TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Marko Lindeberg 221892IVCM

ANALYSIS AND IMPLEMENTATION OF 'APP.4.4: KUBERNETES' FROM THE ESTONIAN INFORMATION SECURITY STANDARD (E-ITS)

Master's Thesis

Supervisor: Siim Vene
MSc

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marko Lindeberg 221892IVCM

EESTI INFOTURBESTANDARDI (E-ITS) 'APP.4.4: KUBERNETES' ANALÜÜS NING TEOSTUS

Magistritöö

Juhendaja: Siim Vene
MSc

Tallinn 2025

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: Marko Lindeberg

18.05.2025

Abstract

As software development has been shifting from monolithic architecture to microservices architecture, containerisation has become one of the most common and popular way for running and managing services efficiently, and independently. However, this comes with an increasing reliance on container orchestration platforms, particularly Kubernetes, that has transformed software deployment and scalability.
The increasing adoption of Kubernetes introduces significant security challenges that many organizations struggle to address as default configuration often fail to meet the security requirements, exposing clusters to vulnerabilities. According to a survey conducted by Red Hat, 42% of respondents suggested that their company does not have enough capabilities to address container security threats [1]. Additionally, the same report highlights that 67% of organizations have delayed or slowed down their deployments and 46% have lost revenue or customers due to Kubernetes security incident [1]. To address these challenges, the Estonian Information Security Standard (E-ITS) has released a module 'APP.4.4: Kubernetes' module in the baseline security catalogue.

E-ITS 'APP.4.4: Kubernetes' offers a strong foundation for improving Kubernetes security posture, however, without a practical guidance, many organizations face challenges to implement the security measures to meet the compliance requirements. A Systematic Literature Review (SLR) is conducted to evaluate the existing research on Kubernetes security, identifying key contribution, limitations and research gaps. A detailed analysis of E-ITS APP.4.4: Kubernetes module to identify the specific security requirements followed by a high-level architectural design and suitable tool selection based on the identified requirements. Finally, a comprehensive technical implementation is performed followed by a validation process to ensure that the implemented Kubernetes cluster meets the E-ITS APP.4.4: Kubernetes requirements. The outcome of this research provides a practical implementation steps that can be used as a template, guidance or reference for organization to achieve E-ITS compliance for their Kubernetes clusters.

The thesis is written in English and is 83 pages long, including 8 chapters, 20 figures and 5 tables.

Annotatsioon

Eesti infoturbestandardi (E-ITS) 'APP.4.4: Kubernetes' analüüs ning teostus

Tarkvaraarendus on liikumas monoliitselt arhitektuurilt mikroteenuste arhitektuuri suunas ning seetõttu on peamiseks ning kõige populaarsemaks teenuste haldamise viisiks saanud konteinerdus. Sellest tingituna, on suurenenud ka sõltuvus konteinerite orkesteerimise platvormide vastu, eelkõige Kubernetese, mis on avaldanud suurt positiivset mõju tarkvara paigalduses ja mastaabitavuses.

Kubernetese järjest kasvav kasutuselevõtt on toonud kaasa olulisi turbealaseid väljakutseid, millega paljud ettevõtted silmitsi seisavad, kuna vaikeseaded ei vasta tihti turvanõuetele ning sellest tingituna võivad jätta Kubernetese klastrid haavatavaks. Red Hat-i poolt teostatud uuringu kohaselt leidis 42% vastajatest, et nende ettevõttel puudub võimekus konteinerduse turvalisuse ohtude vastu võitlemiseks [1]. Samas raportis on tõstatatud, et 67% ettevõtetest on pidanud erinevat Kubernetese turvaprobleemide tõttu viivitama oma tarkvarajuurutusi, ning 46% ettevõtetest on kaotanud kas kliente või tulu [1]. Nende väljakutsete lahendamiseks on Eesti Infoturbestandardi (E-ITS) etalonturbe kataloogis moodul 'APP.4.4: Kubernetes'.

E-ITS 'APP.4.4: Kubernetes' moodul on küll tugev põhi Kubernetese turvaohtude likvideerimiseks, kuid ilma praktilise juhendita seisavad paljud ettevõtted siiski silmitsi probleemiga nende tõhusaks lahendamiseks. Antud lõputöö raames teostatakse süstemaatiline kirjanduse ülevaade, mille käigus analüüsitakse Kubernetese turvalisusega seotud akadeemilist kirjandust, tuvastatakse peamised piirangud ning uuriislüngad. Seejärel analüüsitakse põhjalikult E-ITS 'APP.4.4: Kubernetes' moodulit, et tuvastada konkreetsed turvanõuded ning luuakse arhitektuuriline lahendus ning valitakse välja sobivad tööriistad ja tehnoloogiad, mis põhinevad tuvastatud nõuetel. Järgneb põhjalik tehniline teostus koos valideerimisprotsessiga, mille käigus paigaldatakse ja seadistatakse Kubernetese klaster, mis vastab E-ITS 'APP.4.4: Kubernetes' mooduli nõuetele. Lõputöö tulemuseks on tehniline teostus, mida ettevõtted saavad kasutada praktilise malli, juhendi või näitena, et saavutada Kubernetese klastri vastavus E-ITS mooduli nõuetele.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 83 leheküljel, 8 peatükki, 20 joonist, 5 tabelit.

List of Abbreviations and Terms

ASO	*Azure Service Operator*, a Kubernetes operator that allows managing Azure resources using Kubernetes manifests
BSI	*Bundesamt für Sicherheit in der Informationstechnik*, German Federal Office for Information Security
cgroups	abbreviation from Linux *control groups*
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface, a framework for dynamically configuring network interfaces in Linux containers
CSI	Container Storage Interface, a standard for exposing arbitrary block and file storage systems to containerized workloads
CVE	Common Vulnerabilities and Exposures, publicly disclosed cybersecurity vulnerabilities
etcd	highly-available key-value store for Kubernetes cluster data
GID	Group Identifier, a unique number assigned to each group on Linux
GitOps	an operational framework of DevOps best practices for infrastructure automation
ISO	International Organization for Standardization
kubelet	Kubernetes "node agent", responsible for managing containers created by Kubernetes
mTLS	mutual Transport Layer Security, a method of mutual authentication between client and server
Overlay	an encapsulation of a network protocol that allows for the creation of a virtual network on top of an existing network
Pod	the smallest deployable unit in Kubernetes, a group of one or more containers
RBAC	Role-Based Access Control, a method for restricting system access to authorized users
SIG	Special Interest Group, a community-driven working group in Kubernetes project
UID	User Identifier, a unique number assigned to each user account on Linux

VNet	Azure Virtual Network, a service that provides an isolated network in Azure

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

The increasing adoption of Kubernetes introduces significant security challenges that many organizations struggle to address as default configuration often fail to meet the security requirements, exposing clusters to vulnerabilities. According to a survey conducted by Red Hat, 42% of respondents suggested that their company does not have enough capabilities to address container security threats [1]. Furthermore, the same report highlights that 67% of organizations have delayed or slowed down their deployments and 46% have lost revenue or customers due to Kubernetes security incident [1].

E-ITS "APP.4.4: Kubernetes" offers a strong foundation for improving Kubernetes security posture, however, without practical guidance, many organizations face challenges in implementing these requirements effectively. This research is important because it provides a comprehensive analysis of "APP.4.4: Kubernetes" module and demonstrates how to implement its requirements in practice.

By offering a clear practical solution, this study aims to support organizations in Estonia, and potentially beyond, to improve their security posture by aligning with E-ITS. The outcome of this research will ease the adoption of E-ITS, by making it more accessible for organizations to strengthen their Kubernetes security and overall security posture.

1.2 Research Problem

1.2.1 Research Problem Statement

Organizations are increasingly adopting Kubernetes to manage their containerized applications, however ensuring a comprehensive security compliance still remains a significant challenge. While standards, such as E-ITS define the baseline security requirements, there is a lack of clear practical guidance, methodologies and tools to achieve compliance. Although Kubernetes security has been extensively studied, the existing literature focuses on security vulnerabilities and high-level recommendations, rather than providing actionable steps for organizations. Additionally, there is a lack of research that specifically addresses Kubernetes security on country-specific standards, such as E-ITS. This shows that there is a clear need to identify effective tools, processes and methods that can be used by

organizations to adopt E-ITS 'APP.4.4: Kubernetes' module and improve their security posture.

1.2.2 Research Questions

- **[RQ1]:** How can organizations implement E-ITS 'APP.4.4: Kubernetes' to achieve compliance and enhance the security posture of their Kubernetes clusters?
- **[RQ2]:** What tools, methods, processes can be used for the E-ITS 'APP.4.4: Kubernetes' adoption?
- **[RQ3]:** What are the specific security requirements outlined in E-ITS 'APP.4.4: Kubernetes' and how do they align with Kubernetes best practices?
- **[RQ4]:** What gaps exist between E-ITS 'APP.4.4: Kubernetes' and current Kubernetes security in academic literature?

1.3 Scope and Goal

1.3.1 Main objective

Many organizations, especially in the public sector, seek to improve their Kubernetes security posture by complying with standards such as the E-ITS 'APP.4.4: Kubernetes' module. This thesis aims to bridge the gap between understanding these requirements and practical implementation. Its main objective is to provide organizations with a detailed technical implementation guide that could be used as a template or a reference to achieve compliance with this module. The research involves a comprehensive analysis of the module's requirements and its alignment with academic literature and best practices, followed by a comparative selection of technologies and a technical implementation. The final output will be a practical and actionable guide that is designed to help organizations achieve E-ITS alignment and thereby increase their Kubernetes security posture.

1.3.2 Limitations

One limitation of this research is that it is not explicitly focusing on E-ITS 'SYS.1.6: Containerisation' module. Although, container security is an important topic, it focuses on a rather narrow aspect of security, whereas Kubernetes is a huge and complex ecosystem of numerous layers and aspects. Furthermore, the measures of 'APP.4.4: Kubernetes' module are always discussed together with the measures from module 'SYS.1.6: Containerisation' [2, p. 1.3]. This implies that the container images that are used within the Kubernetes cluster shall meet the module 'SYS.1.6: Containerisation' requirements. Focusing on

SYS.1.6: Containerisation module in addition to APP.4.4: Kubernetes would significantly broaden the scope of this Thesis and risk a scope creep.
Another limitation is that this Thesis does not focus on the managed Kubernetes services offered by public cloud providers, such as Elastic Kubernetes Service (EKS) by Amazon Web Services (AWS), Azure Kubernetes Service (AKS) by Microsoft Azure and Google Kubernetes Engine (GKE) by Google Cloud Platform (GCP). This is due to the reason that each service provider's managed Kubernetes implementation could be different and not allow as granular configuration as needed in E-ITS requirements.
Instead, this Thesis focuses on 'plain vanilla' Kubernetes that is an open-source version of Kubernetes hosted and maintained by the Cloud Native Computing Foundation (CNCF) [3], as this version could be deployed anywhere - private cloud (on-premises), public-cloud or hybrid-cloud.

1.4 Novelty

This study is novel in its focus to align Kubernetes security practices with the Estonian Information Security Standard (E-ITS) "APP.4.4: Kubernetes" - a national framework that lacks guidance for practical implementation. While existing research covers Kubernetes security best practices, there are no studies that address specific requirements of E-ITS 'APP.4.4: Kubernetes' or other national standards, such as BSI 'APP.4.4: Kubernetes' and NIST CSF 2.0. This thesis fills that gap by analysing E-ITS 'APP.4.4: Kubernetes', identifying alignment and contradictions in current practices and developing a comprehensive practical guidance to achieve compliance.

The primary contribution of this study is the development of a practical guidance specifically focused on E-ITS 'APP.4.4: Kubernetes'. On top of providing a practical solution for Estonian organizations to improve Kubernetes security, this guidance offers a model that can be adapted for similar requirements in other country-specific standards.

1.5 Thesis Structure

This thesis is structured as follows:

- Chapter 2 presents a Systematic Literature Review (SLR) which evaluates existing research about Kubernetes security to identify the best practices, key contributions, limitations and research gaps. This review provides essential context and founda-

tional knowledge relevant to **RQ3** and **RQ4** by identifying the Kubernetes best practices and research gaps in the current literature.

- Chapter 3 outlines the research methodology used in this thesis, describing the mixed-method approach in phases, describing the data collection and analysis techniques and the validation methods.
- Chapter 4 provides a comprehensive analysis of E-ITS 'APP.4.4: Kubernetes' module, identifies its specific security requirements and their alignment with Kubernetes best practices. This chapter primarily addresses **RQ3** and **RQ4**, partly based on the SLR findings in the SLR chapter.
- Chapter 5 details the practical phase of this thesis. It describes the design choices, such as resource platform, provides the high-level architecture, evaluates and selects the suitable tools based on the identified requirements and comparative analysis, and outlines the technical implementation steps for deploying and configuring an E-ITS compliant Kubernetes cluster. Thus, contributing to **RQ1** and **RQ2** by providing specific tools, methods and processes that can be used to implement E-ITS 'APP.4.4: Kubernetes' security measures.
- Chapter 6 presents the outcomes of the technical implementation and focuses on validating that the implemented Kubernetes cluster complies with each E-ITS 'APP.4.4: Kubernetes' security measure and therefore addresses **RQ1** and **RQ2**.
- Chapter 7 describes the findings of the research by providing comprehensive answers to **RQ1, RQ2, RQ3, RQ4**, highlights the study limitations and offers recommendations for improvement.
- Chapter 8 summarizes the entire study, reiterates the main contributions and suggests the future work.

2. Literature Review

In this paragraph, a systematic literature review is conducted. This literature review systematically evaluates existing research on Kubernetes security, identifying key contributions, limitations, and research gaps. Given that the E-ITS is a country-specific standard, this review aims to assess how current studies address Kubernetes security and whether they address country-specific frameworks or standards, such as E-ITS, and what aspects remain unexplored in that context. This provides essential background and context for **RQ3** and **RQ4** that will be addressed in the following chapters.

Kubernetes has become the *de facto* standard for container orchestration [2, p. 1.1], but its security remains a critical challenge. Misconfigurations, unauthorized access, and insecure network policies can expose clusters to attacks. Given the growing adoption of Kubernetes, a comprehensive review of existing research is necessary to assess how security challenges are being addressed and to identify gaps that require further study.

2.1 Search Strategy

While doing the initial research, it became clear that there is a fairly limited amount of literature published on E-ITS as the standard itself is quite fresh - the initial version took effect in 2023, and the current 'APP.4.4: Kubernetes' module in use is from 2023. Thus, the research criteria needed to be expanded to find literature on other country-specific standards and frameworks, such as BSI [4], NIST [5] or any type of literature on Kubernetes security in general.

2.1.1 Search Sources

The following databases were used in search strategy to find the literature:

- Google Scholar
- IEEE Xplore
- Scopus
- PRIMO Search Portal (ESTER and TalTech Library)

2.1.2 Search Terms

The following keywords and combinations were used:

Keywords

- kubernetes
- security
- best practices
- E-ITS, EITS, Estonian Information Security Standard
- standard
- framework
- container orchestration

Combinations

- ’"kubernetes security"’
- ’kubernetes security best practices’
- ’kubernetes AND security’
- ’kubernetes framework OR standard’
- ’kubernetes AND "E-ITS"’
- ’"E-ITS" OR "EITS" OR "Estonian Information Security Standard"’
- ’container orchestration AND security’

2.2 Inclusion and Exclusion Criteria

To refine the search results, the following inclusion and search criteria were used:

2.2.1 Inclusion criteria

- Peer-reviewed journals, conference papers, books (by trusted publishers), theses, publications from government agencies
- Studies focused on Kubernetes security, misconfigurations, vulnerabilities and best practices
- Researches discussing frameworks or standards for Kubernetes security
- Studies focusing on E-ITS, BSI, NIST or other standards and frameworks that are related to Kubernetes

2.2.2 Exclusion criteria

- Articles solely focusing on container security or application security but not on the orchestrator
- Articles not written in Estonian or English
- Articles not focusing on Kubernetes
- Articles that are focusing on Kubernetes but not on security
- Articles published before 2020, unless highly relevant (e.g. NIST 800-190)
- Articles behind paywalls
- Articles that are still in-review, blog posts, vendor-specific whitepapers (to avoid bias)
- Articles focusing on tools or technologies not relevant this research
- Articles focusing on only one aspect of the Kubernetes security
- Articles shorter than 4 pages
- Articles that are still in-review/preprint

2.3 Data Extraction Criteria

For consistent and systematic analysis of the selected studies, specific data is extracted from each paper based on the defined criteria. This will help to identify common patterns, trends and gaps in the existing literature. The following data points will be extracted from each selected study:

- **Bibliographic Information**
  Publication title, authors, and year
  Publication type (journal article, conference paper, technical report, etc.)
- **Research Focus and Scope**
  Primary security domain in Kubernetes (e.g., network security, access control, container security)
  Specific Kubernetes components that were addressed
  Methodology used, if applicable
- **Addressed Security Aspects**
  Specific identified vulnerability or attack methods
  Highlighted security misconfigurations
  Discussed security controls
  Proposed frameworks or remediation strategies
- **Implementation Details**
  Provided practical implementation guidelines/steps

Proposed and evaluated tools, technologies and solutions

Validation methods (metrics, benchmarks, etc)

- **Frameworks and Standards**

  References to existing frameworks or standards (NIST, CIS, E-ITS, BSI, etc.)

  Mentined country-specific standards or regulations

  Specific mentions of E-ITS or similar national standards

- **Limitations and Research Gaps**

  Explicitly stated limitations of the study

  Proposed future works or areas by the authors

  Identified gaps in the existing literature

  Described challenges during the research (e.g., when implementing security controls)

- **Target Audience and Context**

  Industry sector (if applicable)

  Targeted audience (practitioners, researchers, etc.)

  Operational context (e.g., enterprise environment, public or private cloud, critical infrastructure, etc.)

  Organizational context (e.g., small businesses, large enterprises, government agencies, if applicable)

- **Technical Details and Relevance**

  The level of detail in the technical part

  Code or configuration examples included

  Specific versions mentioned (if relevant - avoid deprecated or outdated versions)

2.4 Selection

The initial search across multiple databases yielded 20,349 findings. Following the application of inclusion and exclusion criteria, 559 results remained. After applying the inclusion and exclusion criteria, duplicate studies were identified across various sources during the relevance evaluation of the papers. The remaining papers were then systematically evaluated against the data extraction criteria. Following the assessment of relevance and the elimination of duplicate records, manual inspection of abstracts and conclusions, 30 studies remained. After further reading of the studies, 17 studies were excluded due to the lack of relevance to the research topic. The remaining 13 studies were selected for the literature review.

The results can be seen in Table 1.

Table 1. Search results before and after selection process

Database	**Initial**	**Inclusion & Exclusion**	**1st Manual Inspection**	**Final**
Google Scholar	10382	177	17	6
IEEE Xplore	5457	126	6	4
Scopus	3378	202	2	0
PRIMO	1132	54	5	3

2.5 Synthesis

This section focuses on key findings of 13 reviewed studies, frameworks and guidelines and outlines the main security concerns, proposed solutions and limitations to highlight the research gap.

2.5.1 Kubernetes Security Challenges and Common Vulnerabilities

Kubernetes is the most widely uses container orchestration platform in the world [6] that allows organization to deploy and manage their containerized applications at scale. The increasing adoption of Kubernetes across all industries has made it an attractive target for adversaries. As Kubernetes is a complex system with different layers and components, the attack surface is substantial and many organizations struggle to secure their Kubernetes clusters, as any misconfiguration can have significant consequences. According to the results of a survey conducted by Red Hat, nearly 90% of organizations have had at least 1 container or Kubernetes security incident in the last year [7].

As a result, a considerable amount of research has been conducted to explore this area in more detail, with most of the studies emphasizing on the importance of securing Kubernetes environments.

Overall, the most studies in this area share a common goal - highlighting the importance of a strong security controls in Kubernetes. However, the studies vary in their focus areas, as some concentrate of specific security aspects, such as access controls, container runtime security or networking, whereas others might focus on the security at a broader scale.

The following subsections outline the most commonly identified security risks and mitigation strategies from the reviewed literature.

2.5.2 Analysis of Existing Studies

Enhancing Communication Security in Kubernetes-Based Environments

This research [8] emphasizes the critical importance of securing communication in Kubernetes environments. It identifies the main security practices and tools that could be used to enhance the overall Kubernetes cluster security posture. However, the experimental part of the research is limited to implementing Kubernetes Network Policies and an mTLS communication between the applications within the cluster by utilizing Istio Service Mesh with *'AuthorizationPolicy'* [9]. Analysis why Istio Service Mesh was chosen, and not any other solution, is missing. Nevertheless, the research provides valuable insight into service-to-service encryption and overall communication security within the cluster. Additionally, the security practices discussed in the paper give a good overview of the security measures that should be taken into consideration.

Kubernetes Network Policy Engines and Enforcement

This research [7] focuses on the network policies and their enforcement as a solution to security issues in Kubernetes, emphasizing that the wide variety of engines and technologies can be overwhelming for the administrators. A comparative analysis of top 5 CNIs is conducted, identifying the use cases, challenges and limitations of each. Additionally, two Kubernetes policy engines *Open Policy Agent (OPA) with Gatekeeper* [10] and *Kyverno* [11] are thoroughly analyzed and compared. The author could've expanded the research by including additional policy engines, such as *Polaris* [12] and *jsPolicy* [13], to provide a more detailed overview of the available options. Overall, this analysis of CNIs and Kubernetes Policy Engines is valuable and can be used as a reference for organizations looking to implement network policies in their Kubernetes clusters.

Implementation of New Security Features in CMSWEB Kubernetes Cluster at CERN

This research [14] focuses on the implementation of new security features in the CMSWEB Kubernetes cluster at CERN. The primary focus again is on enforcing Kubernetes Network Policies, deploying *Open Policy Agent (OPA) with Gatekeeper* and integrating HashiCorp Vault for Kubernetes secrets management. Although, this paper provides valuable insights into securing Kubernetes clusters through network isolation, policy enforcement and secrets management, it could've been expanded. While OPA is used to enforce compliance at runtime, the paper could've discussed continuous auditing for configuration drift.

Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study

This study [15] investigates security misconfigurations in Kubernetes manifests - YAML-based configuration files used to define Kubernetes resources. The authors identify and categorize 11 types of security misconfigurations [15, p. 2.3] by utilizing *SLI-KUBE* [16], a custom static code analysis tool developed by the authors:

- lack of resource limits
- lack of or privileged *securityContext*
- enabling of *hostIPC*, *hostPID* and *hostNetwork*
- misuse of capabilities (such as *CAP_SYS_ADMIN* that allows privilege escalation [17])
- Docker socket mounting
- allowed privilege escalations for child processes
- hard-coded secrets
- insecure HTTP traffic both inside and outside the cluster

After the analysis, the authors have provided short and concise mitigation strategies for each misconfiguration. As an outcome, the authors have concluded that the practitioners agreed to fix 60 percent of the 10 misconfigurations after receiving the bug report [17, p. 9], which indicates that some misconfigurations are easily fixable. Although the study provides valuable insights into the most common misconfigurations, the authors could've explained the reasoning why they chose to create a custom static code analysis tool instead of using existing tools, such as *KubeLinter* [18], *Kube-score* [19] or *checkov* [20] (any shortcomings on existing tools?).

XI Commandments of Kubernetes Security

This study [21] aims to help people in securing their Kubernetes clusters through a systematization of Kubernetes security practices. A qualitative analysis was conducted on Internet artifacts and the authors were able to identify 11 Kubernetes security practices covering the key areas such as [21, p. 3]:

- Authentication and Authorization
- Kubernetes-specific Security policies
- Vulnerability scanning
- Logging
- Namespace isolation
- *etcd* restriction and encryption

- Continuous updates
- Resource quotas
- Enabling SSL/TLS support
- Workload separation
- Securing metadata APIs

While the research can be used as a foundational reference for Kubernetes security practices, it relies on non-academic sources only, such as blogs, online guides and community posts [21, p. 2]. This means that they might not have been thoroughly reviewed for accuracy and in my opinion, possibly weakens the credibility of this research. Additionally, the security practices discussed in the paper are general and do not provide clear steps or guidance for organizations how to implement any of the recommendations.

KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments

This paper [22] proposes analysis techniques to automatically detect "security smells" (recurring coding patterns that are indicative of security weakness and can lead to security breaches [23]) issues in Kubernetes. The authors proposed a new tool *KubeHound* [24] that complements already existing static and dynamic analysis tools with a specific focus on previously defined "security smells" for microservices [25] that are categorized into following groups:

- Insufficient Access Control
- Publicly Accessible Microservices
- Unnecessary privileges
- Own Crypto Code
- Non-encrypted data
- Hardcoded secrets
- Unsecure service-to-service communication
- Unauthenticated traffic
- Multiple modes of Authentication
- Centralized Authorization

The validation of the study is a controlled experiment that deploys a mock application with a task to deliberately inject security smells into Kubernetes [22, p. 6], and using the proposed tool to successfully detect these issues. This study provides valuable contribution for near real-time monitoring for Kubernetes security by proposing a new tool that combines the static and dynamic analysis techniques, especially in the context of microservices. However, all applications deployed in Kubernetes are not microservices, and focusing on

microservices only might limit the applicability of the proposed tool so that additional tools are needed to cover the possible gaps.

A Container Security Survey: Exploits, Attacks, and Defenses

Jarkas et al. conducted a study on container security by analyzing a dataset of over 200 vulnerabilities and proposing a new framework covering the security from Kernel to the Application Layer (dividing the container architecture into 5 layers) with a holistic security mapping to 4 hardware and 6 software mitigation strategies [26, p. 1]. The 5 container architecture layers of the proposed framework are:

- Orchestration Layer
- Application Layer
- Engine layers
- Host Layer
- Hardware Layer

In the context of E-ITS 'APP.4.4: Kubernetes' module, the focus is on the orchestration layer, engine layer, and host layer, as these discuss the vulnerabilities on container operations, access controls, container runtime configurations, and vulnerabilities on the host level, such as namespace isolation. Taking into consideration that the authors have included a comprehensive list of hardware and software mitigation strategies, this study that can be used as a baseline for organization to improve their security posture.

Uncovering Threats in Container Systems: A Study on Misconfigured Container Components in the Wild

Dongmin et al. conducted a study on security threats by misconfigured container components that are exposed to the Internet, with a focus on Docker and Kubernetes. A huge dataset of more than 1 million public IP addresses are compiled and then classified through distinct type of each component [27, p. 3].
Some of the most notable findings are that organizations often rely on default configurations, with 86% of the found API servers allowing anonymous authentication. Furthermore, the results showed that most commonly used Kubernetes versions at the time of the study are vulnerable to several CVEs that could be used to bypass security restrictions, the top 10 most exposed *etcd* versions that can be exploited to bypass authentication and gain value to Kubernetes secrets. [27, p. 4].
The study highlights the importance of addressing the vulnerabilities in Kubernetes components and the importance of securing the default configurations. Although the study gives a great overview of the most common vulnerabilities that organizations could use

as a starting point, it does not provide any mitigation strategies or solutions how to tackle these vulnerabilities, requiring additional effort from the organizations to find the solutions themselves.

