINATED NODE   READINESS GATES
nextcloud-7b8fc7f7b5-jdcgw   1/1     Running   0          4m10s   10.245.1.24     t-w4
    <none>           <none>
nextcloud-7b8fc7f7b5-l78j5   1/1     Running   0          5m49s   10.245.7.231    t-w5
    <none>           <none>
```

```
kubectl get replicas -n longhorn-system
NAME                                                    DATA ENGINE   STATE     NODE
    DISK                                    INSTANCEMANAGER
                                        IMAGE
    AGE
pvc-7e3eaae9-0be4-4fa2-acc3-a9597d9bbb2f-r-28403235   v1            running   t-w1
    bedcefcd-2cb5-49ea-831b-d2f2ccc7a5a7
    instance-manager-33450b693df8a115228a716893eb532a
    docker.io/longhornio/longhorn-engine:v1.10.2   2m7s
pvc-7e3eaae9-0be4-4fa2-acc3-a9597d9bbb2f-r-729b0b90   v1            running   t-w2
    82b615b0-7c8c-4cba-9f06-a8250620b00b
    instance-manager-df5e256ec2c787e295ac9fff3e41acb9
    docker.io/longhornio/longhorn-engine:v1.10.2   18h
```

The following output shows that nodes `t-cp1`, `t-w1`, and `t-w4` in the affected availability zone transitioned to the `NotReady` state. As a result, Kubernetes rescheduled replacement pods, while the original pods on the failed nodes remained in the `Terminating state`.

```
kubectl get nodes
NAME    STATUS     ROLES           AGE   VERSION
t-cp1   NotReady   control-plane   23h   v1.34.5
t-cp2   Ready      control-plane   23h   v1.34.5
t-cp3   Ready      control-plane   23h   v1.34.5
t-w1    NotReady   infra           23h   v1.34.5
t-w2    Ready      infra           23h   v1.34.5
t-w3    Ready      infra           23h   v1.34.5
t-w4    NotReady   <none>          23h   v1.34.5
t-w5    Ready      <none>          23h   v1.34.5
t-w6    Ready      <none>          23h   v1.34.5

kubectl get pods -n nextcloud -o wide
NAME                         READY   STATUS        RESTARTS   AGE     IP
    NODE   NOMINATED NODE   READINESS GATES
nextcloud-7b8fc7f7b5-jdcgw   1/1     Terminating   0          8m20s   10.245.1.24
    t-w4   <none>           <none>
nextcloud-7b8fc7f7b5-l78j5   1/1     Running       0          9m59s   10.245.7.231
    t-w5   <none>           <none>
nextcloud-7b8fc7f7b5-wrgr9   1/1     Running       0          82s     10.245.4.30
    t-w6   <none>           <none>



kubectl get pods -n shop -o wide
NAME                                 READY   STATUS        RESTARTS   AGE   IP
                NODE   NOMINATED NODE   READINESS GATES
adservice-5967684f6b-d4sw8           1/1     Running       0          59s
    10.245.4.126   t-w6   <none>           <none>
adservice-5967684f6b-g92gp           1/1     Terminating   0          10m
    10.245.1.61    t-w4   <none>           <none>
cartservice-5997576d5b-dnmd9         1/1     Running       0          60s
    10.245.4.105   t-w6   <none>           <none>
cartservice-5997576d5b-ltdhc         1/1     Terminating   0          10m
    10.245.1.212   t-w4   <none>           <none>
checkoutservice-dd8f5b4bb-8lrhl      1/1     Terminating   0          10m
    10.245.1.235   t-w4   <none>           <none>
```

