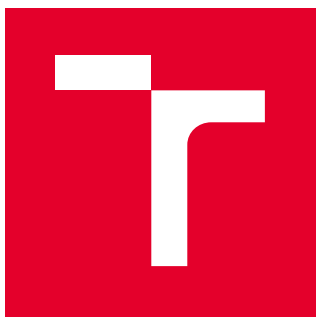




BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF BUSINESS AND MANAGEMENT

FAKULTA PODNIKATELSKÁ

INSTITUTE OF INFORMATICS

ÚSTAV INFORMATIKY

CYBER SECURITY IN CONTAINERIZATION: COMPLIANCE MONITORING

KYBERNETICKÁ BEZPEČNOST V KONTEJNERIZACI: SOULAD S PŘEDPISY

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. Dominika Kuchtová

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. Petr Sedlák

BRNO 2025

Assignment Master's Thesis

Department: Institute of Informatics
Student: **Bc.Dominika Kuchtová**
Supervisor: **Ing. Petr Sedlák**
Academic year: 2024/25
Study programme: Information Management

Pursuant to Act no. 111/1998 Coll. concerning universities as amended and to the BUT Study Rules, the degree programme supervisor has assigned to you a Master's Thesis entitled:

Cyber Security in Containerization: Compliance Monitoring

Characteristics of thesis dilemmas:

Objectives of the work
Theoretical introduction
Description of the current state of the art
Proposed solution
Economic evaluation
Conclusion

Objectives which should be achieved:

The primary goal of this thesis is to design and implement a solution that enhances container compliance within a CI/CD pipeline. This involves developing a system to ensure adherence to security and operational standards while providing real-time insights into containerized applications during their lifecycle

Basic sources of information:

DOUCEK Petr, Martin KONEČNÝ a Luděk NOVÁK, Řízení kybernetické bezpečnosti a bezpečnosti informací, Praha: Professional Publishing, 2020. ISBN 978-80-88260-39-4.

SEDLÁK Petr, Martin KONEČNÝ, Přeměna ISMS v manažerské informatice. Brno: CERM, Akademické nakladatelství, 2023. ISBN 978-80-7623-110-8.

SEDLÁK Petr, Martin KONEČNÝ a kolektiv, Kybernetická (ne)bezpečnost. CERM, Akademické nakladatelství, 2021. ISBN 978-80-7623-068-2.

SEDLÁK Petr, Martin KONEČNÝ a kolektiv, Případové studie řízení kybernetické bezpečnosti. CERM, Akademické nakladatelství, 2024. ISBN 978-80-7623-126-9.

SOUPPAYA, Murugiah; MORRELO, John; SCARFONE, Karen. NIST Special Publication 800190: Application Container Security Guide. Online. 2017. National Institute of Standards and Technology.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2024/25

In Brno dated 9.2.2025

L. S.

doc. Ing. Miloš Koch, CSc.
Branch supervisor

prof. Ing. et Ing. Stanislav Škapa, Ph.D.
Dean

Abstract

The goal of this master's thesis is to propose a solution for a private on-premises cloud infrastructure. The primary focus is identifying and addressing security threats in containerised environments while managing and operating the infrastructure effectively over time. To support this work, an analysis of the external environment, an in-depth look at the internal infrastructure, and a cybersecurity risk assessment have been carried out. Based on the findings' implications, a SWOT analysis has been conducted, and a proposal for the solution has been derived.

Upon reflecting on observations, an architecture for a remediation plan was developed, and automation pipeline steps were established accordingly. The testing environment was assessed using key metrics from the pipeline, including vulnerability capture rates and compliance success rates for patching. During the implementation of the pipeline, we conducted essential tests. Findings showed that the security scan of the environment met a high standard of compliance, which helped support the decision to approve the solution. We also looked at the financial side by evaluating the solution's cost-effectiveness using the Return on Security Investment (ROSI) metric.

The main goal of this master's thesis is to identify and address security risks in container orchestration, while also finding effective ways to manage and maintain the infrastructure over its entire lifecycle. One of the key concerns is containers that are deployed outside of dedicated pipelines, these pose a significant security risk to the overall system. They likely lack validated images, are not adequately secured through vulnerability identification or remediation tools and are not included in systematic patching. Additionally, there is ambiguity regarding the applications running on these containers. Based on these findings, we formulated the architecture of the remediation plan and created automation pipeline steps to support it.

Keywords

Cloud Computing, Container security, Security Pipeline, DevOps, Compliance

Abstrakt

Cloud se v dnešní době rychle vyvíjí a je v podnicích široce rozšířen, takže pojímá stále více pracovních zátěží. Jedním z mnoha důvodů jsou jeho agilní funkce pro vývojové a provozní týmy (DevOps), protože nabízí řešení pro společnosti, které potřebují reagovat na neustále se měnící požadavky vyplývající z externích prostředí nebo interního vývoje. Cloud optimalizuje přerozdělování zdrojů a DevOps maximalizuje návratnost investic díky kontinuálnímu poskytování hodnot.

V minulosti se při vývoji softwaru vyvíjely aplikace v oddělených prostředích. Když vznikla potřeba přenosu, části se přenášely jako binární soubory, což způsobovalo problémy s kompatibilitou související s konfigurací a závislostmi, které bylo třeba ručně upravovat. Tomuto přenosu nepomohl ani nárůst migrací z fyzických strojů na virtuální stroje, ani samotný vznik hybridního prostředí. Se vznikem cloudu přišla efektivnější práce s virtuálními stroji, ale brzy se objevil stejný problém pro cloud. V cloudovém prostředí na virtuálním stroji bylo potřeba efektivně využít veškerý výpočetní výkon, a proto to vedlo k myšlence nasazení více aplikací/služeb na daném serveru. Což však vedlo k dalším problémům s interakcemi nebo potřebou různých verzí knihoven. Tento problém vyřešila kontejnerizace. Kontejner je další formou virtualizace, jako jsou virtuální stroje. Termín je převzat z lodního průmyslu, ale zatímco v lodní dopravě se kontejner používá k fyzickému oddělení nákladu, v IT virtuální kontejnery balí program, konfigurační soubory, knihovny a závislosti, které aplikace potřebuje ke svému běhu. Balí je tak, aby je bylo možné přenášet do různých systémů nebo po síti.

Vývoj využívá otevřený software a cloud k rychlejšímu poskytování řešení v podobě aplikací než kdykoli předtím, přičemž velkou roli hraje kontejnerizace, CI/CD pipelines a infrastruktura jako kód neboli IaC. Nedávné kybernetické útoky zaměřené konkrétně na cloudová prostředí však ukázaly, že bezpečnostní monitorování produkčních prostředí nestačí. Je načase, aby organizace začaly využívat nejen výhod zrychleného doručování, ale také přizpůsobily vhodný přístup k zabezpečení při vytváření infrastruktury od samého počátku.

Tato práce se zabývá velmi aktuálním tématem bezpečnosti při používání technologií, nástrojů a softwaru v prostředí cloudových kontejnerových řešení. Skládá se z nezbytných faktických teoretických podkladů. V další části práce je analyzováno prostředí nadnárodní společnosti. Je zde uveden přehled současného stavu a požadovaný cíl. Dále je provedena analýza vnějšího prostředí z hlediska kontejnerové bezpečnosti, interní analýza, analýza rizik a stanovení priorit na základě zjištěných poznatků. Na základě analýz je v následující části také vypracován návrh řešení s obecnou praktickou implementací pro univerzální použití v podobných případech. Nakonec je toto řešení analyzováno z hlediska ekonomické efektivnosti, kde je použita metrika ROSI. V poslední části jsou uvedeny možnosti vylepšení do budoucnosti a shrnutí přínosů práce.

Klíčová slova

Cloud Computing, Bezpečnost kontejnerizace, Bezpečnostní pipeline, DevOps, Soulad s předpisy

Bibliographic citation

KUCHTOVÁ, Dominika. *Cyber Security in Containerization: Compliance Monitoring* [online]. Brno, 2025 [cit. 2025-03-30]. Available at: <https://www.vutbr.cz/studenti/zav-prace/detail/167904>. Master's Thesis. Brno University of Technology, Fakulta podnikatelská, Ústav informatiky. Supervisor Ing. Petr Sedlák.

Affidavit

I declare that the present master project is an original work that I have written myself. I declare that the citations of the sources used are complete, that I have not infringed upon any copyright (pursuant to Act. no 121/2000 Coll.).

Brno dated 18th May 2025

Bc. Dominika Kuchtová

author's signature

Contents

INTRODUCTION.....	4
1 THEORETICAL BACKGROUND OF THE WORK.....	6
1.1 CLOUD COMPUTING.....	6
1.1.1 Deployment and delivery model.....	7
1.1.2 Virtualisation.....	9
1.1.3 Containerisation.....	10
1.2 CONTAINER SECURITY.....	17
1.2.1 Container Security Standards	18
1.2.2 Common Risks in Container Infrastructure	21
1.3 DEVOPS.....	23
1.3.1 DevSecOps	23
2 ANALYSIS OF THE CURRENT STATE	26
2.1 COMPANY INTRODUCTION	26
2.2 EXTERNAL ENVIRONMENT ANALYSIS	28
2.2.1 Common Threats in the External Environment.....	29
2.3 INTERNAL ENVIRONMENT ANALYSIS	33
2.3.1 Asset definition	33
2.3.2 Methodology.....	38
2.3.3 Identifying Vulnerabilities.....	38
2.3.4 Severity and likelihood of exploitation assessment	40
2.3.5 Risk Assessment of found vulnerabilities	45
2.4 IMPLICATIONS.....	47
2.4.1 Implications from external analysis	47
2.4.2 Implications from internal analysis.....	47
2.4.3 SWOT analysis	48
3 SOLUTION PROPOSAL	50
3.1 AUTOMATED SCANNING SOLUTION FOR VULNERABILITIES AND POLICY ENFORCEMENT	50

3.1.1	<i>Architecture</i>	50
3.1.2	<i>Solution Design</i>	52
3.1.3	<i>Implementation</i>	60
3.1.4	<i>Testing</i>	62
3.2	CONTAINER IMAGE HARDENING.....	67
3.2.1	<i>Architecture</i>	67
3.2.2	<i>Solution Design</i>	69
3.2.3	<i>Implementation</i>	71
3.2.4	<i>Testing</i>	73
3.3	ECONOMIC EVALUATION	75
3.3.1	<i>Economic evaluation of Image hardening pipelines</i>	78
3.4	FUTURE DEVELOPMENT.....	79
CONCLUSION		80
BIBLIOGRAPHY		82
LIST OF TABLES		88
LIST OF FIGURES		89
APPENDIX		I
THREAT ASSESMENTS		I
RISK MANAGEMENT MATRIX		X
AUTOMATED SCANNING SOLUTION OVERVIEW		XIV
ROSI FULL CALCULATION.....		XV

INTRODUCTION

Traditionally, in software development, applications were developed in separate environments. Separation usually meant transferring the application as a binary file. The challenge was that each environment had its own setup, so configurations and dependencies were not usually compatible. As a result, developers often had to spend time manually fixing these issues to get the application running smoothly. This challenge is addressed in containerisation by packaging applications with all necessary dependencies, which allows them to run consistently across any system with a similar OS kernel, whether from a cloud environment or personal computers.

The current fastest delivery of application solutions is heavily driven by containerisation, agile methodologies, CI/CD pipelines, and infrastructure as a code (Iac). According to many institutions like Cloud Native Computing Foundation (CNCF), the use of containers in companies either actively evaluating or piloting them is greater than 90%. These entities rely on containers for most of their cloud-native software development. [\[1\]](#) Out of these primary consumers of the cloud, 40% have reported that security is their leading challenge. Monitoring and observability can become more challenging when considering systems running containers at scale. Robust systems require real-time data collection of metrics, events, and logs to enforce proactive system management.

Cloud computing is widely adopted by enterprises for the agile functions of Development and Operations teams (DevOps) as it has been the solution for companies to meet ever-changing customer demands. Integrating DevOps, cloud computing, and agile methodologies creates a cohesive strategy to enhance software development, improve collaboration, and maintain flexibility in responding to market shifts. Cloud computing optimises resource allocation, and DevOps maximises return on investment (ROI) through continuous delivery of value supported by the agile methodology.

Additionally, cloud computing enables businesses to develop distributed systems with pay-as-you-go, high performance, scalability, ubiquitous availability, and other enabling features. Companies can deploy cloud-based services as an infrastructure for everyday tasks, including critical systems. However, the increased dependence on cloud services has resulted in several availability and reliability problems.

Even though containerisation promotes rapid development, efficiency, and large-scale operations, it is in place to rethink traditional security practices. Unlike mutable servers, containers are immutable, meaning they cannot be modified permanently without rebuilding. Immutability of containers can be approached with pipeline architecture for container security, which integrates security practices into the container deployment process to address vulnerabilities and enhance security throughout the container life cycle.

In addition, container orchestrators, along with their foundational principles, are well-suited to adapt to future renewable energy sources, as they utilise computing hardware far more efficiently than virtual machines.

This thesis explores the container security topic in using technology and software in a cloud container solution environment. It provides a background and includes a review of findings from other scientific articles and theses. The thesis then analyses the environment of a private on-premises environment and proposes a solution for universal use in similar cases. The proposed solution focuses on several aspects of the security of containers. Exploring operational complexity, threat detection, inefficiencies in the resource allocation, reliability issues and policy inconsistency. The thesis also considers the economic efficiency of the proposed solution. The concluding chapter discusses the contributions of the thesis and future work. Overall, the goal of the thesis is to offer a universal approach to address vulnerabilities in private on-premises environments that are not ready for full adoption of Kubernetes, leveraging DevOps principles enhanced with security best practices and tools like Ansible, Tanium, Trivy and Copa.

1 Theoretical Background of the Work

This section contains the core ideas that form the backbone of this thesis's practical and analytical work. Starting with the overall scope and scale of the study, then dive into what cloud computing is, the different forms it can take, how it is used in the real world, and how virtualisation makes it all possible. The next part delves into the specifics of containerisation and its underlying infrastructure needed for smooth and controlled operation. At last, it explains DevOps principles and their integration with the previously discussed technologies, establishing a coherent framework for the following chapters.

1.1 Cloud Computing

Cloud computing enables the use of computing power and resources, such as storage and services, that third parties provide through the Internet as a commodity to enterprises and customers on demand. Resources are managed by the provider responsible for the physical hardware and updates. Users rent the capacity and pay for what they use. In cloud computing, IT is in the form of a service offered to customers by the cloud provider. These services are typically paid on a pay-as-you-go or on a monthly basis. All of this is based on the four cornerstones of quality service. [\[2\]](#)

1. *Efficiency*

Optimising data traffic and latency remains essential to reduce costs and improve performance, especially as data rapidly grows. The aim for most customers is low latency and efficiency at a cost. [\[2\]](#)

2. *Scalability*

Considering the need to scale up to support large user bases, handle peak traffic, and cope with flash crowds. All applications should be designed to serve scalability, mitigating distributed denial-of-service attacks. [\[2\]](#)

3. *Robustness*

High availability, redundancy, and failover strategies ensure consistent services alongside risk management and failure testing to ensure service reliability and minimise downtime. [\[2\]](#)

4. *Security*

Establishing trust by deploying robust security measures is crucial for protecting data and applications from threats, while also fulfilling provider-customer responsibilities and meeting Service Level Agreements (SLA). [\[2\]](#)

1.1.1 Deployment and delivery model

The deployment models can be categorised into five groups: public cloud, private cloud, hybrid cloud, community cloud, and agency cloud. [\[2\]](#)

Public Cloud aligns with the general description of Cloud Computing. Resources are dynamically provisioned based on self-service queries over the Internet by third-party providers who share resources, making them available to multiple users (tenants). Well-known examples of providers are Google Cloud, AWS and Microsoft Azure. [\[2\]](#)

Private Cloud is a dedicated environment that a single organisation exclusively uses. It is usually hosted on-premises or by a third-party provider but remains isolated for the organisation's purposes. Private deployment is used mainly in cases involving sensitive workloads or industries with stringent compliance requirements, such as finance or healthcare. [\[2\]](#)

Hybrid Cloud combines private and public clouds, allowing data and applications to fluctuate between these two environments. Such an establishment provides flexibility to manage sensitive workloads in a private environment while leveraging the public cloud

for scalability. The hybrid deployment type is increasingly utilised due to the rising number of fluctuating demands, the need for disaster recovery, and data separation. [2]

Community Cloud is the cloud infrastructure designed for the exclusive use of a specific community of customers, typically composed of organisations with shared interests or concerns. This infrastructure can be owned, managed, and operated by one or more member organisations within the community, by a third party, or through a collaborative arrangement. Additionally, it may be hosted either on-premises or in an off-premises environment. [3]

Agency Cloud is a specialised form of community cloud tailored for military, defence or government use. Unlike public clouds, it is restricted to authorised agencies and is fully managed by them to ensure high security and compliance with specific operational requirements. For instance, entities like the Defence Information Systems Agency (DISA) might run agency cloud to securely handle sensitive data. [2]

1.1.1.1 Cloud computing delivery

The most common service methods in the cloud can be divided into infrastructure, application platform, and application software. Service models are generally classified into three groups: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). [2]