Automatic Detection of Security Deficiencies and Refactoring Advises for Microservices

Ünver and Britto conducted a study with the objective of developing a set of tools to help developers with microservices security issues [28]. The authors listed the security issues based on the best practices in microservices and Kubernetes, and then compared a set of tools against these security issues. The tools chosen for the Pomegranate suite are:

- Docker Bench for Security
- Trivy
- Kube-hunter
- Kube-bench
- OWASP ZAP
- Nmap
- Terrascan

Unfortunately, the authors did not conclude a detailed selection process of the tools, after they had benchmarked them against the security issues. Initially, the authors stated that they had selected the tools based on their popularity in GitHub, however it is lacking a comprehensive explanation why certain tools were chosen over the others. For example, why *Terrascan* was chosen, even though *Kube-hunter, nmap* and *OWASP ZAP* meet the same requirements?
Furthermore, the objective of the study was to develop a fully automated test suite, however the *bash* script [29] that is provided to run the suite is relying on binaries, such as *Trivy* and *Terrascan* CLI, that are not included in the repository or installation script, or the path to the binary is incorrect.
Additionally, the script is using the *'latest'* tag for *OWASP ZAP*, which should be avoided, especially in the production systems.
As a suggestion, the authors could've used a containerized solution to run the suite, as it would eliminate the need for installing the tools separately and would make it easier to update the tools.
Nevertheless, the authors provided a good overview of the common security issues in Kubernetes and developed a suite of known open source tools to address these issues.

Security Audit of Kubernetes based Container Deployments: A Comprehensive Review

A paper written by Agrawal and Abhijeet identifies various high-level security challenges related to Kubernetes-based deployments and highlights a number of best practices when deploying Kubernetes clusters, such as [30, p. 3]:

- Preparation of worker nodes
- Securing the host system, network and inspecting the containers
- Securing the orchestrator (*etcd, kubernetes API server, kubelet, etc.*)

Although the study includes multiple chapters on different aspects of Kubernetes security, it still lacks some of the aspects. For example, the study focuses on network security, however it does not provide any details about network segmentation or Network Policies to control the traffic between the Pods.
The study could be expanded by including more details about the security aspects that are currently missing, such as Pod lifecycles, resource quotas, security of the service account, system backups etc.

Kubernetes Hardening Guide

The National Security Agency (NSA) and Cybersecurity and Infrastructure Security Agency (CISA) have developed a hardening guide for Kubernetes [31] that aims to help organizations secure their Kubernetes clusters. The guide is focusing on 4 main areas:

- Pod Security
- Network segmentation and hardening
- Authentication and authorization
- Logging and monitoring

Each area is expanded with a list of best practices and recommendations (e.g. using TLS, correctly set-up RBAC, limiting resource usage, setting up central logging system etc.), emphasizing that default configurations are often not secure and lack proper security controls to mitigate the risks. It includes some basic configuration examples that could be used as a starting point, however similar and more refined examples can be found in Kubernetes official documentation. I think that some examples could be either improved or replaced with more relevant ones. For example, it includes an example manifest of a Pod Security Policy (PSP), which is deprecated, but briefly mentions the Pod Security Admission (PSA), which would be a more relevant alternative.
It may lack some practical examples but overall, this guide is a good starting point for

organizations that want to improve their Kubernetes security.

CIS Kubernetes Benchmark

CIS Kubernetes Benchmark focuses on technical configuration settings to increase the Kubernetes security posture [32]. As this is a hands-on guide, it is intended for system administrators, security specialist, auditors, platform engineers, etc., who are deploying or managing Kubernetes clusters [32]. The benchmark is divided into 5 chapters:

- Control Plane components (e.g. API server, controller manager, scheduler)
- Securing the *etcd* database
- Control Plane configurations (e.g. authentication, authorization and logging)
- Data planes (e.g. kubelet and kube-proxy configurations)
- Policies (e.g. RBAC, Network Policies, Secrets Management etc.)

Each chapter contains a list of Kubernetes hardening recommendations with its description, rationale, potential impact and remediation steps. The benchmarks include either a Level 1 or Level 2 Profiles, where Level 1 is considered as a base recommendation to lower the attack surface while keeping machines usable, and Level 2 as "defense in depth" that are stricter but may impact the usability of the system [33].
There are various tools that use CIS Benchmark as a reference to check the system security state - as it is a widely recognized by cybersecurity experts and organizations. Out of all analyzed literature, this is the most detailed and practical guide that organizations can use to improve the cluster security.

Application Container Security Guide

This is a NIST Special Publication 800-190 [5] that provides guidelines for container security. Similarly to CIS Kubernetes Benchmark, it is intended for the same audience. It is divided into 5 major risk sections and even though it is mainly focused on Container Security, it includes sections on orchestration security and underlying host OS risks. Every section expands on the risks and vulnerabilities followed by 5 countermeasure sections for the same risks.
Although each countermeasure is explained, it is more of a high-level overview and focuses rather on the organizational processes than specific technical implementations. It is especially helpful in the design phase, however the thesis author believes that for implementation, it should be used together with other guides, such as CIS Kubernetes Benchmark or Kubernetes Hardening Guide.

2.5.3 Gaps in the existing literature

Despite the significant amount of research conducted on container orchestration security, particularly on Kubernetes, there are still several gaps in the existing literature. The following subsections outline the gaps that were identified during the literature review.

Narrow focus on specific security aspects

Many studies focus only on specific aspects of Kubernetes security. For example, some studies are focusing container security only, leaving Kubernetes security underexplored. For some studies, even if the title suggests the topic of Kubernetes security, further investigation shows that Kubernetes security is either not the main focus of the study or it is not discussed at all. This narrow perspective can have the effect of missing the bigger picture and make it difficult for organizations to understand the threats and implement a comprehensive security strategy.

Limited practical implementation

Most of the analyzed literature is theoretical and focuses on identifying and classifying Kubernetes threats and vulnerabilities. While these studies provide valuable insight into Kubernetes security posture, they lack the mitigation strategies. Generally, when some researches do propose the mitigation strategies, they still conclude with a discussion about these strategies without a practical validation (e.g. controlled experiments). As a result, organizations looking for clear guidelines are left in the dark with the theoretical knowledge and may struggle to implement the security measures in practice.

Lack of country-specific standards

The most significant gap in the literature is the lack of research that focuses on country-specific standards or frameworks. While there are some studies that reference security frameworks, benchmarks or best practices, such as NIST SP 800-190, OWASP Kubernetes Top 10 or CIS Benchmark [34, 35, 32], there is no literature that focuses on country-specific standards, especially on E-ITS.
This is specifically important for Estonian public sector organizations (but not necessarily limited to - as it is also recommended for private sector companies, and could possibly be used as a baseline security measure elsewhere) as they are required to comply with E-ITS.

Contribution to this Thesis

This leads us to:

- **[RQ4]:** What gaps exist between country-specific standards such as E-ITS 'APP.4.4: Kubernetes' module and the Kubernetes security in academic literature?

This review has shown that there is a significant gap in the existing literature regarding country-specific standards, such as E-ITS 'APP.4.4: Kubernetes' module, especially in the context of practical implementation. Existing research focuses on general Kubernetes security best practices, misconfigurations and vulnerabilities but not on country-specific implementations.

This Thesis aims to fill this gap by providing a comprehensive analysis of E-ITS 'APP.4.4: Kubernetes' module, and bridging the theory and practice by offering a practical guidance for organizations to implement the requirements.

3. Research Methods

This study employs a mixed-method research approach to develop a comprehensive practical guidance/reference for organizations that are seeking to comply with E-ITS APP.4.4: Kubernetes module requirements. The methodology includes three phases:

- Exploratory Phase: Initial understanding and definition of the problem space, including the Systematic Literature Review (SLR) findings and detailed document analysis.
- Constructive Phase: Designing and technically implementing a Kubernetes cluster compliant with E-ITS requirements.
- Validation Phase: Verifying and validating the implemented solution to ensure compliance with E-ITS APP.4.4: Kubernetes.

Each phase uses specific research methods to achieve the objectives described below.

3.1 Exploratory Phase

3.1.1 Systematic Literature Review (SLR)

The research begins with an SLR in Chapter 2, which identifies the current state of Kubernetes security, the main threats and challenges organizations face, and the research gaps. The SLR provides insights that guide the next steps in document analysis and design.

3.1.2 Document Analysis

A comprehensive document analysis is conducted focusing specifically on E-ITS APP.4.4: Kubernetes module. This method includes a thorough review of the E-ITS module documentation to identify and break down the requirements into specific technical and procedural tasks. Furthermore, the official Kubernetes documentation, industry-recognized hardening guides and security benchmarks, and other relevant forms of documentation are examined to provide context and enhance the understanding of the E-ITS requirements.

3.2 Constructive Phase

In this phase, the findings from exploratory stage are used to design and implement a technical solution that involves:

- A high-level architecture design based on the identified best practices and specific E-ITS requirements.
- A structured comparative analysis that evaluates the available tools and technologies for Kubernetes and determines the most suitable to be used in the implementation. This analysis includes a comparison of features, limitations, community support and its alignment with E-ITS security measures.
- Executing the designed architecture step by step in a controlled environment, dividing the tools into categories based on their functionality. Initially, the categories are configured incrementally, and each category is systematically validated before moving on to the next. Following this structured approach ensures that any potential issues are identified early and addressed on time, preventing delays in the overall implementation process.

3.3 Validation Phase

Validation ensures that the technical implementation has applied the E-ITS APP.4.4: Kubernetes security measures by checking the cluster compliance against each E-ITS measure with a combination of the following validation methods:

- **Experimental validation:** conducted through controlled tests in the Kubernetes cluster. This includes automated benchmarking tools with predefined security baselines complemented with manual testing scenarios (e.g., creating a resource and verifying whether the policy works as expected) to verify that security measures behave as intended.
- **Empirical validation:** manual reviews of the cluster configurations, Kubernetes manifests, as well as observations and metrics gathered from a live cluster to verify that implemented measures are actually effective and persistent.

By combining these validation methods, the research ensures that the implemented solution is practically effective and aligns with the E-ITS APP.4.4: Kubernetes security measures.

3.4 Data Collection

Data collection process uses various sources to ensure it is comprehensive and reliable. The following sources are used:

- Academic literature identified through the SLR
- Technical documentation such as Kubernetes official documentation, vendor manuals and technical specifications
- Official E-ITS documentation (APP.4.4: Kubernetes module, implementation and auditing guidelines and other relevant documents)
- Recognized Kubernetes security guides, benchmarks and blueprints
- Case studies, such as real-world experiences and public repositories (e.g., GitHub) detailing issues, resolutions and lessons learned regarding Kubernetes security

3.5 Data Analysis Techniques

Data analysis techniques are used to extract meaningful information from the collected data. The following two complementary techniques are used:

- **Thematic analysis** to identify, analyze and synthesize patterns from the SLR, regulatory documents, technical guidance and other relevant literature to align E-ITS requirements with industry best practices.
- **Comparative analysis** to select and evaluate the most suitable tools and technologies - each fulfilling a specific purpose, and complementary to each other - to ensure the alignment with E-ITS security measures.

By employing these phases and methods, this research aims to deliver a thoroughly validated and practically applicable implementation of E-ITS APP.4.4: Kubernetes compliant Kubernetes cluster.

4. Analysis of E-ITS APP.4.4: Kubernetes

Based on the findings in the SLR, this chapter helps to address the **RQ3** and **RQ4** by providing a detailed analysis of E-ITS 'APP.4.4: Kubernetes' module, identifying its specific requirements and examining their alignment with Kubernetes best practices. Additionally, it helps to understand the gaps between this module and current Kubernetes security practices in academic literature.

4.1 Background and Context

The Baseline Security Catalogue of EITS is based on the German BSI IT-Grundschutz. Additionally, EITS complies with the requirements of the ISO/IEC 27001 standard. The 'APP.4.4: Kubernetes' module is part of the E-ITS and focuses on the security threats and measures for Kubernetes clusters. The module is divided into three main sections: Description, Threats, and Measures.
The following paragraph breaks down and analyses the E-ITS 'APP.4.4: Kubernetes' module to understand and explain its requirements in detail. This is needed to understand how it compares with the current best practices, and what kind of tools and technologies are needed to meet these requirements in the technical implementation paragraph. The 'APP.4.4: Kubernetes' module is divided into 3 main paragraphs:

- Description
- Threats
- Measures

The content and purpose of each of these 3 main paragraphs will be expanded below.

4.2 Description

This paragraph is mainly introductory and outlines the Purpose, Responsibility and Limitations of E-ITS 'APP.4.4: Kubernetes' module.

4.2.1 Purpose

The purpose of this module is to present various measures for the containerization automation and management, with the purpose of protecting the data within Kubernetes cluster.

Furthermore, the paragraph describes the basics of Kubernetes, highlighting some of the components and that Kubernetes is a *de facto* container orchestration solution in both public and private clouds [2, Ch.1, Sec. 1.1].

4.2.2 Responsibility

It is emphasized that implementing the security measures for Kubernetes cluster is only the responsibility of IT department and no additional personnel [2, Ch.1, Sec. 1.2].

4.2.3 Limitations

This paragraph outlines the limitations for 'APP.4.4: Kubernetes' module, emphasizing that it covers the measures related to deployment, implementation and administration of Kubernetes cluster. That includes the hardware components, such as Container Network Interface (CNI) and Container Storage Interface (CSI) [2, Ch.1, Sec. 1.3]. Additionally, as already mentioned in the Introduction chapter, this module is always discussed together with the measures from module 'SYS.1.6: Containerisation' [2, Ch.1, Sec. 1.3], and this implies that the container images used inside the Kubernetes cluster shall meet that module's requirements. The paragraph states that when implementing the measures, the choice of Container Runtime is not important, highlighting some examples, such as Docker, containerd, runC and Windows Container. However, since Kubernetes 1.24, Dockershim is deprecated [36], it might be worth noting that using Docker as a Container Runtime works only when using cri-dockerd as a shim for Docker Engine.

4.3 Threats

This chapter focuses on the defined threats in the E-ITS 'APP.4.4: Kubernetes' module that can be encountered in Kubernetes cluster. These threats are divided into five different categories.

4.3.1 Control plane authentication and authorization errors

The control plane which includes the applications that orchestrate the operation of Kubernetes nodes, runtimes and clusters, needs privileged access (administrator permissions) to work, and this access is usually provided through network ports or Unix Socket [2, Ch.2, Sec. 2.1]. Misconfigurations on authentication and authorization of control plane can have widespread impacts, potentially compromising the entire orchestration infrastructure.

4.3.2 Loss of confidentiality of the token of a pod

Pods use tokens for communication with the control plane, and the attack on a Pod could result with the token ending up in the possession of the attacker [2, Ch.2, Sec. 2.2]. For example, by default, kubelet automatically mounts the ServiceAccount's API credentials (unless explicitly configured otherwise) [37] to a Pod, which could then be used by attacker to communicate with the Control Plane, and in case of elevated privileges, make unauthorized changes in Control Plane settings or orchestration.

4.3.3 Conflict of resources caused by a pod

A single Pod can overload the node it is running on disrupt the entire Kubernetes orchestration. As a result, this may compromise the availability of all other Pods on that node or prevent the node from functioning as expected [2, Ch.2 Sec. 2.3]. For example, if there are no resource limits set for a Pod, and it starts consuming more memory than available on the node, the OOM Killer can terminate kubelet process on the node, which causes the node being unable to manage the Pods running on it.

4.3.4 Unauthorised changes in the Kubernetes cluster

For environments where automation processes (such as CI/CD tools) have privileged rights to interact with Kubernetes clusters, there is a risk of unauthorized changes being made in the clusters - whether by the automation or the users controlling them [2, Ch.2, Sec. 2.4]. For example, due to a configuration mistake made by a user, a CI/CD pipeline might deploy this change to a production while it was actually intended for a QA cluster.

4.3.5 Unauthorised access to a pod

By default, all Pods have the capability to communicate with each other, with the nodes in the cluster or with the other external systems (unless explicitly denied otherwise) [2, Ch.2, Sec. 2.5]. For example, a Pod exploited by an attacker can be used to access other Pods, or nodes (including the control plane) in the cluster. This can lead to serious security incidents and other types of attacks.

4.4 Measures

The measures in the E-ITS 'APP.4.4: Kubernetes' module are divided into three measure categories:

- Key measures
- Standard measures
- Advanced measures

Additionally, the measures can also be viewed from the perspective of the Lifecycle of the Kubernetes cluster, which is divided into 3 lifecycles, and additional advanced measures:

- Planning
- Implementation
- Operation
- Additional advanced measures

This Lifecycle categorization logic is done such that Key measures, Standard measures and Advanced measures are all spread across these four lifecycles.
However, some measures are related to the overall processes of the organization and cannot be implemented directly on Kubernetes. For these measures, the industry best practices and standards are used as a direction for organizations to follow.
We'll focus on the measures in the same order they are presented in the E-ITS 'APP.4.4: Kubernetes' module, starting from Key measures, followed by Standard measures and finally Advanced measures. Although advanced measures are applied only when "high" or "very high" protection requirements are identified, we'll analyze them as well, as they are still part of the E-ITS 'APP.4.4: Kubernetes' module.

4.4.1 Base Measures

Base Measures chapter focuses on the measures that need organizational processes and policies to be implemented.

APP.4.4.M1 Designing the partition of applications

This paragraph focuses on how to design the application separation in Kubernetes. Before thinking about deploying any applications, there should be a clear architectural understanding within the organization how the Kubernetes workloads are separated from one another.

- **Development**, **Test**, and **Production** environments shall be separated from one another. As there are no direct recommendations from E-ITS or other frameworks focusing on Kubernetes entirely on Kubernetes security, organizations should follow the best practices and additional available frameworks in the industry. For

example, chapter 'Annex A 8.31 - Separation of Development, Test and Production Environments' of ISO 27002:2022 emphasizes on the importance of separating the development, testing and production environments to prevent various security incidents [38], and could be used as a reference for organizations to follow.

- When designing the architecture for Kubernetes namespaces, meta tags (such as labels and annotations), networks and for the clusters themselves, it is important to base the decisions on the requirements and potential risks of the applications. In the context of decision making on application security requirements and risks, chapter 'Control 8.26' - Application Security Requirements' in ISO 27002:2022 framework [39] can be used as a good reference.
- The decision how many nodes and node groups should be created and how to categorize them, shall be based on the protection requirements of the applications and the potential risks.
- It shall be decided how different resources are separated from each other and how many resources will be allocated, based on the design of the architecture. For example, the CPU, memory and storage resources can be allocated and limited per container, Pods or namespace. It is a common practice and a recommendation to use Resource Quotas and Limit Ranges [31, pp. 50–51].
- The network segmentation is a central concept of Kubernetes and 4 distinct networking problems shall be taken into consideration when designing the network architecture of Kubernetes [31, 40, p. 14]:
  - Pod-to-Pod communication
  - Pod-to-Service communication
  - Service-to-Service communication
  - External-to-internal communication
- Additionally, the applications shall be partitioned based on the general network architecture of the organization and the overall zoning principles of the network [2, Ch.3, Sec. 3.2]. NIST 800-215 [41] and OWASP Network Segmentation [42] are great references for organizations to follow when designing the network architecture.
- Namespaces are a way to partition cluster resources and every application shall run in a designated Kubernetes namespace. Whether there's a namespace for each application, group of applications or any other form of grouping, it is important to have a clear understanding of the namespace structure and how the applications are separated from one another. It is also recommended in CIS Kubernetes Benchmark [32, p. 275], NSA/CISA Kubernetes Hardening Guide [31, p. 14], and official Kubernetes documentation [43] to use namespaces to isolate workloads from one another.
- Each cluster contains only applications with similar security requirements and attack vectors. For example, from both privacy and security perspective, you cannot store

European customer data in a cluster that is located outside the EU. Clusters that collect, process, or transmit individually identifiable electronic protected health information have different compliance and security requirements than clusters that do not process any sensitive data, and this is something that needs to be taken into consideration when designing the application partitioning [44]. This is something that has to be taken into consideration in design phase.

APP.4.4.M2 Automation of the development of applications with the help of CI/CD

When designing the CI/CD pipelines, it is important to understand the GitOps framework and best practices to ensure the standard workflow for applications, increased security, reliability and consistencty across clusters [45]. It is important to follow the least privilege principle - define only the minimum set of permissions needed for tools to work.
Additionally, there must be a process that defines how to secure the data that is processed by the CI/CD tools - how to ensure confidentiality, integrity and availability of that data. A special publication from NIST, SP 800-204D focuses on the security of CI/CD pipelines and provides a great starting point for organizations to follow [46, Ch. 4-5].
CI/CD pipelines shall be used throughout the entire application lifecycle - development, testing, deployment, monitoring and updating. A great introduction to CI/CD pipelines is provided by Codefresh, and this is something that organizations can use as a starting point [47].

APP.4.4.M3 Planning the Kubernetes identity and rights management

API requests are tied to either a user (normal user or service account) or are treated as anonymous requests, and authentication and authorization happens regardless of whether the requests are made through a client, web interface or API. By default, Kubernetes allows anonymous requests [48], however it is worth taking into account whether anonymous discovery is an acceptable risk for the organization or not. If the Kubernetes API is using RBAC authorization, it is considered reasonable to allow anonymous access to the API server for health checks and discovery purposes [31, 32, p. 59], as otherwise the the health probes would not be able to access the API server, and kube-apiserver static Pod would not start. One workaround would be to use long-lived tokens and modify the health probes to use these tokens, however this is not a recommended way in Kubernetes documentation [49], or use an mTLS authentication - however, this would require additional certificate management and distribution that might not be feasible in all environments.
Similarly to official recommendations from Kubernetes documentation, the principle of the least privileges shall be followed, and only permissions explicitly required for their operation should be used [50]. Although Kubernetes documentation does not describe how the identities and access permissions should be managed, E-ITS explicitly requires,

similarly to a NIST CSF recommendation, a separation of duties and access control management only by authorised personnel [51, pp. 19–20].
Kubernetes RBAC best practices state that Persistent Volumes should only be created by *trusted administrator* [50], however E-ITS goes a step further and includes the modification constraint as well. This is a bit contradicting, as PVC (Persistent Volume Claim) is usually a part of the application lifecycle, created and managed together with other Kubernetes resources for that application - assuming that a proper process is defined how the changes are reviewed and approved, and that Storage Class is configured to Retain the data after deletion, and the configuration of Storage Class and underlying storage is managed by designated personnel.

APP.4.4.M4 Partition of pods

This chapter defines the requirement to use Linux namespaces and cgroups that are kernel features used by Container Runtimes for isolation and resource limiting. This is a default requirement by Kubernetes that kubelet and the underlying container runtime (e.g. containerd, CRI-O) need to interface with cgroups in order to enforce resource management for Pods [52]. Namespace isolation (not to confuse with Kubernetes namespaces) limits which resources a container may interact with and ensures that applications and processes within the container only see the resources allocated to that container.
Expanding more on the fundamental isolation (provided by Linux namespaces) requirements by E-ITS - Kubernetes enables most of these by default (PID, IPC, network etc.), and using *hostPID*, *hostIPC* or *hostNetwork* parameters would break this isolation and shall be avoided. However, the current version of E-ITS (2023) does not require the use of User Namespaces which is another Linux feature that isolates the UIDs and GIDs from the host [53].

APP.4.4.M5 Backup of cluster information

Although Kubernetes has a way to back up all objects that are stored in etcd, there is no automatic backup solution for this and Kubernetes documentation together with best practices shall be used to back up the etcd cluster [54]. As E-ITS doesn't specify the backup means and methods (apart from the list of resources that shall be periodically backed up), it is recommended to look at the best practices from Kubernetes documentation or other frameworks and rely on the overall backup policy in the organization, depending on the RTO and RPO.
It is recommended to use the backup capabilities of the underlying storage provider or use the Kubernetes VolumeSnapshot API (not part of the core API toolset) as a standardized way for CSI drivers to create and manage snapshots of Persistent Volumes [55]. Backup solutions like Velero or Veeam Kasten have integrations with various CSI drivers and can

be used to utilize the VolumeSnapshot API for backup and restore [56, 57].

Additionally, it is recommended to separately backup OS-level files on control plane and worker nodes, such as kubelet, kubeadm, container runtime and CNI configuration files by using either configuration management tools or volume/OS-level backup solution. Overall, organizations shall have a backup process in place, and E-ITS, NIST 800-34 provides a great overview and starting point how to plan backup and recovery strategies [58, Ch. 3, Sec. 3.4.1], [59, p. 65].

4.4.2 Standard Measures

APP.4.4.M6 Secure resetting of the pods

This measure aligns with Kubernetes defaults and overall best practices how to handle application startup logic. It means that in case there are any pre-requirements (e.g. additional configuration, file transmitting etc.) that need to be done before the container can start, it shall be done by using initContainers, which are specialized containers that always run to completion before the app containers in a Pod [60] ensuring they start in a clean state.

APP.4.4.M7 Partition of Kubernetes networks

Kubernetes clusters require non-overlapping IP addresses for Pods, Services and Nodes assigned by the following components [61]:

- CNI - configured to assign IP addresses to Pods
- kube-apiserver - configured to assign IP addresses to Services
- kubelet - configured to assign IP addresses to Nodes

However, it doesn't define how the network partitioning should be done - whether to use different networks for each of these components (by utilizing the Overlay network) or not. To increase the security and reduce the attack surface of the cluster, E-ITS requires network segmentation, as this is a common best practice and is recommended by different frameworks.

Control plane, data plane (worker node), cluster (pod network) and administrative networks shall be separated from each other [31, p. 19]. In the cluster network (Pod network), only the ports required for the application functionality shall be open. The communication between Kubernetes namespaces shall be blocked by default, and only the required network connections whitelisted. This can be achieved by utilizing Kubernetes Network Policies with an implicit deny rule to deny all traffics to and from Pods, and then explicitly allowing

the traffic on required ports and namespaces [62].
The ports required to manage the control plane and data planes shall be accessible only from the designated administrative network or designated pods (Kubernetes operators that manage various resources), preferably by using a dedicated computing resource. This is recommended by CIS Control 12 [63, Ch. 12.8] and other frameworks focusing on network best practices.
The CNIs and Network Policies shall be managed only be designated personnel or Kubernetes operators. Either way, this assumes a proper review and approval process in the organization.

APP.4.4.M8 Security of the Kubernetes configuration files

This section sets the requirements for Kubernetes configurations (including all extensions and applications) management throughout their lifecycle. Although Kubernetes does not have a built-in versioning system for resources, it is a common recommendation in Kubernetes documentation [64] and an E-ITS requirement to have versioning for the configuration files. This is usually done by using a version control system (e.g. Git). The most common and recommended way for doing this is to use GitOps principles, which is a set of best practices for managing infrastructure and application configuration [65].
By following the same GitOps principle, the configuration files are managed by special tools (e.g. Argo CD, Flux CD, Octopus Deploy) to deploy the changes in the cluster. As these tools are interacting with Kubernetes API, they need to have the required permissions to do so. It is a requirement in E-ITS and an overall Kubernetes best practice to use the least-privilege principle when granting RBAC permissions [50] these tools, particularly when handling the read-write permissions of the control pane configuration.

APP.4.4.M9 Security of the Kubernetes service accounts

By default, when a Pod is created, Kubernetes automatically assigns the default service account of that namespace unless specified otherwise [66], but this is not recommended from security perspective. Similarly to CIS Benchmark recommendation, E-ITS requires that no Pod should use the default service account and the permissions for 'default' service account shall be revoked entirely together with *'automountServiceAccountToken: false'* to avoid the token being mounted to a Pod by accident [32, Ch. 5].
Furthermore, every application Pod (including the different automation tools) shall have its own service account created, and the permissions shall use the least-privilege principle - only the permissions that are required for the application functionality shall be granted. Additionally, privileged permissions shall only be used for service account managing the control plane or only for Pods where it is otherwise inevitable.
There is a contradicting statement about Pods that do not have a service account - this is

not a possible scenario in Kubernetes, as every Pod will have a service account assigned to it. It is possible, and recommended when the application doesn't need to access the Kubernetes API, to not mount the service account token, but this is not considered as a Pod without a service account.