```
checkoutservice-dd8f5b4bb-jcz6w          1/1     Running       0          59s
    10.245.7.99    t-w5   <none>           <none>
currencyservice-7bc4786db9-79c45         1/1     Running       0          59s
    10.245.4.55    t-w6   <none>           <none>
currencyservice-7bc4786db9-vvwqz         1/1     Terminating   0          10m
    10.245.1.21    t-w4   <none>           <none>
emailservice-f6b44d77b-2v6kw             1/1     Terminating   0          10m
    10.245.1.184   t-w4   <none>           <none>
emailservice-f6b44d77b-xbqnd             1/1     Running       0          59s
    10.245.4.111   t-w6   <none>           <none>
frontend-697676dd64-8scmf                1/1     Terminating   0          10m
    10.245.1.180   t-w4   <none>           <none>
frontend-697676dd64-fc59h                1/1     Running       0          60s
    10.245.4.248   t-w6   <none>           <none>
loadgenerator-6ff96b4c5f-66qwh           1/1     Running       0          59s
    10.245.7.105   t-w5   <none>           <none>
loadgenerator-6ff96b4c5f-hgs4f           1/1     Terminating   0          10m
    10.245.1.5     t-w4   <none>           <none>
paymentservice-75fdd996c5-gffcd          1/1     Terminating   0          10m
    10.245.1.74    t-w4   <none>           <none>
paymentservice-75fdd996c5-kktgs          1/1     Running       0          59s
    10.245.4.119   t-w6   <none>           <none>
productcatalogservice-84d7b7fd4-cfx7p    1/1     Terminating   0          10m
    10.245.1.114   t-w4   <none>           <none>
productcatalogservice-84d7b7fd4-gjs6l    1/1     Running       0          59s
    10.245.4.5     t-w6   <none>           <none>
recommendationservice-7976fc545b-gksxp   1/1     Terminating   0          10m
    10.245.1.192   t-w4   <none>           <none>
recommendationservice-7976fc545b-p8vsz   1/1     Running       0          59s
    10.245.4.224   t-w6   <none>           <none>
redis-cart-58d5cc968-96q8g               1/1     Terminating   0          10m
    10.245.1.250   t-w4   <none>           <none>
redis-cart-58d5cc968-h6csc               1/1     Running       0          59s
    10.245.4.84    t-w6   <none>           <none>
shippingservice-c8c6d4ff8-7fj65          1/1     Running       0          60s
    10.245.4.210   t-w6   <none>           <none>
shippingservice-c8c6d4ff8-8n47w          1/1     Terminating   0          10m
    10.245.1.116   t-w4   <none>           <none>
```

Appendix 8 - Restore Test

```
# Before restore
kubectl exec -n nextcloud nextcloud-6884556767-54nr6 -c nextcloud -- sha512sum /var/www/html/data/peremees/files/2023_eits_english.pdf
473eeadd1d0c6c29adcead57a3427dc5f6bf45cb42a99f3b0f968218430fd4a881de82219da8c43ebc51325f93b506e76b21cfab54a7f8f495bc5d2b929983b5
    /var/www/html/data/peremees/files/2023_eits_english.pdf

# Delete namespace
kubectl delete ns nextcloud
namespace "nextcloud" deleted

# Restore backup
velero restore create nextcloud-restore \
  --from-backup velero-daily-backup-3-days-20260408130249 \
  --include-namespaces nextcloud
Restore request "nextcloud-restore" submitted successfully.
Run `velero restore describe nextcloud-restore` or `velero restore logs nextcloud-restore` for more details.

# After restore
kubectl exec -n nextcloud nextcloud-6884556767-54nr6 -c nextcloud -- sha512sum /var/www/html/data/peremees/files/2023_eits_english.pdf
473eeadd1d0c6c29adcead57a3427dc5f6bf45cb42a99f3b0f968218430fd4a881de82219da8c43ebc51325f93b506e76b21cfab54a7f8f495bc5d2b929983b5
    /var/www/html/data/peremees/files/2023_eits_english.pdf
```

Appendix 9 - Secure Communication Verification

```
# Request from Nextcloud pod
kubectl get pods -n shop -o wide frontend-868d857dc-xwt4z; kubectl get pods -n
    nextcloud nextcloud-6884556767-54nr6 -o wide; kubectl exec -n nextcloud
    nextcloud-6884556767-54nr6 -c nextcloud -- curl -I 10.245.7.2:8080
NAME                        READY   STATUS    RESTARTS   AGE     IP            NODE
    NOMINATED NODE   READINESS GATES
frontend-868d857dc-xwt4z   1/1     Running   0          2m19s   10.245.7.2   t-w5
    <none>           <none>
NAME                           READY   STATUS    RESTARTS   AGE   IP              NODE
    NOMINATED NODE   READINESS GATES
nextcloud-6884556767-54nr6   1/1     Running   0          89m   10.245.10.154   t-w4
    <none>           <none>
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0HTTP/1.1
      200 OK
Set-Cookie: shop_session-id=ca8cea63-bcf4-41da-9eec-e8179b503a21; Max-Age=172800
Date: Sun, 05 Apr 2026 18:01:00 GMT
Content-Type: text/html; charset=utf-8