IaaS or Infrastructure as a Service is the base layer in the cloud service hierarchy. IaaS includes virtual machines, storage, and networking resources, which can be scaled up and down as needed. The cloud provider solely manages the underlying infrastructure, including servers and networking hardware. Users control their operating systems, applications, and configuration via the web interface or API. Examples of IaaS are Amazon Web Services (AWS), Microsoft Azure or Google Compute Engine (GCE). [4]

PaaS or Platform as a Service provides developer teams with an environment and toolset for building, deploying and managing applications without the need to handle the underlying infrastructure. Key features include application development tools and infrastructure management. It offers built-in services like databases, messaging, and workflow engines and supports team collaboration with version control and shared development. The best-known examples of platform as a service are SAP Cloud or Google App Engine. [\[2\]](#) [\[4\]](#)

SaaS or Software as a Service, often commonly referred to as the Application Service Provider model, introduces a different way of application software delivery. Applications in this model are database-centric, integrating database operations seamlessly into service delivery. SaaS runs on top of PaaS, which runs on top of IaaS. The resources and a single instance of object code, application and underlying database are shared. A shared model is beneficial for the support of multiple customers who rely on the scalable infrastructure. The fault tolerance principles, automated recovery, and flexible schemas are utilised to enhance performance. Naturally, it enables extensive customisation of user interfaces, workflows, and persistence. An example of SaaS is Gmail from Google or Microsoft Office 365. [\[2\]](#) [\[4\]](#)

1.1.2 Virtualisation

Virtualisation makes it possible to create virtual machines by dividing and managing physical hardware resources more efficiently. This technology extends beyond computing as it can also be used to virtualise storage, networking, and other IT infrastructure components. [\[2\]](#) [\[4\]](#)

A **virtual machine (VM)** is a software-driven emulation of a physical computer that runs an operating system and executables independently of the underlying hardware. Each VM includes a complete operating system, and virtualised HW (CPU, memory, storage, network). A hypervisor is necessary for VM isolation and abstraction. Hypervisors act as an intermediary layer, manage virtualised resources, and enable multiple virtual machines

to run on a single physical host, which also acts as a hardware-level isolation. The key component for a hypervisor is the host type, which can be either on bare metal, which is more efficient and provides better performance as the hypervisor has direct access to physical resources (VMware ESXi) or on top of a host operating system (Oracle VirtualBox). [\[4\]](#)

1.1.3 Containerisation

Containerisation is a form of virtualisation that enables applications to run alongside their dependencies in isolated environments. Unlike traditional virtualisation, which abstracts the hardware layer, containerisation abstracts the operating system layer, creating lightweight, easily portable runtimes independent of the underlying hardware and OS. [\[2\]](#) [\[4\]](#)

Containers are yet another computing model after bare metal and virtual machines. The container isolation levels vary, but applications generally run with an isolated root filesystem, namespaces and resource allocation from other processes on the host server. The formats of how to package and run them are standardised by the Open Container Initiative (OCI), designed so that software components encapsulate application code and all necessary dependencies, ensuring seamless portability across different computing environments that run independently. They provide a consistent and isolated application environment, ensuring reliability across development, staging and production environments with low-overhead isolation. Applications running in containers share the host OS kernel while remaining isolated. However, their OS specificity is limited: Linux containers operate solely on Linux hosts, and Windows containers exclusively on Windows hosts. They provide far more hardware efficiency and have given engineering teams more flexibility to run cloud-native applications and quickly match demand spikes. [\[5\]](#) [\[6\]](#) [\[7\]](#)

Visualisation of layer difference can be seen in Figure 1 below.

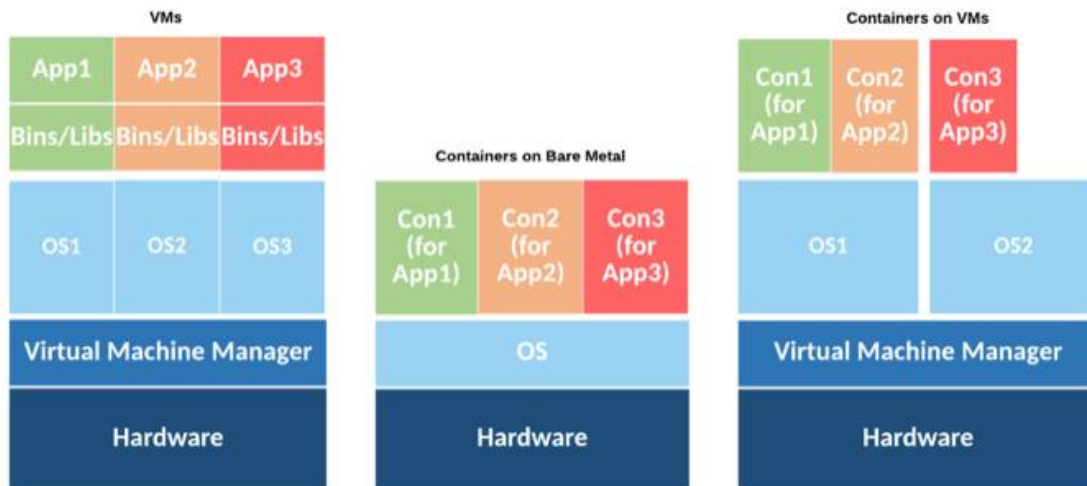


Figure 1: Layer difference overview of container deployments and virtual machine.

Image source [\[7\]](#)

Key Concepts of Containerisation

Containers leverage Linux Kernel features such as namespaces for process isolation and groups for resource allocation, enabling multiple containers to share a host kernel while remaining isolated. [\[7\]](#) [\[8\]](#)

The lifecycle of the container is categorised into three distinct phases:

1. ***Build Phase*** – Developers pack the application and its binaries into images.
2. ***Distribution Phase*** – Images are distributed to public or private registries and archived.
3. ***Run Phase*** – Containers are created from the images pulled from the registry.

Categories of Containers

Containers are categorised based on the type of virtualisation and application services they provide.

System Containers also known as operating system containers, they provide lower-level virtualisation (OS level) for applications and may lack scalability and flexibility. They

mimic virtual machines and are ideal for running monolithic applications¹. It supports Linux-based environments and uses *cgroups*² for resource isolation. However, LXC (Linux Containers) lacks third-party tool support, requires manual updates, and provides lower security due to shared vulnerabilities. [\[6\]](#) [\[7\]](#) [\[8\]](#)

Application Containers are built upon LXC and *libcontainer*, providing application-level virtualisation, and are designed to execute single processes, making them stateless and microservice-oriented. Application container file systems are usually built in layers, so they can use external storage to keep data around even after the system shuts down. These systems are designed to scale out easily, allowing infrastructure to grow by simply swapping old containers for new ones. They work across Linux, macOS, and Windows, making them portable and scalable. Many application containers emphasise CI/CD workflows, offer versioning, and support collaborative development through shared libraries. Popular application containers include Docker or Podman. [\[6\]](#) [\[7\]](#) [\[8\]](#)

Figure 2 below displays differences between these two types of containers.

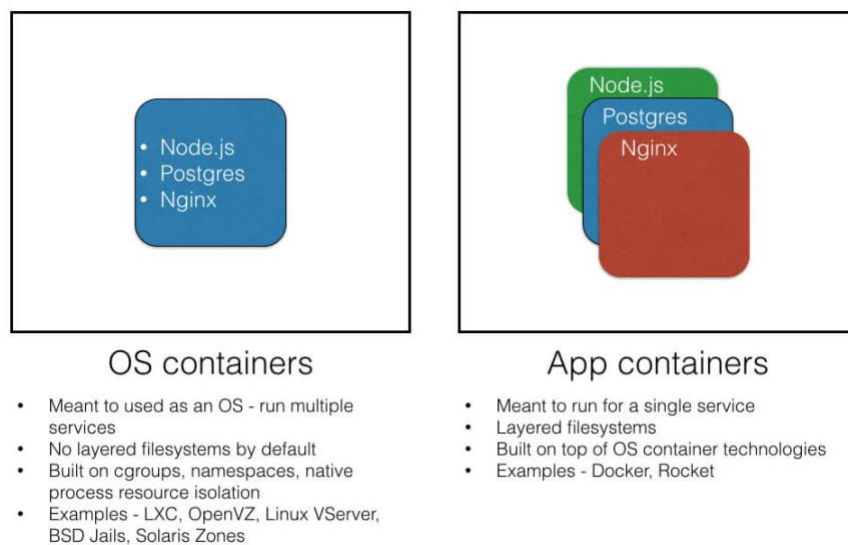


Figure 2: Container types . Image source [\[9\]](#)

¹ Monolithic application is a complex application that is designed to handle multiple related tasks.

² Is a feature that enables to limit or manipulated resources used by the process.

Container Orchestration

The term orchestrator is often used interchangeably with the term scheduler. However, the orchestrator is mainly responsible for resource utilisation in a cluster. Orchestrators are tools typically taking care of many tasks such as health checks, cloud automation, load balancing, scheduling and distribution. Schedulers are a subset of orchestration platforms that oversee which processes and services run on each server or node. [\[10\]](#)

Orchestration tools are platforms for running and managing containers. They enable the running of microservices applications, which are defined as a set of containers on a cluster. The focus is on the lifecycle of containers to provide scalability and high availability. With these tools, scaling up and down or redirecting customer traffic is possible. These tools provide crucial building blocks to define and manage complex Cloud-native applications with high flexibility and complexity.

Kubernetes is one of the leading open-source orchestration platforms that is easily implemented and optimises container-based infrastructure across the globe. Nevertheless, organisations must establish a secure environment, particularly when building architecture on open-source code, usually into third-party applications. The tool itself has an extensive ecosystem that promotes integrating automated, systematic processes into the core of its build and deployment pipelines. Some of Kubernetes's native features promote secure operations, such as RBAC, pod security policies, and network policies. [\[11\]](#)

Pods are the most minor deployable units in Kubernetes, and can contain one or more containers that share storage, network namespace, and resources. [\[11\]](#)

Control plane includes API Server, which exposes Kubernetes API to users and components, Scheduler, which assigns pods to nodes based on resource requirements, Etcd acting as a distributed key-value store for cluster state, and Controller Manager, which manages replication and endpoints. [\[11\]](#)

Nodes can be either physical or virtual host machines for containers. In Kubernetes, a node is managed by the control plane and runs pods. Includes the kubelet component and the kube-proxy component. Kubelet facilitates communication with the control plane to manage the pod lifecycle. Kube-proxy handles networking rules for pods. [\[11\]](#)

Networking in containerised environments

Networking is essential for enabling communication between containers and external environments. Virtual networks act as bridges that ensure seamless container-to-container and container-to-host connectivity. [\[11\]](#)

Container-to-container communication

Containers within a pod share a network namespace, so they see each other as localhost. Visibility between containers is implemented by default; the shared namespace acts as a bridge to dispatch the traffic for every container in a pod. [\[11\]](#)

Pod-to-pod communication

Pod IP addresses are accessible from other pods, no matter which nodes they are on. Pod-to-pod communication within the same node goes through the bridge by default. Let us say we have two pods, which have their network namespaces. First pod packet passes through its namespace and goes to the bridge. The bridge then broadcasts the destination IP to help the packet find its way. [\[11\]](#)

Pod communication across nodes

All nodes must communicate with all containers. Orchestrator delegates the implementation to the container network interface (CNI). Users could choose different implementations like L2, L3, or overlay. Overlay networking is one of the standard solutions, known as packet encapsulation. It wraps a message before leaving the source, gets delivered, and unwraps it at the destination. Cross-node communication leads to a situation where overlay increases the network latency and complexity. [\[11\]](#)

Pod-to-service communication

The key point is that the service exposes the cluster IP to outside traffic from the target, which links to two custom chains with a statistic mode random probability of 0.5. These two custom chains have the DNAT target set to the corresponding pod IP. The DNAT target is responsible for changing the packets' destination IP address. All of this results in a routing behaviour. When the traffic comes to service, iptables will randomly pick one of the pods to route and modify the destination IP from the service IP to the real pod IP, and un-DNAT to go back. [\[11\]](#)

Network types

Bridge Network is usually, a default setup is when containers share the same IP range for internal communications without additional configuration. Secures communication by isolating containers from external networks. [\[12\]](#)

Host Network's namespace is shared by containers, providing maximum network performance and reducing security by bypassing containers' network isolation. [\[12\]](#)

No Network is basically disabled any network communication for containers, which makes it ideal for high-security environments or strict isolation for sensitive workloads. [\[12\]](#)

Overlay Network creates a virtual network that spans the entire cluster, making it ideal for distributed applications with service instances on different hosts. [\[12\]](#)

MAC-Vlan Network assigns each container a unique MAC address, making them appear as physical devices. It provides direct access to physical networks without NAT and is helpful for network services, firewalls, or routers within containers. [\[12\]](#)

IP-Vlan Network shares a single MAC address across multiple containers with the possibility of operating in two modes. In layer 2 mode, the system works like a virtual switch, containers talk to each other within the same subnet using MAC addresses, just like devices on a local network. Furthermore, layer 3 mode acts like a router, allowing containers on different subnets to communicate using IP addresses. Both modes provide network isolation and offer additional flexibility in routing options. [\[12\]](#)

Network policies

Ensures fine-grained control over network traffic, enhancing security within a cluster. The purpose of network policies is that they act as a software firewall for pods and control communication between them by setting explicit rules, as by default, all pods can communicate without restrictions. [\[13\]](#)

1.2 Container Security

Container security reflects the ever-changing nature of IT architecture and the answer to the need for fast adoption of cloud-native computing solutions. The key is adjusting how to secure them, while keeping up with the pace of technology changes. Containers do not have the same robust support as virtual machines, primarily due to a lack of comprehensive security support, sharing the same kernel with multiple applications, and varying access to the host. This shared kernel architecture can lead to vulnerabilities if the kernel is exploited or if containers are misconfigured, such as running with excessive privileges. However, containers provide extra isolation from the hosting machine and each other through namespaces. Namespaces can reduce the impact of a compromised application, as it remains restricted to its containerised scope. Their ephemeral nature may pose challenges for the traditional security measures, including event detection, logging and forensic analysis, as they do not have a centralised log system by default. Nevertheless, not only do the containers create a new risk, but the rise of orchestration platforms like Kubernetes also expands the attack surface and adds layers that require securing. [\[5\]](#) [\[7\]](#) [\[13\]](#)

As induced securing container environment may require a shift in approach compared to traditional security measures, the dynamic and lightweight nature introduces unique security risks that must be addressed at every application lifecycle stage. In the development stage, security must ensure secure coding, scanning for vulnerabilities and a secure CI/CD pipeline and image registry. Enforcing policies and safe orchestration is crucial for the deployment phase validation of container images. In the last phase for runtime, continuous monitoring of containers in production is essential to detect threats while maintaining performance and availability. [\[7\]](#) [\[13\]](#)

All actions are based on these five security practises:

1. Vulnerability Assessment
2. Configuration Management
3. Access Control Enforcement
4. Network Segmentation
5. Continuous Monitoring

Securing a containerised environment means protecting containerised applications and the underlying infrastructure throughout the lifecycle, from development to deployment and runtime. Taking into scope vulnerability scanning on each step, configuration management, access control, network segmentation and active monitoring. Since containers are immutable and cannot be directly modified while running, any updates require a rebuild. This immutability creates an opportunity to adopt a shift-left security approach, integrating security measures early in the CI/CD pipeline. The goal is to maximise the benefits of a containerised environment while minimising risks associated with resource sharing and enlarged attack surface. [\[7\]](#) [\[13\]](#)

1.2.1 Container Security Standards

There are several guidelines from various bodies that can help in securing container environments. These resources fall into two categories: compliance standards and hardening guides. Where compliance standards focus on providing detailed and precise recommendations, like configuration of specific parameters or setting file permissions for a particular product, a hardening guide covers more ground at a higher level. Hardening guides might provide specific details, but not necessarily cover all settings related to the security of one product. [\[14\]](#)

1.2.1.1 Compliance standards

NIST SP 800 Series standards are published by the National Institute of Standards and Technology (NIST). Publications provide comprehensive security guidance. Relevant documents are NIST SP 800-190: Application Container Security Guide, which focuses specifically on container security, and NIST SP 800-53, which covers security and privacy controls for IT systems, with implications for containerised environments. [\[14\]](#)

CVE (Common Vulnerabilities and Exposures)

The objective of the CVE® Program is to systematically identify, define, and catalogue publicly disclosed cybersecurity vulnerabilities. A vulnerability is an internal feature of an asset, and each vulnerability has a unique CVE Record. [\[15\]](#)

Key functions of CVE tools in Containerised Compliance:

- Analyses container images for known CVEs.
- Continuously inspect running containers for vulnerabilities.
- Ensures containers adhere to security policies.
- Alerts and remediates security risks using updated CVE databases.
- Aligns with regulatory frameworks.