APP.4.4.M10 Security of the automation process

This section addresses the security between the automation tools (e.g. CI/CD and pipelines) and the Kubernetes cluster. Similarly to the previous measures, it is required to rely on the best practices [50] and use the least-privilege principle when granting permissions to these automation tools. Additionally, E-ITS focuses on the human element of the automation process and requires the separation of duties - only authorized personnel shall be able to do any changes on the Pod configuration or trigger the automation tools that would deploy the changes. The least-privilege principle and separation duties is a common recommendation in the industry and can be found in various frameworks, such as NIST CSF [51, pp. 19–20] and 800-53 [67, pp. 36–38], NSA/CISA Kubernetes Hardening Guide [31, pp. 23–25] and CIS Kubernetes Benchmark [32, p. 220].

APP.4.4.M11 Monitoring the use of containers

This section focuses on the utilization of Kubernetes health probes - a built-in feature that performs a periodical diagnostic on a container [68]. Although a built-in feature, it is not configured by default and needs to be explicitly defined in the resource manifest by the user. However, it is a common recommendation to use these probes to ensure the liveness and readiness of the application.
E-ITS requires that for each container in a Pod, the health probes (*livenessProbe, readinessProbe, startupProbe*) are defined and configured according to the application requirements and expected behavior. Even though it might seem trivial, it is a common mistake to either not define the probes at all, or use the values that are not suitable for the application, setting either too optimistic or pessimistic values for the probes. This can lead to unnecessary restarts of the application or the application being marked as *'ready'* when it actually isn't. Even though *lifecycle* hooks, particularly the *preStop* hook to ensure that the application is stopped gracefully (e.g. the container needs to perform additional cleanup tasks before termination), are not explicitly required by E-ITS, it can be essential in some cases to ensure the graceful termination of the application - which itself is a requirement. Unfortunately, the topic of health probes and lifecycle hooks are not covered in CIS Kubernetes Benchmark, NSA/CISA Kubernetes Hardening guide or NIST framework, so the best way for organizations to learn about the health probes is to use the official Kubernetes documentation [69, 68, 70, 71].

APP.4.4.M12 Security of infrastructure applications

This measure mandates specific security considerations when 'images' (e.g. system images, disk snapshots) are used for critical infrastructure tasks such as automation, hard drive management or configuration file backups. The focus is on securing these images and the tools that manage them (e.g. CSI Drivers for Volume Snapshots, backup solutions like Velero and Veeam)

- Data exchange and communication:

  Data Exchange - images and its associated sensitive data must be encrypted in-transit. This applies when the images are being copied and moved to external storage (e.g., for backups) Images and associated sensitive data must be encrypted during transit. This applies when images are being copied, moved to external storage (e.g., for backups), or deployed across the network.

  Data Communication - all network communication channels involved in managing or transferring these images must be encrypted. In Kubernetes, API communication is typically encrypted, but it needs to be ensured that pod-to-pod communication is also secured, potentially using a service mesh (e.g. Istio, Linkerd) to enforce mTLS. Communication with any external services handling these images (e.g. image registries, backup storage) must also be encrypted.
- Logging of changes:
  A comprehensive logging of all changes related to these images is required. For example, Kubernetes has the audit logging capabilities, however it is not enabled by default and needs explicit configuration in audit policy manifests [72]. A central logging solution is essential to collect audit and application logs from all applications in the cluster. It should be ensured that Kubernetes logging best practices are followed and containers are logging to stdout/stderr streams, so it can be correctly handled by the container runtime. This ensures that logs are collected and aggregated by the logging agent and sent to a central location for storage [73].

4.4.3 Advanced Measures

This chapter now describes the advanced measures that are required by E-ITS. These measures are more specific and technical and are not focused as much on the organizational process as Key measures and Standard measures, rather they look it from the CIA (Confidentiality, Integrity, Availability) Triad perspective.

APP.4.4.M13 Automatic configuration audits (C-I-A)

This section focuses on proactive approach on maintaining the Kubernetes security through automated checks and continuous monitoring. It requires to continuously compare the live configuration (node settings, Kubernetes components, Pods and other Kubernetes resources) against the desired state. This is something that is not provided by Kubernetes by default and requires organizations to use external tools. There are various tools providing this functionality.
There are various tools that could be used for auditing purposes and policy enforcement. For example, kube-bench, Kubescape can be used to check misconfigurations against various frameworks (e.g. NSA/CISA, CIS Benchmark, MITRE ATT&CK) [74, 75], Kyverno, OPA with Gatekeeper to enforce policies within Kubernetes cluster [11, 10], Argo CD or Flux CD to ensure the desired application state (the application in this context may vary - from Kubernetes resources to entire clusters or external services) with auto-remediation [76, 77]. These are just a few examples of the tools that can be used to fulfill this E-ITS requirement. However, the most important aspect of this requirement is that there should be an organizational process in place that defines the ruleset how to choose and use these tools.

APP.4.4.M14 Use of specialised nodes (C-I-A)

This section complements the APP.4.4.M1 key measure regarding the partitioning of the Pods and focuses entirely on the segmentation of Kubernetes nodes. The main requirement is to use specialized nodes depending on the tasks they are performing. This is not a requirement or recommendation in Kubernetes documentation, however it is a common practice to use specialized nodes in production environments to avoid the *noisy neighbour* issues. The E-ITS requirement specifically emphasizes the following segmentation logic:

- A general specialisation - the nodes shall be grouped based on the needs of the applications and the task they are performing. For example, memory-intensive workloads should be run on nodes with more memory, CPU-intensive workloads on the nodes with a higher CPU core count, GPU-intensive workloads on the nodes with GPU support and so on. After defining the node groups it is important to use the built-in Kubernetes scheduling features to assign the Pods to the correct nodes and prevent them from being scheduled on the wrong nodes. There are various ways to do this, such as node selector, affinity/anti-affinity rules and taints/tolerations. Kubernetes documentation provides a great overview how to use these features [78].
- Nodes for network traffic - specific nodes are designated only for handling inbound and outbound traffic between the cluster and external networks. Having dedicated nodes for this purpose simplifies the network security management, enhances mon-

itoring (easier to pinpoint on issues) and isolates the components that are facing external networks to reduce improve the security posture of the cluster.
- Control plane components - It is an overall best practice that Kubernetes API server, etcd, controller-manager, scheduler shall all run on dedicated nodes and nodes shall not be used for any "regular" application Pods at all.
- Storage nodes - Applications with a primary function of performing backups (e.g. Velero, Veeam operators, Pods doing the actual backups) shall be grouped together and run on dedicated nodes. As these Pods usually have higher I/O demands, the nodes should be optimized for this purpose (when possible). Often times, these nodes are used for running other stateful applications (e.g. databases, message queues) as well. However, basing on E-ITS requirements it would be even better to have separate nodes for these applications as they have different requirements and workloads.

Although NSA/CISA Kubernetes Hardening Guidance briefly touches the topic about the importance of node segmentation [31, pp. 19–20], it does not expand on this topic, and other frameworks/guides do not mention this at all, making it unique to E-ITS in this aspect.

APP.4.4.M15 Partition of applications at the levels of nodes and clusters (C-I-A)

This section requires a strong isolation for highly sensitive/secure applications running on Kubernetes in two possible ways:

- Deploying these applications into a completely separate Kubernetes cluster for maximum isolation. From security perspective, this is the best option as it provides the highest level of isolation, however it causes a lot of operational overhead, requires additional resources for the cluster, which will results with higher costs.
- Isolating these resources to dedicated nodes that are completely isolated from other nodes in the cluster based on correct application of taints, tolerations, node selectors, affinity rules and network policies. This is a more cost-effective solution with less operational overhead, however it requires a lot of planning to ensure that these nodes are isolated from the rest of the cluster.

Overall, the choice between these option depend on the requirements of the organization (e.g. compliance, security, privacy and operational). Although not explicitly recommended by spfecific Kubernetes hardening guides/frameworks, it is best to rely on Kubernetes documentation about multi-tenancy [43] and the general cybersecurity frameworks describing the platform security and data flow controls [51, 67].

APP.4.4.M16 Use of Kubernetes operators (C-I-A)

This section focuses on the use of Kubernetes operators to automate the management of critical applications and control plane components. Although Kubernetes provides the framework for Operators [79], their use is optional but highly recommended for complex and critical workloads.
The operator in this context is a software extension that uses custom resources to manage applications and their components - such as cert-manager for managing TLS certificates, Velero operator for managing backup or Prometheus operator for logging and metric collection. Using these tools can significantly reduce the operational overhead, reduce the risk of human error and help to improve the overall security posture of the cluster simply by automating the tasks that would otherwise be done manually.
Choosing operators can be a difficult task as there are hundreds of different operators available, and it is important to make the decision based on the needs. It is important to secure them properly - use the least-privilege approach when granting RBAC permissions (based on the requirements in APP.4.4.M9) and make sure that they are not running with elevated permissions.
OperatorHub is a common place to find different Kubernetes operators together with their documentation and installation instructions [80] and can be used as a good starting point to find the right operators.
E-ITS is kind of unique in this section as the other guidances and frameworks do not explicitly mention the recommendation to use operators at all.

APP.4.4.M17 Certification of nodes (C-I-A)

This section focuses on the certification aspect of Kubernetes nodes, and most of these requirements are already facilitated by default when deploying the cluster with kubeadm (as all certificates needed for the cluster are generated automatically) [81, 82].
However, it is still possible that some installation methods allow insecure communication between the nodes and the control plane, or custom certificates (including the ones signed by external CA). It is important that certificates are rotated regularly - usually this is done automatically when upgrading the cluster or using the *RotateKubeletServerCertificate* feature on kubelet. However, it is important to note that even though kubeadm renews all certificates during control plane upgrade, it cannot manage certificates signed by external CA [82], and 3rd party tools are needed to approve the CSR.
Another thing to note is that the default kubelet serving certificate is a self-signed certificate and connections from external services like metrics-server to kubelet cannot be secured with TLS [82], and this automatically does not meet the E-ITS requirement and shall be mitigated by using the *'serverTLSBootstrap: true'* configuration when initializing the cluster with kubeadm.

Additionally, TPM-based attestation is preferred whenever possible - instead of the usual CSR process, the node is using the Endorsment Key of the TPM module to create the CSR, which is then signed by the TPM CA on the control plane. However, this is not always possible as virtualization platforms may not offer virtual TPM modules depending on the virtual host. TPM is also mentioned in the NIST 800-190 under orchestrator countermeasures but rather in a general context that software-based security measures can take the organization just so far [5, p. 28], which makes E-ITS unique in this aspect as well.

APP.4.4.M18 Microsegmentation (C-I)

This section sets the requirements on microsegmentation of the Kubernetes cluster with a special focus on the CNI-based segmentation. Some requirements are already covered in section APP.4.4.M7 regarding the utilization of network policies. However, this section expands further and complements it with additional requirements that cannot be achieved with the Kubernetes Network Policies only - such as data filtering based on service accounts or certificate-based authentication.
This can be achieved by using the expanded capabilities of CNIs or Service Meshes, or the combination of both. For example, Calico and Cilium are capable of enforcing network policies based on service account names and other metadata [83, 84].
To comply with the E-ITS requirements in this section, Service Meshes shall be used to expand even further by moving from L3 and L4 to L7 filtering. By utilizing CNI network policies, it is possible to achieve the network policy requirements defined in section APP.4.4.M7 in a more efficient way and with less operational overhead.
Although different frameworks and guidances mention the importance of overall network segmentation, only E-ITS expands on this further and sets the requirements for microsegmentation.

APP.4.4.M19 Guaranteeing the high availability of Kubernetes (A)

This section outlines the requirements for ensuring the availability and disaster recovery of Kubernetes clusters and applications. In case of an outage or failure in one location, the cluster and the applications should continue operating with a minimum downtime or be able to recreated in another location. Meaning that all the configuration files, application images, networks and other resources shall be prepared as much as possible for smooth and quick recovery.
This comes down to the overall recovery requirements of the organization - when designing the system, what is the acceptable RPO (Recovery Point Objective) and RTO (Recovery Time Objective) for the applications running in the cluster.
Expanding on this requirement, it is important to ensure that underlying hardware of the

clusters, Pods, and applications have been distributed across different locations to ensure high-availability. Although the default kubeadm setup creates a single control plane node, it is recommended to use at least 3 control plane nodes for redundancy (either with stacked or external etcd topology) [85, 86].
The data plane (worker) nodes should also be distributed across different locations, and Kuberentes built-in scheduling features such as *podAntiAffinity* and *topologySpreadConstraints* should be used to ensure that applications are not running on the same node.
None of this is a requirement in Kubernetes documentation or in other Kubernetes specific hardening guides - most likely because it is usually considered as a part of reliability and not security. However, it is a common practice in the industry to have high-availability and disaster recovery requirements in place.

APP.4.4.M20 Encryption of the control plane storage space (C)

This section sets the requirements for implementing both OS-level encryption and Kubernetes at-rest encryption for the control plane, with a special focus on the etcd database. This is one of a few E-ITS requirements that is discussed in most Kubernetes frameworks, hardening guides and official documentation because it is unencrypted and contains all cluster information together with secrets in a plaintext format and therefore can be a target for attackers.
For *etcd* encryption, it is best to use the Kubernetes documentation that explains the process in detail with emphasis of the common mistakes that can happen during the configuration [87] or other guides with specific examples [31, pp. 52–53],[32, pp. 111–113]. To encrypt the volumes on OS-level, it is recommended to use either the provider's volume encryption solution or if that is not available, to use software that allows to encrypt the whole disk.

APP.4.4.M21 Periodic restarting of the pods (C-I-A)

This requirement is perhaps the most controversial in the E-ITS Kubernetes section. Although initially Kubernetes was designed to be used for stateless applications, it has now become a common practice to run stateful applications in Kubernetes, including databases and message queues, especially with the increasing popularity of using operators to manage these applications seamlessly.
Although it is critical to ensure the availability of Kubernetes applications during the restart process, and design the applications in a way they can be shut down gracefully, it is not a standard practice to manually restart the Pods periodically. This is either done by the orchestrator based on actual need or shall be done with the help of operators that manage the applications by utilizing Pod Management Policies, Pod Disruption Budgets and other Kubernetes features.

4.5 Conclusion

This chapter provided an overview and analysis on 'E-ITS APP.4.4: Kubernetes' module. Each section was expanded with additional information and context to provide a better understanding of the requirements, how do they compare with the overall best practices and Kubernetes frameworks/guidances and what should be taken into consideration when implementing them.

Although it is usual for standards to set the requirements and leave the implementation up to the organizations, I believe that some requirements should provide more context and explanation for the need, as the interpretation of them can vary from organization to organization, and eventually cause potential issues during audits. Although the author of this thesis has experience with Kubernetes and read the requirements multiple times in Estonian and English, it was still difficult to understand either the requirement itself or the reasoning behind it. It feels like some context is lost in translation from German to Estonian (BSI to E-ITS) and then to English.

For example, the requirement to use CNI policies (a custom resource) for network segmentation, although you'd only need a CNI that supports NetworkPolicies - of course, these custom resources often expand the Kubernetes Network Policies and allow for more defined outcomes. Additionally, the requirement that some Pods should run without a service account, is not possible, as every Pod will have a service account assigned to it and disabling the auto mounting of its token will contradict with the second part of the requirement - using a token to authenticate to the Kubernetes API.

To conclude, the E-ITS Kubernetes module is a great addition to the Kubernetes security landscape and compared with other frameworks, it expands more on the organizational processes and advanced security measures that are not completely covered elsewhere. However, for the technical implementation, organizations still need to rely on the Kubernetes documentation and practical hardening guides (e.g. CIS Benchmark, NSA/CISA Hardening Guide) as standards provide you the objectives for end goal, but not the resources how to achieve it.

5. Technical Implementation

In this technical implementation section, the focus is to deploy a self-managed Kubernetes cluster in E-ITS 'APP.4.4: Kubernetes' compliant way. It is based on the requirements outlined in the previous analysis chapter to find the most suitable tools and technologies to follow the best practices and achieve the desired outcome. Additionally, this chapter provides a significant part of the **RQ2** answer by identifying the tools and demonstrating the implementation process that can be used for E-ITS adoption, and shows a concrete path for addressing the **RQ1**.
Although the advanced measures are applied only when a "high" or "very high" protection requirement is detected, this technical implementation will treat as these protection requirements are detected, and therefore the advanced measures will be implemented. This is done to ensure that in case organizations have a higher protection requirement, this implementation is still applicable and relevant.

5.1 Resource Platform

For the resource deployment platform, Microsoft Azure was selected due to multiple reasons. In addition to providing all the resources that are needed to build a self-managed Kubernetes cluster, the author of this thesis has access to Azure credits which allows the creation of this environment without excessive costs. This is important because the author does not have access to dedicated physical servers and alternatively, using other cloud service providers without credit could become exceedingly expensive and risk the technical implementation of the thesis.

5.2 High-Level Architecture

The high-level architecture designed in this study will be built taking into consideration the best practices of security, scalability and high-availability to align directly with E-ITS requirements. As described in the previous section, the cluster infrastructure will be built in Microsoft Azure to leverage its capabilities, such as providing isolated and segmented network and multiple availability zones to ensure resilience and fault tolerance. This architecture consists of distinct node group, each dedicated to specific functionalities:

- **Control Plane Nodes** - responsible for managing the Kubernetes cluster state. These nodes will be deployed across 3 availability zones to ensure high availability and

fault tolerance.

- **Ingress Nodes** - dedicated nodes to handle external inbound traffic, isolating ingress workloads from other workloads for security and performance reasons (e.g., noisy neighbor effect).
- **Monitoring Nodes** - specific nodes to host monitoring and logging tools that are often resource-intensive and would otherwise affect the performance of other workloads.
- **Storage Nodes** - dedicated for storage-related workloads, such as cluster backups.
- **General Nodes** - general-purpose nodes that are used to run other workloads that do not require specific resources or configuration.
- **Isolated Nodes** - dedicated nodes for running sensitive workloads (high protection requirement) that require additional security measures on network level.

Network design includes a dedicated subnet for control plane nodes, and a separate subnet for data plane nodes to comply with the E-ITS requirements. Additional subnet is created for the management network which is used to access the control plane and data plane nodes.
The following sections will describe the resource deployment tools, cluster deployment planning, expanded node group choices and network design.

5.3 Resource Deployment Tools

Although, there are various tools and ways to deploy Azure resources, such as Terraform, Bicep, AKS Engine, Azure CLI, etc., the chosen tool for this implementation is Cluster API (CAPI) [88] with Cluster API Provider Azure (CAPZ) [89]. The reason for this choice is that although the other tools are great, they do not simplify the cluster provisioning and management as much as CAPI does - by making managing hundreds of clusters look seamless. By using a declarative approach, and GitOps tools like Argo CD, it allows for continuous reconciliation of the desired state of the clusters with their live configuration, without the need of managing a state file (e.g. Terraform) or manually/on-schedule running the tasks to stay in sync (e.g. Ansible). This is a significant advantage over the tools that do require it.
Additionally, Microsoft has shifted its focus from AKS Engine to CAPZ, and they are encouraging people to use it for deploying self-managed Kubernetes clusters in Azure [90].
CAPI itself is an open-source subproject of Kubernetes focused on providing declarative APIs and tooling to simplify the provisioning and lifecycle management of workload clusters by using a management cluster - the cluster with one or more infrastructure providers are installed [88]. On top of that, CAPZ provider allows efficient management at

scale of managed and self-managed Kubernetes clusters in Azure. Additionally, CAPZ is seamlessly integrated with Azure Service Operator (ASO) [91] that allows to manage any Azure infrastructure which could significantly reduce operational overhead as everything can be stored in Git as YAML files, without requiring any additional languages.

5.4 Planning The Cluster Deployment

This section outlines the planning and design considerations for the Kubernetes cluster deployment. It is based on the requirements identified in E-ITS base measures and the main emphasis is on the environment and application partitioning, high-availability (HA), node group planning and CI/CD integration. Development, test and production environments are separated, although this implementation will focus on deploying only the production environment. The following sections will mention different custom resources that are used to deploy the cluster and its components. Every Azure resource is explained in detail in the Self-managed Clusters paragraph in CAPZ documentation [89, Ch. 3], and every non-Azure resource is explained in the CAPI documentation [88].

5.4.1 Node Groups

Control Plane

As one of E-ITS advanced measures is high-availability (HA), each node group needs to be designed accordingly. For the stacked *etcd* topology, three control planes are needed to ensure the HA and help with the leader selection in case of a machine failure [86, 85]. These control plane nodes are spread across three Availability Zones (AZ) to ensure that it can withstand the failure of one zone. Control plane nodes are in a separate subnet from the worker nodes to meet the network segmentation requirement and can be accessed only through the Bastion in a separate management network. During the instance size selection, the recommendations from Kubernetes and *etcd* documentation are taken into consideration to avoid any resource starvation or performance issues [92, 93].

Ingress Node Group

A separate node group is created for the ingress controller to ensure that the external traffic is handled separately. This is important from both security and performance perspective - the ingress controller is exposed to the internet and needs to be secured properly, and traffic spikes on the these nodes do not affect the performance of other workloads. Two nodes are created in this node group, and they are spread across two availability zones to ensure that it can withstand the failure of one zone. During the instance size selection, Azure

recommendations shall be taken into consideration so that compute optimized instances with high network bandwidth and sufficient core count are selected [94].

Monitoring Node Group

Monitoring node group is created for the monitoring stack as these tools (e.g. log and metric collection and aggregation) can be resource-intensive (often memory and I/O) and affect the performance of other workloads if they are not isolated. Two nodes are created in this node group and spread across two availability zones to ensure the HA. Having a dedicated pool for monitoring tools makes it more efficient to manage the underlying resources by choosing the instance size based on the actual workload requirements. Additionally, it complies with the requirement for dedicated nodes for specialized tasks.

Storage Node Group

Storage node group is created for backup agents and controllers (e.g. Velero, Veeam) and other storage-related workloads. It is created to ensure that the backup operations do not affect the overall performance of the cluster. Similarly, with other node groups, two nodes are created in this node group and spread across two availability zones for HA. Backup operations can be heavily I/O and network bandwidth dependent. To avoid potential performance degradation, Azure recommendations shall be taken into consideration so that storage optimized instances with higher I/O capabilities are selected [94].

General Node Group

This node group is the primary pool that is dedicated to running the end-user applications. This node group is using the general purpose instances that have a balance CPU and memory ratio and are suitable for most workloads. A minimum of two nodes are created in this node group and spread across two availability zones to comply with the HA requirement. The instance size and the number of replicas are based on the actual workload needs and can be adjusted either manually or by using the Cluster Autoscaler functionality [95].

Isolated Node Group

This node group is created for running sensitive workloads with high protection requirements. It is created in a separate subnet and access to this subnet is limited by Network Security Group rules, allowing only SSH access from the Bastion host and API Server access with control plane subnet (for cluster management). Traffic between other Kubernetes nodes is not allowed. This node group uses the compute-optimised instances with a minimum of two nodes spread across two availability zones for HA.

5.4.2 Bastion Host

A bastion host is created in a separate *AzureBastionSubnet* subnet, and it is used to access the control plane and data plane nodes through SSH. The access to the bastion network is restricted to specific network ranges. The SSH authorized keys and user configuration are defined in *KubeadmControlPlane, KubeadmConfig, AzureMachinePool* and *AzureMachineTemplate* sections of the Cluster API configuration file and are used to allow access to the control plane and data plane nodes [96]. As the cluster is in private network, access to tooling user interfaces (e.g. Argo CD, Grafana, Prometheus) is provided through *enableTunneling* parameter which enables the native client support for the bastion host [97, 98]. Additionally, it is possible to set up a VPN Gateway to set up either a site-to-site or point-to-site VPN Connection to private network.

5.4.3 Network Design

The network is segmented into multiple subnets to comply with the E-ITS requirements - a subnet for control planes, subnet for data planes, subnet for isolated nodes and a separate subnet for the Bastion host. The subnet sizes are chosen based on the number of nodes in each node group - except for the Bastion host, where the subnet size has to be at least **/26** or larger, and the name of the subnet needs to be *AzureBastionSubnet* [99]. In this implementation, the number of nodes is fixed and quite small, however when designing an environment with potential growth in mind, the subnet masks should be chosen accordingly to facilitate the needed growth. The chosen network configuration can be seen in Table 2.

Table 2. Proposed Network Design Segmentation

Resource name	**Resource type**	**CIDR**
VNet	Virtual Network	10.0.0.0**/22**
control-planes	Subnet	10.0.0.0**/27**
data-planes	Subnet	10.0.0.64**/26**
AzureBastionSubnet	Subnet	10.0.0.128**/26**
isolated data-planes	Subnet	10.0.0.192**/26**

When designing the network rules, it is important to ensure to minimally allow traffic for necessary ports and protocols for the Kubernetes cluster to function properly. These ports include Kubernetes API, etcd, kubelet, kube-proxy, NodePort, CNI (Overlay networking), ingress controller etc., and can be found in documentation [100, 101].

5.4.4 Instance Size and OS Selection

Choosing the Instance Size

In addition to choosing the instance sizes based on the workload requirements, it is important to check the availability of these instance types in the selected region(s) together with the supported *HyperVGenerations* parameter.
This is important to note when choosing the OS image in *AzureMachineTemplate* and *AzureMachinePool* templates, as some images support either Gen1 or Gen2 Virtual Machines, and this would prevent the nodes from booting up if the generations are not compatible.
Additionally, ***vCPUsPerCore*** value should be noted because the vCPU count is not always equal to the number of processor cores - this gets especially important when planning to run CPU-intensive workloads. For example, on general-purpose instances ***vCPUsPerCore: 2*** means that for 2 vCPUs, there is usually only 1 core available (but 2 threads), whereas for compute-optimised instances there are usually 2 cores available.
The final choices for instance sizes can be seen in Table 3 and sizes are based on the recommendations from Azure, Kubernetes, *etcd* and other documentation and the actual workload planned for the cluster.

Table 3. Instance selection

Node Group	**VM Class**	**VM Size**	**Specifications**
control-plane	general purpose	Standard_D2as_v5	vCPU: 2t, RAM: 8 GB
monitoring-node	memory optimised	Standard_D2as_v5	vCPU: 2t, RAM: 8 GB
ingress-node	compute optimized	Standard_F2s_v2	vCPU: 2c, RAM: 4 GB
storage-node	memory optimised	Standard_E2bs_v5	vCPU: 2t, RAM: 16GB, high IOPS
general-node	general purpose	Standard_D2as_v5	vCPU: 2t, RAM: 8GB
isolated-node	compute optimised	Standard_F2s_v2	vCPU: 2c, RAM: 4GB

It is important to note that depending on the Azure Subscription type, the Total Regional *vCPUs* quota may be limited to 20, and this should be increased before deploying the cluster - otherwise the quota is exceeded, and the cluster cannot be deployed. This is

especially important when deploying multiple clusters in the same region, as the quota is shared across all clusters within the subscription.