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0


# Captured network dump indicating TLSv1.3 traffic
talosctl -n t-w5 pcap -i eth0 -o - | tshark -r - -Y 'ip.addr==10.245.10.154 &&
    ip.addr==10.245.7.2 && tls'
  761 2.401590033 10.245.10.154 -> 10.245.7.2   TLSv1.3 275 Client Hello
  763 2.402028704   10.245.7.2 -> 10.245.10.154 TLSv1.3 1079 Server Hello, Change
      Cipher Spec, Application Data
  765 2.402881711 10.245.10.154 -> 10.245.7.2   TLSv1.3 877 Change Cipher Spec,
      Application Data
  766 2.402882968 10.245.10.154 -> 10.245.7.2   TLSv1.3 112 Application Data
  768 2.402944248 10.245.10.154 -> 10.245.7.2   TLSv1.3 360 Application Data
  769 2.403203204   10.245.7.2 -> 10.245.10.154 TLSv1.3 250 Application Data
  770 2.403241240   10.245.7.2 -> 10.245.10.154 TLSv1.3 121 Application Data
  771 2.403263782   10.245.7.2 -> 10.245.10.154 TLSv1.3 101 Application Data
  772 2.403321650   10.245.7.2 -> 10.245.10.154 TLSv1.3 97 Application Data
  773 2.403521575   10.245.7.2 -> 10.245.10.154 TLSv1.3 209 Application Data
  775 2.403727487 10.245.10.154 -> 10.245.7.2   TLSv1.3 97 Application Data
  776 2.403919544 10.245.10.154 -> 10.245.7.2   TLSv1.3 177 Application Data
  865 2.424837934   10.245.7.2 -> 10.245.10.154 TLSv1.3 275 Application Data
  866 2.425557215 10.245.10.154 -> 10.245.7.2   TLSv1.3 97 Application Data
  867 2.426023483   10.245.7.2 -> 10.245.10.154 TLSv1.3 97 Application Data
```

Appendix 10 - Scheduling Restriction Test

```
root@nextcloud-868bdc8d7b-mfm7w:/var/www/html# kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: malicious-pod
  name: malicious-pod
  namespace: nextcloud
spec:
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    operator: Exists
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  containers:
  - image: nginx
    name: malicious-pod
    resources:
      limits:
        cpu: 100m
        memory: 100Mi
      requests:
        cpu: 100m
        memory: 100Mi
  dnsPolicy: ClusterFirst
  restartPolicy: Always
EOF
Error from server: error when creating "STDIN": admission webhook
   "validate.kyverno.svc-fail" denied the request:

resource Pod/nextcloud/malicious-pod was blocked due to the following
   policies

restrict-controlplane-scheduling:
  restrict-infra-node-scheduling: 'validation error: Pods may not use
     tolerations
    which schedule on infra nodes. rule restrict-infra-node-scheduling
       failed at path
    /spec/tolerations/0/key/'
```

Appendix 11 - Privilege Escalation Resistance Test

```
root@nextcloud-868bdc8d7b-mfm7w:/var/www/html# kubectl label namespace
   nextcloud pod-security.kubernetes.io/enforce=privileged --overwrite
Error from server: admission webhook "validate.kyverno.svc-fail"
   denied the request:

resource Namespace//nextcloud was blocked due to the following policies

deny-namespace-modification:
  check-namespace-update: Only approved service accounts may modify
     Namespace information
```

Appendix 12 – E-ITS SYS.1.3 Linux and Unix server: Implementation, Evidence and Result

Table 11. SYS.1.3 module: implementation, validation, evidence and result

E-ITS measure	Validation	Evidence	Result
M2 Proper definition of indicators	Validation not required; enforced by Talos Linux design	Talos Linux does not use traditional UNIX user and group management	Measure not applicable
M3 Avoiding automatic connection of removable devices to the file system	Validation not required; enforced by Talos Linux design	Talos Linux does not support automatic mounting of removable devices	Measure not applicable
M4 Protection of applications	Validation not required; enforced by Talos Linux design	Talos Linux uses a hardened kernel	Measure not applicable
M5 Secure installation of software packages	Validation not required; enforced by Talos Linux design	Talos Linux does not support installation of software packages on the host system. All applications are deployed as container images rather than installed on the operating system.	Measure not applicable
M6 Administration of users and groups	Validation not required; enforced by Talos Linux design	Talos Linux does not use traditional UNIX user and group management.	Measure not applicable
M8 SSH-encrypted data exchange	Validation not required; enforced by Talos Linux design	Talos Linux does not use the SSH protocol	Measure not applicable
M10 Prevention of the unauthorised expansion of rights	Validation based on aggregated policy compliance reports provided by Policy Reporter	Talos node compliance report: “Ensure Pod Security Standards policy is restricted”	Measure achieved

E-ITS measure	Validation	Evidence	Result
M14 Prevention of system and user information intelligence	Validation not required; enforced by Talos Linux design	Talos Linux does not provide interactive user access to the operating system. Access to system information is restricted to controlled API-based mechanisms and not available to general users.	Measure not applicable
SYS.1.3.M16 Additional prevention of the unauthorised expansion of rights	Validation based on aggregated policy compliance reports provided by Policy Reporter	Trivy ConfigAudit results were used to assess workload security configuration related to privilege restrictions and seccomp usage. Explicit seccomp profiles were not configured in the prototype.	Measure partially achieved
SYS.1.3.M17 Additional kernel security	Validation not required; enforced by Talos Linux design	Talos Linux uses a hardened Linux kernel with security-focused configuration	Measure not applicable

Appendix 13 – E-ITS SYS.1.6 Containerisation Measures: Implementation, Evidence and Result

Table 12. SYS.1.6 module: implementation, validation, evidence and result

E-ITS measure	**Validation**	**Evidence**	**Result**
M1 Secure configuration of container platform	Validation based on review of the documented platform architecture design and implemented controls	The platform is implemented using Kubernetes with GitOps-based deployment, supporting isolated workloads across namespaces. Security mechanisms such as network policies, RBAC, and policy enforcement are applied as described in the platform implementation. Platform goals and security risks are addressed, but no explicit cost analysis is documented.	Measure partially achieved
M2 Lifecycle & regular security updates	Validation based on inspection of GitOps configuration, repository structure, and applied security and auditing controls	Application lifecycle management is implemented using Flux GitOps, with deployments, updates, and configuration changes managed through version-controlled repositories. Policy Reporter and Trivy Operator provide continuous security assessment of workloads.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M3 Secure operation of containerised IT systems	Validation based on inspection of isolation mechanisms, network policies, and monitoring configuration	Secure operation is implemented through namespace isolation, network policies enforcing restricted communication, and RBAC-based access control. Monitoring and health checks are validated through Policy Reporter results and workload configurations. High availability is demonstrated through failure testing and workload rescheduling.	Measure achieved
M4 Secure implementation of container images	Validation based on review of deployment manifests and repository configuration	Container images are referenced in Kubernetes manifests and deployed through GitOps. No dedicated process for secure image building or image hardening is implemented in the prototype.	Measure partially achieved
M5 Partitioning the management network of containers	Validation based on configuration review of network policies and policy compliance reports	Implemented using network policies with default-deny behaviour, ensuring that only explicitly defined communication paths between services are allowed. Compliance is shown through Trivy Compliance report results. Hubble observability confirms restricted network flows and visibility of allowed traffic.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M6 Use of secure container images	Validation based on verification of image sources, policy compliance, and vulnerability scanning results	A Kyverno policy “restrict-image-registries” audits the use of approved container image registries. Container images are versioned and deployed through GitOps. Trivy Operator performs vulnerability scanning of images. Full verification of image origin and supplier security practices is not implemented.	Measure partially achieved
M7 Storage of container log data	Validation based on verification of logging configuration and external storage mechanisms	Container logs are handled according to Kubernetes logging architecture, where logs are written to standard output and collected by the container runtime at the node level.	Measure achieved
