NTNU
Norwegian University of
Science and Technology

Intrusion Detection in Kubernetes – A study of tools and techniques

Neuenschwander, Kurt Lukas Vincent

Submission date: July 2024
Main supervisor: Gligorski, Danilo (NTNU)
Co-supervisors: Bromander, Siri (mnemonic AS) and
Paulsen, Ole Henrik (mnemonic AS)

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Intrusion Detection in Kubernetes – A study of tools and techniques

Student: Neuenschwander, Kurt Lukas Vincent

Problem description:

Context: To ensure better scalability and maintainability of their applications in the cloud, more and more companies adopt the microservice-based application architecture. The open-source tool Kubernetes has been widely adopted to manage and orchestrate the containers running these microservices.

Objective: This rise in popularity also leads to new concerns regarding security and privacy in Kubernetes deployments. Numerous tools have emerged to aid in identifying security vulnerabilities within deployment scripts or to conduct scans on container images before deployment. However, the challenge remains: How can intrusions beyond known vulnerabilities and code smells like e.g. a compromised container be detected within a live cluster environment?

Method: In order to address these challenges, potential attacks and vulnerable assets will be enumerated and scrutinized. Subsequently, existing tools will be evaluated and possibly enhanced or new ones will be developed at a Proof of Concept (PoC) stage.

Results: The aim of this thesis is not to devise a comprehensive solution to the problem at hand, but rather to examine potential solutions and generate ideas on how intrusion detection in Kubernetes could be approached.

Approved on: 2024-03-19

Main supervisor: Gligorski, Danilo (NTNU)

Co-supervisors: Bromander, Siri (mnemonic AS) and
Paulsen, Ole Henrik (mnemonic AS)

Abstract

In recent years, more and more companies have started splitting their monolithic applications having everything in a single codebase into small independent pieces called microservices. The de-facto standard solution to orchestrate these microservices running in containers across multiple servers is the open-source tool Kubernetes. The ongoing global increase in cyberattacks raises the question of how these Kubernetes environments can be protected. This problem can be approached from two distinct angles: The proactive angle addresses security concerns like vulnerability scanning, access control, and security policies. However, zero-day exploits show that successful cyberattacks can happen even when all the proactive security measures have been taken. How can these attacks be detected and investigated to allow better protection in the future? This thesis is focused on the reactive angle of Kubernetes security, also known as runtime security, that addresses these issues.

Accordingly, this thesis investigates how intrusion detection can be done in Kubernetes. At first, a high-level risk analysis of the Kubernetes components is conducted to identify the most vulnerable assets and prioritise further work. It concludes with containers and Pods as the most important Kubernetes components to protect, closely followed by the Operating System (OS) on Kubernetes nodes. Since most of the literature on this topic can be related to Kubernetes security software vendors, an unbiased comparison of these tools was missing. Such a comparison of tools independent from vendor sponsoring is given in this thesis.

GCP's microservices-demo is used as a comparison baseline in two lab Kubernetes clusters based on OpenStack Magnum and microk8s. Due to budget limitations, no tests are run in Kubernetes cloud services like AKS, AWS, or GCP. For the most part, freely available open-source tools are evaluated for the same reason. A novel set of criteria is introduced that evaluates how easily each tool could be integrated into existing workflows. In total, 20 different tools providing reactive Kubernetes security features are evaluated in detail based on these criteria.

Initially, different tools are evaluated that support digital forensics and analysis of the context of security alerts in Kubernetes. They cover functionalities like logging and monitoring, networking observability, and mapping out relations between the applications deployed in Kubernetes. Finally, the two available open-source Intrusion Detection Systems (IDSs) for Kubernetes, Falco and Tetragon, are evaluated, along with a discussion of commercial intrusion detection services for Kubernetes. Based on the observations made, the thesis concludes with recommendations of tools to cover the reactive side of Kubernetes security.

Sammendrag

I løpet av de siste årene har flere og flere bedrifter begynt med å splitte opp deres monolitiske applikasjoner der alt er samlet i én kodebase i små, uavhengige deler kalt mikrotjenester. Kubernetes har skilt seg ut som de-facto standard løsning for å orkestrere disse mikrotjenestene som kjører i konteinere fordelt på flere servere. Den pågående globale stigningen i cyberangrep har økt fokuset rundt spørsmålet om hvordan disse miljøene kan beskyttes. Dette problemet kan angripes fra to forskjellige sider: Den proaktive siden omhandler sikkerhetshensyn som sårbarhetsscanning, tilgangssstyring, og sikkerhetspolicies. Likevel viser zero-day-angrep at vellykkede cyberangrep forekommer, selv når alle proaktive sikkerhetstiltak er iverksatt. Hvordan kan disse angrep detekteres og undersøkes for å sikre bedre beskyttelse i fremtiden? Denne oppgaven fokuserer på den reaktive siden av Kubernetes sikkerhet, også kjent som runtime-sikkerhet, som ettergår disse utfordringene.

Følgelig undersøker denne oppgaven hvordan cyberangrepsdeteksjon kan gjøres i Kubernetes. Først blir en overordnet risikoanalyse gjennomført for å identifisere de mest sårbare komponentene og prioritere fremtidig arbeid. Den konkluderer med konteinere og Pods som de viktigste Kubernetes komponenter å beskytte, tett fulgt av operativsystemet (OS) på Kubernetes serverne. Siden mye av litteraturen kan relateres til bedrifter som leverer sikkerhetsprogramvare til Kubernetes, mangler det en uavhengig sammenlikning av disse verktøyene. En slik sammenlikning av verktøy uten innflyt av sponsing fra bedrifter gis i denne oppgaven.

GCP's demo av mikrotjenester brukes som sammenlikningsgrunnlag i to Kubernetes clustere basert på OpenStack Magnum og microk8s. Grunnet et begrenset budsjett blir det ikke gjennomført tester i Kubernetes skymiljøer som AKS, AWS, eller GCP. Av samme grunn evalueres stort sett fritt tilgjengelig programvare med åpen kildekode. Et egenutviklet sett av kriterier introduseres som vurderer hvor lett det er å integrere hvert verktøy i eksisterende arbeidsrutiner. Det evalueres totalt 20 forskjellige verktøy i detalj basert på disse kriteriene.

Innledningsvis evalueres verktøy til digital etterretning og analyse av konteksten rundt sikkerhetshendelser i Kubernetes. De dekker funksjonaliteter som logging og monitorering, innsyn i nettverkshendelser, og kartlegging av relasjonene mellom applikasjoner som kjører i Kubernetes. Til slutt evalueres de to cyberangrepsdeteksjonssystemer (IDSer) for Kubernetes med åpent tilgjengelig kildekode, Falco og Tetragon, fulgt av en diskusjon av kommersielle cyberangrepsdeteksjonstjenester for Kubernetes. Basert på observasjonene konkluderer denne oppgaven med anbefalinger av verktøy til å dekke den reaktive siden av Kubernetes sikkerhet.

Preface

The research presented in this master thesis was conducted between February and July 2024 in the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology (NTNU) in Trondheim, in collaboration with the Norwegian cybersecurity company mnemonic AS. It fulfils my master's degree in Digital Infrastructure and Cybersecurity at NTNU.

Before this degree, I took bachelor's degrees in Electronical Engineering and Computer Science at the University of Stavanger, followed by three years of full-time work. The first was in the Business Computing Group at CERN, followed by two years in mnemonic's Threat Intelligence (TI) Department. My studies and especially my practical working experience as a cybersecurity consultant turned out to be useful among other things to evaluate the many trade-offs addressed in this thesis.

Even though I had almost no previous knowledge about it before I started working on this thesis, the versatility and power of Kubernetes have fascinated me for a long time. It is very popular, and the adoption and market share are still growing. At a meeting in the TI Department where the topic randomly came up, I almost instantly realised that I would like to write my thesis about it.

At this point, I would like to thank my supervisor Siri Bromander, whose feedback really shaped this thesis and always led me back to the right path when I did not know how to proceed. Many thanks also to my colleagues from mnemonic for much inspiration and input based on practical experience, especially Ole Henrik Paulsen, Mats Christensen and Nikolas Papaioannou. I would also like to thank Lars Erik Pedersen from NTNU for his quick answers to my OpenStack-related questions and my friends Kevin Dittrich and Lennart Robert Wilke for reading the final drafts. Finally, I would like to thank my NTNU supervisor Danilo Gligorski for his constructive feedback early in the process.