SOC 2 (System and Organisation Controls 2) is a framework for customer data management defined on five trust principles: privacy, confidentiality, processing integrity, availability and security. [\[14\]](#)

ISO/IEC 27001 provides a framework for ISMS (information security management systems), and in connection with containers, it requires secure handling of containerised applications and infrastructure as part of the overall ISMS. [\[14\]](#)

GDPR (*General Data Protection Regulation*) regulates data protection and privacy in the EU (European Union), containers that handle personal data must ensure compliance with data encryption, access restriction, and monitoring to prevent data breaches. [\[14\]](#)

OpenSCAP is a compliance framework for automated security audits and configuration validation. This framework translates into a containerised environment with tools and guidelines for validating container configurations against established standards like NIST and CIS. [\[14\]](#)

1.2.1.2 Hardening Guides

CIS (*Centre of Internet Security*) benchmarks are designed to help secure systems by offering clear, practical recommendations. They focus on key areas like container images,

runtime environments, and host systems. The CIS Kubernetes Benchmark zeroes in on securing Kubernetes clusters. It includes guidance for critical components such as the API server, worker nodes, and network policies. On the other hand, the CIS Docker Benchmark provides best practices for hardening Docker environments. It covers everything from securing Docker container images to configuring the host and runtime settings properly. [\[14\]](#)

NSA and CISA Hardening Guidance (National Security Agency (NSA) and Cybersecurity and Infrastructure Security Agency) hardening guidance focuses on best practices for securing clusters in the context of national security. Their key recommendations include implementing role-based access control (RBAC), securing API servers and etcd databases, and applying network segmentation while enforcing pod security policies. [\[14\]](#)

RedHat Container Security Guide (Vendor-Specific guide) is focusing on secure deployment of containers on Red Hat platforms like OpenShift. [\[14\]](#)

Kubernetes Security Best Practices (Vendor-Specific guide) was published by Google Cloud as best practices for securing Kubernetes clusters and workloads. [\[14\]](#)

Docker Security Cheat Sheet (Vendor-Specific guide) was published by OWASP (Open Web Application Security Project) to share best practices related to the Docker environment. [\[14\]](#)

Pod Security Standards (PSS) (Vendor-Specific guide) was published by Kubernetes Community has published these guidelines to define and enforce pod security within Kubernetes. [\[14\]](#)

Google Cloud Container Security Whitepaper (Vendor-Specific guide) includes comprehensive security practices for containers deployed in cloud environments, extending the Google Cloud environment. [14]

1.2.2 Common Risks in Container Infrastructure

According to NIST publication 800-190: Application Container Security Guide, six core components pose significant risks within the container infrastructure: container images, registries, orchestrators, container runtime, host operating system and container applications. For some of the found risks, the guide also provides countermeasures to minimise their impact. These details are summarised in Table 1 below. [7] [5]

Table 1: Common risks in container Infrastructure according to NIST 800-190. Table source [7]

Core Component	Major Risks	Countermeasures
Container Images	Vulnerabilities in images; use of untrusted or outdated images.	Regular vulnerability scanning of images, enforce signed images for authenticity, and use trusted repositories only.
Container Registries	Unauthorized access, poorly configured access controls, and untrusted images in registries.	Use access control policies, enforce signed images, and regularly scan stored images in registries for vulnerabilities.
Orchestrators	Misconfigurations leading to privilege escalation, unbounded network access, and lack of node isolation.	Use strong authentication, limit administrative access, and configure orchestration platforms (e.g.,

		Kubernetes) to isolate nodes and control network access.
Container Runtime	Vulnerabilities in the runtime software, misconfigurations, and insufficient monitoring.	Regular runtime vulnerability assessments, apply compliance standards (e.g., CIS Docker Benchmarks), and enforce secure runtime configurations.
Host Operating System	Large attack surfaces, mixing containerized/non-containerized workloads, and lack of isolation between containers.	Use container-specific OSs (e.g., CoreOS) when possible, harden the host OS, and ensure isolation of workloads.
Container Applications	Insecure code, lack of runtime application protection, and exploitation of known application vulnerabilities.	Implement application vulnerability scans, monitor runtime behaviours, and apply security policies tailored for containerized environments.

Some other risks mentioned in the NIST guide do not provide countermeasures and could pose significant challenges. It is important to remember these for further chapters, particularly during the analysis.

Categorisation of Container Security Tools

Container security tools are designed to cover the entire SDLC (software development lifecycle), ensuring that security measures are integrated seamlessly from development to deployment and typically address one or more mitigation phases. The *Container Security Report* by GigaOm highlights the growing importance of dedicated container security tools, especially as cyber threats evolve and become more complex. [\[16\]](#) [\[17\]](#)

Prevention phase, the tool in this phase avoids vulnerabilities or security issues before they manifest. This phase includes vulnerability scanning, policy enforcement (secure configuration), secure image creation, and signing. This phase is usually entirely in the hands of developers. [\[16\]](#) [\[17\]](#)

Detection phase's objective is to identify security issues in real-time or through monitoring. Significant features include threat monitoring, log analysis, and alerting or intrusion detection systems (IDS). [\[16\]](#) [\[17\]](#)

Response phase key is to mitigate or remediate issues once detected, usually via automation, workflows or forensic analysis. [\[16\]](#) [\[17\]](#)

1.3 DevOps

DevOps practices focus on integrating operations and development. The main emphasis is on seamless collaboration between development and operation teams to enhance application integration and delivery.

1.3.1 DevSecOps

DevSecOps protects the agile and automated processes of DevOps from being exploited by attackers by embedding security validation throughout the entire workflow. The approach emphasises integrating security early in the development process and across the entire software development life cycle (SDLC). This shift-left approach of including security in development requires fostering a culture of collaboration and flexibility between development and security teams and incorporating security protocols directly into the development process itself. [\[17\]](#)

1.3.1.1 Six pillars

Provides a framework for integrating security into every phase of the SDLC. Each pillar aligns closely with DevSecOps practices and Continuous Integration/Continuous Deployment (CI/CD) pipelines, ensuring a continuous and integrated software development and deployment aspect. [\[17\]](#)

Collective Responsibility is an approach ensures that security considerations are integrated from the outset and throughout the development process by shared responsibility across all teams involved in SDLC, including development, operations, and security teams. [\[17\]](#)

Collaboration and Integration leverage CI/CD pipeline integration of security tools and practices, enhancing the overall security posture as teams can identify and address vulnerabilities early in the development process. [\[17\]](#)

Pragmatic Implementation of measures that are practical and effective within existing development and operational workflows. [\[17\]](#)

Bridging Compliance and Development by incorporating compliance checks, continuous validation helps ensure that regulatory standards are met without slowing down innovation through CI/CD pipelines. [\[17\]](#)

Automation emphasises the importance of automating security processes to achieve consistent and repeatable security assessments, thereby reducing manual errors and accelerating development. [\[17\]](#)

Continuous Improvement encourages ongoing evaluation, enhancement and refining of security practices. This approach aligns closely with automated security automations achieved by CI/CD pipelines, which introduces trade-offs in the form of overhead and

potential impacts on development speed due to the implementation of runtime monitoring and vulnerability scanning processes. [\[17\]](#)

2 Analysis of the Current State

This thesis chapter analyses an international corporation, examining its internal and external environment, explicitly focusing on its container infrastructure's security and potential threats. The company's state chapter provides an overview of the company's profile to set the stage and provide context for the following analysis. The following section addresses current challenges in the external environment, including an evaluation of common threats and the execution of a risk analysis. The third part focuses on internal threat assessment, incorporating a detailed risk analysis based on the CVE metric. The chapter concludes by synthesising the key findings with a SWOT analysis, providing a foundation for the subsequent chapter.

2.1 Company Introduction

To start with, the introduction of the company itself, enterprise XY, is an international business operating across continents with several decades of history. Its area of entrepreneurship is within many fields, such as ecological materials, location guidance systems, and flight navigation. The infrastructure described in this context supports critical activities, including revenue-generating applications, scientific research computations, and essential operations that ensure the continuous functionality and reliability of the infrastructure.

The analysis and proposed solutions will be developed following the shift-left mindset, focusing primarily on the foundational operations that ensure the seamless functioning of the entire infrastructure.

Containers are elastic and ephemeral and are inherently designed to be more agile and flexible than traditional monolithic applications. The dynamic and lightweight nature of containers introduces unique security risks, which require a different approach for securing containerised applications. These risks need to be addressed at every stage of the application lifecycle.

The company faces significant challenges in aligning its infrastructure and operations capabilities with the growing demand for secure, containerised solutions from its internal customers. In response to these pressures, the organisation has decided on a transformation initiative to modernise its infrastructure while maintaining robust security and compliance. Central to this strategy is developing a purpose-built solution to support containerised applications within a private on-premises cloud while affording the organisation complete control over critical aspects such as data, networking, and security policies. Unlike public cloud options, this approach enables the company to adhere to industry-specific regulatory requirements and optimise performance for internal workloads.

A core challenge lies in the company's reliance on a legacy network architecture characterised by rigid network segmentation. The existing infrastructure is predicated on static VLAN configurations, inflexible firewall rules, and predefined security zones, all of which complicate the integration of modern container orchestration frameworks. Additionally, the security model is predominantly IP-based rather than identity or service-based, limiting the organisation's capacity to effectively adopt zero-trust security principles. These limitations undermine the scalability, and automation benefits typically associated with cloud-native environments.

While containerisation has already been introduced, the organisation is not yet ready for orchestration like Kubernetes due to several key factors. First, network policies and firewall rules are not optimised for dynamic container workloads, requiring extensive reconfiguration before full Kubernetes deployment can be considered. Second, the storage systems and persistent volume management practices are still rooted in legacy technologies, which lack the agility required for scalable, container-native storage solutions. The operational teams are still transitioning to containerised workflows, highlighting the need for further training and process optimisation before the organisation fully embraces the complexities of a Kubernetes-based ecosystem.

Despite these challenges, the company's initiative to develop a solution for containers is a crucial step toward modernising its IT landscape. To bridge the gap between legacy segmentation and modern cloud-native practices, the company is leveraging Ansible for automation, service meshes for enhanced networking, and custom security scanning tools to monitor and enforce compliance. These efforts should lay the foundation for a future Kubernetes adoption, ensuring a smooth transition while maintaining the company's security and operational requirements.

2.2 External Environment Analysis

According to the Stack Overflow survey for 2023, Docker has been one of the most popular technologies amongst professional developers, with more than 58% of votes. [17]

This result from the Stack Overflow survey aligns with a survey conducted by the Cloud Native Computing Foundation (CNCF) that focused solely on cloud consumers to provide a clearer picture of adoption, benefits, and challenges. The study's outputs, concluded by CNCF, imply that Kubernetes is now mainstream, with 66% of cloud consumers using it in production, 18% evaluating, and only 15% of organisations having no plans to adopt it.

The CNCF predicts that Kubernetes will become a foundational component of the global technology ecosystem. This projection is based on its role in improving cloud economics and its critical role in AI development, solidifying its position as a core technology.

Additionally, the study explored challenges in container deployment and usage. As illustrated in the findings in Figure 3 below, a significant proportion of respondents, irrespective of their specific groups, identified security and monitoring as primary pain points. These results suggest that as the scale of a containerised system increases, the complexity of monitoring and observability increases, presenting growing challenges for organisations managing large-scale container infrastructures.

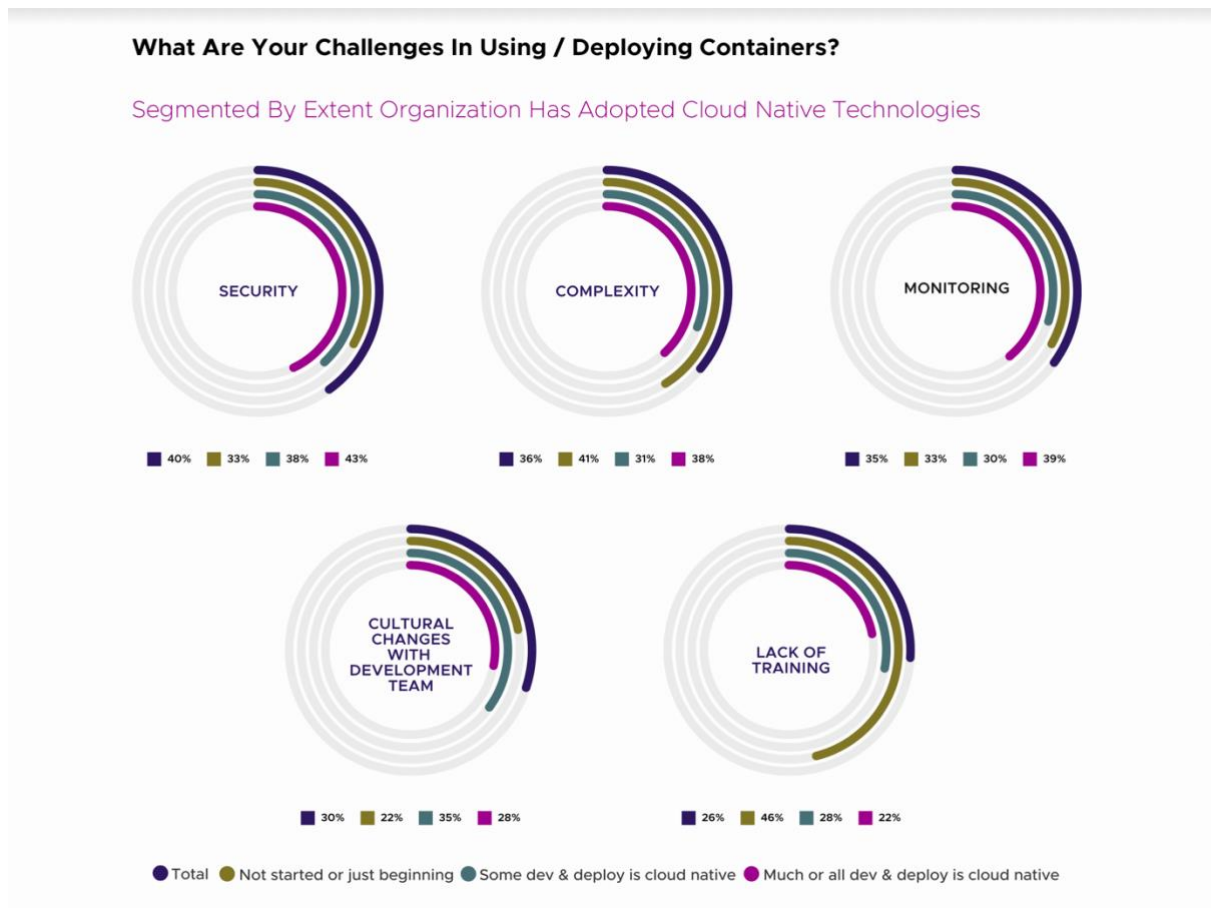


Figure 3: Challenges in Container Deployment. Image source [\[18\]](#)

2.2.1 Common Threats in the External Environment

This subsection evaluates the most pressing threats in the external landscape.

2.2.1.1 Cybersecurity Threats

Attacks

According to the ENISA's (European Union Agency for Cybersecurity) report for 2024 from 3rd December 2024, visualised on Figure 4, these observations can be concluded:

41.1% of the threat landscape is dominated by distributed denial-of-service (DDoS) and Ransom denial-of-service (RDoS) attacks. This domination indicates a significant focus on targeting availability in cloud and online environments. [\[19\]](#) [\[20\]](#)

Ransomware, at 25.9%, remains a critical challenge as it directly impacts data confidentiality and availability.

Other significant threats are data breaches, with 19.1%, highlighting an ongoing challenge in securing sensitive data against unauthorised access or leaks. [\[19\]](#) [\[20\]](#)

Incidents by threat type for the timeframe between July 2023 and June 2024 are shown in Figure 4 below.

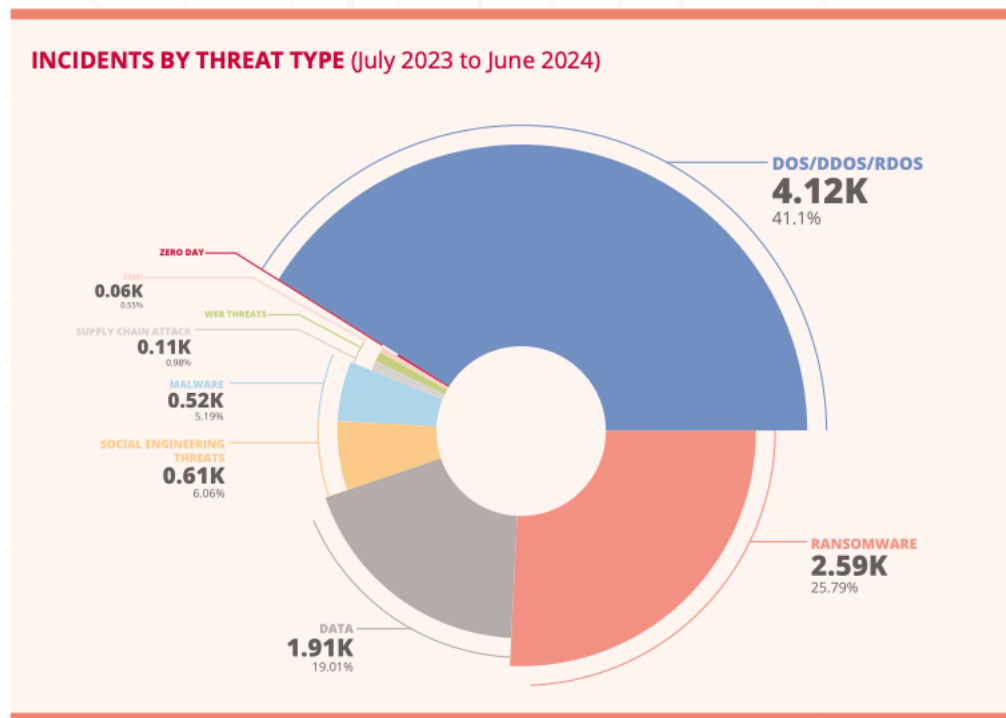


Figure 4: Incidents by threat type. Image source [\[19\]](#)

Overall, the chart emphasises the need for a comprehensive and layered security approach to tackle the diverse range of threats.