M8 Secure management of container access data	Validation based on inspection of secret management configuration and access control policies	Credentials are stored as Kubernetes Secrets and are encrypted in Git repositories using SOPS. Secrets are not embedded in container images and are injected at runtime. Access is restricted through RBAC policies. Implementation is validated through Trivy Compliance report results	Measure achieved
M9 Assessing the suitability of applications	Validation based on testing applications and reviewing deployment behaviour	The laboratory environment was used to test applications deployed in the cluster. Applications such as the Online Boutique and Nextcloud were evaluated for compatibility with platform security mechanisms and operational requirements.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M10 Guidelines for the use of containers	Validation based on policy enforcement checks and compliance reports	Operational rules for containers are enforced through Kyverno policies, including restrictions such as approved image registries and security configurations. These rules are applied through GitOps-managed configurations and validated through Policy Reporter. A formal documented policy for container usage is not created.	Measure partially achieved
M11 Partitioning applications in containers	Validation based on inspection of deployment configurations	Applications in the prototype follow a service-based architecture, where containers are used to provide individual services.	Measure achieved
M12 Secure distribution of container images	Validation based on verification of image sources and policy compliance	A Kyverno policy (restrict-image-registries) audits the use of approved container image registries. Images are referenced using versioned tags and deployed through GitOps-managed configurations. Mechanisms such as image signing or formal approval criteria are not implemented.	Measure partially achieved
M13 Approving container images for use	Validation based on review of deployed images and compliance reports	Container images are deployed through GitOps-managed repositories without a formal pre-deployment approval process. Trivy provides post-deployment visibility into image security. However, no approval workflow is enforced.	Measure partially achieved

E-ITS measure	Validation	Evidence	Result
M14 Updating container images	Validation based on review of version updates and deployment history	Updates to container images are performed by modifying image versions in Git repositories and redeploying workloads through the GitOps workflow, following immutable infrastructure principles. However, no formal change management process or defined update policy is implemented.	Measure partially achieved
M15 Defining the resource limitations of a container	Validation based on inspection of resource configurations and compliance reports	Resource requests and limits are defined and enforced through Kyverno policies. Validation is supported by Trivy ConfigAudit report results.	Measure achieved
M16 Secure remote maintenance of containers	Platform design validation	Remote maintenance operations are performed through the Kubernetes API and Talos API. Containers do not have direct access to host systems, and management is restricted to platform-controlled interfaces.	Measure achieved
M17 Defining the minimum rights required for operating a container	Validation based on inspection of security context configurations and policy reports	Minimal privilege execution is enforced through Kubernetes Pod Security Admission and Kyverno policies, validated through aggregated results in Policy Reporter.	Measure achieved
M18 Restricting the rights of containerised applications	Validation based on compliance and policy reports	Aggregated results in Policy Reporter. Pod Security Standards are enforced and Kyverno policies are used.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M19 Regulating access to data recorders	Validation based on inspection of volume configuration and access restrictions	Containers access storage through Kubernetes persistent volumes backed by Longhorn distributed storage. Access is limited to explicitly defined volumes. Aggregated results in Policy Reporter.	Measure achieved
M20 Security of the configuration information of a container	Validation based on repository history and configuration review	Container configuration is stored and managed in Git repositories using a GitOps approach. All configuration changes are versioned and documented.	Measure achieved
M22 Using container images as evidence	Not applicable	At the time of implementation, Talos Linux did not support the required kernel configuration for CRIU.	Measure not implemented
M23 Ensuring the constancy of containers	Validation based on policy compliance reports	Policy Reporter and Trivy ConfigAudit findings such as "Root File System Is Not Read-Only".	Measure achieved
M24 Host-based attack detection	Not applicable	Not implemented, as no runtime detection mechanisms are deployed in the prototype	Measure not implemented
M25 High availability of containerised applications	Validation based on review of node distribution and scheduling configuration	Cluster nodes are distributed across availability zones, enabling resilient scheduling and maintaining availability in failure scenarios.	Measure achieved
M21 Expanded security rules of containerisation	Validation based on compliance and policy reports	Access rights are enforced through network policies, Pod Security Standards, and Kyverno policies. However, explicit syscall-level controls are not implemented.	Measure partially achieved

E-ITS measure	Validation	Evidence	Result
M26 Further isolation and encapsulation of containers	Not applicable	Not implemented, as advanced isolation mechanisms such as hypervisor-based isolation are not used.	Measure not implemented

Appendix 14 – E-ITS APP.4.4 Kubernetes Measures: Implementation, Evidence and Result

Table 13. APP.4.4 module: implementation, validation, evidence and result

E-ITS measure	Validation	Evidence	Result