July 2024
Lukas Neuenschwander

Contents

List of Figures ix

List of Tables xi

List of Listings xii

Glossary xiii

List of Acronyms xv

1 Introduction 1
- 1.1 From monolithic to microservice architecture . . . 1
- 1.2 From Virtual Machines to containers . . . 2
- 1.3 Running and managing container clusters in Kubernetes . . . 4
- 1.4 Research Questions about securing Kubernetes clusters . . . 5

2 Method and related work 7
- 2.1 Method and outline . . . 7
- 2.2 Tools . . . 8
- 2.3 Related work . . . 10

3 Assets and attack vectors in Kubernetes 13
- 3.1 What is a Kubernetes cluster comprised of? . . . 13
- 3.2 Components of worker nodes . . . 14
  - 3.2.1 Kube-proxy and the Container Network Interface (CNI): Networking . . . 15
  - 3.2.2 Container runtime and kubelet: Infrastructure to run containers . . . 16
  - 3.2.3 Operating System (OS): Infrastructure for Kubernetes components . . . 16
  - 3.2.4 Pod and container: Encapsulation and isolation for microservices . . . 17
- 3.3 Components of master nodes . . . 19
  - 3.3.1 Etcd – etc distributed: Reliable storage for configuration and state . . . 21
  - 3.3.2 Kube-apiserver: Message broker and single point of contact . . . 21
  - 3.3.3 Kube-controller-manager and cloud-controller-manager: Control loops . . . 22
  - 3.3.4 Kube-scheduler: Assigning Pods to nodes . . . 23
  - 3.3.5 Domain Name Service (DNS): Identify Kubernetes Services . . . 24
- 3.4 High-level risk analysis of Kubernetes components . . . 24
- 3.5 Prioritisation of assets and attack vectors . . . 25

4 Lab environment to test tools **27**
4.1 Kubernetes lab environments . . . 27
4.1.1 Homelab on used machines . . . 27
4.1.2 Managed Kubernetes cluster in OpenStack Magnum . . . 28
4.1.3 Microk8s cluster in OpenStack . . . 30
4.2 Setting up a baseline for testing intrusion detection tools . . . 31
4.3 Conclusion: Which lab environment was finally used? . . . 34

5 Observation capabilities in Kubernetes **35**
5.1 Kubernetes security tools: Requirements and evaluation . . . 35
5.1.1 Desirable information for digital forensics in Kubernetes . . . 35
5.1.2 Developing evaluation criteria for Kubernetes tools . . . 36
5.1.3 Novel proposal of evaluation criteria for Kubernetes tools . . . 37
5.1.4 Finding and evaluating Kubernetes tools . . . 39
5.2 Tools for getting an overview of the cluster . . . 40
5.2.1 Kubectl: Universal Kubernetes administration . . . 40
5.2.2 Kubernetes dashboard: A simple overview of the cluster . . . 41
5.2.3 Grafana dashboards with Prometheus operator: Display metrics . . . 43
5.2.4 Integrated Development Environments (IDEs) for Kubernetes . . . 44
5.2.5 K9s: Simplified command-line Kubernetes administration . . . 46
5.2.6 KubeView: Kubernetes cluster visualiser and graphical explorer . . . 49
5.2.7 Jaeger: Tracing to reveal the interdependencies of services . . . 51
5.2.8 Otterize Network Mapper: Map out relations between applications . . . 52
5.2.9 NeuVector: Full lifecycle container security platform . . . 53
5.3 Digression: Access control in Kubernetes . . . 54
5.4 Tools for collecting and viewing logs . . . 55
5.4.1 Elastic Cloud on Kubernetes (ECK): Complete logging stack . . . 55
5.4.2 Fluentd and Fluent Bit: More resource-efficient log-collectors . . . 57
5.4.3 Kube-audit-rest: Retrieve audit-logs in cloud platforms . . . 58
5.4.4 Other tools for viewing logs live in Kubernetes . . . 59
5.5 Tools for collecting and viewing networking information . . . 59
5.5.1 Calico: Kubernetes' initial network driver . . . 60
5.5.2 Cilium: Kubernetes' new de-facto standard network driver . . . 60
5.5.3 Service meshes: Encryption and observability for microservices . . . 63
5.5.4 Kubeshark: Live network traffic analyser for Kubernetes . . . 66
5.6 Tools for remote access to Kubernetes resources . . . 68
5.6.1 Adapted vanilla Kubernetes Pods: Remotely access nodes . . . 68
5.7 Tools for container imaging and backup . . . 70
5.7.1 Kubelet Checkpoint API: Kubernetes native container snapshots . . . 71
5.7.2 Velero: Open-source backup and restore of Kubernetes resources . . . 72
5.8 Tools that were not evaluated or disqualified . . . 72
5.8.1 Tools that were not evaluated because there were alternatives . . . 72
5.8.2 Tools that are no longer maintained or deprecated . . . 73
5.8.3 Tools that are regarded immature or not ready for production usage . . . 74
5.8.4 Tools mainly focused on the proactive side of Kubernetes security . . . 74

6 Intrusion detection tools for Kubernetes **75**
6.1 Falco: Open-source intrusion detection for Kubernetes . . . 76
6.2 Cilium Tetragon: Observability and runtime enforcement . . . 79
6.3 SNORT and Suricata: Network Intrusion Detection . . . 81
6.4 Endpoint Detection and Response (EDR): Node protection . . . 83
6.5 Kubernetes security as a commercial service . . . 84

7 Results, discussion, and conclusion **85**
7.1 Contribution to sustainability . . . 85
7.2 Summary of Results . . . 85
7.2.1 High-level risk analysis of Kubernetes components . . . 85
7.2.2 Lab environments and a baseline for testing Kubernetes tools . . . 86
7.2.3 Novel proposal of evaluation criteria for Kubernetes security tools . . . 86
7.2.4 Evaluation of reactive Kubernetes security tools . . . 86
7.3 Limitations of the results . . . 88
7.4 Future work . . . 88
7.5 Discussion of the results . . . 89
7.5.1 High-level risk analysis, lab environment and evaluation criteria . . . 89
7.5.2 From the degrees of open source to commercial offerings . . . 91
7.5.3 Tools for Kubernetes administration and remote access . . . 92
7.5.4 Tools for networking and mapping out the cluster . . . 93
7.5.5 Tools for monitoring, logging and information gathering . . . 94
7.5.6 Tools for container imaging and backup . . . 95
7.5.7 Tools for intrusion detection in Kubernetes . . . 95
7.5.8 Wrapping up Kubernetes security tools: Recommendations . . . 96
7.6 Conclusion . . . 98

References **99**

Appendix

A Additional figures and technical documentation **107**
A.1 Additional figures . . . 107
A.2 Code and scripts on GitHub and attached to this file . . . 107
A.3 K9s plugin to improve the readability of JSON logs . . . 107
A.4 Deployment of the Elastic Cloud on Kubernetes (ECK) . . . 108
A.4.1 Deployment of the ECK Operator . . . 108
A.4.2 Enable audit logs in microk8s . . . 109
A.4.3 Configure Filebeat to fetch and enrich log-files . . . 109
A.4.4 Setting up Logstash for further processing . . . 110
A.4.5 Persistent storage in volumes for Elasticsearch . . . 111
A.4.6 Visualisation and data access with Kibana . . . 112
A.4.7 Deployment script for Elastic Cloud on Kubernetes (ECK) . . . 113
A.5 Deployment of Fluent Bit logging into ECK . . . 117
A.6 Deployment script for privileged Pods . . . 123
A.7 Deployment of Falco with audit logs and export of events . . . 125

List of Figures

1.1 Comparison of the monolithic and microservice architecture 2
1.2 Comparison of running microservices in VMs and containers 3
1.3 Features provided by container orchestration tools 4

2.1 Terminal multiplexer tmux demonstrating simultaneous code execution 9

3.1 Kubernetes deployment with master and worker nodes 13
3.2 Important components to secure in a Kubernetes worker node 14
3.3 Communication inside a Pod 18
3.4 Important components to secure in a Kubernetes master node 19
3.5 Example of communication between Kubernetes master components 20

4.1 Three-node Kubernetes cluster in home network 27
4.2 High-availability cluster in OpenStack Magnum 28
4.3 High availability microk8s cluster on Ubuntu Server in OpenStack 30
4.4 Screenshot of the checkout process in the online-boutique 32
4.5 Overview of the different microservices that are part of the online-boutique 32

5.1 Pod metrics in Kubernetes dashboard, total for all and individually for each Pod . 42
5.2 Official Grafana Labs Kubernetes dashboard deployed on the microk8s-cluster . . . 43
5.3 Lens dashboard on the microk8s cluster 45
5.4 K9s dashboard displaying the Pods from the microservices-demo 47
5.5 K9s xray for services in the microservices-demo on the microk8s-cluster 48
5.6 Demonstration of a novel k9s plug-in reformatting and displaying JSON logs . . . 49
5.7 KubeView on the microservices-demo 50
5.8 KubeView showing a more complex graph of more resources 50
5.9 Dependency graph of microservices in Jaeger 51
5.10 Timespans each involved microservice takes to process a request 51
5.11 Full map of services generated by the Otterize network mapper 52
5.12 NeuVector network activity graph for the microservices-demo 53
5.13 K9s overview of the Kubernetes resources deployed by Neuvector 54
5.14 Usage of kubectl to retrieve a Secret for a ServiceAccount 54
5.15 Resource consumption of the ECK stack in k9s 56
5.16 K9s resource consumption comparison of the ECK Stack and Fluent Bit 58
5.17 Comparison of the log-volume from kube-audit-rest and full audit-logs in Kibana . 59
5.18 Hubble traffic graph for the microservices-demo 62
5.19 Hubble traffic graph for the Elastic Cloud on Kubernetes Stack 62

5.20 Architecture and communication flow for a service mesh with sidecars 63
5.21 Istio's web GUI Kiali showing a traffic graph for the microservices-demo 65
5.22 Kubeshark showing gRPC, DNS and HTTP traffic in the microservices-demo . . . 66
5.23 Kubeshark service map for the microservices-demo 67
5.24 Demonstration of impersonating the node from a Pod 69

6.1 Falco alerts in Kibana, including an alert from the custom rule 79
6.2 Events from Falco and Tetragon triggering on read access to /etc/hosts in Kibana 81
6.3 Usage of a terminating proxy for in a network intrusion detection system 81

A.1 Workload overview in Kubernetes dashboard deployed in the microk8s-cluster . . . 107
A.2 Kibana on the microk8s-cluster showing container-logs from the last hour 112
A.3 Total volume of audit- and container-logs in the microk8s-cluster in Kibana 113
A.4 Falcosidekick-UI dashboard with alerts from the last 48 hours 126