Choosing the OS

Choosing the OS for the nodes is an important step and there are hundreds of distributions available - whether it is something based on Debian, RHEL (Red Hat Enterprise Linux), CentOS or something else.
Ideally, the OS should be actively maintained, have a good balance between security and stability and should be tested with Kubernetes. For example, Flatcar Container Linux is a minimal OS designed for running containers [102], but it is still in the Incubating stage of CNCF [102] and is not as widely used as other distributions and therefore could make it harder to find help and support.
However, when choosing a general-purpose distribution, the potential attack surface can be larger, as it comes with higher amount of pre-installed applications, and securing these distributions can be difficult, and it is something that shall be taken into consideration. For this implementation, the chosen distribution is Ubuntu 24.04 LTS (Long Term Support), as Canonical (the company developing Ubuntu) is an official Kubernetes Certified Service Provider. Ubuntu supports the latest Kubernetes v1.32, containerd v2.0, is actively maintained and contains the native tooling for CIS benchmarks. [103].

5.5 Initial Bootstrapping

The initial bootstrapping of the workload cluster is done by defining the following resources in the Cluster API configuration file, and then applying them through the management cluster:

- *AzureClusterIdentity* - defines the Azure identity that is used to authenticate and manage the Azure resources (e.g. service principal, managed identity.). For this implementation Service Principal is used to authenticate with Azure by following the CAPZ documentation [104];
- *Cluster* - the main object that defines the cluster configuration (e.g. name, version, cluster network configuration, etc.);
- *AzureCluster* - defines the Azure Cluster configuration (e.g. VNet, subnets, Bastion host, etc.);
- *MachinePool* - defines the set of data plane nodes that are created (e.g. failure domains, number of replicas, Kubernetes version, etc.);
- *AzureMachinePool* - a template for creating Azure Virtual Machine Scale Set that will be used as Kubernetes nodes (e.g. instance size, OS image, disks, etc.);

- *AzureMachineTemplate* - a template for creating Azure Virtual Machines that will be used as Kubernetes nodes (e.g. instance size, OS image, disks, etc.);
- *KubeadmControlPlane* - For managing the configuration and lifecycle of control plane nodes by using kubeadm (e.g. control plane component configuration, disks, users, replicas, etc.);
- *KubeadmConfig* - Defines the kubeadm configuration that is used to bootstrap the data plane node pools (e.g. users, ssh keys, additional scripts, etc.);

Since this is a self-managed Kubernetes cluster, a CNI plugin needs to be installed to enable the networking between the nodes and pods. In this implementation, Calico is used as the CNI plugin. Although multiple CNI plugins are available and were considered, Calico was chosen for its maturity and reliability. While Cilium's eBPF approach is different (can be used without kube-proxy), it still has some limitations (its mTLS support is in Beta) and this can introduce potential issues. Calico on the other hand supports mTLS by integrating with Istio [105].
Calico CNI plugin is installed by utilizing the *HelmChartProxy* bootstrapping functionality [106, 107], followed by the Cloud provider for Azure and Azure Disk CSI driver for Kubernetes (allows Kubernetes to access Azure Disk volumes).
After that, *HelmChartProxy* is used again to bootstrap Argo CD together with its initial app configurations that allows managing rest of the workload cluster applications with the 'app-of-apps' pattern [108]. This allows to seamlessly manage the applications and their configuration with GitOps principles and will always ensure the desired applications state. Additionally, when new applications are needed, creating a new *'Application'* in workload/apps/argoapps directory together with Helm Chart files in the respective workload/apps directory in Git repository [109] is enough to deploy it automatically in the cluster.
The Cluster API manifests, together with additional API Server configuration files, are stored in the mgmt/azure/cluster directory, *HelmChartProxy* manifests are stored in the mgmt/azure/helmproxycharts directory in Git repository [109] and are deployed by the management cluster for bootstrapping.

5.6 Securing the Cluster Components

This section focuses on securing the cluster components, such as the API server, kubelet, etcd, controller-manager, scheduler etc. Securing these components is crucial as it is the backbone of the cluster and if compromised, it can lead to disastrous consequences. Securing the control plane was done by using the *KubeadmControlPlane* resource, which allows to configure each component separately. Achieving this is done by following the CIS Kubernetes Benchmark v1.10.0 (the latest version at the time of writing), using this

widely accepted reference helps in meeting the E-ITS requirements that focus specifically on the cluster component security (and often restricts more than E-ITS requires).

5.6.1 Control Plane and Data Plane Configuration Files

This section focused mainly on configuration file permissions and ownership, as these files are critical for the overall cluster security. Most configuration file permissions were already restricted to minimal access by default (e.g. static pod manifests, CNI configuration files, *etcd* directory permissions, etc.), and the ownership was set to **root:root**. However, Kubernetes PKI key and certificate file permissions were not as restricted as they should've been (by allowing read access from other users), and this was mitigated by setting the permissions to 600.

5.6.2 Kubernetes API Server

Kubernetes API Server configuration was done in the *clusterConfiguration.apiServer.extraArgs* section, and its configuration is shown in Figure 1.

Figure 1. *Kubernetes API Server security configuration.*

```
kubelet-certificate-authority: "/etc/kubernetes/pki/ca.crt"
enable-admission-plugins: "AlwaysPullImages,NodeRestriction,EventRateLimit,
    DenyServiceExternalIPs"
disable-admission-plugins: "AlwaysDeny"
admission-control-config-file: "/etc/kubernetes/config/AdmissionConfiguration.yaml"
encryption-provider-config: "/etc/kubernetes/config/EncryptionConfig.yaml"
profiling: "false"
audit-log-path: "-"
audit-log-maxage: "30"
audit-log-maxbackup: "10"
audit-log-maxsize: "100"
audit-policy-file: "/etc/kubernetes/config/auditlog.yaml"
request-timeout: 60s
service-account-lookup: "true"
tls-cipher-suites: "TLS_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,
TLS_CHACHA20_POLY1305_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
```

This ensures that the API Server verifies the kubelet's serving certificates, sets the list of supported TLS cipher suites, configures the audit logging (will be explained in more detail in the logging section), defines the encryption provider to secure the *etcd* data at rest by utilizing the 'secretbox' provider that ensures a strong encryption [87] (e.g. secrets, config maps, etc.), enables the service account token lookup (by validating that it exists in *etcd*), enables the admission control plugins, such as:

- *AlwaysPullImages* - especially important in multi-tenant environments to ensure that images can only be used by those entities who have credentials to pull them. As a negative side, it increases the load on the image registry and ingress traffic.
- *NodeRestriction* - ensures that kubelet can only modify API objects on their own node, as it prevent them from modifying other node resources. This is additional step for resource isolation.
- *EventRateLimit* - enforces a limit on the number of events that API Server will accept to prevent exhaustion either by DoS attacks or misbehaving workloads. Enabling this plugin requires additional *AdmissionConfiguration* file that references to different configuration files - in this implementation by limiting the number of queries per second per user and namespace.
- *DenyServiceExternalIPs* - prevents users from creating Services that use *externalIPs*, and not add any external IP addresses to already existing services. In case this is needed, a custom policy shall be used to allow it.

An important side note is that enabling kubelet's serving certificate verification on the API Server level requires a kubelet TLS Bootstrapping, otherwise the kubelet won't be able to communicate with the API Server. This is described more in detail in the kubelet configuration section.

5.6.3 Controller Manager

Kubernetes Controller Manager configuration was done in the *clusterConfiguration.controllerManager.extraA* section and its configuration is shown in Figure 2:

Figure 2. *Controller Manager security configuration.*

```
allocate-node-cidrs: "false"
cloud-provider: external
cluster-name: capi-test
terminated-pod-gc-threshold: "500"
profiling: "false"
```

An external (out-of-tree) cloud provider was set to ensure that Cloud provider Azure is used for the cluster. This is important as this service is responsible for managing Azure resources, such as provisioning and managing Azure internal and external load balancers. Disabling profiling was done to prevent exposing system and program details that could be used by the attackers to exploit the system. The default settings of the garbage collector for terminated Pods was lowered to 500 to ensure that the terminated Pods are removed in a timely manner, as the default value of 12500 could lead to performance issue considering the resource specifications of the control plane nodes.

5.6.4 Etcd

In most cases, the default settings were used, as the default configuration mostly met the security requirements (e.g. limited access to configuration files, TLS encryption configuration, etc.). Access to *etcd* was already limited by separating the control plane and data plane networks, and by allowing access to control plane nodes only through the Bastion host. However, additional steps were taken to encrypt the confidential data at rest. Although the *EncryptionConfiguration* file location is defined in the API Server configuration, it increases the security of *etcd*, and hence the configuration is shown here in Figure 3:

Figure 3. *etcd encryption configuration.*

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - secretbox:
          keys:
            - name: key1
              secret: '[REDACTED]'
      - identity: {}
```

Although the primary encryption provider was set to *secretbox*, a fallback configuration was needed to read the unencrypted secrets during the initial migration as described in the official documentation [87]. This was followed by performing a secrets replacement with *kubectl replace* command to ensure that already existing secrets get encrypted.

5.6.5 Scheduler

Configuring the Kubernetes Scheduler was done in the *clusterConfiguration.scheduler.extraArgs* section, and it didn't require many changes from its default configuration to meet the security requirements. Similarly, to *etcd*, Controller Manager, and Kubernetes API server, profiling was disabled to prevent exposing system and program details to reduce the attack surface.

5.6.6 Kubelet

Securing the kubelet was done in multiple resource sections, such as *KubeadmControl-Plane* for the control plane nodes, and *KubeAdmConfig* for each node group. Initially, *KubeletConfiguration* file was patched as shown in Figure 4.

Figure 4. *Kubelet security configuration.*

```
{
  "apiVersion": "kubelet.config.k8s.io/v1beta1",
  "kind": "KubeletConfiguration",
  podPidsLimit: 4096,
  serverTLSBootstrap: true,
  tlsCipherSuites: [
    TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
    TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
    TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
    TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
    TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
    TLS_RSA_WITH_AES_256_GCM_SHA384,
    TLS_RSA_WITH_AES_128_GCM_SHA256
  ],
}
```

Enabling the *serverTLSBootstrap* is important to automatically request a serving certificate signed by the cluster's Certificate Authority for kubelet's HTTPS endpoint. This secures the communication initiated from the Kubernetes API server to kubelet (e.g. logs, exec into Pods) by verifying kubelet's identity and potentially preventing man-in-the-middle attacks. Additionally, this enables the use of the *RotateKubeletServerCertificate* to automatically rotate the certificates.
Due to security reasons, kube-controller-manager does not automatically approve *kubelet-serving* Certificate Signing Requests (CSR), and the requests were approved manually with *kubectl certificate approve* command. Although the kubelet-serving CSR approval can be automated by using a 3rd party solution (e.g. kubelet-csr-approver), it should be carefully considered by the organization as it introduces additional risks. For this implementation, no additional solutions were used to ensure that only administrator-approved nodes can join the cluster and communicate securely with the API Server to comply with E-ITS APP.4.4.M17.

5.6.7 Conclusion of Cluster Component Security

In this section, the focus was on securing the cluster components and configuration files. The full configuration for workload cluster bootstrapping is stored in mgmt/azure/cluster/eits-cluster.yaml file in the Git repository [109]. Cluster API's management cluster features Argo CD that is used to bootstrap and manage the workload cluster and includes the automatic configuration auditing, as automatic sync policy and reconciliation guarantees the desired cluster state in Git backend is in sync with the requirements set by E-ITS. Additionally, Cluster API itself continuously reconciles the workload cluster resources based on their definitions and this ensures prevents the sync drift.

5.7 Deploying the Applications

This section focuses on the applications that are deployed in the workload cluster. The applications are divided into different categories based on their functionality and purpose. Applications are deployed using Argo CD, which is deployed by using the *HelmChartProxy* bootstrapping functionality, and is then used to manage itself and the rest of the applications in GitOps way.
Although there are multiple tools available to manage the applications, such as Flux CD, Octopus Deploy, etc., Argo CD was chosen for its simplicity, functionality compared to other solutions (e.g. sync phases/waves) [110], integration with other Argo Project tools (e.g. Argo Workflows, Argo Events, Argo Rollouts), its ease of use, and its huge open-source community. Additionally, Argo CD has undergone internal and external security reviews and penetration testing, which increases the overall security of application management [111]. The applications are stored in a Git repository and are automatically deployed and managed by Argo CD when the configuration is changed.

5.7.1 Core Applications

Some core applications, such as Calico CNI, Azure Disk CSI driver and Azure Cloud provider are already deployed during the initial bootstrapping phase in the management cluster as they are needed for the cluster to function properly. However, there are additional applications that need to be deployed to ensure its basic functionality and security, such as certificate management and secrets management. Although these could be considered as security or networking applications, they are essential to securely handle the certificates, secrets and other sensitive data in the cluster.

Secrets Management

Managing secrets in Kubernetes can be challenging, especially storing them in a version control system securely, as Kubernetes secrets do not provide any encryption by default and use only base64 encoding. This means that anyone with access to the version control system (e.g. Git) can easily decode the secrets, and this could lead to serious security incidents. Fortunately, there are different tools available to tackle this issue, such as Sealed Secrets [112], External Secrets Operator (ESO) [113], Secrets Store CSI Driver [114], and others.
However, each tool has its own advantages and limitations. For example, Sealed Secrets is a great tool that doesn't require any external backend system to store the secrets, but because it stores the *SealedSecrets* (a custom resource that contains the encrypted value of a secret) locally, it requires resealing the secrets for every cluster that is created, and

this can be cumbersome when managing a huge number of clusters. ESO and Secrets Store CSI Driver require an external backend system to store the secrets (e.g. Azure Key Vault, HashiCorp Vault, AWS Secrets Manager, etc.) and this can introduce another layer of complexity to manage the RBAC permissions. However, when organizations already have a centralized secrets' management system in place, these tools can make it easier to manage secrets in Kubernetes.
Secrets Store CSI Driver is a great tool that allows to mount the secrets directly into the Pods as volumes, without exposing them in the environment variables [114], however the setup can be complex, and it requires additional resources on the cluster (as it needs to run on every node). In Table 4 is the comparison of the 3 most used secret management tools in Kubernetes and the comparison is based on the official documentation of these tools.

Table 4. Secret Management tools comparison

Name	**Sealed Secrets**	**ESO**	**Secrets Store CSI Driver**
Store type	Local	External	External
Dynamic updates	No	Yes	Yes
Dependencies	None	External provider	External provider
Setup	Simple	Moderate, requires provider configuration	Moderate, requires CSI driver setup with a provider
Cost	No additional cost	Provider prices may apply	Provider prices may apply
Advantages	No dependencies, automatic cert rotation	Dynamic secret updates, more providers, centralized secrets	Secrets securely mounted, dynamic updates, centralized secrets
Disadvantages	Difficult to migrate, needs separate manifests for each cluster	Provider depencency	Provider dependency, operational overhead, less providers

Organizations that are already using a centralized secrets' management system, or are planning to have multiple clusters across different environments, should usually consider using one with an external backend to make the management easier and allow better overview of the secrets. As the author of this thesis is already utilizing Azure Key Vault to store secrets, and the cluster is deployed in Azure, ESO has a good documentation with

examples, and a good community support, it was chosen as the secrets' management tool for this implementation.
In Figure 5 is the most important part of the ESO configuration is the *SecretStore* that defines the secret provider configuration, such as the Vault URL, Tenant ID, and Authentication method.

Figure 5. *ESO SecretStore configuration.*

```
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: azure-kv
  namespace: external-secrets
spec:
  provider:
    azurekv:
      tenantId: "${TenantId}"
      vaultUrl: "${MgmtVaultUrl}"
      authSecretRef:
        clientId:
          name: azure-sp-secret
          key: ClientID
        clientSecret:
          name: azure-sp-secret
          key: ClientSecret
```

The full configuration is stored in workload/apps/external-secrets-operator directory of the Author's technical implementation Git repository [109] and is deployed by Argo CD.

DNS and Certificate Management

Although Kubernetes ships with either kube-dns or CoreDNS (depending on Kubernetes version) DNS server, which is needed for the service discovery and resolution within the Kubernetes cluster. However, managing the DNS records outside of Kubernetes cluster (e.g. LoadBalancer and Ingress records, TXT records for domain validation etc.) is not possible with these tools and for this reason, a 3rd party tools are needed. There aren't many options available and the most popular and recommended tool, also listed in the Kubernetes SIGs (Special Interest Groups), for this is ExternalDNS - a tool to synchronize the exposed Kubernetes Services and Ingresses with DNS providers [115].
ExternalDNS supports various DNS providers, including Azure Private DNS - this is important as Cluster-API already utilizes Azure Private DNS for Kubernetes API Server records, so it is possible to use the same DNS provider to manage Ingress and LoadBalancer IP addresses as A type records, and the Provider metadata as TXT records. Figure 6 shows the Azure-specific configuration block for ExternalDNS that defines the Tenant ID, Subscription ID, Resource Group, Service Principal's Client ID and a Secret name containing the secret value.

Securing Ingress and LoadBalancer communication with TLS certificates is crucial to

Figure 6. *ExternalDNS Azure configuration.*

```
provider: azure-private-dns
azure:
    secretName: "azure-sp-secret"
    cloud: "PublicCloud"
    resourceGroup: "capi-ha"
    tenantId: "${TenantId}"
    subscriptionId: "${SubscriptionId}"
    aadClientId: "${ClientId}"
```

ensure the confidentiality and integrity of the transmitted data. Managing the certificates manually is not feasible and can become overwhelming - to simplify the process, a certificate management tool shall be used instead. There are multiple tools available, such as cert-manager [116], certbot [117], HashiCorp Vault [118].

Although Vault is a powerful tool with PKI capabilities, it's a complex solution and unless the organization already has it set up in place, configuring it only to generate TLS certificates is not recommended due to the operational overhead.

Certbot is an easy-to-use tool for generating and managing TLS certificates with Let's Encrypt and even though it supports many DNS plugins [117], it does not support Azure DNS out of the box and relies on a 3rd party plugin to be installed. However, this plugin is lacking the community support, and it has an open issue related to Azure DNS plugin [119] that could lead to issues on this technical implementation.

Cert-manager is a powerful tool to manager X.509 certificates in Kubernetes and it is the most widely used tool for this purpose [116] with a huge community support. It has a clear documentation with well-explained examples, supports various Issuers, the most popular DNS providers for ACME challenges, and multiple out-of-tree DNS providers by using webhooks [120]. Due to these reasons, cert-manager was chosen as the certificate management tool.

To distribute the certificates across Kubernetes cluster, a *ClusterIssuer* resource is created that defines the Issuer configuration and the Let's Encrypt ACME server. The author of this thesis has a domain name *'lndbrg.tech'* registered and managed by Azure DNS in the management resource group that is used by cert-manager to validate the domain ownership and issue the certificates.

As there is a Private Azure DNS zone with the same domain name, cert-manager tries to create the validation TXT record in Private DNS zone, which would fail, as it's not reachable from public internet. To avoid this, recursive nameserver configuration in Figure 7 was used to ensure that the DNS records are created in the public DNS zone. For this configuration, the public DNS servers from Cloudflare and Google were used.

Figure 7. *cert-manager recursive nameserver configuration.*

```
dns01RecursiveNameserversOnly: true
dns01RecursiveNameservers: "1.1.1.1:53,8.8.8.8:53"
```

Full ExternalDNS and cert-manager configurations with additional templates are stored in workload/apps/external-dns and workload/apps/cert-manager directories of the Author's technical implementation Git repository [109] and are deployed by Argo CD.

5.7.2 Networking and Ingress

Networking is one of the most important parts of Kubernetes, as it allows the communication within the cluster and with external services. Although the chosen CNI plugin (Calico) was already described in the previous section, there are additional components that are needed to ensure secure and reliable networking.

Ingress Controller

After the CNI plugin is installed, an Ingress controller is needed to manage the ingress traffic and route it to correct services. There are many Ingress Controllers available, and choosing the most suitable one can be difficult. Fortunately, Kubernetes documentation about Ingress Controllers [121] and CNCF Landscape's Service Proxy and Cloud Native Network sections [122] give a good overview of the most popular and recommended ingress controllers to use in Kubernetes.
Apart from the cloud provider ingress controllers, some of the most popular are nginx, HAProxy, Traefik, Kong (built on top of nginx), Cilium and Istio.

Although Cilium ingress controller introduces additional features compared to nginx, HAProxy and Traefik, setting it up can be a complex task, and since Cilium was not chosen as the CNI plugin, increasing the additional operational overhead for the sake of Ingress controller is not preferred by the author. Nginx and HAProxy are the most widely used ingress controllers that offer simplicity on setup, are well documented and easy to use. Even though this simplicity comes with a cost of limited features, they are still leading the way performance-wise, and there are many benchmark reports published by different organizations [123, 124, 125, 126, 127] and individuals [128, 129], highlighting that the highest performance is usually achieved by nginx and HAProxy. Taking everything into consideration together with author's personal experience, HAProxy Ingress Controller was chosen for this implementation.
The most important parts of the HAProxy controller configuration can be seen in Figure 8 which configures the service type to *LoadBalancer* and annotates the resource to inform the Cloud provider for Azure to create an internal Load Balancer.

The full configuration is stored in the workload/apps/haproxy directory in Git repository

Figure 8. *HAProxy Service and annotation configuration.*

```
service:
  type: LoadBalancer
annotations:
  service.beta.kubernetes.io/azure-load-balancer-health-probe-request-path: "/healthz"
  service.beta.kubernetes.io/azure-load-balancer-internal: "true"
```

[109] is deployed by Argo CD.

Service Mesh

A service mesh is another important layer in the Kubernetes networking stack that provides additional features such as seamlessly enabling the mTLS communication between the application, without having to distribute the certificates manually. On top of that, it allows for more refined traffic management either on L4 or L7 level, depending on the chosen solution. There are multiple service mesh providers available, such as Istio, Linkerd, Cilium, Consul, Traefik etc, however the first three are the most popular and widely used solutions.
Although Linkerd is fastest in terms of performance and latency (according to benchmark reports), it comes with some drawbacks - it is not as feature-rich as Istio and Cilium. For example, both Istio and Cilium support onboarding external services into the service mesh, whereas Linkerd does not. When organizations have external resources that need to be onboarded, Linkerd would not be a suitable choice. It is using a sidecar proxy solution whereas Cilium uses per-node proxy solution and Istio supports both. From the resource consumption perspective, Linkerd is the heaviest as it relies on having a sidecar proxy for every Pod which can become difficult to manage.
While Cilium is good at CNI-level networking with its eBPF support, it is not as feature-rich on Layer 7 traffic management capabilities as Istio with Envoy proxy that has significant customization options. The learning curve of Istio is steeper than Cilium and Linkerd, and should be taken into consideration when choosing the service mesh, however the base functionality to use mTLS support with Ambient mode (proxies on node level not as a sidecar for Pod) is easy to set up and use. As the author of this thesis has previous experience with Istio, and taking into consideration the aforementioned details, Istio in Ambient Mode was chosen as the service mesh for this implementation.

Fortunately, Istio has a good documentation with various examples and Helm charts available that made the setup easier. The most important part to take into consideration when deploying Istio in Ambient mode is to ensure that the underlying operating system supports the required kernel modules, especially when using a custom Linux kernels or nonstandard distributions [130]. As Calico CNI plugin is used, its FelixConfiguration

needs to be patched as in Figure 9 to allow Pods use *allowedSourcePrefixes* to send traffic with a source IP that is not theirs [131]. Additionally, Kiali was deployed to provide a graphical representation of the service mesh and its components. To secure the Kiali access, it was set up in read-only mode so that no changes to the mesh can be made from the UI. Kiali's configuration is stored in the workload/apps/kiali directory in Git repository [109].

Figure 9. *Calico FelixConfiguration patching.*

```
kubectl patch felixconfigurations default \
--type='json' -p= \
'[{"op":"add","path":"/spec/workloadSourceSpoofing","value":"Any"}]'
```

By implementing Istio Ambient mode, and using Argo CD to annotate namespaces with *istio.io/dataplane-mode: ambient*, the communication between the Pods is automatically secured with mTLS. Istio service mesh configuration manifests are stored in the workload-/apps/istio directory in Git repository [109] gets deployed by Argo CD.

5.7.3 Monitoring Stack

This chapter describes the monitoring stack that is used to monitor the cluster resources and applications. It is divided into multiple sections based on the functionality and purpose of the tools. All the components are chosen thoroughly and are well-known and widely used in the Kubernetes community.

Logging Stack

The logging section is divided into two parts, describing their implementation methods. The first part describes the logging stack that is used for collecting and storing applications logs, while the second part focuses on enabling and handling Kubernetes API Server audit logs.
In both scenarios, a central logging system is required to aggregate logs, along with tools to collect the data from log files or container stdout/stderr streams, and then transmit them to central logging system. The most common log collection agents include Fluentd, Fluentbit, Filebeat, Promtail and Grafana Alloy, while the most popular backends for storing logs are ElasticSearch, OpenSearch, Splunk and Grafana Loki. For this implementation, the author selected Grafana Loki, paired with Grafana Alloy for log collection. This choice was made on several advantages over other tools:

- **Resource efficiency:** Loki is designed to be lightweight and efficient, without indexing the full content of logs and focusing only on metadata indexes. This significantly reduces storage requirement and compute resources.

- **Operational simplicity:** Architecture can be deployed in multiple ways, depending on the needs - either simplifying and reducing the operational overhead by using more of the monolithic approach, or scaling up each component separately, and using cloud services as a storage backend.
- **Ecosystem integration:** Both Loki and Alloy are part of the Grafana observability stack and are designed to work seamlessly together. This allows for easy integration with other tools such as Grafana Mimir and Tempo.
- **Cost:** Both Loki and Alloy are open-source and free to use which eliminates software licensing fees compared to Splunk.

Therefore, Grafana Loki and Alloy were chosen as the logging stack for this implementation as it provides a good balance between performance, cost, ease of use and its suitability with Kubernetes.
To avoid additional operational overhead and scope creep, the author decided to use the Loki's 'monolithic deployment' approach (although it is not monolithic and still uses different components) that allows utilizing the persistent volumes for storage backend instead of relying on external storage solutions, such as S3.
Grafana Alloy was configured to collect logs from Kubernetes nodes and applications, including Kubernetes audit logs, and ship them to Loki. Loki and Alloy configuration manifests are stored in the workload/apps/monitoring/grafana/loki and workload/apps/-monitoring/grafana/alloy directories in Git repository [109].

Audit Logs

In addition to application logs, capturing Kubernetes API Server audit logs is critical for security monitoring, compliance and to overall understand the changes in the cluster. Audit logs were configured directly on Kubernetes API Server by using specific parameters. In this implementation, it is configured in *KubeadmControlPlane* resource, which is used to manage the control plane nodes. The parameters used to configure the audit logs are shown in Figure 10, and perhaps the most important aspect is that *audit-log-path* is set to *"-"* which means that the audit logs are written to stdout and can be collected by Alloy.

Figure 10. *Auditlog configuration.*

```
clusterConfiguration:
  apiServer:
    extraArgs:
      audit-log-path: "-"
      audit-log-maxage: "30"
      audit-log-maxbackup: "10"
      audit-log-maxsize: "100"
      audit-policy-file: "/etc/kubernetes/config/auditlog.yaml"
```

Additionally, a specific audit policy file (auditlog.yaml) was created to have a more

balanced approach guided by the industry best practices and recommendations [32] to ensure that critical events are captured without overloading the system with too much data. The most important part of the audit policy file is shown in Figure 11 to guarantee that changes on the Secrets, ConfigMaps and TokenReviews are logged, but no sensitive data is exposed - this is done by logging only on Metadata level.

Figure 11. *Audit log Policy.*