M1 Designing the partition of applications	Validation based on inspection of network policies, resource quotas, and cluster architecture	Applications are deployed in separate Kubernetes namespaces representing different tenants and environments. Network policies enforce restricted communication between namespaces, and resource quotas and limits are applied to control resource usage. Configuration is defined declaratively in Git repositories and applied through GitOps.	Measure achieved
M2 Automation of the development of applications with the help of CI/CD	Validation based on inspection of GitOps workflow and repository configuration	Application deployment and lifecycle management are implemented using Flux GitOps. Application manifests are stored in version-controlled repositories and automatically applied to the cluster. Access to repositories and deployment processes is restricted to authorised users.	Measure achieved
M3 Planning the Kubernetes identity and rights management	Validation based on inspection of RBAC configuration and access control policies	RBAC configuration is defined declaratively in version-controlled Git repositories, including roles and role bindings, and applied through GitOps. Aggregated results in Policy Reporter.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M4 Partition of pods	Validation not required; enforced by Kubernetes and container runtime design, supported by Talos Linux	Pod isolation is provided by Kubernetes and the container runtime using Linux namespaces and cgroups. Workloads are deployed as isolated pods with separate process, network, and file system contexts.	Measure achieved
M5 Backup of cluster information	Validation based on verification of backup configuration and restore testing	Backup is implemented using Velero with configuration stored in Git repositories. Backup schedules and restore tests confirm functionality.	Measure achieved
M6 Secure resetting of the pods	Validation based on inspection of workload manifests for init container usage	The platform supports secure resetting of applications through Kubernetes init containers and pod lifecycle mechanisms.Implementation is application-specific.	Measure achieved
M7 Partition of Kubernetes networks	Validation based on inspection of network policies and connectivity testing	Network segmentation implemented using with default-deny network policies. Only explicitly defined communication paths are allowed. Hubble confirms traffic flows.	Measure achieved
M8 Security of the Kubernetes configuration files	Validation based on analysis of repository history and access control configuration	Configuration is managed through GitOps using version-controlled repositories. No manual changes to cluster resources are performed.	Measure achieved
M9 Security of the Kubernetes service accounts	Validation based on review of Policy Reporter outputs for Kyverno policies enforcing service account token restrictions	Policy Reporter results for Kyverno policy “Disable-Automountserviceaccounttoken”.	Measure achieved
M10 Security of the automation process	Validation based on analysis of repository separation and RBAC configuration	Flux multi-tenancy configuration with separate repositories and access controls.	Measure achieved
M11 Monitoring the use of containers	Validation through aggregated policy compliance reports via Policy Reporter	Policy Reporter results for Kyverno policy "require-pod-probes".	Measure achieved

E-ITS measure	Validation	Evidence	Result
M12 Security of infrastructure applications	Validation through analysis of configuration, access control, and backup mechanisms	RBAC configuration, Istio mTLS communication, Kyverno reports, Velero backups, and GitOps configuration history.	Measure achieved
M13 Automatic configuration audits	Validation through aggregated policy compliance reports via Policy Reporter	Aggregated policy compliance reports via Policy Reporter.	Measure achieved
M14 Use of specialised nodes	Validation based on analysis of node roles, labels, and workload placement	Node labels, taints, and workload placement across control plane and worker nodes. While infrastructure nodes are used for ingress components, they are shared with other infrastructure services. No dedicated bastion nodes are implemented.	Measure partially implemented
M15 Partition of applications at the levels of nodes and clusters	Validation based on analysis of cluster architecture and workload placement	Cluster architecture and node role configuration provide separation between infrastructure and application workloads. However, tenant workloads are scheduled on shared worker nodes, and no tenant-specific node-level partitioning is enforced. A single Kubernetes cluster is used under the assumption that all applications have similar protection requirements.	Measure partially implemented
M16 Use of Kubernetes operators	Validation based on observation of reconciliation behaviour	Flux reconciliation ensures automatic deployment and drift correction.	Measure achieved

E-ITS measure	Validation	Evidence	Result
M17 Certification of nodes	Validation based on analysis of node certificate configuration	Talos manages node certificates and enforces mTLS, ensuring that only nodes with valid certificates can authenticate and communicate with the control plane. However, the solution does not implement hardware-based attestation at boot or runtime.	Measure partially implemented
M18 Microsegmentation	Validation based on connectivity testing and network policy enforcement	Connectivity tests and Hubble flow visibility confirm restricted communication.	Measure achieved
M19 Guaranteeing the high availability of Kubernetes	Validation based on failure testing and system behaviour analysis	Failure testing with simulated zone disruption and workload rescheduling.	Measure achieved
M20 Encryption of the control plane storage space	Validation through compliance reports (Trivy, Talos)	Trivy and Talos compliance reports in Policy reporter confirming encryption settings.	Measure achieved
M21 Periodic restarting of the pods	Validation based on observation of pod lifecycle events	Pod lifecycle events showing periodic rescheduling.	Measure achieved