List of Tables

3.1 Summary of risks in worker components 25
3.2 Summary of risks in master components 25

5.1 Likert rating scale in five steps to evaluate tools 37
5.2 Rating of the command line Kubernetes administration tool kubectl 41
5.3 Rating of the Kubernetes dashboard displaying resources and metrics 43
5.4 Rating of Grafana dashboards for Kubernetes metrics 44
5.5 Rating of Integrated Development Environments (IDEs) for Kubernetes 46
5.6 Rating of the command line Kubernetes administration tool k9s 49
5.7 Rating of the Kubernetes resource relation visualiser KubeView 50
5.8 Rating of the tracing application Jaeger revealing service interdependencies 51
5.9 Rating of the Otterize network mapper revealing relations between applications . . 53
5.10 Rating of the full lifecycle container security platform NeuVector 54
5.11 Rating of the logging stack Elastic Cloud on Kubernetes (ECK) 57
5.12 Rating of the data collectors Fluent Bit and Fluentd 58
5.13 Rating of the audit log fetcher kube-audit-rest 59
5.14 Rating of the Kubernetes network driver Calico 60
5.15 Rating of the Kubernetes network driver Cilium with Hubble 63
5.16 Rating of the service mesh Istio 66
5.17 Rating of the live Kubernetes network traffic analyser Kubeshark 67
5.18 Rating of adapted vanilla Kubernetes Pods for remote access to nodes 70
5.19 Rating of the Kubernetes resource backup tool Velero 72

6.1 Rating of the Kubernetes intrusion detection system Falco 79
6.2 Rating of the observability and runtime enforcement tool Cilium Tetragon 81

7.1 Summary of the ratings for all evaluated tools 87

List of Listings

6.1 Custom rule for Falco detecting when /etc/hosts has been accessed 78
6.2 Custom rule for Tetragon detecting when /etc/hosts has been accessed 80

A.1 K9s plug-in to improve the readability of JSON logs 108
A.2 Additional command-line arguments to activate audit-logs in microk8s 109
A.3 Audit policy file kube-api-audit-policy-all.yaml 109
A.4 Elasticsearch Index Lifecycle Management (ILM) policy 111
A.5 Applying the Elasticsearch ILM template to indices starting with logs-* 111
A.6 Deployment YAML-script for the Elastic Cloud on Kubernetes (ECK) 113
A.7 Deployment YAML-script for Fluent Bit logging into ECK's Elasticsearch 118
A.8 Deployment YAML-script for a privileged Pod on each node in the cluster 123
A.9 Deployment YAML-script for Pods mounting the node's file system root 124
A.10 Helm command and -values for Falco 127

Glossary

bare-metal If Kubernetes (k8s) nodes are installed directly on physical servers, without any Virtual Machine (VM)-hypervisor or cloud platform in between, they are installed ~.

CIA triad The ~ refers to protecting the confidentiality, integrity and availability of a system.

CNCF maturity level Cloud Native Computing Foundation (CNCF) projects have three maturity levels: Sandbox, Incubating, & Graduated. To reach graduation, many criteria must be fulfilled like high adoption rates and many contributors from independent organisations.

code smell Typical misconfiguration that is introduced by developers again and again, like hardcoded credentials.

container-registry A ~ hosts and distributes container images, based on which the file system of containers is initialised. Docker Hub is the biggest public ~. It is also possible to set up a private one that is only accessible locally and thus not exposed to the Internet.

crypto mining The process of using complex mathematic calculations with a high resource demand to generate money in electronic cryptocurrencies and validate transactions.

data exfiltration Cyberattack where data is stolen from personal or corporate devices.

Helm Package manager for Kubernetes, allowing to deploy applications in a cluster easily with few commands. Packages are configured in special YAML files called Helm-charts.

homebrew Package manager primarily for installing command-line (cmd) applications on Mac, but also available for Linux. In the cmd, it is called by the command `brew`, not `homebrew`.

lateral movement Compromising other systems from one infected system to enlarge and spread a cyber attack.

load balancer One or several servers, which efficiently distribute incoming network traffic across a group of backend servers.

microk8s Kubernetes distribution from Canonical mainly targeted to developers, which allows installing a full Kubernetes cluster on a single workstation.

mnemonic Norwegian cyber security company with over 300 employees headquartered in Oslo, mostly active in Western and Northern Europe. ~ is the initiator behind this thesis.

NetFlow A network protocol developed by Cisco collecting IP traffic information. It allows to see which IPs communicated with others, and how much data they exchanged.

on-premises Local software that is installed ~ runs on servers in the company's own IT environment and not remotely in a cloud environment provided by e.g. Microsoft Azure.

open source Source code that is made freely available for modification and redistribution, often maintained by a community of volunteers. There are multiple different ~ licenses.

OpenStack Open-source cloud platform managing distributed network and storage resources, allowing e.g. to deploy Virtual Machines (VMs) on a self-hosted server.

port forwarding Forward traffic from a local port to a remote Service or vice versa. E.g. if a remote web server lays behind a firewall, it can be made available on the local machine.

privilege escalation An attacker gaining unauthorized privileged access to a system, for example by gaining root-privileges from a user that is not entitled to it.

replica For redundancy and to increase performance, applications in k8s run in multiple ~, which are equivalent instances of the application running on different servers.

retention A surveillance camera has a ~ period of 48 hours if it always keeps data from the last 48 hours, overwriting the oldest data to add new data.

sandbox A system allowing an untrusted application to run in a controlled and isolated environment with only restricted access to other resources like the file system or network.

sidecar ~ is the name for the additional containers which for example service meshes add to an application A's Pod to perform a certain function without modifying A's code.

signature In the context of Intrusion Detection Systems (IDSs), ~ is the technical term for an intrusion detection rule.

vanilla A ~ software installation has not been extended with add-ons or modified from its original form, in other words, a clean installation.

webhook A ~ allows event-driven communication between two application-APIs A and B. Instead of A constantly asking B for updates, B notifies A via A's ~ when it has updates.

zero trust A security concept assuming that by default, nobody is trusted from inside or outside the network. Verification is required from everyone trying to access resources.

zero-day A ~ exploit is a new security vulnerability that has recently been discovered and not been fixed by the product vendor yet. Until that fix is widely adopted, the ~ is exploited.

List of Acronyms

AI Artificial Intelligence: A field of research in computer science developing methods enabling machines to perceive their environment and learn and adapt from it.

AKS Azure Kubernetes Service: Kubernetes as a service in Microsoft's Azure cloud.

API Application Programming Interface: Set of rules and protocols defining how different software components can interact with each other.

arg argument: A parameter which defines and modifies the behaviour of a function or command-line-tool can also be called ~.

ARM Advanced RISC (Reduced Instruction Set Computer) Machine: A processor architecture different from and incompatible with the most widely used x86 / x64-architecture. ~ is optimised for lower energy consumption, especially in embedded devices.

AWS Amazon Web Services: Amazon's service to run applications in its cloud environment.

CLI Command Line Interface: Text-based interface, in which commands can be entered to administrate a system. Command-line (cmd) tools are often referred to as ~ tools.

cmd command-line: Another word for a terminal, a text-based user interface allowing a programmer to issue text-based commands, resulting in text-based output.

CNCF Cloud Native Computing Foundation: Open source vendor-neutral hub of cloud-native computing, hosting projects like Kubernetes or Prometheus.

CNI Container Network Interface: Kubernetes specification and libraries to write plugins for network interfaces in Linux containers.

CPU Central Processing Unit: Core computational unit (processor) in a server.

CRD Custom Resource Definition: Extension of the Kubernetes (k8s) API that is not necessarily available in vanilla k8s. Can simplify the configuration of particular resources.

CRI Container Runtime Interface: Interface between container runtime (like Docker) and Kubernetes, allows to use different container runtimes.

DNS Domain Name Service: Translates domain names like google.com into IP addresses.

DoS Denial of Service: Attack that limits the accessibility of a service, for instance by overloading it with fake requests. Can make a service partly or totally inaccessible.

eBPF extended Berkeley Packet Filter: A technology that runs sandboxed programs in a Virtual Machine (VM) in the Linux Operating System (OS) kernel to extend its capabilities.

ECK Elastic Cloud on Kubernetes: A Kubernetes (k8s) Operator for the Elastic logging-stack in k8s, simplifying deployment and management of Elastic resources in k8s.

EDR Endpoint Detection and Response: Cybersecurity technology that continuously monitors an endpoint, e.g. a mobile phone, workstation, or server, to mitigate cyber security threats.

EOL End Of Life: A particular release version of a software reaches its ~ when it is no longer actively maintained and no longer receives security patches.

FN False Negative: In the context of an Intrusion Detection System (IDS), a ~ occurs when a real attack takes place and the IDS does not alert on it.

FP False Positive: In the context of an Intrusion Detection System (IDS), a ~ occurs when it wrongly alerts on an anomaly that is no real attack.

GCP Google Cloud Platform: Google's service to run applications in its cloud environment.

GUI Graphical User Interface: A form of interface for users to interact with electronic devices, which uses graphical icons and animations and is often used with a mouse or touchpad.

HA High Availability: Property of an IT system or application ensuring that it can operate continuously and without interruptions. Usually achieved by redundancy: If one server fails or needs to reboot for an update, the remaining servers keep the service available.