```
- level: Metadata
  resources:
  - resources: ["secrets", "configmaps", "tokenreviews"]
```

The full audit log policy together with configuration parameters are stored in mgmgt/cluster/eits-cluster.yaml file in Git repository [109] and is used by the management cluster to bootstrap the workload cluster.

Telemetry Stack

For telemetry and observability, the most widely used and recommended tools in Kubernetes are Prometheus for telemetry collection and Grafana for visualization. There are a few alternatives, such as New Relic, Zabbix, Nagios, VictoriaMetrics, etc. and they differ in purpose and functionality. Apart from VictoriaMetrics and New Relic, the others are not really cloud native, and the main use case is not for Kubernetes.
New Relic is a powerful solution that works well in Kubernetes, but it is mostly a paid solution and which gets expensive quickly when scaling up. It does have a free tier, but it is limited to 100 GB of data ingestion per month [132], which is usually not enough for production workloads. Additionally, it is a SaaS model, which might not be suitable for some organizations, especially in the public sector.
VictoriaMatrics is a fast a cost-effective monitoring solution and is perhaps the best alternative to Prometheus, has some features that Prometheus is lacking (e.g. long-term storage, horizontal scalability for data ingestion) [133] and organizations that need to ingest high amount of data should consider using it. However, it is not yet as widely used and does not have as big community support as Prometheus and the learning curve can be steep, Prometheus fits the needs of this implementation better and is easier to set up and manage.

Fortunately, there is a collection of Kubernetes manifests, Prometheus rules with documentations and scripts to have an end-to-end cluster monitoring by Prometheus [134]. It is especially developed for Kubernetes and simplifies the deployment and management by having an Operator that manages the custom resources. The installation comes with a set of dependencies that are needed for the monitoring stack to work properly, such as

kube-state-metrics and Prometheus Node Exporter.
Kube-prometheus-stack was chosen for this implementation, and the configuration was modified so that Pods are scheduled on 'monitoring' node group by using the nodeSelector and tolerations that allow scheduling on these nodes. The most important difference from the default deployment was to use a persistent storage instead of *emptyDir* and this is shown in Figure 12. This is important to ensure that the data is not lost in case the Pod is restarted or scheduled to another node.

Figure 12. *Prometheus persistent storage definition.*

```
storageSpec:
  volumeClaimTemplate:
    spec:
      storageClassName: standard
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 50Gi
```

The full configuration manifest is stored in workload/apps/monitoring/kube-prometheus-stack directory in Git repository [109].

Visualization and Reporting

The central platform for visualizing and interacting with collected monitoring data (telemetry, logs) is Grafana. It is the primary user interface that can be used by developers, DevOps engineers, product managers and other stakeholders to have insights into the systems health and performance.
The selection of Grafana was based on several factors that make it well-suited for this implementation:

- **Native integrations:** Grafana offers out-of-the-box integration for various data sources, including Prometheus and Loki, that are used in this implementation.
- ***De facto* standard:** Grafana can be considered as the *de facto* standard for observability in Kubernetes, that comes with pre-built dashboards and various integrations.
- **Unified visualization:** The goal was to have a single interface for viewing and analyzing logs and telemetry, and Grafana provides this capability by allowing custom dashboards with mixed-data sources.
- **Extensibility:** While primary use case in this implementation is to visualize telemetry and logs, it has additional features, such as alerting, reporting, and data transformation that provide flexibility.

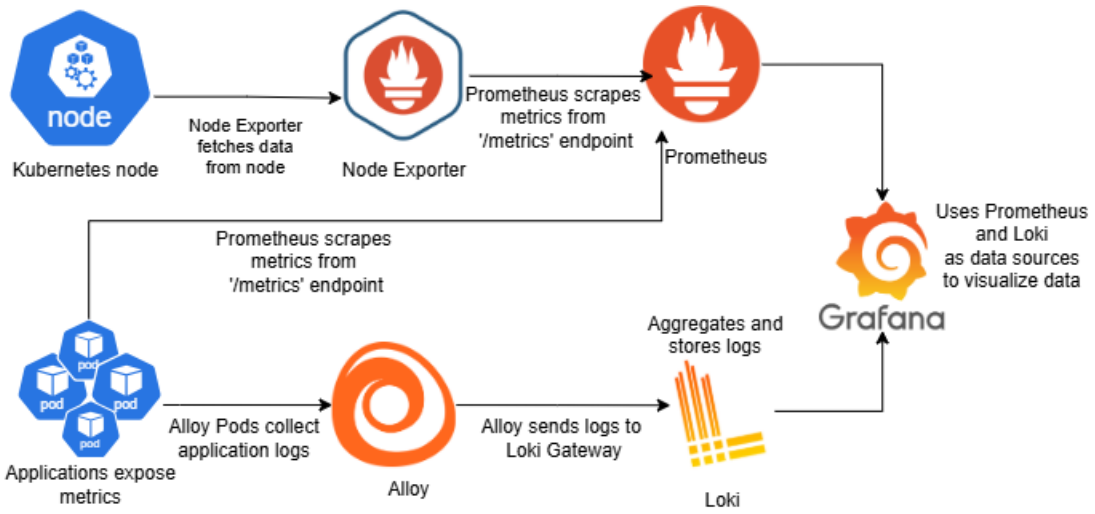
Apart from configuring the Loki and Prometheus data sources, an Ingress resource was created to expose Grafana outside of Kubernetes cluster. Additionally, the default adminis-

trator password was changed to a more secure one.

Monitoring Architectural Overview

This section provides the architectural overview of the monitoring stack that was used in this implementation. On a high level, containers are exposing logs and telemetry data to stdout/stderr streams, and metrics endpoints, which are then collected by the log collection agents (Grafana Alloy) and the exposed telemetry (either by dedicated exporters or the application endpoints) is then scraped by Prometheus and stored in its time-series database. The collected logs are then shipped to a central logging system (Grafana Loki). The logs and telemetry data are then visualized in Grafana that uses Loki and Prometheus as data sources. The architecture diagram is shown in Figure 13.

Figure 13. *Monitoring architecture overview.*



The diagram shows the main components and their interactions, including the Kubernetes node and node exporter, the applications as Pods, the log collection agents, telemetry scrapers, and central logging system. The arrows indicate the direction of the data, and although some components use push-based models rather than pull-based model, the overall data flow direction is the same.

5.7.4 Backup and Restore

Ensuring the resilience, availability and recoverability of the Kubernetes cluster is critical for any production workloads, and a reliable backup solution must capture both - the Kubernetes resource configurations and the persistent data stored in the volumes, and preferably in a way that allows to store the backups outside cluster in a central location with restoring capabilities.

For this implementation, Velero was chosen - an open-source tool to safely back up,

restore and perform disaster recovery on Kubernetes resources and persistent volume [135]. Although Velero was chosen, several alternatives were considered:

- **Kasten K10:** A powerful and feature-rich backup solution to back up Kubernetes resources, including application-aware backups, disaster recovery [136]. However, the free version is limited to maximum of 5 nodes which does not fit the needs of this implementation. Additionally, it is a commercial solution and can be expensive, especially when scaling up.
- **KubeStash:** A cloud-native open-source backup and recovery solution for Kubernetes workloads. Although it is a good and simple alternative to Velero, it does not support automatic backups and database backups in the community edition [137], and the enterprise edition is not free, which doesn't make it suitable for this implementation.
- **Portworx Backup:** A commercial backup solution that provides application-aware and granular backup and restore capabilities for Kubernetes. It allows cloud snapshots, native CSI snapshots (for both on-premises and cloud solutions), and local backups that can later be offloaded to external storage [138]. However, a free tier can be used for 30 days, after which it requires a license. Unfortunately, the pricing is not publicly available, and using a commercial solution is not preferred in this implementation.
- **kube-dump:** A simple open-source solution that backups Kubernetes resources as YAML manifests [139]. Unfortunately, this solution does not support persistent volumes, and although it could be used together with another solution to back up persistent volumes, it isn't actively maintained anymore (last commit was in 2022), and from security perspective, it is not recommended to use anything that does not at least get security updates.

Therefore, considering the limitations and drawbacks of other solutions, Velero was chosen as the backup tool for this implementation. The key features that made Velero a suitable choice include - it is free to use and open source, actively maintained and is a widely used solution in the Kubernetes community, is well documented, and a complete solution (backups, restore, disaster recovery, etc.), supports various storage providers, including Azure Blob Storage and Azure Managed Disks that are suitable for this implementation.

Furthermore, due to seamless integration with Azure, it supports secure data backups and restores by transmitting the already encrypted data over TLS to Azure Blob Storage, which configuration is shown in Figure 14, where the data is again encrypted at-rest, which meets the E-ITS requirements for secure data exchange and communication.

Figure 14. *Azure Storage Account and Storage Container configuration.*

```
az storage account create \
    --name $AZURE_STORAGE_ACCOUNT_ID \
    --resource-group $AZURE_BACKUP_RESOURCE_GROUP \
    --sku Standard_GRS --encryption-services blob \
    --https-only true --min-tls-version TLS1_2 \
    --kind BlobStorage --access-tier Hot --location northeurope
az storage container create \
    --name velero --public-access off \
    --account-name $AZURE_STORAGE_ACCOUNT_ID
```

After the role creation, Velero documentation was followed further to configure the Backup Storage Location and Volume Snapshot Location, together with a Kubernetes secret for Azure Storage Account access. The most important part of the configuration is shown in Figure 15 about initContainer settings - without this, Velero is unable to recognize the backup locations and the provider plugin is not installed. Additionally, a schedule was created to schedule a full backup every day at 1:00 AM (UTC). The full Velero configuration is stored in the workloads/apps/velero directory in Git repository [109].

Figure 15. *initContainer configuration for Azure provider.*

```
initContainers:
- name: velero-velero-plugin-for-microsoft-azure
  image: velero/velero-plugin-for-microsoft-azure:v1.11.0
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - mountPath: /target
      name: plugins
```

5.7.5 Security and Compliance Tools

While security has been the foundational consideration throughout this implementation, this chapter details the tools that are essential for ensuring the cluster's compliance to defined security policies and standards. This includes the policy enforcement for defining and managing specific cluster and application configurations, followed by security assessment tools for measuring against the recognized security benchmarks.

Policy Enforcement

Kyverno was selected for policy enforcement in this implementation due to its Kubernetes-native approach. It manages policies as Kubernetes resources using YAML, making it easy to integrate with existing Kubernetes workflows. Kyverno provides a comprehensive set of tools to manage the Policy-as-Code (PaC) lifecycle for Kubernetes by allowing to validate, mutate, generate and clean up any Kubernetes resource [11]. Although Kyverno is not the only tool available for this purpose, it was chosen over the following alternatives:

- **OPA Gatekeeper:** Although OPA Gatekeeper is a powerful policy engine that allows policy enforcement, it doesn't provide built-in policy templates and uses Rego language for writing policies, which can be difficult to learn in this implementation context. Additionally, the mutation capabilities are limited compared to Kyverno.
- **Kubewarden:** It is a fast and powerful, developer-centric policy engine that allows policies to be written in any programming language that compiles to WebAssembly (Wasm). Although it is flexible and allows using libraries from programming languages, compiling to Wasm introduces additional layer of complexity.
- **jsPolicy:** Another developer-centric policy engine that allows policies to be written in programming language - JavaScript or TypeScript. Although it is flexible and allows using libraries from JavaScript ecosystem, from non-developer perspective, learning JavaScript or TypeScript can be difficult.
- **Pod Security Admission:** A built-in Pod Security admission controller that enforces Pod Security Standards at namespace level. As the name suggests, it is limited to work only on Pods, and does not provide the same level of flexibility and customization as Kyverno.

Therefore, this makes Kyverno the most suitable choice for policy enforcement tool in this implementation.

Enforced Policies With Kyverno

This section describes the policies that were enforced by Kyverno in this implementation. There were multiple types of policies used in this implementation that included validating, mutating and generating policies. The most important policies with their objectives are listed in the table below:

- **restrict-default-sa-automount:** This policy restricts automounting the 'default' service account token in each namespace by mutating the existing and new service account resources with *automountServiceAccountToken: false*. This ensures that the default service account token is not automatically mounted into a Pod when it is created without specifying a service account. Enforcing this policy complies with 'APP.4.4.M9 Security of the Kubernetes service account' security measure.
- **disallow-default-namespace:** This policy disallows creating resources in the 'default' namespace by validating the resource creation and denying it if the namespace is 'default'. It is required by E-ITS that each application shall have their own namespace, and not using a default namespace is a good practice that supports this requirement.
- **generate-networkpolicy-exitsing:** As the name implies, this policy generates a

network policy for each existing namespace and new namespace that is created. The generated policy allows egress traffic within its own namespace, internal DNS resolution on TCP and UDP port 53 to 'kube-system' namespace and ingress traffic from the link-local address *169.254.7.127/32* that is used by Istio Ambient mesh for mTLS capabilities, and everything else is denied. This supports the 'APP.4.4.M7 Partition of Kubernetes networks' requirement and complies with 'APP.4.4.M18 Microsegmentation (C-I)'.

- **require-pod-probes:** This policy requires that all Pods have liveness and readiness probes defined to ensure that the application is healthy and ready to serve the traffic, and can be automatically restarted when needed. This policy was set to 'Audit' mode, as not all applications have endpoints that can be used for probes. Although it is a good practice to have them, and a custom probe can be created (relying on telemetry or logs), it is not always reliable and can end up in a situation where the application is healthy, but the probes are failing. This policy supports 'APP.4.4.M11 Monitoring the use of containers' security measure.
- **require-requests-limits:** This policy requires that all containers have resource requests and limits defined. It was set to 'Audit' mode, as not all applications have init containers or sidecar containers defined, so it's not possible to enforce this policy explicitly on all containers. However, it reports that there are containers that do not have the requests and limits defined and can be used as a reference to set them. This helps to avoid conflicts created by applications that could consume all resources and starve other applications.
- **generate-quotas:** This policy generates Limit Range and Resource Quota for each namespace, ensuring that the number of resources created into a namespace is controlled, and provides the default resource limitations for the applications. This supports the 'APP.4.4.M1 Designing the partition of applications' base measure.
- **generate-peerauthentication:** This ClusterPolicy generate a PeerAuthentication configuration for Istio Ambient mTLS configuration for each namespace.

The content of these and other policies are stored in the workload/apps/kyverno/templates directory in Git repository [109] and are deployed by Argo CD. The policies were created to ensure that the cluster is compliant with security standards and best practices. As for the enforced Network Policy - to still allow the cluster services to work without disruptions, additional Network Policies were created for each application, taking into consideration the explicit ports that are used by the application. These policies are either defined in the values file of the Helm chart or stored as a template in application's directory.

Tools for Security Assessment

For security posture assessment, kube-bench was selected. It's an open-source tool developed by Aqua Security that checks the Kubernetes cluster against the CIS Kubernetes Benchmark [74], which is vital for meeting the security requirements like E-ITS, and in some measures is more strict. The tool is designed to be easy to use and provides a comprehensive report on the security posture of the cluster, including recommendations for remediation.
Although it is not the only tool available for this purpose, it was chosen over the following alternatives:

- **Kubescape:** A comprehensive open-source security platform developed by ARMO with wide range of security scanning capabilities, such as vulnerability scanning and compliance checks against multiple frameworks [140]. While it now supports CIS Kubernetes Benchmark, the most common tool for this purpose is still kube-bench as it is specifically designed for that purpose. Additionally, the reporting does not provide the same level of explanation and detail for remediation as kube-bench.
- **Trivy:** Also developed by Aqua Security, is an open-source scanner for vulnerabilities, misconfigurations, secrets and other types of security issues [141]. It is richer in features than kube-bench, but it is not allow specifying the Benchmark version, which means that cloud provider managed clusters (usually without access to control plane) and self-managed clusters are scored in the same way. Additionally, kube-bench runs as a lightweight Kubernetes Job, and can be easily managed.
- **Kube-score:** A tool for static code analysis of Kubernetes objects that provides a list of potential issues and recommendations for improvement [19]. While it is a useful tool for identifying issues in Kubernetes manifest definitions, it doesn't check the life state of the cluster resources or its components (e.g. API Server, etcd, kubelet, etc.) and therefore cannot be used to assess the security posture in the same way as kube-bench.

Therefore, kube-bench was selected as the primary tool for security assessment in this implementation due to its simplicity, feature-rich CIS Benchmark checks, and the fact that it is widely used in the Kubernetes community.

5.8 Conclusion of Technical Implementation

The technical implementation has described the whole process of setting up the cluster. Starting with a selection and the preparations of the resource platform and provisioning tools, followed by a strategic planning of the cluster architecture and its components, initial

bootstrapping with the custom resource definitions, then securing the cluster components according to the security requirements and best practices. After that, the applications were categorized into different groups based on their functionality and purpose. For each category, the most common, recommended and widely used tools were thoroughly evaluated, and the most suitable ones were selected based on the outcome of the evaluation and requirements of E-ITS.

The technical implementation started by provisioning the workload cluster in the management cluster's Argo CD instance. After the resources were provisioned and validated, Calico CNI and Cloud Controller Manager were deployed by using the *HelmChartProxy* resources in management cluster's Argo CD instance. Then Azure Bastion host was used to access the workload cluster's control plane to approve the kubelet-serving CSRs. This was followed by doing the final modifications on kubelet configuration - running the script shown on Figure 16 on each Kubernetes node to ensure that the *tlsPrivateKeyFile* and *tlsCertFile* values are appropriate:

Figure 16. *Additional Kubelet TLS Configuration.*

```
#!/usr/bin/env bash
set -euo pipefail

CONFIG_FILE="/var/lib/kubelet/config.yaml"

# Append the TLS settings to kubelet config
cat <<EOF >> "$CONFIG_FILE"
tlsPrivateKeyFile: /var/lib/kubelet/pki/kubelet-server-current.pem
tlsCertFile: /var/lib/kubelet/pki/kubelet-server-current.pem
EOF

# Reload systemd and restart kubelet
systemctl daemon-reload
systemctl restart kubelet

echo "Appended TLS configuration."
```

The management cluster's Argo CD instance was then used to bootstrap the ArgoCD instance to workload cluster with its first Application (app-of-apps, containing the logic for every other application deployment) by using *HelmChartProxy* resource, and *ClusterResourceSet* feature to automatically apply an initial set of secrets to the workload cluster. This new Argo CD instance in the workload cluster then bootstrapped some core applications in the workload cluster - Azure Disk CSI Driver, HAProxy Ingress Controller, cert-manager, ExternalDNS and External Secrets Operator. After that, Calico CNI's FelixConfiguration was patched to allow Istio Ambient Mesh to work properly with the CNI. The rest of the applications were then deployed by manually clicking on "Sync" button in Argo CD.

An important thing to note is that the workload cluster is deployed to a private Azure network, and a pre-requirement is that the management cluster is either in the same network or has connectivity to that network. For this purpose, a VNet Peering was created

automatically (by Cluster API resource) between the management cluster network and the workload cluster network. In this example, a single-node AKS cluster was used as the management cluster, but it can be any Kubernetes cluster as long as it has connection to the workload cluster's private network.
Additionally, as this is a private cluster, accessing the exposed services (Ingress, Load Balancer) such as Argo CD, Grafana, etc. is not possible without having a VPN connection set up to the private network. For this implementation, a Gateway Subnet, Azure VPN Gateway and a Point-To-Site VPN connection were created to allow the author to access the services. However, Azure Private DNS Zone is used for storing DNS records. To resolve the service names, a DNS Resolver is needed and as the VPN Gateway is already expensive, and Azure DNS Resolver is expensive, the most cost-effective solution is to manually add the host entries in the hosts file. Of course, the services can still be exposed to public internet by modifying the HAProxy Load Balancer Service annotations, but for this implementation, the author decided to keep the services private and not expose them to public internet for increased security.

The practical outcome of this implementation will be presented and discussed in detail in the subsequent Results chapter.

6. Results

This chapter delves into the results of this Kubernetes cluster implementation, with a focus on validating compliance with the E-ITS APP.4.4: Kubernetes implemented measures. The findings in this chapter provide crucial part for answering the **RQ1** and **RQ2**.

6.1 Validation Strategy

To ensure comprehensive assurance of the implemented E-ITS measures. Two key validation methods were used:

- Experimental validation: controlled tests in the cluster (e.g., kube-bench tests, policy validation runs, etc.) to prove that security measures behave as expected under the defined conditions.
- Empirical validation: observations and metrics gathered from live cluster (e.g., manually checking *etcd* encryption, container cgroup configuration, etc.) and inspecting the defined Kubernetes configuration files (e.g., manifests, cluster component configurations, etc.) to verify that the implemented security measures are effective and persistent.

6.2 Validating E-ITS APP.4.4 Implemented Measures

To confirm that this Kubernetes cluster meets the APP.4.4 requirements, both automated and manual verification assessments were used. First and foremost, a resource-level validation was performed to ensure that the Azure resources were correctly provisioned and configured. This involved checking the 'Cluster' resource state in the management cluster and verifying the provisioned resources in Azure portal as shown in Appendix 13 – Provisioned resources in Azure Portal.

6.2.1 Cluster Component Configuration Validation With 'kube-bench'

Then, kube-bench was used to perform automated and manual check against the latest (at the time of writing v1.10) CIS Kubernetes Benchmark, verifying the security posture of control plane configuration files, API Server, Controller Manager, Scheduler, etcd,

control plane authentication and authorization, audit logging, together with worker node configuration files, kubelet and kube-proxy configuration.
This supports the overall validation process, as many controls are related to E-ITS APP.4.4 measures. Additionally, it provides remediation steps that can be used for verification.
The tests were run with the following targets: 'master, node, etcd, controlplane' to cover all cluster components. The test results are shown in Appendix 4 – kube-bench test results. Although most tests were automatically validates, some tests required manual validation (either) and those will be discussed in the next subsections.

False Positives

The number of false positives encountered during the validation was low, but they were still present. The details of these false positives with manual verification and explanations are shown in Figure 17.

Figure 17. *False positive errors in kube-bench.*

```
[FAIL] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (
    ↪ Automated)
# etcd is in stacked configuration and located on control plane node, which does not
    ↪  have 'etcd' user at all. Instead' root' user is used as expected.

[FAIL] 4.1.1 Ensure that the kubelet service file permissions are set to 600 or more
    ↪  restrictive (Automated)
# Ownership is restricted, kubelet_service_config variable is set, still gives error
    ↪ . Manual verification shows the kubelet service configuration file location
    ↪ and then its permissions that are restricted to 600:

root@capi-ha-control-plane-h692g:~/kube-bench# systemctl status kubelet.service |
    ↪ grep -A1 "Drop-In"
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             10-kubeadm.conf

root@capi-ha-control-plane-h692g:~/kube-bench# stat -c %a  /usr/lib/systemd/system/
    ↪ kubelet.service.d/10-kubeadm.conf
600
```

Manual Checks

Although the majority of the tests were automated, the following tests needed manual verification. The details with the manual validation results are shown in Appendix 8.4.

Even though Kubernetes API server is configured to use anonymous authentication, as it is considered reasonable as long as RBAC authorization is used, a work-around has been provided to globally disable anonymous authentication and enable this feature only for specific paths, granting that health probes are still working without issues.

RBAC and Other Pod Security Standards Checks

The remediation steps of kube-bench Policy tests were used to thoroughly validate that the RBAC roles are configured by following the least-privilege principle. Additional checks were performed to validate that *hostPath, hostIPC, hostPID, hostNetwork* were not used where it was not needed, and the applications did not allow privilege escalation. After manually running the commands provided in each remediation step, and investigating the outcomes thoroughly, it was confirmed that these parameters were only used by Calico CNI and Prometheus Node Exporter. Calico CNI requires it for networking purposes, and Prometheus Node Exporter requires it to access the host system metrics. This is also confirmed by the official documentation of the application [142]. To conclude this, the manual steps were performed to check that applications are not using elevated privileges, do not allow privilege escalation and are configured by following the industry best practices, and with the thorough validation, it was confirmed.

6.2.2 Validation of each E-ITS APP.4.4 implemented measure

This section provides a detailed validation of each E-ITS APP.4.4 measure by taking into account the results of kube-bench tests, and showing the additional manual validation steps that were taken to ensure that the measures were implemented correctly.

The measures are grouped in the same way they are grouped by lifecycle in the E-ITS APP.4.4 module:

Planning

- **APP.4.4.M1 Designing the partition of applications:** ResourceQuota and LimitRange definitions are stored in the workload/apps/kyverno/templates/resourceQuotaLimitRange.yaml file [109] and are automatically applied to the cluster by Kyverno.
  Figure 18 shows the automatic generation of ResourceQuota and LimitRange for a new namespace that was created in the cluster.
  Networks segmentation is defined in the *networkSpec.subnets* field of the *AzureCluster* resource in the mgmt/azure/cluster/eits-cluster.yaml file [109]. Each application is assigned to work in their own respective namespace, except for the Kubernetes system tools which are assigned to the *kube-system* namespace. The cluster contains the applications with a similar security requirements - no specific applications are deployed that would require different security posture.
- **APP.4.4.M2 Automation of the development of applications with the help of CI/CD :** Argo CD is used to manage the applications in the cluster, and not manual

Figure 18. *ResourceQuota and LimitRange generation.*

```
root@capi-ha-control-plane-ph7w5:~# kubectl get clusterpolicies.kyverno.io | grep "
    ↪ add-ns-quota"
add-ns-quota                                          true        true         True
    ↪ 2m45s   Ready
root@capi-ha-control-plane-ph7w5:~# kubectl create ns "test-quotas"
namespace/test-quotas created
root@capi-ha-control-plane-ph7w5:~# kubectl get resourcequotas -n test-quotas
NAME                    AGE   REQUEST                                       LIMIT
default-resourcequota   13s   requests.cpu: 0/3, requests.memory: 0/12Gi   limits.
    ↪ cpu: 0/4, limits.memory: 0/16Gi
root@capi-ha-control-plane-ph7w5:~# kubectl get limitranges -n test-quotas
NAME                CREATED AT
default-limitrange   2025-05-03T20:24:06Z
```

deployments can be done, as the access to the control plane is limited to only approved users through the Bastion host (the user accounts have to be created beforehand, and SSH keys need to be provisioned to the host). Access to Argo CD instance is limited to only administrator and no one else. The data managed by Argo CD (logs) are collected in the central logging system and are stored for 14 days. Access to logging environment is limited to only administrator and no one else.

- **APP.4.4.M3 Planning the Kubernetes identity and rights management:** RBAC authorization is used for the Kubernetes API Server, AlwaysALlow authorization mode is disabled. This was automatically validated by kube-bench 1.2.6-1.2.8 tests, and the results are shown in Appendix 4 – kube-bench test results. Kubernetes Storage Class is used to manage the Persistent Volumes in the cluster, and its *reclaimPolicy* is configured to Retain the data, so even when Persistent Volumes are deleted, the data is retained.

  Storage Class is protected against accidental pruning in Argo CD by using the following annotation: *argocd.argoproj.io/sync-options: Delete=false*, and this can be modified only by Argo CD administrator. Storage Class configuration is stored in workload/apps/azure-disk-csi/templates/storageclass.yaml [109].

Implementation

- **APP.4.4.M4 Partition of pods:** Kubelet and containerd are configured to use the systemd cgroup v2 driver, ensuring the Linux namespace isolation. Cgroup configuration and container isolation validation steps are shown in Appendix 6 – Kubelet cgroup configuration and namespace isolation.
- **APP.4.4.M6 Secure resetting of the pods:** This measure shall not be enforced, as it is not applicable in most cases. This is because most applications do not require *initContainers*, as usually the configurations have already been done while building the container image. However, when applications did require additional configurations before starting, *initContainers* were used. This can be validated in

Figure 15.

- **APP.4.4.M7 Partition of Kubernetes networks:** This was partially validated in APP.4.4.M1, however Appendix 7 – Azure Bastion (Management), control plane node, isolated node and other data plane node subnets shows subnet configuration in Azure portal. Accessing control-plane, isolated data-plane nodes, and other data-plane subnets is allowed only through the Bastion host, which is configured in *AzureCluster* resource [109]. Additionally, isolated-data nodes allow traffic from control-plane subnets for cluster management (as required by Kubernetes documents) and traffic within its subnet, but prohibits any other inbound traffic for isolation purposes.

  CNI permissions restrictions were already validated with kube-bench 1.1.9 and 1.1.10 tests, and the results are shown in Appendix 5 – kube-bench cluster component manual verification steps. Additionally, Kyverno enforces 'default-deny' Network Policies for each namespace which are automatically generated for each new namespace that is created in the cluster.

Figure 19. *Kyverno ClusterPolicy for NetworkPolicy generation.*