Supply chain vulnerabilities targeting orchestration platforms

The feature that makes cloud containers indispensable to modern software development also renders them vulnerable to security threats. Containers rely on open-source

components and shared libraries and can inherit vulnerabilities that may result in an exploit.

A supply chain attack aims to infiltrate an organisation's network by targeting weaker systems in its supply chain, usually third-party software or services that a company uses in operations. [\[21\]](#)

According to the latest Microminder Cyber Security (mcs) report, dated 9th January 2025, supply chain attacks on containers have increased by 430%. [\[21\]](#)

Attackers often target container images, libraries, and dependencies to insert malicious code, which can compromise the entire containerised environment. A notable example is the exploitation of container images in public repositories like Docker Hub, where malicious actors upload infected images that, when deployed, can lead to unauthorised access or control over cloud resources. Images are stored in registries using standardised formats such as the Open Container Initiative (OCI). Research indicates that 87% of container images exhibit high-severity flaws. [\[21\]](#)

2.2.1.2 Geopolitical Risks

Geopolitical tension often escalates cybersecurity threats, particularly in containerised and virtualised environments that depend on interconnected global networks.

Trade wars

The rise in tech rivalries risks limiting access to specific containerisation tools, virtualisation technologies, or cloud providers due to trade restrictions, which may result in vendor lock-in, where the organisation is forced to rely on a specific provider or technology. Vendor lock-in can significantly reduce flexibility, hinder innovation and system interoperability, and increase long-term costs. [\[22\]](#)

Data sovereignty requirements

Evolving sovereignty regulations that enforce geographic restrictions on data storage pose significant challenges for organisations relying on virtualised environments.

Containerisation offers an isolation for data and applications, creating a space for better compliance with regional laws. However, successful implementation requires robust orchestration to address the complexities of varying jurisdictional requirements while maintaining data security and accessibility. [\[22\]](#)

Regional spread and segmentation according to geographic restrictions necessitate a sophisticated orchestration system to ensure compliance with diverse regional regulations for containerised environments. Non-compliance exposes organisations to legal penalties and reputational damage. Furthermore, adapting to localised compliance infrastructure can increase infrastructure costs and reduce operational efficiency. [\[23\]](#)

2.3 Internal Environment Analysis

This section examines the internal threats inherent in a containerised compute environment. The analysis focuses on identifying and evaluating security risks within the system, including those stemming from misconfigurations, unpatched images, insider threats, and insecure dependencies.

The environment analysis chapter predefined that the changes will be made primarily in the scope of containers in a private cloud on-premises infrastructure. Therefore, it is essential to define assets further for the upcoming analysis.

2.3.1 Asset definition

The environment comprises various hardware, software, data and policy components across different owners. In general, it can be divided into five parts:

1. Core Infrastructure and Hosting Environment

The container host servers, managed by the Cloud Admin team, run containers on physical servers. Security measures include host hardening, patch management, and least privilege access policies.

The container runtime, such as Docker or containerd, is the responsibility of the DevOps team and serves as the engine that executes containerised workloads. As things stand, no security controls have been applied.

Containerised applications, managed by application owners, represent the workloads deployed in the environment. Their security is ensured by static and dynamic analysis, dependency scanning, and Role-Based Access Control. In general, there are no centrally managed security controls targeting deployed applications from either the DevOps or Security teams' perspectives.

2. Image Storage and Management

Container images, managed by the DevOps and Security teams, are the core artifacts used in deployments. Image signing protects their integrity.

The DevOps and Security teams also manage the container registry, which stores and distributes container images. At the moment, there are no security controls in place.

3. Security and Compliance Monitoring

The container security scanning tools are not fully centrally deployed and are not operated by the Security team, but are distributed and utilised by product teams.

The container security scanning tools are owned by the Security team, but are not centralised and operated by the product and development teams. These tools enforce security through regular vulnerability scans, image signing, and policy enforcement. Controls include setting CVE thresholds and blocking images that fail to meet compliance standards.

Runtime security monitoring, managed by the Security team, identifies potential threats during container execution. It relies on behavioural analysis and ensures the isolation of suspicious containers.

IT Operations handles a centralised log and alerting system (SIEM) to collect and manage security alerts. This system provides log integrity and anomaly detection and implements RBAC for secure access.

Security event logs, monitored by the SIEM security team, track security incidents. These logs support threat detection through correlation and the generation of anomaly alerts.

The security team maintains compliance and audit logs, which are protected by log integrity checks and subject to strict retention policies.

4. CI/CD Pipeline and Automation

The CI/CD pipeline, managed by the DevOps team, automates code deployment into production. Its security is reinforced through automated scanning, secrets management, and code signing practices.

The DevOps team owns infrastructure as Code (Iac) templates, facilitating infrastructure deployment automation. These templates undergo regular code reviews and are secured through RBAC and Git-based access controls.

5.Data Protection and Disaster Recovery

IT Operations maintains the backups and disaster recovery systems to ensure recovery readiness. Backups are encrypted, stored in immutable formats, and form part of a broader disaster recovery plan. For a table view, see Table 2 below.

Table 2: Asset definition

Name	Owner	Asset Type	Location	Criticality Level	Security Controls Applied
Container Host Servers	Cloud Admin	Hardware	On-Prem Cloud Servers	High	Host Hardening, Patch Management, Least Privilege Access
Container Runtime (Docker, containerd)	DevOps	Software	Container Host Servers	High	None
Container Images	DevOps / Security Team	Software Artifact	Container Registry	High	Image signing
Containerized Applications	Application Owners	Software	Running on	High	Application Security Testing (SAST, DAST),

			Containers		Dependency Scanning,
Container Registry	DevOps / Security Team	Software	Registry Servers	High	None
Container Security Scanning Tool	Security Team	Software	On-Prem Security Servers	High	None
CI/CD Pipeline (GitHub)	DevOps Team	Software	On-Prem DevOps Servers	High	Secrets Management, Code Signing
Scanning Pipeline Servers	Cloud Admin	Hardware	On-Prem Cloud Servers	High	Endpoint Security, Network Segmentation, Access Controls
Image Scanning Policies	Security Team	Policy	Security Pipeline	High	None
Runtime Security Monitoring	Security Team	Software	On-Prem Security Servers	High	Behavioural Analysis
Log & Alerting System (SIEM)	IT Operations	Software	On-Prem Logging Servers	High	Log Integrity, Anomaly Detection, RBAC

Configuration & Secrets Management	Security Team	Software	Secure Vault	High	Encryption, Least Privilege Access
Infrastructure as Code (IaC) Templates (Ansible)	DevOps	Software	Secure Repository	Medium	Code Reviews, RBAC, Git Security
Compliance & Audit Logs	Security Team	Data	SIEM / On-Prem Logs	High	Log Integrity, Retention Policies
Application Logs	IT Operations	Data	On-Prem Log Storage	High	Log Retention Policies, Encryption
Security Event Logs (SIEM)	Security Team	Data	On-Prem Security Servers	High	Log Correlation, Threat Detection, Anomaly Alerts
Backups (Images, Configs, Logs)	IT Operations	Data	On-Prem & Offsite Storage	High	Encrypted Backups, Immutable Storage, Disaster Recovery Plan
Disaster Recovery Systems	IT Operations	Software	Offsite Backup Location	High	Regular Testing, Automated Recovery Workflows

2.3.2 Methodology

The methodology involves leveraging the Common Vulnerabilities and Exposures (CVE) parameter to assess potential security risks associated with containerised environments. By mapping these risks to known vulnerabilities, the analysis seeks to provide a comprehensive understanding of the threats.

The outcomes of this analysis will detail the severity, impact, and frequency of occurrence of these vulnerabilities, thereby offering a robust framework for assessing security risks in the containerised compute environments.

Steps:

1. Identifying vulnerabilities.
2. Assess severity based on CVEs, common vulnerability scoring system (CVSS), and EPPS percentile and likelihood of exploitation.
3. Pick the top 15 depending on severity and frequency of occurrence.
4. Risk assessment based on severity and impact of CVEs.
5. Implications for the further parts of a thesis.

2.3.3 Identifying Vulnerabilities

The initial analysis phase involves theoretical mapping of potential threats and identification of potential actors, followed by applying scanning tools to thoroughly examine the environment.

Container runtime security threat mainly consist of vulnerabilities in runtime software that remain unpatched. It can be in a form of an escalation attack originating from within the container targets the host system.

The base image security threat is the presence of malware embedded within unpatched container images with known vulnerabilities.

Secrets management threat is the use of hardcoded credentials or inadequately stored secrets, such as those in environment variables. Exposure of secrets via container logs or insecure storage mechanisms.

Application dependencies can pose security threats by leveraging outdated libraries or frameworks, which can expose the application to known vulnerabilities. Pulling dependencies from untrusted or unofficial repositories might include malicious code or a backdoor. Or occurrence of unknown transitive dependency vulnerabilities due to not utilising the correct tools for dependency tracking.

A networking security threat can be a misconfiguration or network policies that result in unauthorised inter-container communication, data leaks between tenants, or unintended exposure of internal services to external entities. Exposed management interfaces mainly use APIS to the public or, in general, untrusted networks.

Identity and access management (IAM) security threat can have the form of overprivileged roles or misconfigured role-based access control (RBAC) settings. Insufficient separation of duties among tenants within the IAM framework.

Runtime configuration security threat occurs in cases when containers are run with elevated privileges, giving them full access to the host system. Host directories are mounted without restrictions, allowing access to paths like /etc/shadow and the possibility of overwriting sensitive data.

Thread actors

External attackers typically target exposed APIs or vulnerabilities to gain unauthorised access. Exploiting container vulnerabilities for privilege escalation and lateral movement within the network.

Insiders are employees or contractors who misuse their elevated privileges, or accidental misconfigurations, resulting in security breaches. These breaches usually have the most significant impact, therefore educating employees and implementing best practices on environmental cooperation is crucial.

Automated threats are malware, bots or service accounts taking advantage of default configurations or unpatched vulnerabilities.

2.3.4 Severity and likelihood of exploitation assessment

The assessment will be concluded by scanning the private cloud infrastructure with the accessible scanning tools. Currently, Trivy, Checkov, GitHub Dependabot, and custom scripts are available for use.

Found vulnerabilities will be examined based on CVSS severity score, type of impact, likelihood of exploitation, and EPSS percentile. Then mapped to the vulnerability groups per vulnerability type category. Further assessment will prioritise vulnerabilities based on these benchmarks: frequency of occurrence within the environment, exploitability score, severity score and overall likelihood of exploitability. [\[24\]](#)

CVSS Score (Common Vulnerability Scoring System) scoring system [\[24\]](#):

- Critical (9.0–10)
- High (7.0–8.9)
- Medium (4.0–6.9)
- Low (0.1–3.9)

EPSS (Exploit Prediction Scoring System) is a data-driven model designed to assess the likelihood that a Common Vulnerability and Exposure (CVE) will be exploited within 30 days. Moreover, a percentile that shows how vulnerability compares to the rest regarding exploitability probability. [\[25\]](#)

Threat Intelligence Sources aggregates data from real-world exploit activity, malware feeds, honeypots, and cyber threat reports. Machine Learning Models use historical CVE exploitation data to predict which vulnerabilities will likely be targeted. Furthermore, scoring Output generates a probability score between 0 and 1, where:

- **0.00** = Extremely unlikely to be exploited.
- **1.00** = Almost certain to be exploited.

The EPSS can be obtained in two ways: either by directly downloading the file from their website or by leveraging their API and automating it to add them to the CVES found in the scans. [\[25\]](#)

The Figure 5 shows the JSON output from the Api call with the specific CVE as a parameter in the endpoint URL. The example API call is shown in the image below.

```
{
  "status": "OK",
  "status-code": 200,
  "version": "1.0",
  "access-control-allow-headers": "x-requested-with",
  "access": "public",
  "total": 1,
  "offset": 0,
  "limit": 100,
  "data": [
    {
      "cve": "CVE-2024-21626",
      "epss": "0.015620000",
      "percentile": "0.870710000",
      "date": "2025-02-09"
    }
  ]
}
```

Figure 5: Example API call for CVE metrics retrieval

2.3.4.1 Base images threat assessment

The vulnerabilities in container images reveal significant security concerns. Many identified vulnerabilities have a high severity level, meaning they can cause serious issues such as privilege escalation, unauthorised access, host system compromise, and data leakage.

One particularly concerning vulnerability is CVE-2024-23497, which is classified as critical. This vulnerability poses a significant risk due to its potential for privilege escalation via a Linux kernel.

The likelihood of exploitation, measured using the EPSS, varies across vulnerabilities. Some, like CVE-2022-0847 (Dirty Pipe), have a high percentile score (0.91), indicating a greater chance of exploitation. Others have a much lower likelihood, reducing the immediate risk but still requiring monitoring.

Several vulnerabilities are specific to infrastructure components. For example, CVE-2024-21626 and CVE-2021-28475 highlight ongoing security issues with container runtime components like `runc` and the image build process. Findings suggest that containerised infrastructures remain exposed, necessitating improved security measures. Details are shown in Table 11: Container image threat assessment in [Appendix](#).

2.3.4.2 Runtime threat assessment

Many vulnerabilities related to container runtimes have a high or critical severity level, emphasising their significant risks. Notable examples include CVE-2021-36222, CVE-2021-30810, and CVE-2019-5736, which are classified as critical. These vulnerabilities can lead to severe consequences such as container escapes, arbitrary code execution, and unauthorised host access.

Privilege escalation remains a recurring issue, as seen in vulnerabilities like CVE-2022-0811, CVE-2019-14287, and CVE-2020-14386. This trend underscores the need for stronger access controls within containerised environments to prevent unauthorised privilege elevation.

The likelihood of exploitation varies among vulnerabilities. CVE-2019-14287, a `sudo` vulnerability, has the highest EPSS percentile (0.9761), indicating an urgent need for mitigation. Similarly, CVE-2021-36222 also has a high likelihood of exploitation (0.8793), reinforcing the importance of timely patching.

The range of attack vectors affecting container runtimes is diverse. Some vulnerabilities, such as CVE-2020-8558, involve network-level packet injection, while others, like CVE-2018-15664, expose filesystem weaknesses that allow arbitrary writes. More details are shown in Table 12: Runtime threat assessment in [Appendix](#).

2.3.4.3 *Secrets Management threat assessment*

Several vulnerabilities in this category have been identified with either high or medium severity, underscoring the risks associated with exposed secrets and unauthorised access. CVE-2020-13692 and CVE-2021-33001 stand out due to their high severity and potential impact.

Unauthorised access to sensitive data is a recurring issue within this category. Vulnerabilities such as CVE-2020-13692, CVE-2022-29154, and CVE-2021-25741 highlight the risks of data leaks and privilege escalation resulting from misconfigurations or the use of hardcoded secrets.

Infrastructure misconfigurations contribute to security risks, such as improperly configured environment variables.

Furthermore, the evolving threat landscape for secrets management includes diverse attack vectors. Examples include hardcoded secrets in Helm charts and vulnerabilities in misconfigured Vault storage. More details can be found in Table 14: Secrets threat assessment in [Appendix](#).

2.3.4.4 *Supply Chain threat assessment*

Most identified vulnerabilities fall within the high or critical severity categories, signifying their substantial threat from the supply chain. CVE-2021-44228 (Log4j2) and

CVE-2022-22965 (Spring4Shell) are rated as critical, demonstrating their capability for remote code execution and widespread compromise across the supply chain ecosystem.

Log4Shell and Spring4Shell are characterised by exceptionally high EPSS percentiles, registering at 0.9989 and 0.9998, respectively. A percentile above 0.9 indicates that these vulnerabilities are most likely to be exploited and necessitate immediate and prioritised remediation measures due to their significant potential impact and extensive exploitation risks.

The identified vulnerabilities encompass diverse impacts, including remote code execution (e.g., CVE-2020-11979, CVE-2021-44228), image tampering (e.g., CVE-2020-8594), dependency compromise (e.g., CVE-2021-25124), and privilege escalation (e.g., CVE-2021-4034). Particular vulnerabilities, such as CVE-2021-30810 (affecting supply chain workflows) and CVE-2022-31478 (impacting package repositories), highlight the prevalence of infrastructure misconfigurations. More details are shown in Table 13: Supply chain threat assessment in [Appendix](#).

2.3.5 Risk Assessment of found vulnerabilities

The analysis so far has identified several areas that require attention to enhance the security of containerised environments. A good starting point is to examine the risks in the current setup closely. From there, the proper steps can be planned and implemented to manage those risks effectively. Prioritisation guides the allocation of resources and the formulation of mitigation strategies.

The cybersecurity risk management matrix will help guide the process of prioritising vulnerabilities. It is a categorising system for organising and managing cybersecurity risks by assessing various dimensions of cyber risks, including internal and external vulnerabilities, the threat landscape, the impact of exposure, and the likelihood of exploitation. This matrix helps decision-makers allocate resources wisely and put the right protective measures in place.