HTTP Hyper Text Transfer Protocol: Protocol used for a substantial part of the data transmissions on the Internet. Often also used in Application Programming Interfaces (APIs).

IDE Integrated Development Environment: A software with many features that support software development, for example, integrated debugging and compiling of an application.

IDS Intrusion Detection System: A system that detects and alerts on abnormal behaviour indicating attacks in the system it protects.

IP Internet Protocol: Communication protocol on the Internet, allowing to route traffic from one host to another. ~ is often used synonymously for the ~ addresses of each host.

IPC Inter Process Communication: Allows processes on a computer to communicate with each other using signals like SIGTERM or SIGKILL.

IRT Incident Response Team: A group of security analysts conducting digital forensics to figure out how a cyber attack took place.

JSON JavaScript Object Notation: Human-readable text-based data exchange format, which can be used to serialise uncomplicated data structures.

k8s Kubernetes: Open-source container orchestration platform maintained by the Cloud Native Computing Foundation (CNCF).

mTLS mutual Transport Layer Security: Method for mutual authentication and encryption between microservices in a Kubernetes (k8s) cluster.

NAT Network Address Translation: A mechanism that allows several computers to share the same IP, for instance in a home network where all computers share the home router's IP.

NTNU Norwegian University of Science and Technology, headquartered in Trondheim.

OS Operating System: Manages the hardware on a computer and divides it between the different applications running on the system.

PoC Proof of Concept: Realising a certain method, principle or idea to demonstrate its viability or feasibility, or a demonstration to verify that a concept or theory has practical potential.

POLP Principle of Least Privilege: Security principle stating that any user of a system only should have the access that is strictly required to do their job, and no more.

PR Pull Request: After a review of the change proposals for software from a ~ by other collaborators, the changes can be merged to become part of a future release of the software.

RAM Random Access Memory: Very fast, but volatile memory used by the processor to keep the state of the Operating System (OS). Not persistent and thus erased on reboot.

RBAC Role-Based Access Control: Security system that grants access to resources based on roles assigned to users or groups. Each user or group can have multiple roles.

rpi Raspberry Pi: Very small, cheap and lightweight Advanced RISC Machine (ARM)-based embedded computer about the size of a big matchbox.

SSH Secure Shell: A protocol allowing to remote control the command-line (cmd) on another machine securely authenticated.

TLS Transport Layer Security: A protocol providing encryption and authentication for secure communication on the Internet.

TP True Positive: In the context of an Intrusion Detection System (IDS), a ~ occurs when it correctly alerts on a real attack on the system.

vCPU Virtual Central Processing Unit: Used in cloud environments to emulate processor cores.

VM Virtual Machine: Software-based environment simulating a hardware-based environment. Allows e.g. to run a Linux OS in a window on a Windows host computer.

VPN Virtual Private Network: Mechanism to create a securely encrypted point-to-point link (tunnel) e.g. between a remote workstation and a company network via the Internet.

YAML YAML Ain't Markup Language: A human-readable data serialisation language, which is commonly used for configuration files.

Chapter 1
Introduction

This chapter gives an introduction to microservices and containerisation: What motivated their usage and how can container orchestration tools like Kubernetes manage them? In the end, different approaches to secure these environments are presented, along with the research questions addressed in this thesis. This chapter also lays the foundation for the scoping of the topics addressed in this thesis.

1.1 From monolithic to microservice architecture

Since the beginning of software development in the 1970s and 1980s, most applications were built using the monolithic software architecture. A monolith is a singular executable software application, where all the code required to run the application like the user interface, application logic and database is bundled into one single codebase, all of which is usually served from an on-premises server ([Won22]).

While this approach works well for smaller applications, many companies faced challenges as their monoliths grew bigger. According to Bigelow and Gillis ([BG23]), it can be difficult for developers to narrow down and isolate errors in a big monolith. Scaling the monolith is also difficult as there are limits to how big and resourceful a single on-premises server can be.

To overcome these and other challenges, the monolith can be split into smaller, loosely coupled, independent components called microservices; see Figure 1.1. Each microservice represents a business function and they communicate with each other through well-defined Application Programming Interfaces (APIs). According to Bigelow et al. ([BG23]), Wong ([Won22]) and Lumetta ([Lum18]), this has the following benefits:

1. **Independence**: As long as the APIs do not change, each microservice can use its own technology stack. Ergo, each development team can choose what suits their needs best.
2. **Isolation**: As the communication goes through well-defined APIs, it is easier to narrow down and locate errors. It is also easier to monitor the performance of the components in the application.
3. **Lifecycle**: Each microservice can be updated and deployed individually, so it is no longer necessary to update the whole application to update one component.

4. **Scalability**: If one microservice experiences a high demand, more instances of it can be deployed to handle it, so it is no longer necessary to scale the whole application when only one of its services experiences high demand.
5. **Testability**: The clear enclosure of the microservices also simplifies unit and resilience testing with tools like Netflix's Chaos Monkey.

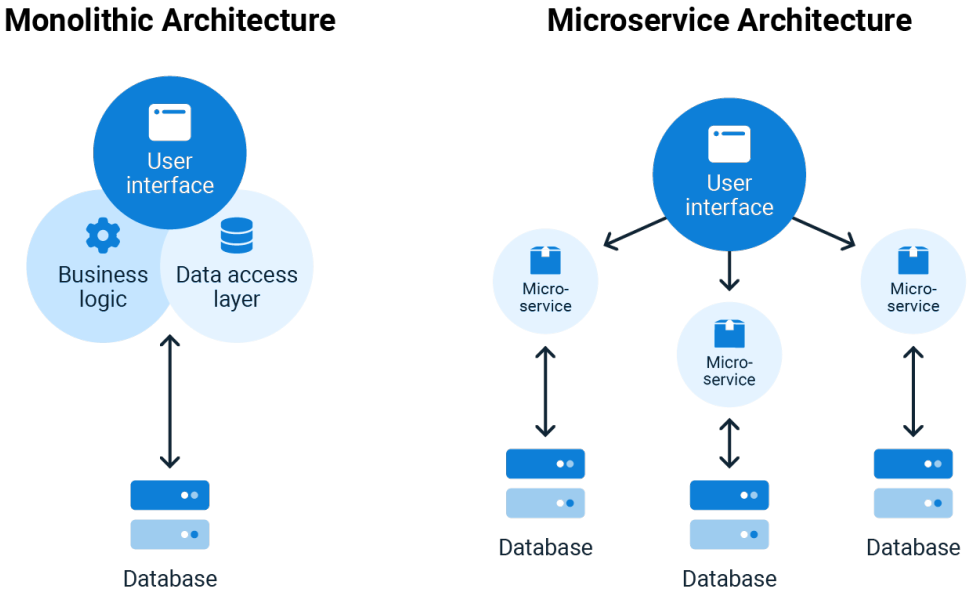


Figure 1.1: Comparison of the monolithic and microservice architecture, figure from [Won22]. Each microservice represents a business function and communicates with its own instance of the database.

Al-Dabagy and Martinek ([AM18]) state that a lot of big companies like Netflix, eBay and Amazon now use microservices and the adoption is growing. However, the monolith can still be easier to deploy for smaller businesses, because microservices add some additional overhead.

1.2 From Virtual Machines to containers

How can microservices be managed and run? One possible way is to run each microservice on its own Virtual Machine (VM). One or several physical servers run the VM hypervisor software, which is used to emulate the VMs. Each VM runs its own full Operating System (OS), ensuring a complete separation of the host OS on which the VM runs and the guest OS inside the VM.

When the containerisation technology Docker was released in 2013, it introduced a new approach to run different microservices on the same machine separately from each other. Instead of emulating the physical hardware to the VM like the VM-hypervisor does, the containerisation engine emulates the OS to the container ([Kum22]). As a result, the container itself does not need to incorporate the whole OS, it only wraps the application and all its dependencies.

In Figure 1.2, Kumar ([Kum22]) shows some of the most important differences between VMs and containers. First, as it does not incorporate the OS, a container's disk space requirements are reduced from several gigabytes for a VM to some megabytes. As containers share the host OS's kernel, they also require less Random Access Memory (RAM) and time to start and stop.