```
root@capi-ha-control-plane-ph7w5:~# kubectl get clusterpolicies.kyverno.io grep "
    ↪ generate-networkpolicy"
NAME                            ADMISSION   BACKGROUND   READY   AGE   MESSAGE
generate-networkpolicy-existing   true        true         True    65m   Ready
```

Operation

- **APP.4.4.M5 Backup of cluster information :** Cluster resources are automatically backed up with Velero - *etcd* backup is stored in Azure Blob Storage container, and Volume Snapshots of Persistent Volumes are stored directly in the management resource group. The backup validations are shown in Appendix 8 – Velero backup validation in Azure portal
- **APP.4.4.M8 Security of the Kubernetes configuration files:** Kubernetes application configurations include a version number with a short description of the application - this is done in the Helm Chart configuration files and can be validated with any Helm Chart that is stored in workload/apps/ directory [109].
- **APP.4.4.M9 Security of the Kubernetes service accounts:** Each application that needs a service account is using a dedicated service account. By following the official documentation of the applications, service account permissions are set to the minimum required for the application to work. Kyverno disables *automountServiceAccountToken* for the 'default' service account in each namespace, and this is validated in Appendix 9 – Disabled automount for default service accounts.
- **APP.4.4.M10 Security of the automation process:** Argo CD is using cluster-wide

permissions to manage the applications within the cluster. This is required as it manages every namespace in the cluster. However, Argo CD instance can only be accessed through internal network and only by the administrator.

- **APP.4.4.M11 Monitoring the use of containers:** Kyverno ClusterPolicy requires that all containers (including init containers, sidecar containers and ephemeral containers) have defined health probes. However, the policy is set to 'Audit' mode, as some applications only have one endpoint defined (e.g. Grafana Alloy supports only readinessProbe), and enforcing this policy would cause the applications to fail. A validation of the health probe requirement is shown in Appendix 10 – Kubernetes health probe requirement validation.
- **APP.4.4.M12 Security of infrastructure applications:** Infrastructure applications are secured, and each application is using a dedicated service account with minimal permissions for the application to work (based on the official documentation). Velero backups are encrypted in-transit by utilizing TLS encryption and the backup data is encrypted at rest by Microsoft-managed keys. The encryption of backups was already validated in APP.4.4.M5, and the encryption configuration is shown in Appendix 16 – Velero backup output with *EncryptionAtRestWithPlatformKey*. Additionally, Istio Ambient is enabled in the cluster to use mTLS encryption for pod-to-pod communication. This can be validated by checking from Kiali UI or by using the istioctl command.

Additional advanced measures

- **APP.4.4.M13 Automatic configuration audits (C-I-A):** kube-bench performs daily automated audits via CronJobs by checking its configuration against the CIS Kubernetes Benchmark v1.10. The results were already shown it Appendix 4 – kube-bench test results, and application configuration specifics are located in workload/apps/kube-bench/templates directory [109]. For continuous auditing and reporting, Kyverno is used. Kyverno rules are defined in the workload/apps/kyverno/templates directory [109] and the applied policies are shown in Figure 20. The final layer of auditing capabilities is provided by Argo CD that verifies the state of applications and against the desired state in Git.
- **APP.4.4.M14 Use of specialised nodes (C-I-A) :** The following node groups were created in the cluster: monitoring, ingress, storage, isolated and general, and are shown in Appendix 11 – Specialised Kubernetes nodes with their configuration defined in the respective *KubeadmConfig* section. Each application is configured to run on the designated nodes based on the configured tolerations and nodeAffinity rules in the Helm Chart configuration files. Access to Kubernetes nodes is allowed only through the Bastion, which was explained in APP.4.4.M7.

Figure 20. *Deployed Kyverno ClusterPolicies.*

```
root@capi-ha-control-plane-ph7w5:~# kubectl get clusterpolicies.kyverno.io
NAME                                ADMISSION   BACKGROUND   READY   AGE     MESSAGE
allow-specific-kube-system          true        true         True    2m36m   Ready
allow-specific-kyverno              true        true         True    2m36m   Ready
add-ns-quota                        true        true         True    2m36m   Ready
disallow-default-namespace          true        true         True    3h1m    Ready
generate-networkpolicy-existing     true        true         True    3h1m    Ready
generate-peerauthentication         true        true         True    3h1m    Ready
restrict-default-sa-automount       true        true         True    3h1m    Ready
require-pod-probes                  true        true         True    3h1m    Ready
require-requests-limits             true        true         True    3h1m    Ready
```

- **APP.4.4.M15 Partition of applications at the levels of nodes and clusters (C-I-A):** Although there are no applications that require very high protection requirements, a dedicated isolated node group was created in the cluster to facilitate the potential need for such applications in the future. The creation of such node group can be validated in Appendix 11 – Specialised Kubernetes nodes, and its Network Security Group Rules is shown in Appendix 14 – Isolated node group Security Group Rules. The isolated node group is configured to use a dedicated subnet, which is defined in the *networkSpec.subnets* field of the *AzureCluster* resource, and the configuration is stored in the mgmt/azure/cluster/eits-cluster.yaml file [109]. Additional manual validation was performed to ensure that the isolated node group is not accessible from the other node groups outside the control-plane subnet, and the outcome is shown in Appendix 15 – Isolated node group connection testing.
- **APP.4.4.M16 Use of Kubernetes operators (C-I-A):** This implementation uses various operators to efficiently manage core application in Kubernetes cluster. The most notable ones are: Velero, Calico CNI (Tiger Operator), Prometheus Operator (in kube-prometheus-stack), and External Secrets Operator.
- **APP.4.4.M17 Certification of nodes (C-I-A):** Each kubelet is authenticating to the API Server using a serving TLS certificate, which was signed by the cluster CA. Both kubelet and API server are configured to use strong cryptographic ciphers, and kubelet server TLS certificates are configured to automatically rotate. This measure was validated by automatically passing the following kube-bench tests:

  API Server *1.2.4, 1.2.5, 1.2.24, 1.2.29 and 1.3.6*

  kubelet: *4.2.3, 4.2.10, 4.2.11 and 4.2.12*

  The whole content of the kube-bench test results is shown in Appendix 4 – kube-bench test results. Additionally, TPM attestation was not used in this implementation and will be explained in more detail in Discussion chapter.
- **APP.4.4.M18 Microsegmentation (C-I):** The 'default-deny' network policies are automatically generated for each namespace and then enforces by Kyverno. Each application has additional network policies defined based on the actual need and are stored in each workload/apps/ directory [109].

- **APP.4.4.M19 Guaranteeing the high availability of Kubernetes (A):** The cluster is highly available - it is using three control-plane nodes, and two data-plane nodes for each node group. This is shown in Appendix 11 – Specialised Kubernetes nodes. Additionally, control-plane nodes are provisioned across three availability zones and data-plane nodes are provisioned across two availability zones. Configuration is done in *KubeadmControlPlane* for control-plane nodes and *MachinePool* for data-plane nodes, which is stored in the mgmt/azure/cluster/eits-cluster.yaml file [109]. Backups are stored off-site and were already validated in APP.4.4.M5.
- **APP.4.4.M20 Encryption of the control plane storage space (C):** The underlying virtual machines are using end-to-end encryption by utilizing the encryption at host feature, which are configured in *AzureMachineTemplate* section for the control plane nodes and *AzureMachinePool* for each data plane node group. Additionally, *etcd* data is encrypted at rest by configuring *EncryptionConfiguration* for the API Server. This is automatically validated by passing the *1.2.27* and *1.2.28* tests. Additional manual validation was performed, and the results are shown in Appendix 2 – Secret encryption verification in etcd.
- **APP.4.4.M21 Periodic restarting of the pods (C-I-A):** Although manual application restarting is not common in Kubernetes, as it's supposed to be automatic based on its defined configurations (e.g. health probes, etc.), or there are operators managing the resources and restarting the Pods when needed, a feature for this measure was still implemented. A dedicated ServiceAccount, ClusterRole (permissions following the least-privilege principle), ClusterRoleBinding and a CronJob were created to restart the specified applications in the cluster periodically. The CronJob is configured to run once a day at 01:00 and restart only the resources that have been specified in the Helm values file, to avoid restarting the applications that do not require it.

  Appendix 17 – Validation of 'restarter' CronJob shows the deployed CronJob, a Pod created by the CronJob, and the logs confirming that the applications were restarted.

6.2.3 Conclusion of the Validation

In this chapter, the author demonstrated through the use of automated tools (kube-bench, Kyverno, Argo CD, etc.) and targeted manual validation checks that the deployed and configured Kubernetes cluster has fully implemented all E-ITS APP.4.4: Kubernetes measures. The detailed pass/fail results of the automated tests are shown in Appendix 4 – kube-bench test results, manual validation results are shown throughout this chapter, and the Table Appendix 12 – **PEARO** principle validation applies the **PEARO** principle by following the E-ITS application guide [143] to conclude the implementation status of each measure.

This thesis delivered a detailed and practical implementation of Kubernetes cluster that fully complies with E-ITS APP.4.4: Kubernetes requirements (has implemented each measure) and it can serve organizations as a guidance, reference deployment or a template that can be adapted to their needs and requirements to achieve E-ITS compliance on their Kubernetes clusters. All configuration manifests, scripts and other relevant resources are published in the public Git repository [109].

For privacy and security reasons, some values such as Tenant ID, Subscription ID, *etcd* encryption keys and other sensitive values have been redacted. Prior or during the workload cluster deployment, organizations must provision any required secrets, such as Argo CD repository credentials, External Secrets Operator and External DNS authentication credentials either by using the ClusterResourceSet in the management cluster (as demonstrated in the technical implementation chapter) or with their preferred bootstrapping method. While this implementation used Argo CD for management cluster, it can be substituted with other GitOps tools such as Flux, or simply by using 'kubectl apply' command to deploy and bootstrap the workload cluster.

Additionally, regions, instance types, network configurations, storage type and sizes, and other resource parameters can be adjusted to fit the organization's environment and requirements.

7. Discussion

This chapter discusses the findings of the research by addressing the research questions, highlighting the study limitations and potential recommendations for improvement.

7.1 Findings

The following section addresses the research questions set in the Introduction chapter, and they are answered based on the findings of previous chapters.

7.1.1 Answers to Research Questions

[RQ1] How can organizations implement E-ITS 'APP.4.4: Kubernetes' to achieve compliance and enhance the security posture of their Kubernetes clusters?

Organizations can achieve E-ITS compliance through a structured approach that includes exploratory analysis, constructive design and implementation and a thorough validation. Initially, a Systematic Literature Review (SLR) could be combined with detailed document analysis to provide clarity on specific E-ITS requirements and help to align them with Kubernetes best practices. Then, a high-level architecture design is created based on the identified requirements. Organizations should then follow a structured and incremental implementation process (to simplify the implementation tasks) supported by a comparative analysis to select the most suitable tools. Finally, a validation process through automated benchmarks and manual validation ensures that the E-ITS compliance and enhanced security posture. The technical implementation together with the validation process outlined in this study can be used as a practical guideline or reference for organizations trying to achieve similar compliance requirements.

[RQ2] What tools, methods, processes can be used for the E-ITS 'APP.4.4: Kubernetes' adoption?

The adoption of E-ITS compliance can leverage structured methods and various tools identified in this study. Initially, each E-ITS measure should be reviewed to understand the specific security requirements and to determine the suitable combination of tools to address them effectively. To simplify the implementation process, the tools can be grouped into different categories based on their functionality, such as core applications (for the cluster to function), secrets management, monitoring, DNS and certificate management, service mesh

and others. An iterative implementation process should be used to deploy and validate the tools in each category before moving to the next one. This step-by-step approach simplifies the troubleshooting process and addresses the security issues systematically, reducing the risk of misconfigurations. The validation process should include both automated security benchmarks and manual validation process to ensure that each measure is fully covered, and the final outcome is a compliant Kubernetes cluster. The technical implementation chapter can be used to get a detailed overview of possible tools that can be used to achieve the E-ITS compliance.

[RQ3] What are the specific security requirements outlined in E-ITS 'APP.4.4: Kubernetes' and how do they align with Kubernetes best practices?

E-ITS APP.4.4: Kubernetes outlines several detailed security requirements that are based on the 5 identified threat categories - control plane authentication and authorization, Pod confidentiality breach, resource conflicts, unauthorized changes in the cluster and unauthorized Pod access. The measures aim to mitigate these threats by implementing security controls across different layers of Kubernetes architecture. Generally, the requirements to implement the outlined security measures are aligned with widely recognized Kubernetes hardening guides (e.g., CTR Hardening Guide, CIS Kubernetes Benchmark) and overall best practices.

[RQ4] What gaps exist between E-ITS 'APP.4.4: Kubernetes' and current Kubernetes security in academic literature?

The primary identified gaps between E-ITS and existing literature identified during the SLR is the lack of possible security measures (with a few exceptions) after the threats have been identified. The current academic literature often focuses on specific security aspects or technical issues, whereas the scope of E-ITS is considerably broader. Additionally, there is no literature that focuses on practical implementation of Kubernetes security measures that are aligned with regulatory frameworks or country-specific standards. This study bridges this gap by providing a comprehensive practical implementation that is tailored for E-ITS APP.4.4: Kubernetes and can be used as a template, guidance or reference for organizations trying to achieve the same or similar compliance requirements.

7.2 Limitations

A several limitations were identified during this research which potentially affected the scope and execution of the study.

- **Lack of TPM-based attestation:** APP.4.4.M17 Certification of nodes (C-I-A) mea-

sures preferred the use of Trusted Platform Module (TPM) for node attestation. This was not implemented due to limited availability of Confidential Virtual Machine instances in the chosen region or incompatibility of the supported Hyper-V generation with the OS images used for this implementation. The instance sizes that were available in the region and were compatible with the Hyper-V generation were not practical to use due to their resource specifications - minimum of 24 cores. This made them impractical to use due to inefficient resource utilization and a huge cost.

- **Limited Scope to IPv4:** The implementation was limited to IPv4-only network configurations. No IPv6 or dual-stack configurations were implemented. Although this was considered by the thesis author at the beginning of architecture design phase, it was decided to limit the scope to IPv4-only to not extend the complexity of the implementation and potentially delay the outcome of the project.
- **Azure Dependency:** The practical implementation utilized Azure cloud services as underlying infrastructure. While the core Kubernetes cluster configurations (e.g., API Server, etcd, kubelet, etc.) are platform-agnostic, certain custom resources (e.g., AzureCluster, AzureMachinePool) are Azure specific. This introduces a platform dependency, however, these can be replaced with other provider's resources (e.g., AWSCluster and AWSMachinePool) to achieve the same functionality
- **Budgetary constraints:** Due to limited Azure credits, the full production-sized cluster could not remain running for the entire duration of the research to avoid running out of Azure credits and paying a hefty price for the resources. After each deployment category was completed, the cluster was dismantled to avoid unnecessary costs. This on-off cycle of provisioning and deprovisioning the resources significantly slowed down the progress and prevented the author from keeping the cluster available during the writing phase of the next category.

7.3 Recommendations

Based on the findings throughout the research, several recommendations are proposed to enhance the E-ITS APP.4.4: Kubernetes module and its implementation process:

- **Wording improvement:** Some E-ITS requirements include a bit unclear wording or potential translation ambiguities (possibly lost in translation from BSI sources) and could benefit from a review to enhance the clarity. Additionally, some measures could include more context or examples to help organizations understand the intention behind them.
- **Revision of APP.4.4.M9 Security of the Kubernetes service accounts:** This measure focuses on securing the Kubernetes service accounts. However, one of the requirements states the following - "Those pods which do not require a service

account do not have one. Such pods use tokens for communicating with the Kubernetes control plane." [59]. This statement is misleading, as Pods will always have a service account, either the default one or a designated one. It can be assumed that the intention was to disable the automatic service account token mounting for Pods that do not require it, but this does not equal to not having a service account at all. The second part is missing the context - if there is no service account token mounted, then which token is described in the second part of the statement? The recommendation is to revise the wording of this measure to avoid confusion and clarify the intention.

8. Conclusion

8.1 Summary

This thesis provided a structured and practical approach for organizations seeking to implement and achieve E-ITS APP.4.4: Kubernetes compliance. The exploratory phase, which included a Systematic Literature Review and document analysis, identified key security challenges, industry best practices, and most notably the lack of concrete and actionable guidance for compliance with standards such as E-ITS. The constructive phase involved designing a high-level architecture, evaluating and selecting tools, followed with a step-by-step technical implementation of E-ITS compliant Kubernetes cluster. The validation phase included automated checks against the CIS Kubernetes Benchmark complemented with a thorough manual verification process to confirm that the cluster has been configured according to the E-ITS requirements.

Every configuration file used for technical implementation in this thesis (including Cluster API resources and Kubernetes manifests) is stored in the author's public GitHub repository. It contains all the necessary files to deploy an E-ITS compliant Kubernetes cluster on Microsoft Azure using Cluster API. The repository is publicly accessible for anyone to use and adapt for their own needs, and it can be accessed at: `mlndbr6/e-its-kubernetes/`.

8.2 Contribution

This work bridges the gap between theoretical security measures and a real-world implementation, providing a practical solution for organizations aiming to comply with E-ITS APP.4.4: Kubernetes requirements and enhance their Kubernetes security posture. By providing a validated technical solution, this thesis delivers an actionable template, guidance or reference for organizations to deploy and configure secure Kubernetes clusters that are E-ITS compliant in a structured and a simplified way.

In addition to contributing to the academic literature on Kubernetes security with a specific focus on national security standards, it delivers a valuable practical solution to the community, supporting organizations in building secure and compliant Kubernetes clusters.

8.3 Generalization

Although the implementation was based on Microsoft Azure infrastructure, the core methods and configurations (e.g., cluster components, Kubernetes resources) are platform-agnostic and can be adapted to other cloud providers or on-premises environments. This shows the solution's usefulness outside the specific context of this study, making it applicable to a wide range of environments.

8.4 Future Work

While this implementation focused on identified requirements, the future research opportunities include two identified limitations in Chapter 7. For example, implementing a TPM-based node attestation as recommended in 'APP.4.4.M17 Certification of nodes (C-I-A)', extending the current implementation to support dual-stack networking configurations.
Additionally, the solution could be further enhanced by completely automating the kubelet TLS bootstrapping process using the 'kubelet-csr-approver' [144].
Furthermore, the existing Helm charts could be improved by modifying the current static templates to a more dynamic and adaptable state, which would reduce code duplication and enhance the flexibility of application deployments.
Finally, the research could be expanded to explore the use of security-focused operating systems for the underlying Kubernetes nodes. As these systems are designed with security in mind, they may limit in some functionalities and this needs to be studied further to determine their suitability for this use case.

References

[1] *The State of Kubernetes Security in 2024*. en. URL: https://www.redhat.com/en/blog/state-kubernetes-security-2024 (visited on 11/16/2024).

[2] *E-ITS*. URL: https://eits.ria.ee/et/versioon/2023/eits-poohidokumendid/etalonturbe-kataloog/app-rakendused/app4-aerirakendused/app44-kubernetes (visited on 01/08/2025).

[3] *Pure, upstream Kubernetes is the best Kubernetes*. en-US. Jan. 2023. URL: https://www.cncf.io/blog/2023/01/30/pure-upstream-kubernetes-is-the-best-kubernetes/ (visited on 02/16/2025).

[4] BSI. *APP.4.4 Kubernetes*. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-GS-Kompendium_Einzel_PDFs_2023/06_APP_Anwendungen/APP_4_4_Kubernetes_Edition_2023.pdf?__blob=publicationFile&v=4 (visited on 02/09/2025).

[5] Murugiah Souppaya, John Morello, and Karen Scarfone. *Application container security guide*. NIST SP 800-190. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 25, 2017, NIST SP 800–190. DOI: 10.6028/NIST.SP.800-190. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf (visited on 02/21/2025).

[6] *What Is Container Orchestration? | IBM*. en. Oct. 2024. URL: https://www.ibm.com/think/topics/container-orchestration (visited on 02/23/2025).

[7] *Kubernetes Documentation*. en. URL: https://kubernetes.io/docs/home/ (visited on 11/16/2024).

[8] Vladyslava Shekula and Edmund Laugasson. "Kommunikatsiooni turvalisuse parandamine Kubernetesi põhistes keskkondades". In: (May 2024). URL: https://digikogu.taltech.ee/et/item/bbe036c3-551a-4b7f-a672-e01794b4de5d (visited on 11/16/2024).

[9] 12 Minute Read. *Authorization Policy*. en. URL: https://istio.io/latest/docs/reference/config/security/authorization-policy/ (visited on 01/22/2025).

[10] *Gatekeeper | Gatekeeper*. URL: https://open-policy-agent.github.io/gatekeeper/website/ (visited on 03/04/2025).

[11] *Kyverno*. URL: https://kyverno.io/ (visited on 03/04/2025).

[12] Fairwinds Ops Inc. *Polaris | Open Source Policy Engine for Kubernetes*. URL: https://www.fairwinds.com/polaris (visited on 03/06/2025).

[13] *Easier & Faster Kubernetes Policies | jsPolicy*. URL: https://www.jspolicy.com/ (visited on 03/11/2025).

[14] Aamir Ali et al. "Implementation of New Security Features in CMSWEB Kubernetes Cluster at CERN". en. In: *EPJ Web of Conferences* 295 (2024). Ed. by R. De Vita et al., p. 07026. ISSN: 2100-014X. DOI: 10.1051/epjconf/202429507026. URL: https://www.epj-conferences.org/10.1051/epjconf/202429507026 (visited on 11/16/2024).

[15] Akond Rahman et al. "Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study". In: *ACM Trans. Softw. Eng. Methodol.* 32.4 (May 2023), 99:1–99:36. ISSN: 1049-331X. DOI: 10.1145/3579639. URL: https://doi.org/10.1145/3579639 (visited on 12/15/2024).

[16] *akondrahman/sli-kube - Docker Image | Docker Hub*. URL: https://hub.docker.com/r/akondrahman/sli-kube (visited on 03/11/2025).

[17] *Configure a Security Context for a Pod or Container*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ (visited on 03/11/2025).

[18] *Introduction*. URL: https://docs.kubelinter.io/#/ (visited on 03/13/2025).

[19] *kube-score - Kubernetes object analysis with recommendations for improved reliability and security*. URL: https://kube-score.com/ (visited on 03/12/2025).

[20] *What is Checkov? - checkov*. URL: https://www.checkov.io/1.Welcome/What%20is%20Checkov.html (visited on 03/13/2025).

[21] Md Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. *XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices*. Issue: arXiv:2006.15275 arXiv:2006.15275. June 2020. DOI: 10.48550/arXiv.2006.15275. URL: http://arxiv.org/abs/2006.15275 (visited on 10/04/2024).

[22] Giorgio Dell'Immagine, Jacopo Soldani, and Antonio Brogi. "KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments". en. In: *Future Internet* 15.7 (July 2023). Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, p. 228. ISSN: 1999-5903. DOI: 10.3390/fi15070228. URL: https://www.mdpi.com/1999-5903/15/7/228 (visited on 11/16/2024).

[23] Akond Rahman, Chris Parnin, and Laurie Williams. "The Seven Sins: Security Smells in Infrastructure as Code Scripts". In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). ISSN: 1558-1225. May 2019, pp. 164–175. DOI: 10.1109/ICSE.2019.00033. URL: https://ieeexplore.ieee.org/document/8812041 (visited on 03/20/2025).

[24] *Home - KubeHound*. URL: https://kubehound.io/ (visited on 03/14/2025).

[25] Francisco Ponce et al. "Smells and refactorings for microservices security: A multivocal literature review". In: *Journal of Systems and Software* 192 (Oct. 1, 2022), p. 111393. ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111393. URL: https://www.sciencedirect.com/science/article/pii/S016412122200111X (visited on 03/20/2025).

[26] Omar Jarkas et al. "A Container Security Survey: Exploits, Attacks, and Defenses". In: *ACM Comput. Surv.* 57.7 (Feb. 20, 2025), 170:1–170:36. ISSN: 0360-0300. DOI: 10.1145/3715001. URL: https://dl.acm.org/doi/10.1145/3715001 (visited on 03/21/2025).

[27] Choi Dongmin et al. *Uncovering Threats in Container Systems: A Study on Misconfigured Container Components in the Wild | IEEE Journals & Magazine | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/10788674 (visited on 03/21/2025).

[28] Burak Ünver and Ricardo Britto. "Automatic Detection of Security Deficiencies and Refactoring Advises for Microservices". In: *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. 2023 IEEE/ACM International Conference on Software and System Processes (ICSSP). May 2023, pp. 25–34. DOI: 10.1109/ICSSP59042.2023.00013. URL: https://ieeexplore.ieee.org/document/10169061 (visited on 03/22/2025).

[29] *Pomegranate-suite/pomegranate-suite.sh at main · ramsessw/Pomegranate-suite*. GitHub. URL: https://github.com/ramsessw/Pomegranate-suite/blob/main/pomegranate-suite.sh (visited on 03/24/2025).

[30] Mayank Agrawal and Kumar Abhijeet. “Security Audit of Kubernetes based Container Deployments: A Comprehensive Review”. In: 07.6 (2020). URL: https://www.irjet.net/archives/V7/i6/IRJET-V7I6649.pdf.

[31] *Updated: Kubernetes Hardening Guide | CISA*. Aug. 30, 2022. URL: https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF (visited on 03/26/2025).

[32] *CIS Kubernetes Benchmarks*. CIS. URL: https://www.cisecurity.org/benchmark/kubernetes (visited on 03/24/2025).

[33] *CIS Benchmarks® FAQ*. CIS. URL: https://www.cisecurity.org/cis-benchmarks/cis-benchmarks-faq/ (visited on 04/15/2025).

[34] Murugiah Souppaya, John Morello, and Karen Scarfone. *Application container security guide*. NIST SP 800-190. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 25, 2017, NIST SP 800–190. DOI: 10.6028/NIST.SP.800-190. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf (visited on 03/24/2025).

[35] *OWASP/www-project-kubernetes-top-ten*. original-date: 2022-03-31T16:26:46Z. Mar. 23, 2025. URL: https://github.com/OWASP/www-project-kubernetes-top-ten (visited on 03/24/2025).

[36] *Updated: Dockershim Removal FAQ*. Kubernetes. Section: blog. Feb. 17, 2022. URL: https://kubernetes.io/blog/2022/02/17/dockershim-faq/ (visited on 03/25/2025).

[37] *Configure Service Accounts for Pods*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/ (visited on 03/25/2025).

[38] *Separation of Development, Test and Production Environments*. https://www.isms.online/. URL: https://www.isms.online/iso-27002/control-8-31-separation-of-development-test-and-production-environments/ (visited on 03/26/2025).

[39] *Control 8.26, Application Security Requirements | ISMS.online*. https://www.isms.online/. URL: https://www.isms.online/iso-27002/control-8-26-application-security-requirements/ (visited on 03/26/2025).

[40] *design-proposals-archive/network/networking.md at main · kubernetes/design-proposals-archive*. GitHub. URL: https://github.com/kubernetes/design-proposals-archive/blob/main/network/networking.md (visited on 03/26/2025).

[41] Ramaswamy Chandramouli. *Guide to a Secure Enterprise Network Landscape*. NIST Special Publication (SP) 800-215. National Institute of Standards and Technology, Nov. 17, 2022. DOI: 10.6028/NIST.SP.800-215. URL: https://csrc.nist.gov/pubs/sp/800/215/final (visited on 03/26/2025).

[42] *Network Segmentation - OWASP Cheat Sheet Series*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Network_Segmentation_Cheat_Sheet.html#network-segmentation-cheat-sheet (visited on 03/26/2025).

[43] *Multi-tenancy*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/security/multi-tenancy/ (visited on 03/26/2025).

[44] *Container and Kubernetes compliance considerations*. URL: https://www.redhat.com/en/topics/containers/compliance (visited on 03/26/2025).

[45] *What is GitOps?* URL: https://www.redhat.com/en/topics/devops/what-is-gitops (visited on 03/26/2025).

[46] Ramaswamy Chandramouli, Frederick Kautz, and Santiago Torres-Arias. *Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines*. NIST Special Publication (SP) 800-204D. National Institute of Standards and Technology, Feb. 12, 2024. DOI: 10.6028/NIST.SP.800-204D. URL: https://csrc.nist.gov/pubs/sp/800/204/d/final (visited on 03/26/2025).

[47] *CI/CD: Complete Guide to Continuous Integration and Delivery*. Codefresh. URL: https://codefresh.io/learn/ci-cd/ (visited on 03/26/2025).

[48] *Authenticating*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/reference/access-authn-authz/authentication/ (visited on 03/26/2025).

[49] *Managing Service Accounts*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/ (visited on 04/19/2025).

[50] *Role Based Access Control Good Practices*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/security/rbac-good-practices/ (visited on 03/26/2025).

[51] "Cybersecurity Framework". In: *NIST* (Nov. 12, 2013). Last Modified: 2025-03-14T14:55-04:00. URL: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf (visited on 03/26/2025).

[52] *Cluster Architecture*. Kubernetes. URL: https://kubernetes.io/docs/concepts/architecture/ (visited on 03/28/2025).

[53] *User Namespaces*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/workloads/pods/user-namespaces/ (visited on 03/28/2025).

[54] *Operating etcd clusters for Kubernetes*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/ (visited on 03/28/2025).

[55] *Volume Snapshots*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/storage/volume-snapshots/ (visited on 03/28/2025).

[56] *velero/design/Implemented/csi-snapshots.md at main · vmware-tanzu/velero*. GitHub. URL: https://github.com/vmware-tanzu/velero/blob/main/design/Implemented/csi-snapshots.md (visited on 03/28/2025).

[57] *Storage Integration — Veeam Kasten 7.5.8*. URL: https://docs.kasten.io/latest/install/storage.html (visited on 03/28/2025).

[58] "NIST SP 800-34". In: *NIST* (Jan. 12, 2020). Last Modified: 2021-04-23T09:21-04:00. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf (visited on 03/28/2025).

[59] RIA. *E-ITS English*. URL: https://eits.ria.ee/api/2/main_menu/asset/2023_eits_english.pdf (visited on 02/08/2025).

[60] *Init Containers*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/workloads/pods/init-containers/ (visited on 03/28/2025).

[61] *Cluster Networking*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/cluster-administration/networking/ (visited on 03/28/2025).

[62] *Network Policies*. en. Section: docs. Nov. 2024. URL: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (visited on 10/13/2024).

[63] *CIS Control 12: Network Infrastructure Management - CIS Controls Assessment Specification for Controls v8.1*. URL: https://cas.docs.cisecurity.org/en/latest/source/Controls12/ (visited on 03/28/2025).

[64] *Configuration Best Practices*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/configuration/overview/ (visited on 03/28/2025).

[65] *GitOps with Kubernetes: Why It's Different and How to Adopt It*. Codefresh. URL: https://codefresh.io/learn/gitops/gitops-with-kubernetes-why-its-different-and-how-to-adopt-it/ (visited on 03/28/2025).

[66] *Configure Service Accounts for Pods*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/ (visited on 03/28/2025).

[67] Joint Task Force. *Security and Privacy Controls for Information Systems and Organizations*. NIST Special Publication (SP) 800-53 Rev. 5. National Institute of Standards and Technology, Dec. 10, 2020. DOI: 10.6028/NIST.SP.800-53r5. URL: https://doi.org/10.6028/NIST.SP.800-53r5 (visited on 03/29/2025).

[68] *Pod Lifecycle*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/ (visited on 03/29/2025).

[69] *Configure Liveness, Readiness and Startup Probes*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/ (visited on 03/29/2025).

[70] *Liveness, Readiness, and Startup Probes*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/configuration/liveness-readiness-startup-probes/ (visited on 03/29/2025).

[71] *Container Lifecycle Hooks*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/ (visited on 03/29/2025).

[72] *Auditing*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/ (visited on 03/29/2025).

[73] *Kubernetes logging best practices*. CNCF. July 3, 2023. URL: https://www.cncf.io/blog/2023/07/03/kubernetes-logging-best-practices/ (visited on 03/30/2025).

[74] *Kube-bench*. URL: https://aquasecurity.github.io/kube-bench/v0.6.15/ (visited on 11/17/2024).

[75] *kubescape/kubescape*. original-date: 2021-08-12T10:39:29Z. Mar. 31, 2025. URL: https://github.com/kubescape/kubescape (visited on 03/31/2025).

[76] *Argo CD - Declarative GitOps CD for Kubernetes*. URL: https://argo-cd.readthedocs.io/en/stable/ (visited on 03/31/2025).

[77] *Flux - the GitOps family of projects*. URL: https://fluxcd.io/ (visited on 03/31/2025).

[78] *Assigning Pods to Nodes*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/ (visited on 03/31/2025).

[79] *Operaptor pattern*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (visited on 03/31/2025).

[80] *OperatorHub.io | The registry for Kubernetes Operators*. URL: https://operatorhub.io/ (visited on 03/31/2025).

[81] *Securing a Cluster*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/ (visited on 03/31/2025).

[82] *Certificate Management with kubeadm*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-certs/ (visited on 03/31/2025).

[83] *Use service accounts rules in policy | Calico Documentation*. URL: https://docs.tigera.io/calico/latest/network-policy/policy-rules/service-accounts (visited on 03/31/2025).

[84] *Using Kubernetes Constructs In Policy — Cilium 1.18.0-dev documentation*. URL: https://docs.cilium.io/en/latest/security/policy/kubernetes/ (visited on 03/31/2025).

[85] *Creating Highly Available Clusters with kubeadm*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/ (visited on 04/01/2025).

[86] *Options for Highly Available Topology*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/ (visited on 04/01/2025).

[87] *Encrypting Confidential Data at Rest*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/ (visited on 04/01/2025).

[88] *Introduction - The Cluster API Book*. URL: https://cluster-api.sigs.k8s.io/ (visited on 04/04/2025).

[89] *Introduction - The Cluster API Provider Azure Book*. URL: https://capz.sigs.k8s.io/ (visited on 04/04/2025).

[90] Craig Peters Francis Jack. *Introducing the Cluster API Provider for Azure (CAPZ) for Kubernetes cluster management*. Microsoft Open Source Blog. Dec. 15, 2020. URL: https://opensource.microsoft.com/blog/2020/12/15/introducing-cluster-api-provider-azure-capz-kubernetes-cluster-management/ (visited on 04/18/2025).

[91] *Azure Service Operator v2*. Azure Service Operator. URL: https://azure.github.io/azure-service-operator/ (visited on 04/18/2025).

[92] *Recommended Labels*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/ (visited on 03/26/2025).

[93] *Hardware recommendations*. etcd. Section: docs. URL: https://etcd.io/docs/v3.3/op-guide/hardware/ (visited on 04/04/2025).

[94] mattmcinnes. *Virtual machine sizes overview - Azure Virtual Machines*. Nov. 19, 2024. URL: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/overview (visited on 04/04/2025).

[95] *Autoscaling - The Cluster API Book*. URL: https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling.html?highlight=autoscaler#using-the-cluster-autoscaler (visited on 04/18/2025).

[96] *SSH Access to nodes - The Cluster API Provider Azure Book*. URL: https://capz.sigs.k8s.io/self-managed/ssh-access.html?highlight=bastion#azure-bastion (visited on 04/04/2025).

[97] cherylmc. *Connect to a VM using Bastion - Windows native client - Azure Bastion*. Jan. 28, 2025. URL: https://learn.microsoft.com/en-us/azure/bastion/connect-vm-native-client-windows (visited on 04/04/2025).

[98] *v1beta1 API - The Cluster API Provider Azure Book*. URL: https://capz.sigs.k8s.io/reference/v1beta1-api (visited on 04/04/2025).

[99] cherylmc. *About Azure Bastion configuration settings*. Mar. 14, 2025. URL: https://learn.microsoft.com/en-us/azure/bastion/configuration-settings (visited on 04/04/2025).

[100] *AKS as management cluster - The Cluster API Provider Azure Book*. URL: https://capz.sigs.k8s.io/developers/tilt-with-aks-as-mgmt-ilb.html?highlight=ports#challenges-and-solutions (visited on 04/04/2025).

[101] *Ports and Protocols*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/reference/networking/ports-and-protocols/ (visited on 04/04/2025).

[102] *Project Metrics*. CNCF. URL: https://www.cncf.io/project-metrics/ (visited on 04/06/2025).

[103] *Security Compliance & Certifications for 24.04 | Ubuntu*. URL: https://ubuntu.com/security/certifications/docs/2404 (visited on 05/05/2025).

[104] *Getting Started - The Cluster API Provider Azure Book*. URL: https://capz.sigs.k8s.io/getting-started.html?highlight=contributor#setting-up-your-azure-environment (visited on 05/06/2025).

[105] *Adopt a zero trust network model for security | Calico Documentation*. URL: https://docs.tigera.io/calico/latest/network-policy/adopt-zero-trust (visited on 04/05/2025).

[106] *Workload bootstrap using GitOps - The Cluster API Book*. URL: https://cluster-api.sigs.k8s.io/tasks/workload-bootstrap-gitops.html#bootstrap-managedcluster-using-argocd (visited on 04/05/2025).

[107] *cluster-api-provider-azure/templates/addons/cluster-api-helm/cloud-provider-azure.yaml at main · kubernetes-sigs/cluster-api-provider-azure*. GitHub. URL: https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/main/templates/addons/cluster-api-helm/cloud-provider-azure.yaml (visited on 04/06/2025).

[108] *Cluster Bootstrapping - Argo CD - Declarative GitOps CD for Kubernetes*. URL: https://argo-cd.readthedocs.io/en/latest/operator-manual/cluster-bootstrapping/ (visited on 04/05/2025).

[109] Marko Lindeberg. *E-ITS technical implementation*. URL: https://github.com/mlndbr6/e-its-kubernetes.

[110] *Argo CD vs. Flux: 6 Key Differences and How to Choose*. Codefresh. URL: https://codefresh.io/learn/argo-cd/argo-cd-vs-flux-6-key-differences-and-how-to-choose/ (visited on 04/18/2025).

[111] *Overview - Argo CD - Declarative GitOps CD for Kubernetes*. URL: https://argo-cd.readthedocs.io/en/stable/operator-manual/security/ (visited on 04/18/2025).

[112] *bitnami-labs/sealed-secrets: A Kubernetes controller and tool for one-way encrypted Secrets*. URL: https://github.com/bitnami-labs/sealed-secrets (visited on 04/19/2025).

[113] *Introduction - External Secrets Operator*. URL: https://external-secrets.io/latest/ (visited on 04/19/2025).

[114] *Introduction - Secrets Store CSI Driver*. URL: https://secrets-store-csi-driver.sigs.k8s.io/ (visited on 04/19/2025).

[115] *external-dns*. URL: https://kubernetes-sigs.github.io/external-dns/latest/ (visited on 05/01/2025).

[116] *cert-manager*. cert-manager. URL: https://cert-manager.io/ (visited on 05/01/2025).

[117] *Certbot*. URL: https://eff-certbot.readthedocs.io/en/stable/using.html (visited on 05/01/2025).

[118] *Vault | HashiCorp Developer*. Vault | HashiCorp Developer. URL: https://developer.hashicorp.com/vault (visited on 05/01/2025).

[119] *Issue with Handling Multiple DNS Challenges in Azure DNS (TXT Record Concatenation) · Issue #51 · terricain/certbot-dns-azure*. URL: https://github.com/terricain/certbot-dns-azure/issues/51 (visited on 05/01/2025).

[120] *DNS01*. cert-manager. URL: https://cert-manager.io/docs/configuration/acme/dns01/#supported-dns01-providers (visited on 05/01/2025).

[121] *Ingress Controllers*. Kubernetes. Section: docs. URL: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/ (visited on 05/03/2025).

[122] *CNCF Landscape*. URL: https://landscape.cncf.io/ (visited on 05/03/2025).

[123] Guest Author. *Benchmarking 5 Popular Load Balancers: Nginx, HAProxy, Envoy, Traefik, and ALB | Loggly*. Log Analysis | Log Monitoring by Loggly. Dec. 10, 2018. URL: https://www.loggly.com/blog/benchmarking-5-popular-load-balancers-nginx-haproxy-envoy-traefik-and-alb/ (visited on 05/03/2025).

[124] *haproxytech/ingress-controller-benchmarks*. original-date: 2020-08-17T06:18:15Z. Apr. 3, 2025. URL: https://github.com/haproxytech/ingress-controller-benchmarks (visited on 05/03/2025).

[125] *Testing the Performance of NGINX Ingress Controller for Kubernetes – NGINX Community Blog*. Apr. 11, 2019. URL: https://blog.nginx.org/blog/testing-performance-nginx-ingress-controller-kubernetes (visited on 05/03/2025).

[126] *Benchmarking Linkerd and Istio*. Linkerd. Section: blog. May 27, 2021. URL: https://linkerd.io/2021/05/27/linkerd-vs-istio-benchmarks/ (visited on 05/03/2025).

[127] Emre Çalışkan. *Is Nginx dead? Is Traefik v3 20% faster than Traefik v2?* Beyn Technology. Apr. 12, 2023. URL: https://medium.com/beyn-technology/is-nginx-dead-is-traefik-v3-20-faster-than-traefik-v2-f28ffb7eed3e (visited on 05/03/2025).

[128] *gaplo917/load-balancer-benchmark: Apache Httpd vs Nginx vs Traefik vs HAProxy*. URL: https://github.com/gaplo917/load-balancer-benchmark (visited on 05/03/2025).

[129] Gary Lo. *gaplo917/load-balancer-benchmark*. original-date: 2020-02-08T11:47:50Z. Apr. 8, 2025. URL: https://github.com/gaplo917/load-balancer-benchmark (visited on 05/03/2025).

[130] 2 Minute Read Page Test4. *Platform Requirements*. Istio. URL: https://istio.io/latest/docs/ops/deployment/platform-requirements/ (visited on 05/03/2025).

[131] *Felix configuration | Calico Documentation*. URL: https://docs.tigera.io/calico/latest/reference/resources/felixconfig (visited on 05/03/2025).

[132] *Free-Tier Pricing | Get More Observability For Less*. URL: https://newrelic.com/pricing/free-tier (visited on 05/03/2025).

[133] *VictoriaMetrics*. URL: https://docs.victoriametrics.com/victoriametrics/ (visited on 05/03/2025).

[134] *helm-charts/charts/kube-prometheus-stack at main · prometheus-community/helm-charts*. GitHub. URL: https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack (visited on 05/03/2025).

[135] *Velero*. URL: https://velero.io/ (visited on 05/04/2025).

[136] *Free Kubernetes: Start Your Journey with Veeam Kasten for Kubernetes*. Veeam Software. URL: https://www.veeam.com/products/free/kubernetes.html?ck=1731487151821 (visited on 05/04/2025).

[137] AppsCode Inc. *KubeStash - Backup and Recovery Solution for Kubernetes*. URL: https://kubestash.com/ (visited on 05/04/2025).

[138] *Backup types by driver - Portworx Documentation*. May 2, 2025. URL: https://docs.portworx.com/portworx-backup-on-prem/concepts/backup-types (visited on 05/04/2025).

[139] Maxim Levchenko. *WoozyMasta/kube-dump*. original-date: 2021-02-03T00:53:49Z. Apr. 16, 2025. URL: https://github.com/WoozyMasta/kube-dump (visited on 05/04/2025).

[140] *Kubescape*. URL: https://kubescape.io/ (visited on 01/16/2025).

[141] *aquasecurity/trivy: Find vulnerabilities, misconfigurations, secrets, SBOM in containers, Kubernetes, code repositories, clouds and more*. URL: https://github.com/aquasecurity/trivy (visited on 05/10/2025).

[142] *kube-prometheus security*. GitHub. URL: https://github.com/prometheus-operator/kube-prometheus/security/policy (visited on 05/06/2025).

[143] *E-ITS*. URL: https://eits.ria.ee/et/abimaterjalid/rakendusjuhend (visited on 05/07/2025).

[144] *postfinance/kubelet-csr-approver*. original-date: 2021-11-19T16:17:46Z. May 6, 2025. URL: https://github.com/postfinance/kubelet-csr-approver (visited on 05/07/2025).

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

I Marko Lindeberg

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Analysis and Implementation of 'APP.4.4: Kubernetes' from the Estonian Information Security Standard (E-ITS)", supervised by Siim Vene
   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.

18.05.2025

[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 – Secret encryption verification in etcd