As shown in Table 15: Cyber Security Risk Management Matrix in [Appendix](#), several assets exhibit an high initial RPN (Risk Priority Number), which is the outcome of multiplying the probability and consequence. Assets that indicate a high RPN include Container Infrastructure, External Cyber Threats, and Public Registries.

Matrix outlines mitigation strategies that align with industry best practices such as:

- Image signing for Containers to ensure integrity and authorisation.
- Enforcing least privilege to limit unauthorised access.
- Implementation of automated compliance, auditing, and encryption.

In the post-mitigation phase, most residual risks were reduced to an acceptable level (≤ 8). However, some risks persist at this level, particularly in Container Networking, due to evolving attack vectors and external cyber-threats, where new vulnerabilities emerge rapidly.

Analysis indicated that the identified risks primarily impact data integrity, confidentiality, and availability and do not pose a direct safety risk. However, cybersecurity risks with the same measures as cloud have been considered a safety risk in critical infrastructure.

Further enhancement of security posture requires additional focus in the following areas:

- Strengthen CI/CD Pipeline Security with automated code scanning, image build scanning, signed commits.
- Enable container networking protection to mitigate network-layer threats.

Several challenges, primarily of a financial nature, must be considered during the process. Resource constraints arise, as implementing advanced security measures typically requires specialised expertise and significant financial investment. Compliance costs are another factor, as meeting regulatory requirements necessitates additional auditing and ongoing compliance efforts, which can be costly. Workflow adjustments may be necessary, as incorporating trusted registries and enhanced container image management may necessitate modifications to workflows and alignment with DevOps practices.

2.4 Implications

This chapter briefly concludes the analyses and provides a foundation for the solution, stating directions and alignment with company initiatives and goals.

2.4.1 *Implications from external analysis*

The complexity of containerised environments necessitates enhanced observability and monitoring frameworks to mitigate organisations' large-scale deployments. Organisations implement robust security protocols, including runtime protection, image scanning, and secure configuration management, to ensure container integrity and resilience against emerging threats.

There has been a significant increase in cybersecurity threats, particularly DDoS, RDoS, and ransomware attacks. There has also been a surge in supply chain attacks on containerised environments, with a high prevalence of image vulnerabilities (87% exhibiting critical flaws).

2.4.2 *Implications from internal analysis*

The company's goal is to modernise on-premises cloud infrastructure as a foundation for adopting a more dynamic Kubernetes approach in the future, aligning with the newest market standards.

An internal environment analysis has identified several priority disturbances.

The vulnerabilities demonstrate the wide range of attack vectors that threat actors can exploit. These include container escape, resource abuse, and supply chain compromises. As a result, it is crucial to implement strong security practices throughout the entire container lifecycle, from image building to storage and distribution. The company should reinforce the importance of correctly implementing security measures across CI/CD pipelines and orchestration platforms.

In conclusion, this analysis underscores the importance of proactive security strategies in containerised environment. Company should prioritise patching, continuous vulnerability minimisation, and regular security controls to minimise risk and maintain a secure infrastructure.

2.4.3 SWOT analysis

The insights from the previous chapters have been built into a SWOT analysis that shows clear directions, strengths, and threats. The SWOT analysis is shown as a Table 3 below.

Table 3: SWOT analysis

POSITIVE	NEGATIVE
Strength	Weaknesses
Segmented infrastructure	Manual scanning for vulnerabilities
Granular access	Manual policy enforcing
Awareness of security importance	Overly complicated setup
Opportunities	Threats
Regular Automated Reporting (Audit framework)	Human Error
Implementing automation for scanning and policy enforcement	Supply Chain
Securing container images	Increased Attack Surface

Notable weaknesses have been identified, especially the heavy dependence on manual processes to scan for vulnerable infrastructure. Relying on people to do this work by hand

not only slows things down but also increases the risk of mistakes and can undermine all previous security efforts.

The company faces significant threats, including human error, vulnerabilities in the supply chain, and an expanded attack surface, all of which can undermine its overall security resilience.

The organisation demonstrates several strengths contributing to a good predisposition for a stout security posture, including segmented infrastructural access control mechanisms and organisationally appropriate security principles.

Integrating automated vulnerability scanning and policy enforcement, which would reduce unnecessary manual effort, can offer improvement opportunities. Automating reports within an audit framework could also make it easier to stay on top of compliance and improve overall visibility. Implementing safer image practices could help minimise chain threats, represents an area of potential advancement.

3 Solution Proposal

Based on the previous chapters and thorough analyses of both external and internal environments, which highlighted threats and weaknesses and served as proof to supply areas for improvement, the solution will be proposed in this chapter.

The proposed solution will be implemented using a structured approach. To enhance security monitoring and compliance, an automated vulnerability scanning and policy enforcement system will be deployed. In parallel, a container image hardening process and a reusable example pipeline will be introduced to facilitate users' adoption.

3.1 Automated scanning solution for vulnerabilities and policy enforcement

Regular scanning is crucial since newly created or updated images may introduce vulnerabilities, and previously scanned images can become susceptible as new security issues emerge. Scanning should maintain a proactive security posture rather than relying on image pulls or application deployment, and ensure ongoing threat detection.

This solution will aim to regularly scan containers for known vulnerabilities using automated tools alongside policy enforcement and establish a patch management process for vulnerabilities discovered.

The scanning solution will be conducted within the private GitHub repository with the tools: Ansible, Tanium, Trivy, Copa.

3.1.1 Architecture

This section holistically describes the highest overview of the proposed solution for the found issues in the infrastructure.

Figure 21: High-level overview of the automated scanning solution in [Appendix](#), displays the simplified architecture proposal of the container scanning and policy enforcement solution.

The solution is divided into five sequential phases.

1. Host discovery
2. Scan
3. Remediation
4. Post-remediation scanning
5. Reporting

In the first phase, host discovery involves connecting via the Tanium server to the Tanium clients across the segments, leveraging the Tanium API to retrieve important information about the servers and save it in a file in the runtime artifacts.

It filters the information about the servers that have the containers deployed and retrieves contact information about the asset owner, asset address, and network segment.

The second phase scan retrieves the artifact's data, creating a list for the Ansible playbook.

Then, it connects to the jump host with the Ansible control node and performs the vulnerability and policy scan on the servers.

After this scan, the Trivy and policy scan reports are saved into the runtime artifact.

The third remediation phase is retrieving data from the saved reports and utilising Copa and a custom script to patch images.

Fourth phase post-remediation scanning is running the scan for vulnerabilities again, saving the findings into the runtime artifacts.

Fifth phase reporting is conducting the difference operation to compare the first scan results with the second scan after remediation, calculates the patch success rate and sends out communication to the owners with the vulnerability report and to the DevOps and security team about the success of the patching with the necessary manual actions if needed.

3.1.2 Solution Design

This part outlines the solution design for a five-phase process aimed at host discovery, vulnerability scanning, remediation, post-remediation scanning, and reporting. The solution leverages Tanium, Ansible, Trivy, and Copa for automation and security assessments. Each phase has been structured to ensure vulnerabilities are detected, addressed, documented systematically and organised.

Host discovery

The Tanium server and client have been chosen for host discovery as this technology has already been adopted and is ready to use. The Tanium server uses a custom sensor to detect the presence of containers. During this process, the system retrieves important asset details, including server information such as IP Address, MAC Address, Operating System, Network Interfaces, and Subnet Mask. Additionally, it collects the asset owner's contact details and identifies the network segment to which the server belongs. The following Figure 6 is an example of a declaration of a sensor:

Name: Container Presence
Category: Custom – Containers
Description: Detect whether a system is running container
Result type: Boolean (TRUE/FALSE)

Figure 6: Sensor declaration example

After gathering this data, it is stored in runtime artifacts for use in subsequent phases. The Tanium sensor is created with specific configurations to ensure effective recognition and easy reuse. The sensor, Container Presence, is categorised under Custom – Containers and is designed to return a Boolean result (TRUE/FALSE) indicating whether a system is running a container. The subsequent Figure 7 of the code snippet is an example of a sensor verification script:

```

$ query.sh
1  #!/bin/bash
2  if pgrep -x "dockerd" > /dev/null || pgrep -x "containerd" > /dev/null; then
3      exit 0 # Returns True (0 exit status)
4  else
5      exit 1 # Returns False (non-zero exit status)
6  fi
7

```

Figure 7: Sensor verification script

To achieve this, the sensor executes a shell script that checks whether dockerd or containerd is running on the machine. The following Figure 8 is an example of a Tanium query used for this purpose:

```

Get Computer Name, IP Address, MAC Address, Operating System,
OS Platform,

Network Interfaces, Subnet Mask, Running Processes, Running
Services,

Container Name, Container Status, Owner Email Address

from all machines

where Container Presence = True

```

Figure 8: Example of a Tanium query

Once the Tanium API retrieves the results, they are processed into JSON format for further use in the next phases of the solution.

Scan

The scanning phase is resource intensive as it requires authentication, trust establishment, and the execution of Ansible playbooks to perform vulnerability and policy scanning.

Establishing trust is complex, as the environment is heavily segmented. Therefore, the trust may require a different setup. Luckily, the previous step provided enough information to prepare the playbook for Ansible accordingly and recognise beforehand

not just the segment and prepare necessary authorisation and authentication steps, but also at the OS level. Therefore, the connecting protocols used, and the version of scanning scripts can be distributed accordingly.

Once trust is established, the system executes the Ansible playbook, which performs two key functions:

- Vulnerability scan using containerised Trivy
- Policy compliance check

Results are then stored as runtime artifacts for later analysis and reporting.

Execution workflow

The vulnerability scan phase begins by verifying whether containerised Trivy is located on the server. Suppose Trivy is not present or is outdated. In that case, the system ensures it is installed or updated to the latest version by pulling a hardened image from the Private GitHub OCI artifactory before proceeding. The following Figure 9 is an example of how the hardened image can be pulled from a private GitHub OCI repository:

```
echo "$GITHUB_PAT" | docker login ghcr.io -u "$GITHUB_USERNAME" --password-stdin
docker pull ghcr.io/$GITBUB_ORG/trivy-hardened:1.01
```

Figure 9: Image pull from repository

Containerised Trivy has been chosen because it eliminates dependency issues and compatibility issues. It can be used across different environments and operating systems without installation conflicts, which ensures a predictable and consistent state of Trivy execution. It also ensures better isolation from the host system and reduces the risk of modifying system files or interfering with other applications. As Trivy runs inside a container, it can be run as an ephemeral container. Ephemeralization of the container means it is automatically removed once the container is no longer needed, leaving no

residual files on the system. The following snippet displayed in Figure 10 of the code is an example usage of the containerised, ephemeral Trivy to scan local images:

```
# authenticate to GHCR
echo "$GITHUB_PAT" | docker login ghcr.io -u "$GITHUB_USERNAME" --password-stdin
# pull the hardened Trivy Image, this phase is not always necessary
docker pull $TRIVY_IMAGE
# run hardener Trivy as ephemeral container to scan
docker run --rm $TRIVY_IMAGE image nginx:latest
```

Figure 10: Example scanning tool usage

The vulnerability scan is then run right after local images have been identified and stored in a variable through which Trivy scan performs a loop, and stores reports in the following format: <image_name>-<tag>-IP-before.json. Locally, uploaded to the runtime as artifacts and deleted from local storage. The Figure 11 below shows an example of a Trivy report exported into the terminal.

```
python:3.4-alpine (alpine 3.9.2)
=====
Total: 37 (UNKNOWN: 0, LOW: 4, MEDIUM: 16, HIGH: 13, CRITICAL: 4)
```

LIBRARY	VULNERABILITY ID	SEVERITY	INSTALLED VERSION	FIXED VERSION	TITLE
expat	CVE-2018-20843	HIGH	2.2.6-r0	2.2.7-r0	expat: large number of colons in input makes parser consume high amount... -->avd.aquasec.com/nvd/cve-2018-20843
	CVE-2019-15903			2.2.7-r1	expat: heap-based buffer over-read via crafted XML input -->avd.aquasec.com/nvd/cve-2019-15903
libbz2	CVE-2019-12900	CRITICAL	1.0.6-r6	1.0.6-r7	bzip2: out-of-bounds write in function BZ2_decompress -->avd.aquasec.com/nvd/cve-2019-12900
libcrypto1.1	CVE-2019-1543	HIGH	1.1.1a-r1	1.1.1b-r1	openssl: ChaCha20-Poly1305 with long nonces -->avd.aquasec.com/nvd/cve-2019-1543
	CVE-2020-1967			1.1.1g-r0	openssl: Segmentation fault in SSL_check_chain causes denial of service -->avd.aquasec.com/nvd/cve-2020-1967
	CVE-2021-23840			1.1.1j-r0	openssl: integer overflow in CipherUpdate -->avd.aquasec.com/nvd/cve-2021-23840

Figure 11: Example Trivy output. Image source [\[26\]](#)

In parallel, the policy compliance scan is executed. This scan evaluates key security aspects, including:

- Root access enforcement ensures that no containerised application operates with root privileges.
- Open ports detection focuses on identifying and flagging non-compliant open ports to prevent unauthorised access or communication.
- Running processes as root is prohibited to ensure that no processes execute with root privileges inside containers, reducing the attack surface and risk of privilege escalation.
- Disk usage monitoring involves tracking and identifying excessive disk consumption to prevent system resource exhaustion and ensure optimal performance.
- To detect and mitigate unauthorised modifications that could compromise the system, file integrity checks are performed on critical files such as `/etc/passwd`, `/etc/shadow`, and `/etc/hosts`.
- Verification of sudoers file permissions ensures that the file granting elevated privileges to users adheres to strict security policies and permissions.
- CIS compliance checks validate container configurations against the CIS Docker benchmark with Docker Bench for security script, which concludes checks for common best practices around the deployment of containers in production. This aims to maintain best practices and align with standards for secure container deployments. [\[27\]](#)
- Host filesystem protection safeguards sensitive host files by preventing unauthorised container access, ensuring the integrity and security of the host environment.

Once the compliance check is completed, a policy violation report is generated and stored locally in the format `policy-violation-IP-before.json`. It is then uploaded to the runtime as artifacts and deleted from local storage. The same script also enforces the policy.

Remediation

The remediation phase addresses vulnerabilities identified in the previous phase by efficiently patching container images. Its aim is to mitigate the majority of detected security flaws to maintain the system's integrity.

The remediation process begins with retrieving the scan reports. Scan reports are analysed directly in the CI/CD pipeline using Copa, which helps to prioritise the vulnerabilities for remediation. The report created by Copa serves as input for the patching commands on the designated endpoints.

The following example command displayed in Figure 12 is used for this analysis:

```
copa analyze --trivy-report python-3.9-<IP>-before.json --output copa-report-python-3.9-<IP>.json
```

Figure 12: Copa example command

Before proceeding with patching, the system must have Copa installed and BuildKit enabled. BuildKit is a native Docker builder that works with Docker and Podman and can build on both Linux and experimental scope support for building Windows containers (WCOW).

Once the environment is validated and the necessary setup is finished, identified vulnerabilities are patched using Copa, and then the container image is tagged accordingly. Copa is applying patches to a specific image, based on a vulnerability report, and then producing a patched image. It will attempt to apply fixes based on the vulnerabilities listed in the report, including upgrading packages or switching to more secure versions of the base image. At first, it tries to retrieve the hardened image from the private OCI registry. If the image is absent, it retrieves the image from the source, usually Docker Hub.

The following example in Figure 13 shows how Copa would patch with the image from a private repository:

```
copa patch -i ghcr.io/$GITHUB_ORG/python-hardened:1.01 -r copa-report-python-3.9-<IP>.json -t python-3.9-patched -a buildx://buildx
```

Figure 13: Copa patch command private repository

Moreover, this example in Figure 14 shows how Copa would patch the image from Docker Hub:

```
copa patch -i docker.io/library/python:3.9 -r copa-report-python-3.9-<IP>.json -t python-3.9-patched -a buildx://buildx
```

Figure 14: Copa patch command external repository

In both cases, the previous step for authentication to the registries is the same, as shown in the retrieval of the Trivy image. After patching, the image is pushed to the registry.

The final check of this step ensures that the patched container image is present and fully functional before concluding that the operation is successful. If this step fails, it first tries to pull the image from the private registry 2 times in 10-minute intervals, as this may be an issue of overload of traffic. If the issue persists, an immediate report is sent to the DevOps team for action.

Post-remediation scan

A secondary scan is conducted to confirm and express by a quantitative metric that the process was successful and verify whether the vulnerabilities and policies have been effectively addressed. This phase is essential for validating the effectiveness of the patching process.

During this phase, Trivy and policy scans are used again to scan patched container images. The results are stored in the runtime artifacts named `<image_name>-<tag>-IP-after.json` and `policy-violation-IP-after.json` for the final reporting phase.

Reporting

The final phase of the solution generates reports and calculates the overall success rate of the patching process. These reports are essential for tracking security improvements and identifying any remaining vulnerabilities that require further action.

Reports `<image_name>-<tag>-IP-before.json,` `policy-violation-IP-before.json` and `<image_name>-<tag>-IP-after.json,` `policy-violation-IP-after.json` are compared with a custom script to determine the effectiveness of the remediation.