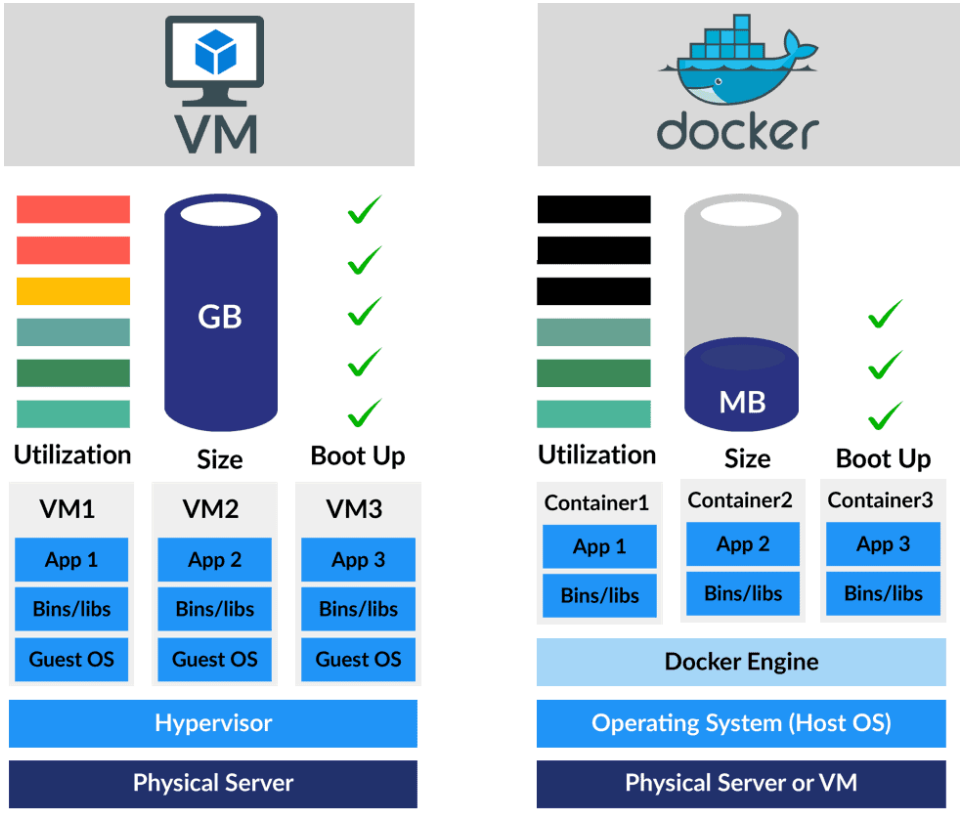


Figure 1.2: Running microservices in Virtual Machines (VMs) versus containers, figure from [Kum22]

Container processes directly run on the host OS[1], the containerisation engine only ensures that they run in a separate namespace. This namespace limits the container process's resources and also which other processes it can see, but as Huang and Jumde ([HJ20], page 67) mention, there is a possibility for container processes to "escape" their namespace and take control over the host. Consequently, containers are less isolated from the host and from each other than VMs.

For the most part, containers are constructed based on images, static files including the executable code running on a container. The image contains the system libraries, tools and other settings the microservice needs to run ([Ear23]). Containers are by definition ephemeral and thus short-lived; if a container crashes or more capacity is needed to handle the load, new containers are started based on the aforementioned image.

No data is stored permanently in the container. If it dies, everything saved directly in the container's file system is lost. Volumes can be used to store data persistently outside a container, but the container itself is not persistent ([Doc24]). For VMs, the data is stored on a virtual hard disk and thus stays available if after a reboot. Accordingly, it is usually more cumbersome to spin up additional VMs than starting new containers to scale up or down a service.

To conclude this comparison of VMs and containers, the adaption of containers is still growing ([Kre20]). However, both VMs and containers have their pros and cons and in most companies, both technologies are used to some extent for different use cases.

[1] This is true for Linux hosts only. On Windows and Mac, a special VM is used to provide the running containers with a kernel, as these OSs do not support containers natively.

1.3 Running and managing container clusters in Kubernetes

Several different containerisation engines are available providing the backend to run containers. Since it came in 2013, Docker has been the most commonly used and widely adopted one ([WHG21]), but there are also alternatives like PodMan or containerd. Yet, the containerisation engine itself only provides the runtime for the containers. But what can be used actually to manage the containers and, for example, start more containers if there is a higher demand for a particular microservice?

The technical term for this kind of software is container orchestration tool. Figure 1.3 shows the features they enable: Load balancing, scheduling, networking, automated rollouts and rollbacks, health monitoring, and scaling based on workload in a cluster of nodes running the containers ([Fas24]). On each cluster node, the container orchestration tool runs on top of the containerisation engine to manage the containers. Several container orchestration tools have appeared on the market recently, including Docker Swarm, Rancher, Helios and OpenShift.

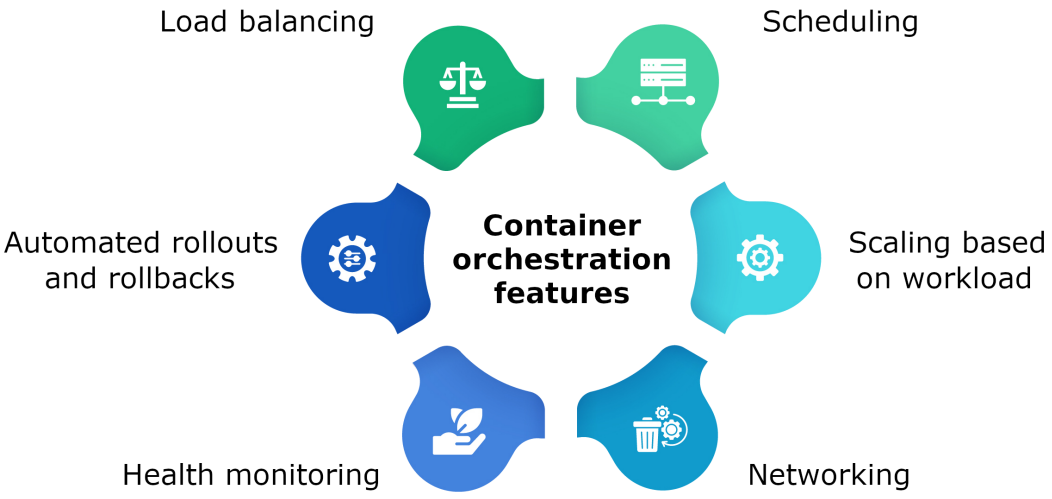


Figure 1.3: Features provided by container orchestration tools, figure adapted from [Kum23]

Nevertheless, the by far most used and best-known container orchestration tool is Kubernetes, having a market share of almost 80 % in 2019[2] according to Sysdig ([Car19]). It was originally started as an open-source project by Google, but in 2015, the development and responsibility were handed over to the Cloud Native Computing Foundation (CNCF) ([Fas24]). According to Al-Debagy and Martinetk ([AM18]), many big companies like Netflix, Amazon and eBay have already adopted Kubernetes and they also suggest that many others are likely to follow.

[2] Almost no reliable recent numbers were available, Statista's report ([Vai23]) found 97 % of Sysdig's customers using Kubernetes in 2022. The important point here is that Kubernetes dominates the market.

1.4 Research Questions about securing Kubernetes clusters

As mentioned in the previous section, Kubernetes can automate many cumbersome tasks for developers and system engineers. It can be seen as a monolithic management application solving the new problems that arise through the use of microservices. However, Kubernetes can also be a "complicated beast", as Sysdig's CEO Loris Degioanni states in the foreword to Huang and Jumde ([HJ20]). A big modern Kubernetes cluster can comprise thousands of containers, which all need to be configured properly not to create new security vulnerabilities in the cluster deployment. This is confirmed in Palo Alto's "2023 State of Cloud Native Security Report" ([Pal23]), which found that security is the top challenge for Kubernetes users. Hereby, the central issue this thesis seeks to address is introduced: Given its complexity and versatility, what are the best methods to secure Kubernetes deployments? This problem can be approached from two distinct angles:

- **Proactive angle**: How can misconfigurations and code smells or vulnerabilities in image files be identified before they are deployed to the cluster? What is the most efficient approach to configure the security and trust boundaries in Kubernetes?
- **Reactive angle**: Errors can occur regardless of the efforts invested into doing everything right beforehand. This has been demonstrated repeatedly by zero-day exploits. Therefore, what are the outcomes if an exploit still occurs? What potential consequences might it have and how can it be exposed to ensure proper and quick remediation?

For this thesis, the focus will be on the reactive angle since intrusion detection belongs to the reactive side of Kubernetes security. As decided in the thesis preparation project ([Neu23]), the following research questions (RQs) will be addressed:

RQ1: What are the most important assets to be protected in Kubernetes and what are the most impactful attacks on them?

RQ2: Is there a reliable tool available for intrusion detection in Kubernetes? And if not – what should be considered for the implementation of such a tool?

The research questions imply the following hypotheses (Hs) to be evaluated in this thesis:

H1: There is a Kubernetes component clearly standing out as the most important one to protect.

H2: There is a technology stack working better than others for intrusion detection in Kubernetes.

Chapter 2
Method and related work

This chapter describes how the results found in this thesis were obtained, and important findings from work related to it.

2.1 Method and outline

This thesis facilitates principles of the widely used scientific method, a description of which was found in a paper from Dodig-Crnkovic ([Dod02]). Below, it is explained how the steps in this method have been addressed in this thesis. The first chapter introduces the topic, along with the need to further investigate intrusion detection capabilities in k8s. Section 1.4 sums up the chapter and formulates the research questions: What are the most important assets to protect in Kubernetes (k8s), which tools are available to do so, and if there are none what would be important to consider for implementing a new tool? Furthermore, a corresponding hypothesis is formulated for each research question. Contrary to Dodig-Crnkovic's proposal of steps for the scientific method ([Dod02]), no predictions were formulated for the hypotheses to minimise bias in the evaluation of all possible outcomes.

Guided by the research questions, the literature review presented in section 2.3 was conducted. The studied literature was found on different sources: Published scientific papers and books were queried through Google Scholar and oria.no, the Norwegian universities' online search catalogue. When only insufficient published scientific sources could be found, the results were replenished with other websites found on Google. Furthermore, the formal documentation of k8s and other tools was consulted. This literature review formed the basis to answer the first research question RQ1 in chapter 3, concluded by a high-level risk analysis of the assets in k8s to identify what to focus on in the following and evaluate hypothesis H1. Additionally, it provided the theoretical knowledge of k8s' inner workings better to understand k8s tools and documentation.