```
root@capi-ha-control-plane-j5v62:~#
ETCDL_API=3 etcdctl \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
get /registry/secrets/external-dns/azure-sp-secret \
| hexdump -C | awk '{print $18}' | head -n 5

|/registry/secret|
|s/external-dns/a|
|zure-sp-secret.k|
|8s:enc:secretbox|
|:v1:key1:.Q..!.~|
```

Appendix 3 – Velero role definition

```
{
"Name": "Velero",
  "Description": "Velero permissions for backups, restores and deletions",
"Actions": [
  "Microsoft.Compute/disks/read",
  "Microsoft.Compute/disks/write",
  "Microsoft.Compute/disks/endGetAccess/action",
  "Microsoft.Compute/disks/beginGetAccess/action",
  "Microsoft.Compute/snapshots/read",
  "Microsoft.Compute/snapshots/write",
  "Microsoft.Compute/snapshots/delete",
  "Microsoft.Storage/storageAccounts/listkeys/action",
  "Microsoft.Storage/storageAccounts/regeneratekey/action",
  "Microsoft.Storage/storageAccounts/read",
  "Microsoft.Storage/storageAccounts/blobServices/containers/delete",
  "Microsoft.Storage/storageAccounts/blobServices/containers/read",
  "Microsoft.Storage/storageAccounts/blobServices/containers/write",
  "Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey/action",
],
"DataActions" :[
  "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete",
  "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read",
  "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write",
  "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/move/action",
  "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action"
],
"AssignableScopes": ["/subscriptions/$AZURE_SUBSCRIPTION_ID"]
}
```

Appendix 4 – kube-bench test results

```
root@capi-ha-control-plane-h692g:~/kube-bench# kubectl logs -n kube-bench kube-bench-
    master-n4vc4
[INFO] 1 Control Plane Security Configuration
[INFO] 1.1 Control Plane Node Configuration Files
[PASS] 1.1.1 Ensure that the API server pod specification file permissions are set to
    600 or more restrictive (Automated)
[PASS] 1.1.2 Ensure that the API server pod specification file ownership is set to root:
    root (Automated)
[PASS] 1.1.3 Ensure that the controller manager pod specification file permissions are
    set to 600 or more restrictive (Automated)
[PASS] 1.1.4 Ensure that the controller manager pod specification file ownership is set
    to root:root (Automated)
[PASS] 1.1.5 Ensure that the scheduler pod specification file permissions are set to 600
     or more restrictive (Automated)
[PASS] 1.1.6 Ensure that the scheduler pod specification file ownership is set to root:
    root (Automated)
[PASS] 1.1.7 Ensure that the etcd pod specification file permissions are set to 600 or
    more restrictive (Automated)
[PASS] 1.1.8 Ensure that the etcd pod specification file ownership is set to root:root (
    Automated)
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set to 600
     or more restrictive (Manual)
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to root:
    root (Manual)
[PASS] 1.1.11 Ensure that the etcd data directory permissions are set to 700 or more
    restrictive (Automated)
[FAIL] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (
    Automated)
[PASS] 1.1.13 Ensure that the default administrative credential file permissions are set
     to 600 (Automated)
[PASS] 1.1.14 Ensure that the default administrative credential file ownership is set to
     root:root (Automated)
[PASS] 1.1.15 Ensure that the scheduler.conf file permissions are set to 600 or more
    restrictive (Automated)
[PASS] 1.1.16 Ensure that the scheduler.conf file ownership is set to root:root (
    Automated)
[PASS] 1.1.17 Ensure that the controller-manager.conf file permissions are set to 600 or
     more restrictive (Automated)
[PASS] 1.1.18 Ensure that the controller-manager.conf file ownership is set to root:root
     (Automated)
[PASS] 1.1.19 Ensure that the Kubernetes PKI directory and file ownership is set to root
    :root (Automated)
[PASS] 1.1.21 Ensure that the Kubernetes PKI key file permissions are set to 600 (Manual
    )
[INFO] 1.2 API Server
[PASS] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)
[PASS] 1.2.2 Ensure that the --token-auth-file parameter is not set (Automated)
[PASS] 1.2.3 Ensure that the --DenyServiceExternalIPs is set (Manual)
[PASS] 1.2.4 Ensure that the --kubelet-client-certificate and --kubelet-client-key
    arguments are set as appropriate (Automated)
[PASS] 1.2.5 Ensure that the --kubelet-certificate-authority argument is set as
    appropriate (Automated)
[PASS] 1.2.6 Ensure that the --authorization-mode argument is not set to AlwaysAllow (
```

```
    Automated)
[PASS] 1.2.7 Ensure that the --authorization-mode argument includes Node (Automated)
[PASS] 1.2.8 Ensure that the --authorization-mode argument includes RBAC (Automated)
[PASS] 1.2.9 Ensure that the admission control plugin EventRateLimit is set (Manual)
[PASS] 1.2.10 Ensure that the admission control plugin AlwaysAdmit is not set (Automated
    )
[PASS] 1.2.11 Ensure that the admission control plugin AlwaysPullImages is set (Manual)
[PASS] 1.2.12 Ensure that the admission control plugin ServiceAccount is set (Automated)
[PASS] 1.2.13 Ensure that the admission control plugin NamespaceLifecycle is set (
    Automated)
[PASS] 1.2.14 Ensure that the admission control plugin NodeRestriction is set (Automated
    )
[PASS] 1.2.15 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.2.16 Ensure that the --audit-log-path argument is set (Automated)
[PASS] 1.2.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate
     (Automated)
[PASS] 1.2.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as
    appropriate (Automated)
[PASS] 1.2.19 Ensure that the --audit-log-maxsize argument is set to 100 or as
    appropriate (Automated)
[WARN] 1.2.20 Ensure that the --request-timeout argument is set as appropriate (Manual)
[PASS] 1.2.21 Ensure that the --service-account-lookup argument is set to true (
    Automated)
[PASS] 1.2.22 Ensure that the --service-account-key-file argument is set as appropriate
    (Automated)
[PASS] 1.2.23 Ensure that the --etcd-certfile and --etcd-keyfile arguments are set as
    appropriate (Automated)
[PASS] 1.2.24 Ensure that the --tls-cert-file and --tls-private-key-file arguments are
    set as appropriate (Automated)
[PASS] 1.2.25 Ensure that the --client-ca-file argument is set as appropriate (Automated
    )
[PASS] 1.2.26 Ensure that the --etcd-cafile argument is set as appropriate (Automated)
[PASS] 1.2.27 Ensure that the --encryption-provider-config argument is set as
    appropriate (Manual)
[PASS] 1.2.28 Ensure that encryption providers are appropriately configured (Manual)
[PASS] 1.2.29 Ensure that the API Server only makes use of Strong Cryptographic Ciphers
    (Manual)
[INFO] 1.3 Controller Manager
[PASS] 1.3.1 Ensure that the --terminated-pod-gc-threshold argument is set as
    appropriate (Manual)
[PASS] 1.3.2 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.3.3 Ensure that the --use-service-account-credentials argument is set to true (
    Automated)
[PASS] 1.3.4 Ensure that the --service-account-private-key-file argument is set as
    appropriate (Automated)
[PASS] 1.3.5 Ensure that the --root-ca-file argument is set as appropriate (Automated)
[PASS] 1.3.6 Ensure that the RotateKubeletServerCertificate argument is set to true (
    Automated)
[PASS] 1.3.7 Ensure that the --bind-address argument is set to 127.0.0.1 (Automated)
[INFO] 1.4 Scheduler
[PASS] 1.4.1 Ensure that the --profiling argument is set to false (Automated)
[PASS] 1.4.2 Ensure that the --bind-address argument is set to 127.0.0.1 (Automated)

== Summary master ==
53 checks PASS
1 checks FAIL
5 checks WARN
0 checks INFO
```

```
[INFO] 2 Etcd Node Configuration
[INFO] 2 Etcd Node Configuration
[PASS] 2.1 Ensure that the --cert-file and --key-file arguments are set as appropriate (
    Automated)
[PASS] 2.2 Ensure that the --client-cert-auth argument is set to true (Automated)
[PASS] 2.3 Ensure that the --auto-tls argument is not set to true (Automated)
[PASS] 2.4 Ensure that the --peer-cert-file and --peer-key-file arguments are set as
    appropriate (Automated)
[PASS] 2.5 Ensure that the --peer-client-cert-auth argument is set to true (Automated)
[PASS] 2.6 Ensure that the --peer-auto-tls argument is not set to true (Automated)
[PASS] 2.7 Ensure that a unique Certificate Authority is used for etcd (Manual)

== Summary etcd ==
7 checks PASS
0 checks FAIL
0 checks WARN
0 checks INFO

[INFO] 3 Control Plane Configuration
[INFO] 3.1 Authentication and Authorization
[WARN] 3.1.1 Client certificate authentication should not be used for users (Manual)
[WARN] 3.1.2 Service account token authentication should not be used for users (Manual)
[WARN] 3.1.3 Bootstrap token authentication should not be used for users (Manual)
[INFO] 3.2 Logging
[PASS] 3.2.1 Ensure that a minimal audit policy is created (Manual)
[WARN] 3.2.2 Ensure that the audit policy covers key security concerns (Manual)

== Summary controlplane ==
1 checks PASS
0 checks FAIL
4 checks WARN
0 checks INFO