Comparison script analyses Trivy vulnerability scan reports by loading JSON data, extracting vulnerability details, and comparing scan results before and after patching. It identifies fixed vulnerabilities by computing the set difference between the vulnerability IDs in the `before` and `after` reports. Specifically, it constructs dictionaries mapping `VulnerabilityID` to `PkgName`, then determines resolved issues using `set(before_vulns.keys()) - set(after_vulns.keys())`. The script calculates the remediation success rate, outputs vulnerability counts and lists fixed issues and their affected packages. The script calculates the success rate of patching with the [Formula 3.1](#):

$$\text{Success Rate} = \left(\frac{\text{Fixed Vulnerabilities}}{\text{Total Vulnerabilities Before}} \right) \times 100 \quad (3.1)$$

This formula can be translated into script as shown in the example show in Figure 15:

```
success_rate = (fixed_count / total_before * 100) if total_before > 0 else 0
```

Figure 15: Success rate formula in code form

The success rate formula ensures that the percentage of fixed vulnerabilities is calculated if there were vulnerabilities before patching (`total_before > 0`). If there were no vulnerabilities before patching (`total_before = 0`), it avoids division by zero and sets the success rate to 0%.

Finally, the script prints the counts of vulnerabilities before and after patching, the number of fixed vulnerabilities, the success rate, and a list of the fixed vulnerabilities and their associated packages.

The policy script is calculated similarly, calculating the occurrence of compliance breaks before and after the enforcement and patch. This conducts an overall compliance ratio of the whole infrastructure. Both reports are sent to the DevOps team and Security.

Vulnerability scan reports from after patching are sent to the asset owners based on the information provided from the initial host discovery scrape. In the reports, only information connected to the asset they own is included.

3.1.3 Implementation

This section provides an overview of the deployment strategy, detailing the setup within the GitHub pipeline and configuration of rules, policies and compliance baselines. Additionally, it outlines the timeline for implementation and orchestration.

The solution will be deployed in a private, on-premises cloud environment, focusing on scanning deployed containers for security vulnerabilities and compliance violations. This scanning process will complement the existing security team's efforts, which include scanning for misconfigurations and image vulnerabilities.

The GitHub actions pipeline will automate the scanning and enforcement of security policies for containerised environments. Process includes:

- Scheduled scans and policy enforcement detecting vulnerabilities and non-compliant container configurations and enforcing security policies.
- Artifact generation and management where pipeline generate, store artifacts such as scan reports, compliance logs, and conduct local data handling. These artifacts will store vulnerabilities for further analysis, retain compliance reports to meet regulatory and security audit and provide historical data for trend analysis and policy adjustments.

Implementation timeline

The implementation followed an Agile Scrum methodology, ensuring flexibility and continuous improvement. The project was divided into six two-week sprints, each delivering incremental progress towards the final deployment.

Sprint 1-2: Proof of Concept (Weeks 1-4)

The project began by setting initial scanning and policy enforcement in a controlled test environment, with Trivy and Copa deployed to conduct vulnerability and policy scans. Detection capabilities and reporting accuracy were validated to ensure the tools functioned effectively. A retrospective was conducted to gather insights, and adjustments were made to the backlog for improvement.

Sprint 3-4: Pilot Deployment (Weeks 5-8)

The pilot deployment occurred in the following two sprints, weeks five to eight. Scanning tools were deployed on a limited number of production containers, and policy enforcement was tested on real workloads to evaluate practical applicability. During this phase, scanning results were also integrated into the GitHub Actions pipeline alongside the artifacts storage. Stakeholder feedback was collected, and policies were refined based on the pilot findings.

Sprint 5-6: Full GitHub Pipeline Integration (Weeks 9-12)

During sprints five and six, covering weeks nine to twelve, full GitHub pipeline integration was planned to be achieved. Prioritising the automation of policy enforcement within the CI/CD pipeline. Scheduled runs were validated to ensure proper execution and artifacts storage functionality. Processes of vulnerability remediation were automated. A mid-project review was held to assess overall progress and make necessary adjustments to the project scope.

Sprint 7-8: Performance and Scalability Testing (Weeks 13-16)

Sprints seven and eight, which covered weeks thirteen and sixteen, prioritised scalability testing. The system's performance impact was assessed during the large-scale scans, and scanning efficiency was optimised to reduce overhead and improve processing times. Benchmarking against industry standards confirmed compliance with expectations for speed and efficiency.

Sprint 9-10: Security and Compliance validation (Weeks 17-20)

Security testing simulated attack scenarios, while compliance was successfully validated against CIS Benchmarks, ISO 270001, NIST, and GDPR standards. A comparison script was used to analyse compliance results before and after enforcement with patching. Stakeholder demonstrations were conducted to gather feedback prior to final deployment, and necessary adjustments were implemented.

Sprint 11-12: Full Deployment and Handover (Weeks 21-24)

The final phase, comprising Sprints 11 and 12 and spanning weeks 21 to 24, was focused on full deployment and handover. The solution was scaled across almost all production environments. However, as is often the case, some exceptions had to be made for business-critical segments, which have been postponed and slowed down the project.

Security and DevOps teams were trained to interpret scan results and respond to enforcement actions and long-term monitoring.

The project lasted almost 30 weeks, considering the autonomous functioning of support teams and full deployment, omitting some exceptions. The last six weeks were dedicated to education and special-use case deployment.

3.1.4 Testing

This subchapter of implementation provides an overview of the tested measurements used, the progression of the tests, and the accepted ratios for production.

Compliance verification tested adherence to compliance standards (CIS, ISO 27001, NIST).

Below, in Table 4, are test cases created for compliance verification with the expected result measurement metric, achieved, and accepted rate per measurement displayed. The first test case was to run compliance scans before vulnerability CI/CD pipeline implementation, which showed an unsatisfactory 55 % compliance alignment that was not accepted. The post-implementation achieved rate is 95%, which shows significant improvement. At last, the accuracy of compliance script checks met the expected 89%.

Table 4: Compliance Verification outcome

Test Case	Expected Result	Measurement Metrics (%)	Achieved rate (%)	Accepted rate (%)
<i>Run compliance scans before vulnerability implementation</i>	Identify baseline compliance before implementation.	Pre-implementation compliance score	55	Not accepted
<i>Run compliance scans after enforcement</i>	Identify baseline compliance violations.	Post-implementation compliance score	95	90
<i>Compare results using the compliance script</i>	Script should generate a compliance report.	Accuracy of compliance checks	89	80

Policy enforcement objective is to ensure security policies are enforced correctly, preventing misconfigurations.

Table 5: Policy enforcement

Test case	Expected result	Measurement Metrics (%)	Achieved rate (%)	Accepted rate (%)
<i>Detect misconfigurations</i>	Any container with misconfiguration should be flagged	Detection rate	89	80
<i>Block containers with open ports</i>	Unauthorized open ports should be blocked.	False positive/negative rate	2	5
<i>Enforce sudoers file permissions</i>	Modified sudoers file should be corrected automatically.	Enforcement success rate	96	90
<i>Verify automated remediation</i>	Non-compliant containers should be patched or stopped.	Remediation success rate	90	90

As shown in Table 5 above, misconfiguration detection exceeded expectations with an 89% success rate against the accepted 80%, demonstrating the system's ability to identify vulnerabilities effectively. Unauthorised open ports were successfully blocked with a low false positive/negative rate of 2%, well below the 5% threshold, ensuring minimal disruptions. The enforcement of sudoers file permissions showed a near-perfect 99% accuracy. Automated remediation met the expected 90% success rate, ensuring non-compliant containers were patched or stopped as required.

Post-patching success measures how effectively vulnerabilities are mitigated, as shown in Table 6 below.

Table 6: Post-patching outcomes

Test Case	Expected Result	Measurement Metrics (%)	Achieved rate (%)	Accepted rate (%)
<i>Patch known vulnerabilities</i>	Vulnerability that has accessible remediation should be removed after patching.	Patch success rate	81	80
<i>Rescan patched images</i>	Previously flagged issues should not reappear.	False negative rate	10	10
<i>Regression testing after patching</i>	Application functionality should remain intact.	Post-patch failure rate	10	10
<i>Compare vulnerability counts pre/post patch</i>	Number of vulnerabilities should decrease.	Reduction in CVEs	82	80
<i>Patch rollback test</i>	System should revert to a stable state if patch fails.	Rollback success rate	90	90

Post-patching results demonstrate that mitigation efforts have been made more effective, though with some room for optimisation. The patch success rate reached 80%, meeting expectations. However, the false negative rate for rescanning patched images remained at 10%, suggesting that some vulnerabilities may still go undetected, or the vulnerability may not have a known fix yet. Regression testing confirmed that application functionality

remained mostly stable after patching, with a post-patch failure rate of 10%. The number of vulnerabilities was successfully reduced by 80%.

3.2 Container image hardening

The container image hardening pipelines for custom and third-party images were created as add-ons to the vulnerability pipeline to pursue a shift left in the security mindset and help end-users adapt faster to security best practices.

3.2.1 Architecture

Container image hardening is a standalone task that needs to be addressed. Both DevOps and Security teams must implement best practices that are easily adaptable, repeatable, and secure. This alignment of teams ensures that users can more easily adapt, fostering good habits.

Two reusable, ready-made pipelines have been created: one for creating and hardening custom images, and the other for using third-party images not stored or scanned in the private GitHub Container Registry (GHCR).

The Image Hardening workflow is a GitHub Actions workflow designed to automate the process of building, scanning, and hardening container images. The workflow builds a Docker image, scans it for image vulnerabilities and checks with the CIS benchmarks for misconfigurations using Trivy, applies security patches using Copa, and finally pushes the hardened image to GHCR. The image hardening workflow is shown in the Figure 16 below.

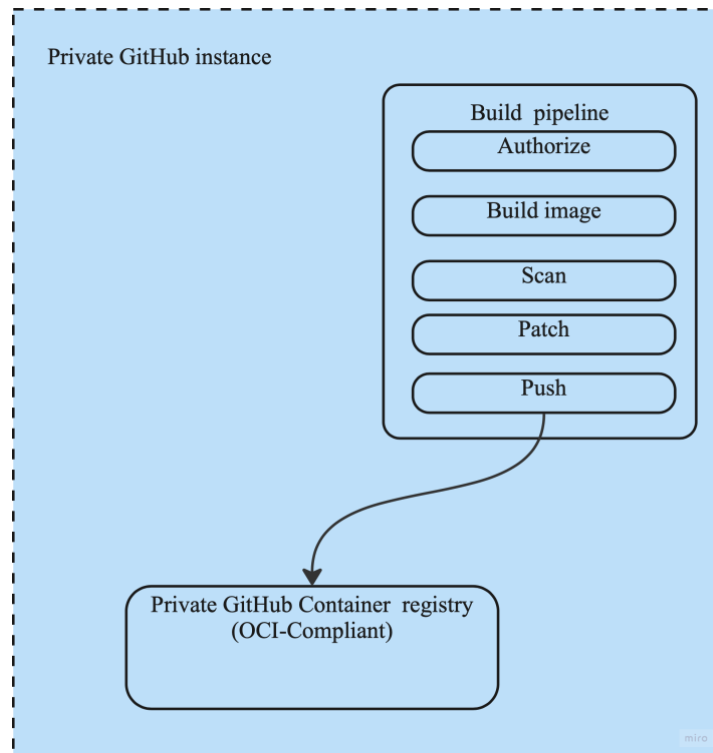


Figure 16: Image hardening diagram

The pull image hardening workflow is designed to automate the process of pulling, scanning, hardening, and deploying images from third-party image repositories. This image is not saved in the GHCR, as it is intended for one-time use. If users need to use it again, they can utilise the partially pushed image from the image hardening workflow. This scanning of third-party images enables precaution for the supplier chain vulnerabilities injected in the images and third-party dependencies to be scanned and remediated before the container gets deployed. It demonstrates a shift left security mindset, implementing security as a proactive measure in creating assets that may become or be vulnerable. The visualisation is shown in the Figure 17: Pulled image hardening

diagram

below.

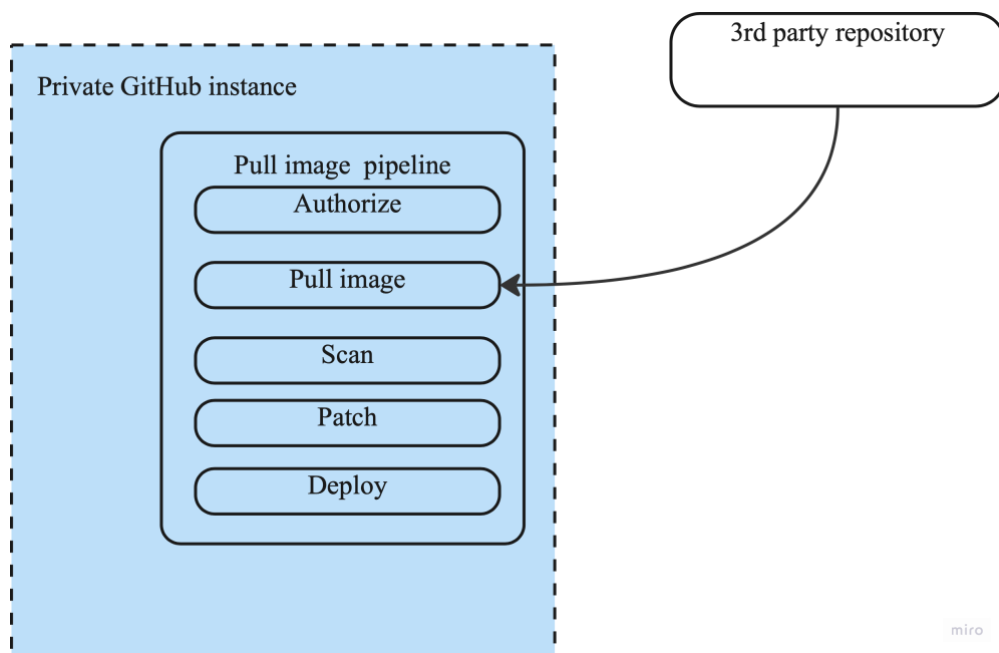


Figure 17: Pulled image hardening diagram

3.2.2 Solution Design

This chapter introduces a deeper insight into the workflows for image hardening. Both follow a structured process that includes scanning and patching container images, but with different inputs and outputs.

Image Hardening Workflow

Consists of five consecutive phases.

1. Authorization
2. Build image
3. Scan
4. Patch
5. Push

Image hardening workflow is a GitHub actions pipeline designed to enhance containerised application deployment security. It follows a systematic process comprising image creation, construction, vulnerability scanning, patching, and the subsequent pushing of container images to the GitHub container registry (GHCR). Each workflow stage is structured as an independent job that enables reuse across different applications.

Workflow is initiated with authorisation, utilising GitHub actions secrets for secure authentication with GHCT. The process then proceeds to the image building phase, which leverages BuildKit to optimise and streamline the construction of container images. In the scanning phase, Trivy is utilised to detect vulnerabilities and ensure compliance with Docker CIS security policies. During the patching phase, Copa is used to automate the application of security patches, mitigating the need for a complete image rebuild.

Finally, the hardened image is tagged and pushed to the GHCR, ensuring appropriate tagging and secure storage for future use and deployment.

Pull Image Hardening Workflow

Similarly to Image Hardening Workflow also this workflow consists of five consecutive phases.

1. Authorize
2. Pull Image
3. Scan
4. Patch
5. Deploy

The Pull Image Hardening Workflow is another GitHub Actions pipeline aimed at securing externally sourced container images before deployment. It begins by authenticating with a third-party container registry using a short-lived token, followed by the local retrieval of the specified container image. The scanning phase then utilises Trivy to identify vulnerabilities and confirm compliance with the security policies. Afterwards,

Copa is employed to apply necessary security patches to address the identified risks. The hardened image is then deployed based on the user's requirements and specific use cases.

Both workflows are designed to be modular, allowing for independent execution or selective integration. Thus, they offer flexibility and reusability for various container images and deployment contexts.

3.2.3 Implementation

The solution can be implemented by the users using example template for full script Image hardening, as shown in the Figure 18.

```
name: Full Image Hardening

on:
  workflow_dispatch:

jobs:
  full-hardening:
    uses: your-repo/.github/workflows/image_hardening.yml@main
    with:
      image_name: my-image
    secrets:
      GHCR_USERNAME: ${ secrets.GHCR_USERNAME }
      GHCR_TOKEN: ${ secrets.GHCR_TOKEN }
```

Figure 18: Example template for image hardening

Alternatively, users may opt to partially reuse specific sections, as illustrated in the Figure 19 below.

```

name: Scan Image Only

on:
  workflow_dispatch:

jobs:
  scan-only:
    uses: your-repo/.github/workflows/image_hardening.yml@main
    with:
      image_name: my-image
    secrets:
      GHCR_USERNAME: ${ secrets.GHCR_USERNAME }
      GHCR_TOKEN: ${ secrets.GHCR_TOKEN }
    needs: [scan-image]

```

Figure 19: Example - partial usage of template

This GitHub Actions workflow template is designed to trigger a reusable workflow that applies security hardening to a container image. It waits for a manual trigger, as `workflow_dispatch` is declared. Users then need to navigate to the Actions tab in the repository and click the "Run workflow" button. The workflow accepts input parameters, enabling users to specify their base container image, Dockerfile, and build context, allowing users to customise the execution.