Discussions with experienced coworkers helped to prioritise which use cases to focus on for answering the second research question RQ2 and evaluate hypothesis H2. A clear and structured testing methodology with well-defined criteria was needed to improve the consistency and reproducibility of test results and allow a quantitative comparison of tools facilitating intrusion detection in k8s. No standardised criteria were found for evaluating intrusion detection tools or software in general. Therefore, a novel evaluation framework for k8s intrusion detection tools was developed. Details about the process can be found in section 5.1. It is based on installing and configuring the tools as a Proof of Concept (PoC) of production usage.

Moreover, a lab environment was required to install and test k8s tools, the considerations and details of its deployment can be found in chapter 4. The tools needed for intrusion detection in k8s were divided into two categories: Forensics tools providing an overview of the services running in the cluster and other information about it are evaluated in chapter 5. Tools allowing to detect attacks and abnormal behaviour in k8s and alerting on it are evaluated in chapter 6. Based on this, the second research question RQ2 could be addressed in chapter 7, followed by an evaluation of the second hypothesis H2.

According to Dodig-Crnkovic ([Dod02]), the last step of the scientific method involves several similar iterations trying to solve the problem. At first, one approach is tried, and if it fails other approaches are tried until a better solution is found. The method of this thesis was not iterative in that sense since there were no repeated iterations to solve the same problem. Nevertheless, the results from each chapter guided and influenced the work further on, and the evaluation methodology for the k8s security tools was refined along the way.

2.2 Tools

This section gives an overview of the most important tools used in this thesis.

Text production

Mostly, the online Latex-editor Overleaf was used to write this thesis. Sometimes, a local installation of TexStudio was used when no steady Internet connection was available while travelling. References for websites not being published scientific papers were generated with Chegg's bibme, which performs automated scrapes to find necessary data for the citation. Furthermore, Thesaurus.com and the PONS-dictionary were consulted for language-related questions.

The text-generating Artificial Intelligence (AI) tools ChatGPT, Grammarly and deepl.com were only used to verify grammar and figuring out good ways to formulate certain aspects, but never to generate longer text fragments or retrieve knowledge. Other than that, the AI tools ChatGPT and GitHub Copilot were referenced as a source of inspiration for code generation and solving technical issues.

Many of the included images are screenshots, if necessary they were adapted using GIMP or Inkscape. To create certain figures such as 3.2, PlantUML was used. The decision to utilize it was influenced by the ChatGPT-based AI from chatuml.com, which provided the first sketch for Figure 3.2. However, it was eventually found to be more efficient to manually write and adapt the PlantUML code locally in a PlantUML Docker container without relying on chatuml.com.

Technology stack for testing Kubernetes tools

A lab environment was necessary to evaluate k8s intrusion detection tools for answering the second research question RQ2. NTNU provided an OpenStack server with a dedicated project, on which Virtual Machines (VMs) could be run to deploy k8s. In the end, two clusters were deployed: One in OpenStack's native k8s deployment tool Magnum, and another one with microk8s on three dedicated VMs running Ubuntu-Linux. More about the deployment of this lab environment and the considerations behind it can be found in chapter 4.

Most of the k8s cluster administration was done via command-line (cmd), as the Operating Systems (OSs) on the cluster nodes did not have Graphical User Interface (GUI) libraries installed. Linux-, Mac and Windows machines were used to remotely connect to the cluster nodes via Secure Shell (SSH). The main entry point to administrate the clusters in OpenStack was `ntnu-shell`, a Ubuntu Linux-based server available to all students at NTNU via SSH. As it has access to everything that is deployed on NTNU's premises, it can be used as an alternative to establishing a Virtual Private Network (VPN) connection to NTNU for reaching the k8s clusters.

SSH config files were used to make each node reachable via a simple name like `master-1`. The node's IP address was stored in the SSH config file along with other configuration like the default username, so it did not need to be specified and remembered manually each time a node was accessed. One problem with SSH sessions to a remote host is that they can "die" if the development host is shut down or has an unstable Internet connection. Any process that was started from an SSH session that died is also terminated immediately[1]. Consequently, a development host must stay online with stable Internet constantly until a complex upgrade operation on a remote server via SSH is finished, even if it takes several hours.

To address this issue, the terminal multiplexer tmux was installed on `ntnu-shell`. It starts a special session on the server, from which SSH connections to the clients are initiated, and they are likely to remain stable: As `ntnu-shell` is a server maintained by NTNU, it is rarely rebooted and the network connection to the nodes is unlikely to break. With tmux, a developer can start an upgrade operation, turn off their machine and return the day after to check the results: The SSH session from `ntnu-shell` stays alive, and so does the upgrade process. In tmux, several terminal windows can be opened, here one was opened for each node. There are also many other features, for example, the same commands can be run on multiple hosts simultaneously. Figure 2.1 demonstrates this for `kubectl get nodes` on all the nodes in the microk8s cluster.

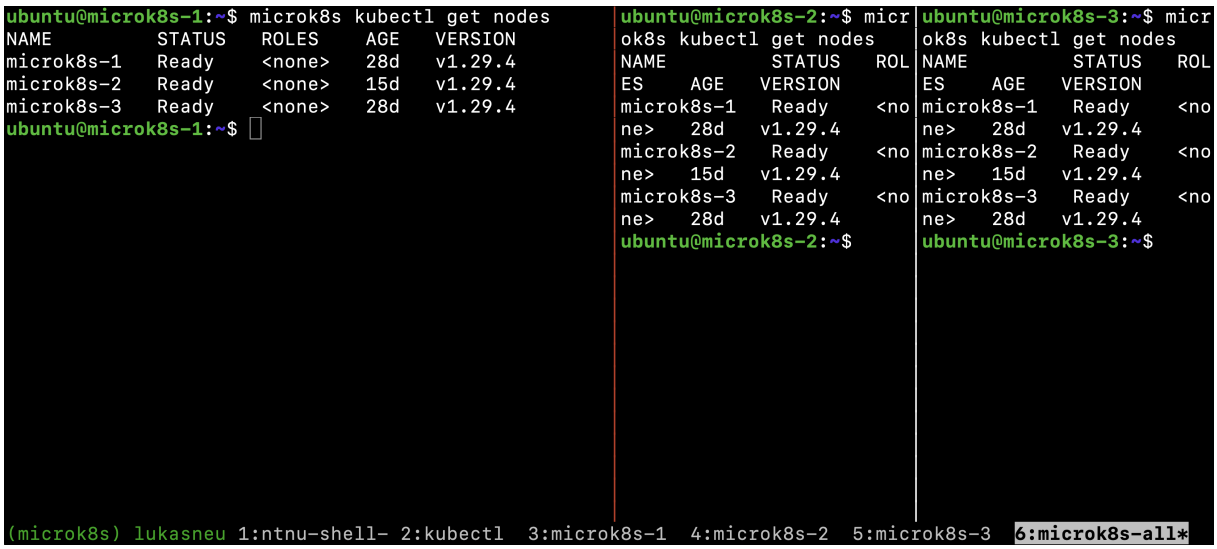


Figure 2.1: Terminal multiplexer tmux running on `ntnu-shell`, demonstrating how the same command can be run on the three microk8s cluster nodes simultaneously.

[1] At least on Linux servers, where all the child processes are killed with the parent process who started them.

During testing and installation, mostly typical Linux tools available in the Linux cmds bash and zsh were used. For each mentioned installation or evaluated k8s tool, the homepage is linked in the beginning as a citation. No particular mention of the installation process implies that it went through smoothly based on its documentation. Details are brought up if undocumented errors or dependencies were discovered or the installation was found particularly complicated. Moreover, undocumented solutions to installation problems are pointed out on discovery.

For k8s tools, the simplest deployment strategy was tried first. Consequently, deployments via Helm-chart only requiring a single command for installation were preferred and used if possible. Helm-deployments are also easily reproducible and the whole deployment process is updated along with the tool itself. Other important tools for k8s deployments and administration were kubectl and the Integrated Development Environments (IDEs) Visual Studio Code and Lens, depending on the use case. K9s was quickly integrated into the workflow after its discovery, as it noticeably accelerated tasks like port forwarding or enumerating and jumping between k8s resources. Details about all these k8s tools can be found in section 5.2 in chapter 5.

2.3 Related work

As the adoption of Kubernetes (k8s) is growing worldwide, securing k8s deployments also has become an actively researched field ([Kar+23], [Ngu+20]). Querying papers about k8s security reveals that a lot of the research is very recent, with most papers published in the last five years. This section reuses parts of the research that I conducted in the preparation project for this thesis ([Neu23]).

To get a first overview of what k8s is and how it works, several YouTube tutorials from "Techworld with Nana" by Nana Janashia ([Jan20]) were studied, like for example "Kubernetes: Zero to Hero". Bigelow and Gillis ([BG23]), Lumetta ([Lum18]), Wong ([Won22]), and Al-Debagy and Martinek ([AM18]) were examined to get a hold of the microservices architecture being one of the main reasons to deploy and use k8s. Earls ([Ear23]) and Kumar ([Kum22]) were consulted for the exact difference and technical details of containers and Virtual Machines (VMs). The first queries for literature were about "Kubernetes Security", first on oria.no and Google Scholar, and later on Google.

Huang and Jumde's peer-reviewed book "Learn Kubernetes Security" from 2020 ([HJ20]) provides a detailed overview of many aspects that are important to consider about security in k8s environments, like how the "Principle of Least Privilege" can be applied, security and trust boundaries, and scanning of images in a deployment pipeline. For all the different k8s components, the function and interaction with other components in the k8s architecture is highlighted. As the book focuses on the security aspect, typical vulnerabilities in components or configurations are pointed out wherever applicable. Moreover, there are sections about different security tools for k8s along with many practical examples.