[INFO] 4 Worker Node Security Configuration
[INFO] 4.1 Worker Node Configuration Files
[FAIL] 4.1.1 Ensure that the kubelet service file permissions are set to 600 or more
    restrictive (Automated)
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root:root (
    Automated)
[WARN] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 600 or more
    restrictive (Manual)
[WARN] 4.1.4 If proxy kubeconfig file exists ensure ownership is set to root:root (
    Manual)
[PASS] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are set to 600
    or more restrictive (Automated)
[PASS] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set to root:
    root (Automated)
[WARN] 4.1.7 Ensure that the certificate authorities file permissions are set to 600 or
    more restrictive (Manual)
[PASS] 4.1.8 Ensure that the client certificate authorities file ownership is set to
    root:root (Manual)
[PASS] 4.1.9 If the kubelet config.yaml configuration file is being used validate
    permissions set to 600 or more restrictive (Automated)
[PASS] 4.1.10 If the kubelet config.yaml configuration file is being used validate file
    ownership is set to root:root (Automated)
[INFO] 4.2 Kubelet
[PASS] 4.2.1 Ensure that the --anonymous-auth argument is set to false (Automated)
[PASS] 4.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow (
    Automated)
```

```
[PASS] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate (Automated)
[PASS] 4.2.4 Verify that the --read-only-port argument is set to 0 (Manual)
[PASS] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to
    0 (Manual)
[PASS] 4.2.6 Ensure that the --make-iptables-util-chains argument is set to true (
    Automated)
[PASS] 4.2.7 Ensure that the --hostname-override argument is not set (Manual)
[PASS] 4.2.8 Ensure that the eventRecordQPS argument is set to a level which ensures
    appropriate event capture (Manual)
[PASS] 4.2.9 Ensure that the --tls-cert-file and --tls-private-key-file arguments are
    set as appropriate (Manual)
[PASS] 4.2.10 Ensure that the --rotate-certificates argument is not set to false (
    Automated)
[PASS] 4.2.11 Verify that the RotateKubeletServerCertificate argument is set to true (
    Manual)
[PASS] 4.2.12 Ensure that the Kubelet only makes use of Strong Cryptographic Ciphers (
    Manual)
[PASS] 4.2.13 Ensure that a limit is set on pod PIDs (Manual)
[INFO] 4.3 kube-proxy
[PASS] 4.3.1 Ensure that the kube-proxy metrics service is bound to localhost (Automated
    )

== Summary node ==
21 checks PASS
1 checks FAIL
2 checks WARN
0 checks INFO

== Summary total ==
77 checks PASS
3 checks FAIL
15 checks WARN
0 checks INFO
```

Appendix 5 – kube-bench cluster component manual verification steps.

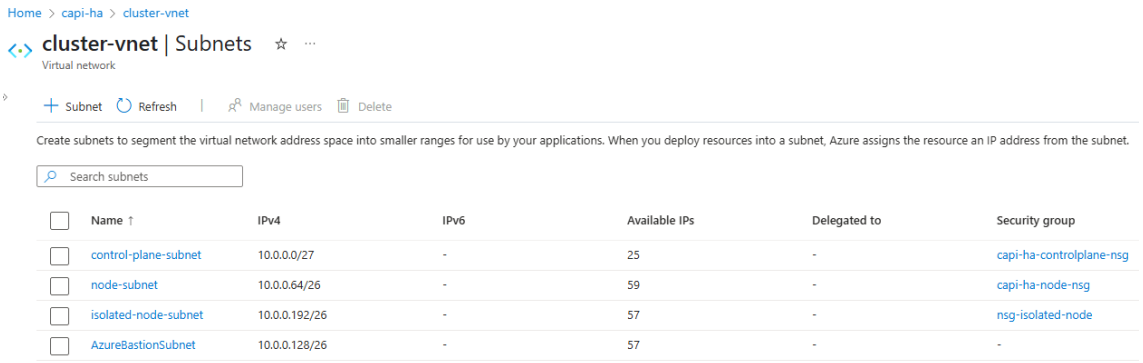
```
# Manual validation shows correct permissions for Container Network Interface files:
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set to 600
    ↪  or more restrictive (Manual)
root@capi-ha-control-plane-h692g:~/kube-bench# stat -c %a /etc/cni/net.d/
600

# Manual validation shows correct ownership of Container Interface files:
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to root:
    ↪ root (Manual)
root@capi-ha-control-plane-h692g:~/kube-bench# stat -c %U:%G /etc/cni/net.d/
root:root

# Manual validation shows correct permissions for PKI certificates as expected:
[WARN] 1.1.20 Ensure that the Kubernetes PKI certificate file permissions are set to 600
    ↪  or more restrictive (Manual)
root@capi-ha-control-plane-h692g:~/kube-bench# stat -c '%a' /etc/kubernetes/pki/*.crt
600
600
600
600
600
600

# Anonymous auth is enabled on Kubernetes API server, as it is generally considered
    ↪ reasonable as long as RBAC authorization is enabled.
# Additionally, this isn't explicitly required by E-ITS, as long as the administrative
    ↪ operations are performed by approved users.
[WARN] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)

# It is set to 120s in KubeadmControlPlane configuration
[WARN] 1.2.20 Ensure that the --request-timeout argument is set as appropriate (Manual)
root@capi-ha-control-plane-ph7w5:~# cat /etc/kubernetes/manifests/kube-apiserver.yaml |
    ↪ grep request-timeout
- --request-timeout=120s

# 3.1.1-3.1.3 are not used for users in this cluster. Access to the cluster is through
    ↪ Bastion host
# with SSH, and ~/.kube/config file is used for authentication.
[WARN] 3.1.1 Client certificate authentication should not be used for users (Manual)
[WARN] 3.1.2 Service account token authentication should not be used for users (Manual)
[WARN] 3.1.3 Bootstrap token authentication should not be used for users (Manual)

# This is covered in audit logging configuration configuration file in
    ↪ KubeadmControlPlane configuration.
[WARN] 3.2.2 Ensure that the audit policy covers key security concerns (Manual)

# This is not applicable as kube-proxy kubeconfig parameters are configured as
    ↪ Kubernetes ConfigMap.
[WARN] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 600 or more
    ↪ restrictive (Manual)
[WARN] 4.1.4 If proxy kubeconfig file exists ensure ownership is set to root:root (
```

```
    ↪ Manual)

# Manually validated that the permissions are set to 600.
[WARN] 4.1.7 Ensure that the certificate authorities file permissions are set to 600 or
    ↪ more restrictive (Manual)
root@capi-ha-control-plane-h692g:~/kube-bench# stat -c %a /etc/kubernetes/pki/ca.crt
600
```

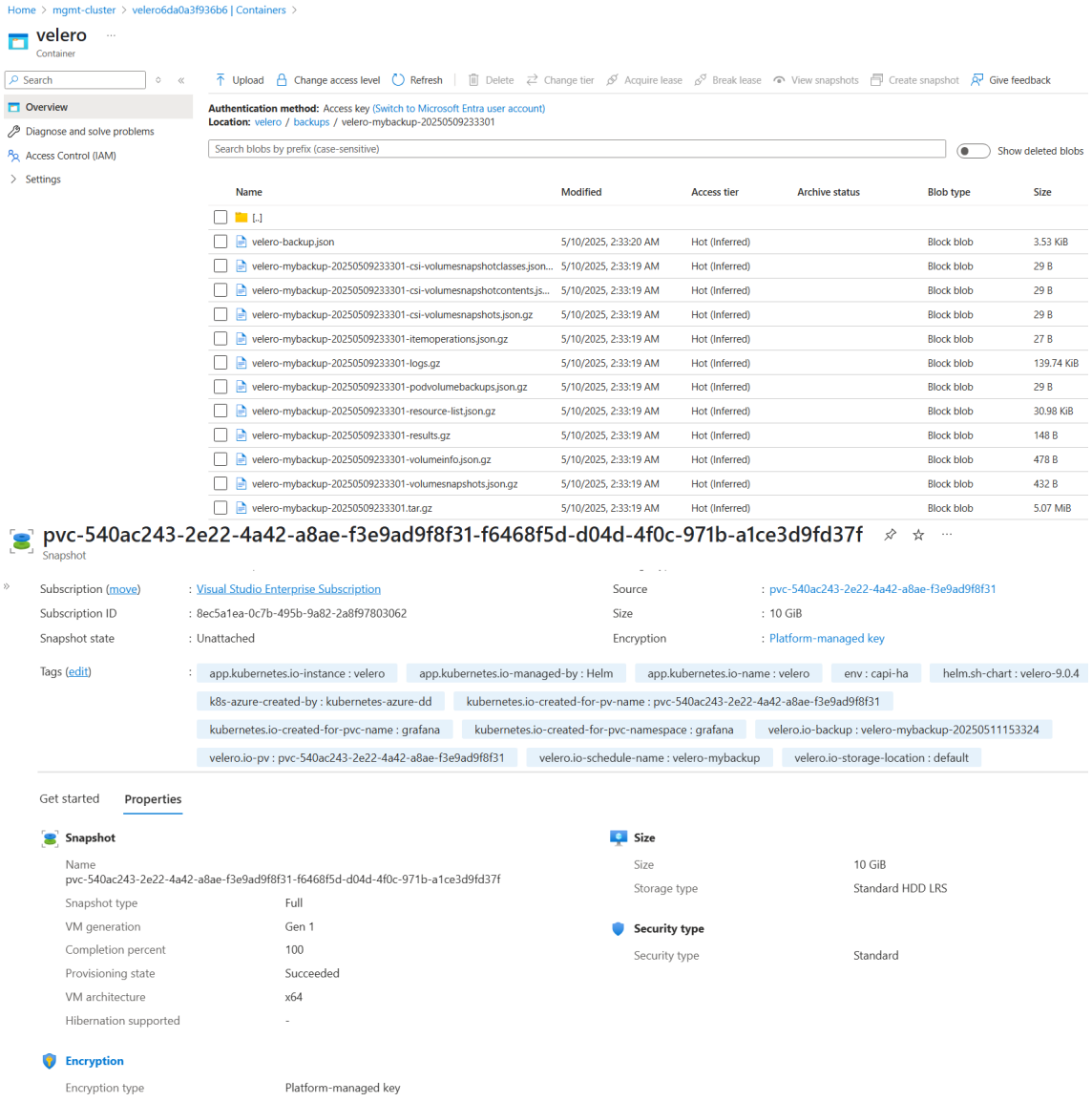
Appendix 6 – Kubelet cgroup configuration and namespace isolation.

```
root@capi-ha-control-plane-ph7w5:~# stat -fc %T /sys/fs/cgroup/
cgroup2fs
root@capi-ha-control-plane-ph7w5:~# cat /var/lib/kubelet/config.yaml | grep cgroup
cgroupDriver: systemd

root@capi-ha-control-plane-ph7w5:~# crictl ps --name=alloy -q
8afd1cf8a6737649f225584f200357cfe154d565455160463ac09870f1252bd6
root@capi-ha-control-plane-ph7w5:~# crictl inspect --output go-template --template '{{.
    ↪ info.pid}}' 8afd1cf8a6737649f225584f200357cfe154d565455160463ac09870f1
252bd6
29620
root@capi-ha-control-plane-ph7w5:~# ls -l /proc/29620/ns
total 0
lrwxrwxrwx 1 root root 0 May 11 20:45 cgroup -> 'cgroup:[4026532821]'
lrwxrwxrwx 1 root root 0 May 11 20:45 ipc -> 'ipc:[4026532817]'
lrwxrwxrwx 1 root root 0 May 11 20:45 mnt -> 'mnt:[4026532819]'
lrwxrwxrwx 1 root root 0 May 11 20:45 net -> 'net:[4026532746]'
lrwxrwxrwx 1 root root 0 May 11 20:45 pid -> 'pid:[4026532820]'
lrwxrwxrwx 1 root root 0 May 11 20:45 pid_for_children -> 'pid:[4026532820]'
lrwxrwxrwx 1 root root 0 May 11 20:45 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 May 11 20:45 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 May 11 20:45 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 May 11 20:45 uts -> 'uts:[4026532816]'
```

Appendix 7 – Azure Bastion (Management), control plane node, isolated node and other data plane node subnets.



Appendix 8 – Velero backup validation in Azure portal.



Appendix 9 – Disabled automount for default service accounts.

```
NAME                            ADMISSION   BACKGROUND   READY   AGE    MESSAGE
restrict-default-sa-automount   true        true         True    3h1m   Ready

root@capi-ha-control-plane-ph7w5:~# kubectl get sa -A -o=jsonpath='{range .items[*]}{@.
    ↪ metadata.name}:{@.metadata.namespace}:{@..automountServiceAccountToke
n}\n{end}' | grep "default:"
default:alloy:false
default:argocd:false
default:calico-apiserver:false
default:calico-system:false
default:cert-manager:false
default:default:false
default:external-dns:false
default:external-secrets:false
default:grafana:false
default:haproxy-controller:false
default:istio-system:false
default:kiali:false
default:kube-node-lease:false
default:kube-prometheus-stack:false
default:kube-public:false
default:kube-system:false
default:kyverno:false
default:loki:false
default:test-quotas:false
default:test:false
default:tigera-operator:false
default:velero:false
```

Appendix 10 – Kubernetes health probe requirement validation.

```
root@capi-ha-control-plane-ph7w5:~# kubectl get policyreports.wgpolicyk8s.io -n test-
    quotas 845b7b1f-344b-42db-a478-c1c50b7bad99 -o yaml
apiVersion: wgpolicyk8s.io/v1alpha2
kind: PolicyReport
metadata:
  creationTimestamp: "2025-01-11T23:29:15Z"
  generation: 4
  labels:
    app.kubernetes.io/managed-by: kyverno
  name: 845b7b1f-344b-42db-a478-c1c50b7bad99
  namespace: test-quotas
  ownerReferences:
  - apiVersion: v1
    kind: Pod
    name: tmp-shell
    uid: 845b7b1f-344b-42db-a478-c1c50b7bad99
  resourceVersion: "272574"
  uid: c335d246-0be9-4f50-bca8-0232022ec6fe
results:
- category: Multi-Tenancy
  message: validation rule 'validate-namespace' passed.
  policy: disallow-default-namespace
  properties:
    process: background scan
  result: pass
  rule: validate-namespace
  scored: true
  severity: medium
  source: kyverno
  timestamp:
    nanos: 0
    seconds: 1747007156
- category: Best Practices
  message: 'validation failure: Liveness, readiness, or startup probes are required
    for all containers.'
  policy: require-pod-probes
  properties:
    process: background scan
  result: fail
  rule: validate-probes
  scored: true
  severity: medium
  source: kyverno
  timestamp:
    nanos: 0
    seconds: 1747007156
- message: validation rule 'validate-resources' passed.
  policy: require-requests-limits
  properties:
    process: background scan
  result: pass
```

```
  rule: validate-resources
  scored: true
  severity: medium
  source: kyverno
  timestamp:
    nanos: 0
    seconds: 1747007156
scope:
  apiVersion: v1
  kind: Pod
  name: tmp-shell
  namespace: test-quotas
  uid: 845b7b1f-344b-42db-a478-c1c50b7bad99
summary:
  error: 0
  fail: 1
  pass: 2
  skip: 0
  warn: 0
```

Appendix 11 – Specialised Kubernetes nodes.

```
root@capi-ha-control-plane-tx9fg:~# kubectl get nodes
NAME                           STATUS   ROLES           AGE     VERSION
capi-ha-control-plane-krkpk    Ready    control-plane   4m51s   v1.32.0
capi-ha-control-plane-szk82    Ready    control-plane   107s    v1.32.0
capi-ha-control-plane-tx9fg    Ready    control-plane   9m      v1.32.0
capi-ha-general000000          Ready    <none>          7m37s   v1.32.0
capi-ha-general000001          Ready    <none>          7m31s   v1.32.0
capi-ha-ingress000000          Ready    <none>          7m30s   v1.32.0
capi-ha-ingress000001          Ready    <none>          7m32s   v1.32.0
capi-ha-isolated000000         Ready    <none>          7m39s   v1.32.0
capi-ha-isolated000001         Ready    <none>          7m33s   v1.32.0
capi-ha-monitoring000000       Ready    <none>          7m36s   v1.32.0
capi-ha-monitoring000001       Ready    <none>          7m36s   v1.32.0
capi-ha-storage000000          Ready    <none>          7m38s   v1.32.0
capi-ha-storage000001          Ready    <none>          7m37s   v1.32.0
```

Appendix 12 – PEARO principle validation.

Table 5. PEARO principle validation

Measure	**Implementation**	**Comment**
APP.4.4.M1	R	
APP.4.4.M2	R	
APP.4.4.M3	R	
APP.4.4.M4	R	
APP.4.4.M5	R	
APP.4.4.M6	R	*initContainers* used where needed
APP.4.4.M7	R	
APP.4.4.M8	R	
APP.4.4.M9	R	
APP.4.4.M10	R	
APP.4.4.M11	R	In 'Audit' mode
APP.4.4.M12	R	
APP.4.4.M13	R	
APP.4.4.M14	R	
APP.4.4.M15	R	
APP.4.4.M16	R	
APP.4.4.M17	R	
APP.4.4.M18	R	
APP.4.4.M19	R	
APP.4.4.M20	R	
APP.4.4.M21	R	CronJob created, although usually applications are managed by operators and Kubernetes itself

Appendix 13 – Provisioned resources in Azure Portal.

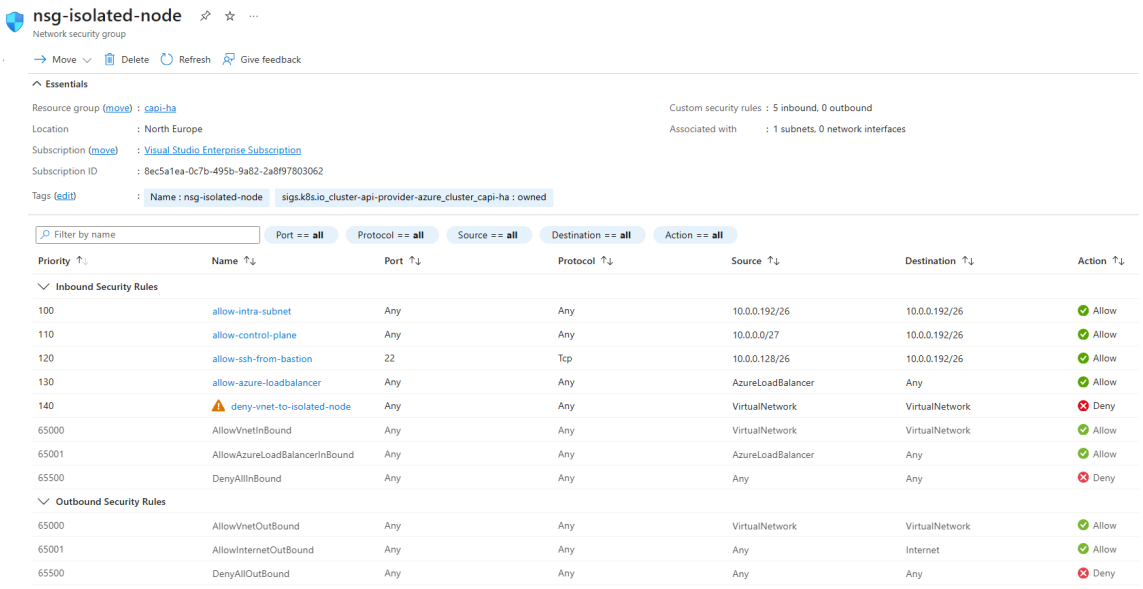
Showing 1 to 37 of 37 records. Show hidden types

No grouping

Name	Type	Resource group
cluster-vnet-link (lndbrg.tech/cluster-vnet-link)	Virtual network link	capi-ha
mgmt-cluster-link (lndbrg.tech/mgmt-cluster-link)	Virtual network link	capi-ha
cluster-vnet	Virtual network	capi-ha
capi-ha-general	Virtual machine scale set	capi-ha
capi-ha-ingress	Virtual machine scale set	capi-ha
capi-ha-isolated	Virtual machine scale set	capi-ha
capi-ha-monitoring	Virtual machine scale set	capi-ha
capi-ha-storage	Virtual machine scale set	capi-ha
capi-ha-control-plane-krkpk	Virtual machine	capi-ha
capi-ha-control-plane-szk82	Virtual machine	capi-ha
capi-ha-control-plane-tx9fg	Virtual machine	capi-ha
capi-ha-node-routetable	Route table	capi-ha
capi-ha-cluster-azure-bastion-pip	Public IP address	capi-ha
pip-capi-ha-controlplane-outbound	Public IP address	capi-ha
pip-capi-ha-node-natgw-1	Public IP address	capi-ha
pip-capi-ha-node-natgw-2	Public IP address	capi-ha
pip-capi-ha-node-outbound	Public IP address	capi-ha
lndbrg.tech	Private DNS zone	capi-ha
capi-ha-controlplane-nsg	Network security group	capi-ha
capi-ha-node-nsg	Network security group	capi-ha
nsg-isolated-node	Network security group	capi-ha
capi-ha-control-plane-krkpk-nic	Network Interface	capi-ha
capi-ha-control-plane-szk82-nic	Network Interface	capi-ha
capi-ha-control-plane-tx9fg-nic	Network Interface	capi-ha
capi-ha-node-natgw-1	NAT gateway	capi-ha
capi-ha-node-natgw-2	NAT gateway	capi-ha
capi-ha	Load balancer	capi-ha
capi-ha-cluster-internal-lb	Load balancer	capi-ha

< Previous Page 1 of 1 Next >

Appendix 14 – Isolated node group Security Group Rules.



nsg-isolated-node

Network security group

Move | Delete | Refresh | Give feedback

Essentials

Resource group (move) : capi-ha
Location : North Europe
Subscription (move) : Visual Studio Enterprise Subscription
Subscription ID : 8ec5a1ea-0c7b-495b-9a82-2a8f97803062
Tags (edit) : Name : nsg-isolated-node | sigs.k8s.io_cluster-api-provider-azure_cluster_capi-ha : owned

Custom security rules : 5 inbound, 0 outbound
Associated with : 1 subnets, 0 network interfaces

Filter by name | Port == all | Protocol == all | Source == all | Destination == all | Action == all

Priority	Name	Port	Protocol	Source	Destination	Action
Inbound Security Rules						
100	allow-intra-subnet	Any	Any	10.0.0.192/26	10.0.0.192/26	Allow
110	allow-control-plane	Any	Any	10.0.0.0/27	10.0.0.192/26	Allow
120	allow-ssh-from-bastion	22	Tcp	10.0.0.128/26	10.0.0.192/26	Allow
130	allow-azure-loadbalancer	Any	Any	AzureLoadBalancer	Any	Allow
140	deny-vnet-to-isolated-node	Any	Any	VirtualNetwork	VirtualNetwork	Deny
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny
Outbound Security Rules						
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Appendix 15 – Isolated node group connection testing.

```
# A successful connection from the control plane to the isolated node
root@capi-ha-control-plane-tx9fg:~# nc -zv capi-ha-isolated000000 22 -w 10
Connection to capi-ha-isolated000000 (10.0.0.196) 22 port [tcp/ssh] succeeded!

# A failed connection from the general node (in regular node subnet) to the isolated
    node
root@capi-ha-general000000:~# nc -zv capi-ha-isolated000000 22 -w 10
nc: connect to capi-ha-isolated000000 (10.0.0.196) port 22 (tcp) timed out: Operation
    now in progress
```

Appendix 16 – Velero backup output with *EncryptionAtRest-WithPlatformKey*.


```
{
    "name": "pvc-3317be7c-9b3e-4b89-bf2e-106124d94f3c-36fc38e9-3bc5-4c16-a1aa-395
        d0cef9603",
    "id": "/subscriptions/${subscriptionId}$/resourceGroups/mgmt-cluster/providers/
        Microsoft.Compute/snapshots/pvc-3317be7c-9b3e-4b89-bf2e-106124d94f3c-36fc38e9-3
        bc5-4c16-a1aa-395d0cef9603",
    "type": "Microsoft.Compute/snapshots",
    "location": "northeurope",
    "tags": {
        "app.kubernetes.io-instance": "velero",
        "app.kubernetes.io-managed-by": "Helm",
        "app.kubernetes.io-name": "velero",
        "argocd.argoproj.io-instance": "velero",
        "env": "capi-ha",
        "helm.sh-chart": "velero-9.0.4",
        "k8s-azure-created-by": "kubernetes-azure-dd",
        "kubernetes.io-created-for-pv-name": "pvc-3317be7c-9b3e-4b89-bf2e-106124d94f3c",
        "kubernetes.io-created-for-pvc-name": "storage-loki-0",
        "kubernetes.io-created-for-pvc-namespace": "loki",
        "velero.io-backup": "velero-mybackup-20250509233301",
        "velero.io-pv": "pvc-3317be7c-9b3e-4b89-bf2e-106124d94f3c",
        "velero.io-schedule-name": "velero-mybackup",
        "velero.io-storage-location": "default"
    },
    "sku": {
        "name": "Standard_LRS",
        "tier": "Standard"
    },
    "properties": {
        "creationData": {
            "createOption": "Copy",
            "sourceResourceId": "/subscriptions/${subscriptionId}/resourceGroups/capi-ha
                /providers/Microsoft.Compute/disks/pvc-3317be7c-9b3e-4b89-bf2e-106124
                d94f3c",
            "sourceUniqueId": "dbaf148c-085a-4bcf-896f-feb4535212df"
        },
        "diskSizeGB": 10,
        "encryption": {
            "type": "EncryptionAtRestWithPlatformKey"
        },
        "incremental": false,
        "networkAccessPolicy": "AllowAll",
        "publicNetworkAccess": "Enabled",
        "timeCreated": "2025-05-09T23:33:11.0358671+00:00",
        "provisioningState": "Succeeded",
        "diskState": "Unattached",
        "diskSizeBytes": 10737418240,
        "uniqueId": "372a3d94-e3be-4a57-a752-1a2c4a47a8ae"
    },
    "apiVersion": "2022-03-02"
}
```

Appendix 17 – Validation of 'restarter' CronJob.

```
# Created CronJob that restarts defined applications in the cluster based on schedule
root@capi-ha-control-plane-tx9fg:~# kubectl get cronjob -n restarter
NAME                  SCHEDULE     TIMEZONE   SUSPEND   ACTIVE   LAST SCHEDULE   AGE
restart-applications   20 * * * *   <none>     False     0        71s             3m3s

# Pod created by CronJob
root@capi-ha-control-plane-tx9fg:~# kubectl get pods -n restarter
NAME                                   READY   STATUS      RESTARTS   AGE
restart-applications-29123360-c8qns   0/1     Completed   0          77s

# Pod logs confirming that 'argocd-server' Deployment and 'argocd-application-controller
    ' StatefulSet were restarted
root@capi-ha-control-plane-tx9fg:~# kubectl logs -n restarter restart-applications
    -29123360-c8qns
Restarting Deployment argocd-server in namespace argocd
deployment.apps/argocd-server patched
Restarting StatefulSet argocd-application-controller in namespace argocd
statefulset.apps/argocd-application-controller patched

# Verifying from "AGE" column that indeed the applications were restarted
root@capi-ha-control-plane-tx9fg:~# kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          73s
argocd-applicationset-controller-84449fb775-589kf   1/1     Running   0          11m
argocd-redis-7459ddd4fc-jjb2l                       1/1     Running   0          11m
argocd-repo-server-7c66dd8687-5xvxb                 1/1     Running   0          11m
argocd-server-7b7c78b7b6-j472c                      1/1     Running   0          76s
```