A solution for pulling a third-party image can be used as shown in the template snippet in Figure 20 below.

```

name: Pull and Scan Image

on:
  workflow_dispatch:

jobs:
  pull-and-scan:
    uses: your-repo/.github/workflows/pull_image_hardening.yml@main
    with:
      image_name: my-image
      image_tag: latest
      registry: my-registry.io
    secrets:
      REGISTRY_USERNAME: ${ secrets.REGISTRY_USERNAME }
      REGISTRY_TOKEN: ${ secrets.REGISTRY_TOKEN }
    needs: [pull-image, scan-image]

```

Figure 20: Example - 3rd party pull and scan Image

3.2.4 Testing

This implementation subchapter provides the results of end-user adaptation testing with the reusable templates compared to the part that has not utilised them.

Table 7: Templates test cases resolutions, below, demonstrate significant advantages of incorporating reusable templates for security image-building tasks. When templates are used, users exhibit higher success rates in task completion, with a 93% accomplishment rate compared to 70% without templates. Tasks are completed efficiently, reducing time on task by 58%, and error rates drop by 78%.

The templates' ease of transition is 48 % higher, which signifies a quicker understanding of the task, reduced cognitive load, and greater consistency.

Table 7: Templates test cases resolutions

Test Case	Expected Result	With Templates (%)	Without Templates (%)	Improvement (%)
Task Completion Rate	Users successfully complete tasks	93	70	23
Time on Task	Tasks are completed faster	58	0	58
Error Rate	Mistakes reduction	78	0	78
User Satisfaction	Higher user satisfaction	82	60	22

Time Savings	Time on task reduction	47	0	47
Ease of Transition	Users quickly adapt	88	40	48
Perceived Usefulness	Users find templates helpful	85	50	35
Ease of Learning	Users quickly understand	83	60	23
Cognitive Load Reduction	Tasks feel easier (reduction in load)	18	0	18
Consistency of Output	More uniform results	92	70	22
Final Output Error Rate	Fewer errors in results	75	0	75

3.3 Economic Evaluation

The value of implementing the solution for securing the containers in private cloud on-prem infrastructure will be measured via ROSI (return on security investment). This calculation is more complex since the effectiveness of the potential financial loss the security investment can incur must be expressed.

ROSI assesses the financial impact of a particular incident and compares it to the cost of implementing a security measure to reduce that risk. It combines the quantitative risk assessment and the cost of implementing security countermeasures for this risk.

It is important to note that ROSI calculation is based on many approximations. Therefore, this evaluation is not considered an exact calculation but a well-thought-out approximation, mainly relying on historical statistical data. Mathematical equation of ROSI is displayed in [Formula 3.2](#). [\[28\]](#) [\[29\]](#)

$$ROSI = \frac{\text{Monetary loss reduction} - \text{Cost of Security Investment}}{\text{Cost of Security Investment}} \quad (3.2)$$

Cost of Security investment quantification

This cost will be calculated based on the findings in the True Cost of CVE management paper, based on the average salary for the programmer, architect and the cost of software used. [\[30\]](#)

Table 8: Resources and assets, expresses yearly costs for resources and assets. It is important to note that these costs will not be applied during the solution's entire six-year lifecycle.

Table 8: Resources and assets

Resources and Assets	Cost of Security investment (thousands €/year)
----------------------	---

Architect	350
Programmer	250
Software	100
Maintenance	300

Monetary loss reduction or, also known as the benefit of security investment quantification, is shown in [Formula 3.3](#):

$$MLR = ALE * MR . \quad (3.3)$$

MR – mitigation ratio represents the percentage of risks the security investment addresses.

Annual loss Expectancy formula is calculated accordingly to equation shown in [Formula 3.4](#):

$$ALE = SLE * ARO . \quad (3.4)$$

ALE - Annual Loss Expectancy.

SLE – Single Loss Expectancy, represents direct costs of remediation, lost productivity, or indirect costs.

ARO - Annual Rate of Occurrence.

Table 9 displays a shortened overview of the ROSI outcomes from the 6-year perspective. The full calculation can be found in [Appendix](#).

Table 9: ROSI calculation (partial). Table calculation sources [\[28\]](#) [\[29\]](#) [\[30\]](#) [\[31\]](#) [\[32\]](#) [\[33\]](#) [\[34\]](#) [\[35\]](#) [\[36\]](#) [\[37\]](#)

Automated Security Scanning and policy enforcement					
Incident Category/Metric	Total ALE (thousands €)	Effectiveness of Solution	MLR (thousands €)	Cost of the Solution (thousands €)	ROSI Analysis (6 years) (%)
External attacks (Image threats, Container escape, dependency poisoning)	108000	0.9	97200	314.285714	308.272727
Internal misuse/abuse (Insider Threats, Privileged User Misuse)	19 200	0.9	17280	314.285714	53.9818182
Theft (Exposed credentials or API keys)	28 800	0.6	17280	314.285714	53.9818182
Malfunctions (Misconfigurations, System failure)	9 600	0.7	6720	314.285714	20.3818182
Service interruption (DDoS, Cryptojacking)	16 200	0.7	11340	314.285714	35.0818182
Human error (Exposed secrets in GitHub repositories, Negligence)	14 500	0.7	10150	314.285714	31.2954545
GDPR	60 000	0.7	42000	314.285714	132.636364

The ROSI calculation for Automated Security Scanning and policy enforcement predicts a return on security investment ranging from approximately 308% at 0.9 effectiveness for

the external attack incident category, to around 20% at 0.6 effectiveness for theft. Overall, the ROSI analysis over six years across all incidents demonstrates a return of over 635% on the investment in the security solution.

3.3.1 Economic evaluation of Image hardening pipelines

This section examined the economic benefit from the second part of the solution. Essential measures that need to be considered are cost reduction and process efficiency.

They are calculated by the Relative Performance Index (RPI) which equation is displayed in [Formula 3.5](#), which shows improvement over a baseline and expresses a percentage change.

$$RPI = \left(\frac{\text{Current Performance} - \text{Baseline Performance}}{\text{Baseline Performance}} \right) * 100 \quad (3.5)$$

Current Performance – consists of the time/money spent on the given task and the money and time saved by using templates for image hardening pipelines, which has been achieved by omitting errors and ensuring consistency.

Baseline Performance – consists of time/money spent on the same task and the additional time/money spent fixing issues connected with the faulty utilisation.

As shown in Table 10, below, templates lower operational costs by 28%, improve process efficiency by 38%, and reduce errors in final outputs by 75%.

Table 10: Operational outcomes

Measure	Expected Result	Improvement (%)
Cost Reduction	Lower operational costs	28
Process Efficiency	Faster, more efficient workflows (improvement)	38

3.4 Future Development

As declared in the environment analysis, the company is not ready yet for full Kubernetes and zero-trust implementation. However, it intends to continue educating and innovating the infrastructure to make it less rigid and more suitable for the zero-trust and Kubernetes model.

The current solution generates post-remediation reports that detail several variables, including patching success rate. To improve visibility, a dashboard could be implemented that provides real-time metrics on patching efficiency, critical vulnerabilities, or failed remediation attempts.

Strengthening proactive monitoring by deploying an integrated observability stack for log monitoring of containers would be a great step forward. Alongside alerts for misconfiguration and possible remediation actions by the time the alert occurs, Kubeshark presents a viable solution for network traffic analysis due to its compatibility with service mesh frameworks, facilitating the detection of inbound/outbound traffic anomalies.

Given that no vulnerability or misconfiguration scanning tool achieves 100% accuracy, a machine learning (ML)-based reconciliation layer could be introduced. By cross-referencing results from multiple scanning tools, an ML model could minimise false positives/negatives, improving overall assessment reliability.

Alongside implementing Kubernetes and adopting a more agile and IaC approach for container deployment, it would be good practice to implement detached repositories, such as Jfrog, and tools for better visibility for development teams for continuous development, like Argo CD.

Nevertheless, implementing a rollback mechanism in case of an automated patch in production would be very beneficial and would be easily implemented once the infrastructure is onboarded on Kubernetes.

Finally, a re-evaluation of runner architecture and trust models is advised to address potential resource exhaustion issues and ensure scalability and resilience in dynamic environments.

Conclusion

This thesis presented an extensive solution for enhancing container security by aligning technical practices with organisational goals for improving security posture, reducing operational overhead, and ensuring system reliability, by enforcing policies for image creation and compliance, which included image signing, vulnerability scanning, and runtime protection mechanisms. The project established a robust framework for maintaining the integrity, confidentiality, and reliability of containerised assets.

Key outcomes demonstrated tangible improvements across multiple dimensions. First, integrating automated scanning tools and policy enforcement directly into the CI/CD pipeline eliminated manual patching, ensured consistent compliance, and significantly reduced exposure to vulnerabilities. Secondly, implementing a governance framework that includes permanent registry tags, CVE threshold policies, and continuous monitoring made it easier to comply with industry and regulatory standards. Third, introducing reusable pipeline templates and practical examples fostered a cultural shift, accelerating secure practices among DevOps and Security teams while mitigating risks posed by internal threats.

The impact of this solution was particularly evident in addressing critical gaps in the previous environment, which lacked security measures for container runtimes, registries, and images. Post-implementation assessments confirmed that security policies were effectively enforced, vulnerabilities were mitigated efficiently, and automation improved compliance adherence.

Automated security scanning and policy enforcement demonstrated a compelling Return on Security Investment (ROSI), with projections from 308% (at 0.9 effectiveness for external attacks) to 20% (at 0.6 effectiveness for theft). Over six years, the cumulative ROSI across all incident categories exceeded 635%, underscoring the cost-effectiveness of the implemented controls. Operational efficiencies were equally significant.

Standardised templates and automation reduced operational costs by 28%, improved process efficiency by 38%, and minimised errors in final outputs by 75%.

Despite the successes, opportunities for refinement remain. The solution lays a foundation for continuous evolution towards Zero Trust architecture and deeper integration with Kubernetes-native paradigms. While the current framework addresses immediate security and operational gaps, enhancements in ML-driven vulnerability reconciliation, IaC/GitOps deployment models, and self-healing rollback mechanisms would be an excellent addition for further reduction of operational overhead while hardening the system against emerging threats.

In summary, this thesis bridges the gap between DevOps agility and security rigour, delivering a reusable, automated framework that safeguards containerised ecosystems against evolving threats. This framework can be used for regular audit records while maintaining operational efficiency.

Bibliography

- [1] THE LINUX FOUNDATION. Cloud Native Computing Foundation. Online. 2025. Available at: <https://www.cncf.io>. [cit. 2025-04-13].
- [2] Towards Application-Specific Service Level Agreements: Experiments in Clouds and Grids. Online. 2010, s. 361-371. Available at: https://doi.org/10.1007/978-1-84996-241-4_21. [cit. 2025-04-13].
- [3] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Special Publication 800-145, *The NIST Definition of Cloud Computing*. United States of America: Computer Security Division - Information Technology Laboratory, 2011.
- [4] ERL, Thomas; PUTTINI, Ricardo a MAHMOOD, Zaigham. *Cloud Computing: Concepts, Technology & Architecture*. Pearson, 2013. ISBN 9780133387520.
- [5] SOUPPAYA, Murugiah; MORRELO, John; SCARFONE, Karen. *NIST Special Publication 800-190: Application Container Security Guide*. Online. 2017. National Institute of Standards and Technology. Available at: <https://doi.org/10.6028/NIST.SP.800-190>. [cit. 2025-01-02]
- [6] Container security: Precaution levels, mitigation strategies, and research perspectives. Online. 2023, no. 135. ISSN 0167-4048. Available at: <https://www.sciencedirect.com/science/article/pii/S0167404823004005>. [cit. 2025-04-13].

[7] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NIST Special Publication 800-190, *Application Container Security Guide*. United States of America: Computer Security Division - Information Technology Laboratory, 2017.

[8] *Containerization Technology: Types, Advantages, Applications, and More*. Online. SIMFORM. 2022. Available at: <https://www.simform.com/blog/containerization-technology> . [cit. 2025-04-13].

[9] *Os-vs-app-containers.jpeg*. Online. In: . Available at: <https://blog.risingstack.com/wp-content/uploads/2021/07/os-vs-app-containers.jpeg>. [cit. 2025-04-13].

[10] GARRISON, Justin a NOVA, Kris. *Cloud Native Infrastructure*. O'Reilly Media, 2017. ISBN 9781491984307.

[11] YILMAZ, Onur a AKBAS, Suleyman. *Introduction to DevOps with Kubernetes*. Packt Publishing, 2019. ISBN 9781789808285.

[12] headintheclouds. Docker networking: All types explained with real examples! *Dev Genius* [online]. Available at: https://dev.to/tusharops_29/docker-networking-basics-network-types-examples-5ed7. [cit. 2025-01-02].

[13] What Is Container Security? | Palo Alto Networks. Online. Available at: <https://www.paloaltonetworks.com/cyberpedia/what-is-container-security>. [cit. 2025-04-13].

[14] CONTAINER SECURITY SITE. *Container & Kubernetes Security Tools*. Online. Available at: <https://www.container-security.site>. [cit. 2025-04-13].

[15] MITRE CORPORATION. *CVE®*. Online. Available at: <https://www.cve.org>. [cit. 2025-03-17].

[16] Container security: Precaution levels, mitigation strategies, and research perspectives. Online. ISSN 0167-4048. Available at: <https://www.sciencedirect.com/science/article/pii/S0167404823004005#se0090>. [cit. 2025-04-13].

[17] What is DevSecOps and How Can You Start or Mature A Program? Online. 2022. Available at: <https://www.securitycompass.com/blog/what-is-devsecops-and-how-can-you-start-or-mature-a-program>. [cit. 2025-04-13].

[18] STACK OVERFLOW. *Developer survey*. Online. Available at: <https://survey.stackoverflow.co/2024/technology#most-popular-technologies-tools-tech-prof>. [cit. 2024-11-12].

[19] CLOUD NATIVE COMPUTING FOUNDATION. *Annual Survey 2023*. Online. Available at: <https://www.cncf.io/reports/cncf-annual-survey-2023/#findings>. [cit. 2024-12-12].

[20] ENISA. *2024 REPORT ON THE STATE OF CYBERSECURITY IN THE UNION*. Online. 2024. Available at: <https://www.enisa.europa.eu/sites/default/files/2024-11/2024%20Report%20on%20the%20State%20of%20Cybersecurity%20in%20the%20Union%20-%20Condensed%20version.pdf>. [cit. 2025-04-13].

[21] MICROMINDER CYBER SECURITY (MCS). *The Impact of Supply Chain Attacks on Cloud Container Security*. Online. 2025. Available at: <https://www.micromindercs.com/blog/supply-chain-attacks-on-cloud-container-security>. [cit. 2025-04-13].

- [22] FEDERAL RESERVE BANK OF ST. LOUIS. *Trade Wars, Tech Rivalry and Geopolitical Tensions*. Online. 2024. Available at: <https://www.stlouisfed.org/on-the-economy/2024/sep/trade-wars-tech-rivalry-geopolitical-tensions>. [cit. 2025-04-13].
- [23] DELOITTE. *Supply chain resilience in the face of geopolitical risks*. Online. Deloitte Insights. 2023. Available at: <https://www2.deloitte.com/us/en/insights/economy/us-china-trade-war-supply-chain.html>. [cit. 2025-04-13].
- [24] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Vulnerability Metrics*. Online. 2024. Available at: <https://nvd.nist.gov/vuln-metrics/cvss>. [cit. 2025-05-17].
- [25] FORUM OF INCIDENT RESPONSE AND SECURITY TEAMS. *EPSS*. Online. 2017, 17.03.2025. Available at: <https://www.first.org/epss/>. [cit. 2025-05-17].
- [26] *Trivy output*. Online. Available at: <https://alphasec.io/container-vulnerability-scanning-with-trivy/>. [cit. 2025-04-13].
- [27] DOCKER. *Docker-bench-security*. Online. Available at: <https://github.com/docker/docker-bench-security/blob/master/README.md>. [cit. 2025-04-13].
- [28] ENISA. *Introduction to Return on Security Investment*. Online. 2012. Available at: <https://www.enisa.europa.eu/sites/default/files/publications/Return%20On%20Security%20Investment.pdf>. [cit. 2025-04-13].

- [29] CERT PODCASTS: SECURITY FOR BUSINESS LEADERS: SHOW NOTES: *The ROI of Security*. Carnegie Mellon University, 2006. Available at :https://insights.sei.cmu.edu/documents/4575/2006_016_102_66890.pdf.
- [30] CHAINGUARD. *CVE Management is Painful - research report*. Online. 2024. Available at: <https://get.chainguard.dev/true-cost-of-vulnerability-management>. [cit. 2025-04-13].
- [31] STATIONX. *75+ Surprising Cloud Security Statistics You Should Know in 2025*. Online. 2024. Available at: <https://www.stationx.net/cloud-security-statistics/>. [cit. 2025-04-13].
- [32] ENISA. *Threat Landscape*. Online. 2024. Available at: <https://www.enisa.europa.eu/topics/cyber-threats/threat-landscape?utm#contentList>. [cit. 2025-04-13].
- [33] SENTINELONE. *50+ Cloud Security Statistics in 2025*. Online. 2024. Available at <https://www.sentinelone.com/cybersecurity-101/cloud-security/cloud-security-statistics>. [cit. 2025-04-13].
- [34] STATISTA. *Distribution of cyber incidents in organizations worldwide as of September 2024, by type*. Online. 2024. Available at: <https://www.statista.com/statistics/1483769/global-cyber-incidents-by-type>. [cit. 2025-04-13].
- [35] ENTERPRISEAPPSTODAY. *Cloud Security Statistics 2024 – Detailed Analysis*. Online. 2024. Available at: <https://www.enterpriseappstoday.com/stats/cloud-security-statistics.html>. [cit. 2025-04-13].