Sysdig covers similar topics in their book "Kubernetes security guide", however, it is less detailed and focuses more on practical examples ([Sys21]). Another introduction to Kubernetes security is given by Kaelble in his book "Kubernetes Security for dummies" ([Kae24]). It is more superficial than Huang and Jumde ([HJ20]) and does not provide just as detailed explanations

of the underlying technology or real-world examples. Gupta's "Introduction to Kubernetes Networking and Security" ([Gup21a]) puts more emphasis on the networking aspects of k8s security, specifically on the extended capabilities of the k8s network driver Calico.

Infrastructure as Code (IaC) is an innovative method for provisioning resources like servers and databases using predefined, versionable configuration files. Kubernetes also utilizes this approach. Rahman and Akond identified and analyzed seven common code smells in IaC deployments, referred to as "The Seven Sins" ([RPW19]). Building on this work, Dell'Immagine et al. introduced the KubeHound tool, which can automatically detect these code smells in Kubernetes deployment scripts ([DSB23]).

Chondamrongkul et al. ([CSW20]) and Zdun et al. ([Zdu+23]) propose automated security analysis methods for microservice architectures, identifying potential attack vectors. Rahman and Akond provide another overview of misconfigurations, specifically targeted to Kubernetes misconfigurations ([RPW19]). Zahnoor et al. delve deeper into security policies in Kubernetes, discovering that they are often redundantly defined at different levels. Their approach uses a programming formalism called Event-Calculus to clean up these policies ([Zah+23]). To focus the search results more on the reactive side of k8s security, the search terms were now refined to "Intrusion detection in Kubernetes".

Hyder et al. ([HAA22]) explore the possibilities of Moving Target Defence (MTD) to mislead attackers and give the defenders more time to plan their steps. Darwesh et al. ([DHV23]) identify a current lack of real-time detection capabilities for Kubernetes clusters. In their paper, a proof of concept for an intrusion detection agent is presented, which successfully managed to collect 24 security metrics in a short amount of time. Aly et al. ([Aly+24]) propose an advanced AI multi-class detection method based on the Naïve Bayes algorithm, which can classify various types of security threats within k8s environments. Their detections run on a fixed dataset to ensure verifiability and comparability of the results. In "Practical Cloud Native Security with Falco", Degioanni and Grasso present Falco, an open-source threat detection tool for k8s ([DG22]).

As they contain recent, relevant research, two Master's theses are also included here even though they are not scientifically peer-reviewed: Kramer ([Kra24]) examines the implications of deploying service meshes and eBPF programs into k8s clusters, both of which can be used for intrusion detection and other purposes. Tinney ([Tin20]) explores the usage of honeypots and port scanners for intrusion detection in k8s.

Most of the literature about security in Kubernetes environments can be divided into the following three categories, some of it fits into multiple categories simultaneously:

1. **Focused mainly on the proactive side of k8s security**: Most of the research articles that were found fall into this category, dealing with topics like these: How can design flaws and misconfigurations be detected as early as possible? How can vulnerability and image scanning be integrated into the development pipeline? For example, the aforementioned papers by Rahman and Akond ([RPW19], ([RPW19]), Dell'Immagine et al. ([DSB23]), Chondamrongkul et al. ([CSW20]) and Zdun et al. ([Zdu+23]) belong to this category.

2. **Linked to a commercial product or company**: Many of the available books, websites and sometimes even research papers about k8s security are in some way related to or sponsored by a company selling a k8s security product. Some examples: Huang and Jumde ([HJ20]) work for Sysdig, which also released another k8s security guide ([Sys21]). Kaelble's book ([Kae24]) is sponsored by and advertises wiz. Tigera, the company behind Calico, sponsored Gupta's book about k8s networking ([Gup21b]), which puts special emphasis on Calico's paid enterprise capabilities.
3. **New algorithms or strategies for k8s threat detection**: Papers in this category propose new threat detection or -defence strategies, for instance, using AI or new technology. Aforementioned examples: Aly et al. ([Aly+24]), Hyder et al. ([HAA22]), Darwesh et al. ([DHV23]), Tinney ([Tin20]) and Kramer ([Kra24]).

After having studied some articles in the first category, research in this direction was suspended because this thesis is focused on reactive k8s security, not proactive. Literature in the second category bears the risk of being biased towards specific products or solutions, thus not covering the whole picture and requiring cross-verification. Knowledge from the third category can often be applied to extend, improve or build new k8s security tools. As long as the findings have not been applied to a readily available tool yet, their relevance for this thesis is limited: The main focus is evaluating available tools, not developing new intrusion detection mechanisms.

Most of the available literature on the topic is from the last few years, indicating significant research activity in the area of k8s security. However, looking back at the three categories a large proportion of k8s security literature can be assigned to, a gap in the current literature becomes apparent: What are the different advantages and disadvantages of the tools available for the reactive side of k8s security, compared without bias from the companies that sell them? This thesis seeks to address this issue, providing the reader with a solid basis to choose their preferred technology stack for intrusion detection in k8s.

Chapter 3
Assets and attack vectors in Kubernetes

This chapter presents the different components of Kubernetes (k8s) and how they interact. Meanwhile, these assets' possible attack vectors and vulnerable properties are identified and analysed. The chapter concludes with a list of assets and attack vectors, prioritised by risk and impact of compromise. This high-level risk analysis is conducted to identify what to focus on for the remainder of this thesis.

3.1 What is a Kubernetes cluster comprised of?

K8s is based on a client-server architecture, in which multiple master nodes control multiple worker nodes ([HJ20], page 7). It is possible to run the worker and master functions on the same machine for testing. However, separate machines are recommended to ensure redundancy and high availability in production clusters.

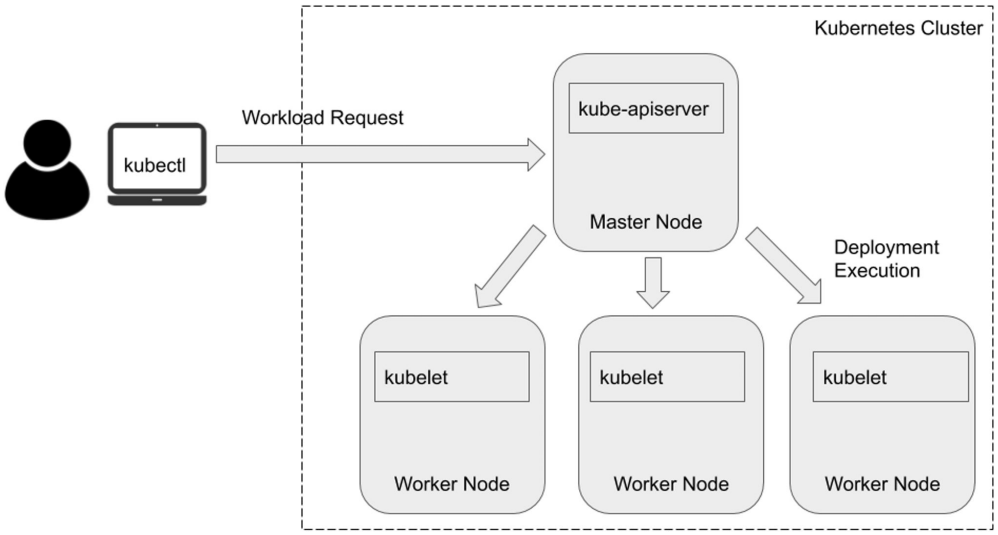


Figure 3.1: Kubernetes deployment with master and worker nodes. Figure from [HJ20], page 7

One of the most important components of the master nodes is the kube-apiserver, the single point of contact for controlling and configuring k8s clusters. Consequently, any GUI or command-line tool for controlling k8s like kubectl connects to it. Figure 3.1 shows how kube-apiserver then forwards the requests to the worker-nodes, where they are executed by the node agent kubelet ([HJ20], page 7).

3.2 Components of worker nodes

In k8s, the microservices run in their respective containers on the worker nodes. Big k8s clusters can incorporate thousands of worker nodes running millions of containers. This section highlights the most relevant components to secure in k8s worker nodes, see Figure 3.2 for an overview.