[36] STATISTA. *Largest fines issued for General Data Protection Regulation (GDPR) violations as of February 2025, by type of violation*. Online. 2024. Available at: <https://www.statista.com/statistics/1172494/gdpr-fines-by-type-violation/>. [cit. 2025-04-13].

[37] IBM. *Cost of a Data Breach Report 2024*. Online. 2024. Available at: <https://www.statista.com/statistics/1172494/gdpr-fines-by-type-violation/>. [cit. 2025-04-13].

List of Tables

Table 1: Common risks in container Infrastructure according to NIST 800-190. Table source [7]	21
Table 2: Asset definition	35
Table 3: SWOT analysis	48
Table 4: Compliance Verification outcome	63
Table 5: Policy enforcement	64
Table 6: Post-patching outcomes	65
Table 7: Templates test cases resolutions	73
Table 8: Resources and assets	75
Table 9: ROSI calculation (partial). Table calculation sources [28] [29] [30] [31] [32] [33] [34] [35] [36] [37]	77
Table 10: Operational outcomes	78
Table 11: Container image threat assessment	I
Table 12: Runtime threat assessment	IV
Table 13: Supply chain threat assessment	VI
Table 14: Secrets threat assessment	VIII
Table 15: Cyber Security Risk Management Matrix	X
Table 16: ROSI full table calculation	XV

List of Figures

Figure 1: Layer difference overview of container deployments and virtual machine. Image source [7]	11
Figure 2: Container types . Image source [9].....	12
Figure 3: Challenges in Container Deployment. Image source [18]	29
Figure 4: Incidents by threat type. Image source [19]	30
Figure 5: Example API call for CVE metrics retrieval.....	41
Figure 6: Sensor declaration example.....	52
Figure 7: Sensor verification script.....	53
Figure 8: Example of a Tanium query	53
Figure 9: Image pull from repository	54
Figure 10: Example scanning tool usage	55
Figure 11: Example Trivy output. Image source [26]	55
Figure 12: Copa example command	57
Figure 13: Copa patch command private repository	57
Figure 14: Copa patch command external repository	58
Figure 15: Success rate formula in code form	59
Figure 16: Image hardening diagram.....	68
Figure 17: Pulled image hardening diagram.....	69
Figure 18: Example template for image hardening	71
Figure 19: Example - partial usage of template	72
Figure 20: Example - 3rd party pull and scan Image.....	72
Figure 21: High-level overview of the automated scanning solution	XIV

Appendix

Threat Assesments

Table 11: Container image threat assessment

Image Threats/CVE	Description	Severity (CVSS)	Attack Vector	Infrastructure frequency of occurrence	Likelihood of exploitation (EPSS)	EPSS Percentile
CVE-2024-21626	File descriptor leak in <code>runc</code> allowing potential host system access	High	Host system access	High	0.01562	0.87
CVE-2021-28475	Vulnerability in image build process leading to privilege escalation	High	Privilege escalation, unauthorized image builds	High	0.01650	0.87
CVE-2021-34444	Docker vulnerability allowing unauthorized access to image storage locations	High	Image tampering, unauthorized access	High	0.00242	0.63

CVE-2024-23653	Enables attackers to break out of containers during image-building process	High	Container escape	Medium	0.00153	0.52
CVE-2024-23497	Linux kernel vulnerability allowing privilege escalation	Critical	Privilege escalation	High	0.00064	0.30
CVE-2024-23878	Flaw in <code>libarchive</code> leading to arbitrary code execution	High	Code execution	High	0.00043	0.11
CVE-2024-23879	Vulnerability in <code>libarchive</code> causing denial of service	Medium	Denial of service	Medium	0.00064	0.30
CVE-2021-21284	Docker images with exposed secrets leading to potential unauthorized access	High	Credential leakage, unauthorized access	High	0.00047	0.19
CVE-2020-13434	Malicious container image vulnerability leading to host compromise	High	Host compromise, data exposure	High	0.00148	0.51

CVE-2021-21334	Malicious images exploiting Helm charts leading to resource abuse	Medium	Resource abuse, data leakage	Moderate	0.00119	0.47
CVE-2022-0847	Dirty Pipe vulnerability affecting image layers	High	Unauthorized access, privilege escalation	High	0.03123	0.91
CVE-2021-41091	Misconfigured Docker image leading to sensitive data leakage	High	Sensitive data leakage, unauthorized access	High	0.00047	0.19
CVE-2020-2025	Vulnerability in containerized image distribution leading to unauthorized modification	High	Image modification, supply chain compromise	Moderate	0.00044	0.14

Table 12: Runtime threat assessment

Runtime Threats/CVE	Description	Severity	Attack Vector	Frequency	Likelihood of exploitation (EPSS)	EPSS Percentile
CVE-2022-0811	<code>crun</code> (container runtime) vulnerability allowing privilege escalation.	High	Privilege escalation within containers.	Moderate	0.0012	0.4812
CVE-2021-36222	Container runtime vulnerability allowing arbitrary code execution in runtime processes.	Critical	Arbitrary code execution, container escape.	High	0.0174	0.8793
CVE-2021-30810	Vulnerability in container isolation allowing access to sensitive files.	Critical	Unauthorized file access, privilege escalation.	High	0.0016	0.5443
CVE-2020-8558	Container networking vulnerability enabling	Medium	Packet injection, network compromise.	Moderate	0.0009	0.4184

	unauthorized packet injection.					
CVE-2020-2025	Malicious script execution vulnerability in container runtime processes.	High	Code injection, privilege escalation.	High	0.0004	0.1533
CVE-2020-15257	Vulnerability in container orchestration leading to unauthorized service restarts.	Medium	Service disruption, privilege escalation.	Moderate	0.0004	0.1835
CVE-2020-14386	Linux kernel vulnerability affecting container runtimes, leading to privilege escalation.	High	Container escape, privilege escalation.	High	0.0004	0.1684
CVE-2019-5736	RunC vulnerability allowing malicious container escapes.	Critical	Unauthorized host access, container escape.	High	0.0038	0.7298
CVE-2019-14287	Sudo vulnerability in containerized environments	High	Privilege escalation, unauthorized access.	High	0.4665	0.9761

	leading to privilege escalation.					
CVE-2018-15664	Docker image tar extraction vulnerability enabling arbitrary file to write.	Critical	Arbitrary file writes on host.	Moderate	0.0008	0.3726

Table 13: Supply chain threat assessment

Supply Chain Attacks/CVE	Description	Severity	Attack Vector	Frequency	Likelihood of exploitation (EPSS)	EPSS Percentile
CVE-2020-11979	Maven dependency vulnerability leading to arbitrary code execution.	Medium	Supply chain compromise, code execution.	Moderate	0.0024	0.6219
CVE-2020-8594	Vulnerability in image signing process allowing unauthorized modifications.	High	Image tampering, supply chain compromise.	Moderate	0.0008	0.3816

CVE-2021-25124	Dependency vulnerability leading to the installation of malicious components.	High	Dependency compromise, resource abuse.	Moderate	0.0004	0.1242
CVE-2021-28474	Vulnerability in Helm chart dependencies leading to unauthorized access.	High	Supply chain compromise, data leakage.	Moderate	0.0126	0.8576
CVE-2021-30810	Misconfigured supply chain workflows exposing sensitive resources.	High	Supply chain compromise, sensitive data leakage.	High	0.0016	0.5443
CVE-2021-4034	Polkit vulnerability allowing privilege escalation in containerized environments.	High	Privilege escalation, lateral movement.	High	0.0014	0.5035
CVE-2021-44228	Log4Shell vulnerability leading to RCE and supply chain compromise.	Critical	Remote code execution, supply chain compromise.	High	0.9711	0.9989

CVE-2021-45105	Log4J denial-of-service vulnerability affecting dependency chains.	Medium	Denial of service, resource exhaustion.	Moderate	0.9508	0.9951
CVE-2022-22965	Spring4Shell vulnerability affecting supply chains through RCE.	Critical	Remote code execution, data theft.	High	0.9745	0.9998
CVE-2022-31478	Misconfigured package repositories allowing malicious code injection.	High	Code injection, supply chain compromise.	High	0.0005	0.2255

Table 14: Secrets threat assessment

Secrets Management/CVE	Description	Severity	Attack Vector	Frequency	Likelihood of exploitation (EPSS)	EPSS Percentile
-------------------------------	--------------------	-----------------	----------------------	------------------	--	------------------------

CVE-2022-29154	Misconfigured environment variables in CI/CD exposing credentials.	Medium	Secret exposure, data leaks.	Moderate	0.0007	0.3278
CVE-2020-13692	Gitlab secret exposure vulnerability impacting pipelines.	High	Unauthorized access to sensitive data.	Moderate	0.0105	0.8423
CVE-2021-23219	Hardcoded secrets in Helm charts allowing privilege escalation.	High	Credential leakage, privilege escalation.	Moderate	0.0004	0.1533
CVE-2021-33001	Vault misconfiguration allowing unauthorized access to secret storage.	High	Secret leakage, privilege escalation.	High	0.0039	0.7336

Risk Management Matrix

Table 15: Cyber Security Risk Management Matrix

Asset	Threat	Vulnerability	Existing Controls	Consequences	Probability	Consequence	RPN	Risk Treatment	Residual Probability	Residual Consequence	Residual Risk
Container Infrastructure	Supply chain attack via malicious images	Insecure third-party dependencies	Limited image scanning	Unauthorized access, data breach	5	5	25	Implement image signing and trusted registries	2	4	8
Private Cloud Containers	Privilege escalation via runtime	Weak IAM settings	Role-based access control (RBAC) enabled	Unauthorized access, system	4	5	20	Enforce least privilege policies &	1	4	4

	vulnerabilities		but lacks granularity	compromise				strong RBAC			
CI/CD Pipelines	Injection of malware into pipeline	Insecure pipeline configurations	Basic security hardening applied	Supply chain compromise, unauthorized software deployments	4	5	20	Enforce pipeline security, restrict access, and audit logs	2	4	8
Secrets Management	Exposure of credentials via logs	Hardcoded secrets in environment	Manual secrets rotation	Credential theft, unauthorized access	5	4	20	Implement secret vaulting and	1	4	4

		ent variables						encrypted storage			
Container Networking	Unauthori zed lateral movement	Misconfig ured network policies	Default network settings in place	Data exfiltratio n, lateral movement inside network	4	5	20	Enforce network policies on the level of containers	2	4	8
Data Sovereignty Compliance	Geographi c restriction s on data storage	Misconfig ured storage settings, lack of complianc e	Limited complianc e measures	Legal penalties, reputation al damage	3	5	15	Implemen t automated complianc e auditing & encryptio n	2	4	8

External Cyber Threats	Ransomware attack on container environments	Weak backup & response plans	Basic backup strategy	Data loss, service downtime	5	5	25	Implement immutable backups, disaster recovery plan	2	4	8
Container Security (Public Registries)	Malicious container images in public repositories	Poorly vetted open-source images	Basic image scanning	Supply chain compromise, unauthorized access	5	5	25	Enforce trusted registries, automated security scans	2	4	8

Automated scanning solution overview

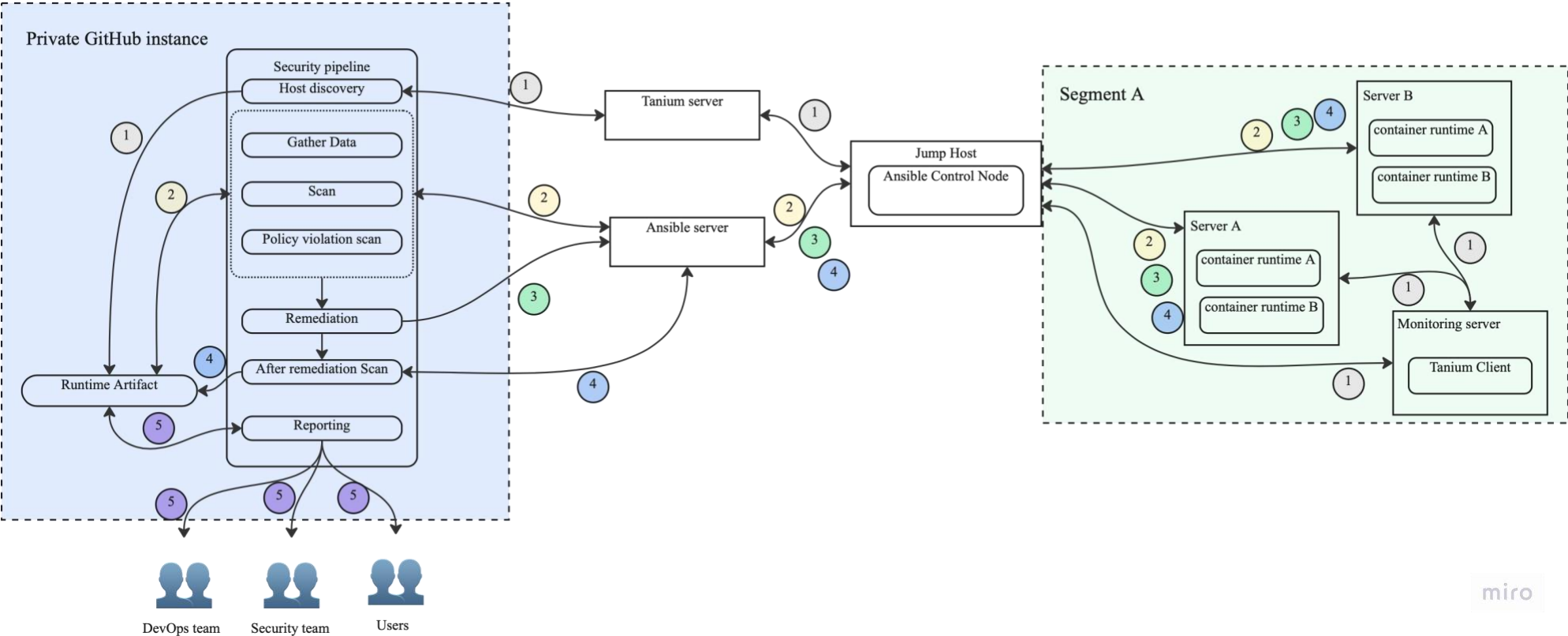


Figure 21: High-level overview of the automated scanning solution

ROSI full calculation

Table 16: ROSI full table calculation

Automated Security Scanning and policy enforcement (thousands €)	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Total ALE (thousands €)				
Incident Category	<i>SLE</i>	<i>SLE</i>	<i>SLE</i>	<i>SLE</i>	<i>SLE</i>	<i>SLE</i>		<i>Effectiveness of Solution</i>	<i>MLR</i>	<i>Cost of the Solution (thousands €)</i>	ROSI Analysis (6 years) (%)
External attacks (Image threats, Container escape, dependency poisoning)	4 500	4 500	4 500	4 500	4 500	4 500	108 000	0.9	97200	314.2857143	308.2727273
<i>ARO</i>	4	4	4	4	4	4					

Internal misuse/abuse (Insider Threats, Privileged User Misuse)	3 200	3 200	3 200	3 200	3 200	3 200	19 200	0.9	17280	314.2857143	53.98181818
ARO	1	1	1	1	1	1					
Theft (Exposed credentials or API keys)	4 800	4 800	4 800	4 800	4 800	4 800	28 800	0.6	17280	314.2857143	53.98181818
ARO	1	1	1	1	1	1					
Malfunctions (Misconfigurations, System failure)	1 600	1 600	1 600	1 600	1 600	1 600	9 600	0.7	6720	314.2857143	20.38181818
ARO	1	1	1	1	1	1					

Service interruption (DDoS, Cryptojacking)	2 700	2 700	2 700	2 700	2 700	2 700	16 200	0.7	11340	314.2857143	35.08181818
<i>ARO</i>	1	1	1	1	1	1					
Human error (Exposed secrets in GitHub repositories, Negligence)	2 900	2 900	2 900	2 900	2 900	2 900	14 500	0.7	10150	314.2857143	31.29545455
<i>ARO</i>	1	1	1		1	1					
GDPR	10 000	10 000	10 000	10 000	10 000	10 000	60 000	0.7	42000	314.2857143	132.6363636
<i>ARO</i>	1	1	1	1	1	1					
<i>ALE per Year</i>	43 200	43 200	43 200	40 300	43 200	43 200	256 300				
<i>Cost of the solution*(for a year) (thousands €)</i>	700	300	300	300	300	300				2 200	

<i>Effectiveness of Solution</i>	0.6	0.8	0.99	0.9	0.87	0.7					
<i>MLR</i>	25 920	34 560	42 768	36 270	37 584	30 240					
ROSI Analysis (for the year) (%)	36.02857143	114.2	141.56	119.9	124.28	99.8					635.6318182

*

- Software
- Pay programmer
- Pay architect and maintenance DevOps team, potentially can be all the same people at the end