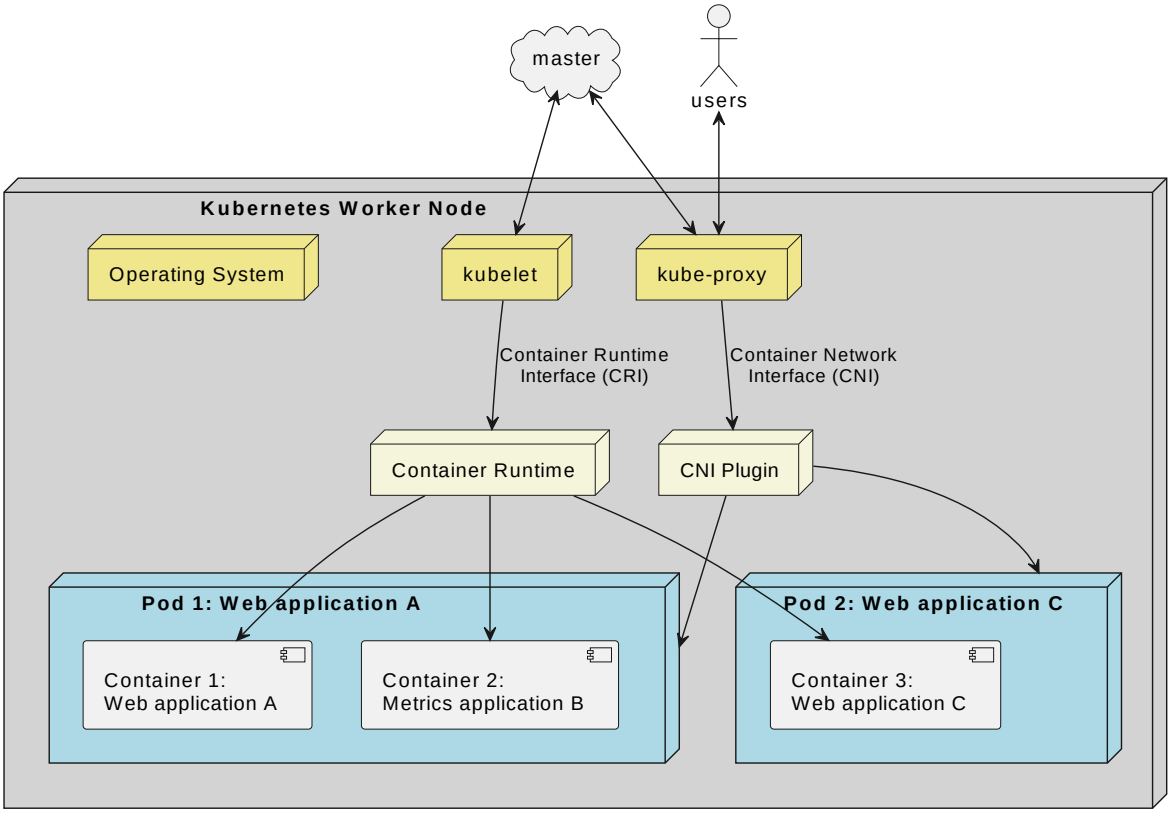


Figure 3.2: Important components to secure in a k8s worker node. Only a few interactions between components are visualized for readability. Users are entities outside the cluster, who utilise applications running in Pods like web application C. Usually, only one container runs in each Pod, but, for instance, web app A is bundled together in the same Pod with metrics app B that collects metrics about it.

K8s was designed to be independent of the container runtime it is running on, this is achieved through the Container Runtime Interface (CRI). It allows k8s to start, stop and run containers, for instance, in Docker, contained or Podman. Another level of abstraction is accomplished through Pods, the basic building blocks of k8s. A Pod usually groups one, sometimes several tightly coupled containers together and is the smallest unit that can be scheduled in k8s ([HJ20], pages 9-10). Throughout this chapter, an example of a web application A coupled with its metrics-collection application B in the same Pod will be used if applicable.

The Container Network Interface (CNI) defines the specification to implement plug-ins to configure network interfaces in Linux containers. The kube-proxy on each worker node is in charge of forwarding traffic to the right Pods and managing the networking rules. On behalf of the k8s-master, kubelet creates and monitors the Pods to safeguard their health ([HJ20], page 26).

3.2.1 Kube-proxy and the Container Network Interface (CNI): Networking

With the aid of a CNI-plugin, each Pod is assigned its own Internet Protocol (IP). By default[1], a Pod can communicate directly with the Pods on all other nodes without requiring Network Address Translation (NAT). Agents such as kubelet can communicate with Pods in the node they are running in. The Pod's IP is private, meaning that it is not publicly accessible from the Internet ([HJ20], page 22).

A microservice in k8s like web application A can run on many replicated Pods at the same time. K8s Pods are dynamic and ephemeral, so the number of Pod replicas varies because it is adapted to the load and Pods may crash. As a consequence, finding and saving Pod IPs to access a microservice would not be efficient. This challenge is addressed by the k8s Service, an abstraction grouping a set of Pods with a definition of how to access these Pods.

The Service is assigned a virtual IP[2], which is more stable and thus less likely to be changed frequently. So instead of manually having to keep track of all the Pods currently running web application A to access it, A's Service IP can be used. One of kube-proxy's tasks is to forward traffic directed to a Service's virtual IP to the Pods associated with it. For that, it watches the k8s master for the addition or removal of Services and Pods ([HJ20], page 26).

There are different CNI-plugins with different capabilities that can be deployed. The same is true for the kube-proxy, which can be deployed in different modes depending on the requirements. Compromising any of these components on a node could lead to loss of:

- **Confidentiality**: Several CNI-plugins and kube-proxy support encryption. If they are compromised, an attacker could see the traffic unencrypted.

  Another possible result of attacking the confidentiality could be data exfiltration. In theory, kube-proxy could redirect traffic from the k8s-network out to another server on the Internet, given that the firewall protecting the cluster does not block this communication.
- **Integrity**: A compromised kube-proxy could alter or invent and insert malicious traffic.
- **Availability**: A compromised CNI-plugin or kube-proxy could stop working and thus make the Service and the Pods behind it inaccessible.

[1] This can be restricted using namespaces or security policies like Pod or NetworkPolicies.

[2] It is a virtual IP as there is no namespace or network interface directly associated with it.

3.2.2 Container runtime and kubelet: Infrastructure to run containers

Considering the susceptibility of the node, the choice of container runtime plays an important role. According to Huang and Jumde ([HJ20], page 54), one possible vulnerability in the container runtime can be the lack of egress filters, which can allow malicious images to be fetched on the node. For instance, if the container image for the metrics-application B is fetched from public sources, it should be verified which other images can be fetched from there. Reeves et al. ([Ree+21]) demonstrated that container "escapes" are another major security hazard to consider, where attackers manage to overcome the isolation mechanisms between the container and the host OS leading to privilege escalation.

Voulgaris et al. ([Vou+22]) compare Docker and the relatively new container runtime Podman. Internally Docker uses containerd, which is also widely used on its own. Containerd relies on a daemon running in the background with root privileges, establishing a single point of failure. Podman improves performance by not relying on a daemon and running containers directly in the OS. This also allows containers to run in user mode without root privileges, which is considered more secure: An attacker managing to escape a Pod will not get root privileges immediately. Newly, containerd also released user mode for its daemon as a feature, but it is not widely used yet.

By directly interacting with the Container Runtime Interface (CRI), kubelet manages the Pods on the worker nodes. Additionally, kubelet monitors the Pods to safeguard their health and registers the worker node on kube-apiserver. They communicate bidirectionally: Kube-apiserver submits change requests to kubelet, and kubelet updates kube-apiserver about the current state of the Pods on the worker node ([HJ20], pages 48-49). An attacker who manages to compromise kubelet on a node practically gets full access to that node. As kubelet has access to the container runtime, it can start and stop containers and Pods, while still acting as if nothing had happened towards kube-apiserver. If the attacker also manages to compromise kube-proxy, all the implications listed in the last section 3.2.1 would be possible here, too.

To summarise, compromising kubelet or the container runtime can have the following impacts:

- **Privilege escalation**: The possible consequences of containers escaping their boundaries to gain unauthorized access are affected by the container runtime and its configuration.
- **Arbitrary code execution**: Without proper filtering, the risk of malicious container images being fetched and run increases.

3.2.3 Operating System (OS): Infrastructure for Kubernetes components

Containers incorporate microservices with all their dependencies, so they do not depend on much more than the container runtime itself on the host OS. Hence, deploying a k8s worker node for instance on the Ubuntu server OS leads to unnecessary overhead since most of the applications this OS leverages are not necessary. Therefore, several lightweight[3] Linux OSs like Fedora CoreOS and AWS Bottlerocket have been developed especially to run containerized workloads.

[3] Here, this implies they only contain what is strictly needed to run containers and keep the OS up to date.

Fedora CoreOS is completely designed around containers. Its file system is immutable, and its package manager OSTree is based on file system layers just like container images themselves ([Fed24]). When a new package or update is installed, all the changes to the filesystem are collected in a new file system layer. To activate it, a reboot is required. Installing packages is discouraged in CoreOS ([Bru20]), if possible applications that would run as a Service on general-purpose Linux OSs should run in a container. Even though this sounds tiresome at first glance, this process makes updating the nodes easier and more controllable. It also gives an attacker who gains root access to a node a harder time: Firstly, fewer tools are available on the OS compared to a general-purpose OS. Secondly, installing additional packages is hard and requires an easily detectable reboot.

To summarize, compromising the OS on a worker node can have the following consequences:

- **Privilege escalation**: A vulnerable OS can provide an attacker full access to the node, and ease further steps in the attack like lateral movement. Specialized container OSs increase in popularity and adoption due to cluster security and overall performance improvements and can be used for worker and master nodes. This shows that the choice of OS for the nodes in a k8s cluster plays an important role in hardening the attack surface.
- **Loss of confidentiality**: A compromised OS allows to see everything inside it.
- **Loss of integrity**: A compromised OS can alter traffic and files inside it.
- **Loss of availability**: Services running in a compromised OS can be disrupted.

3.2.4 Pod and container: Encapsulation and isolation for microservices

Pods are the basic building blocks of k8s and part of its true power. They can be configured to run almost any software on different types of hardware and OSs, ensuring the software in each Pod runs isolated from others. But how exactly are Pods constructed, and how does communication work inside and between them? And what is important to consider when securing them?

Usually one, sometimes several tightly related containers are gathered in a Pod. Pods are the smallest unit that can be scheduled in k8s, and configured in a Podspec. Each Pod is assigned exactly one IP, so two containers inside a Pod cannot listen on the same port. If several containers are inside a Pod, they usually run different applications listening on different ports. This is demonstrated for web application A and metrics-application B in Figure 3.3. Additionally, A and B can communicate using Inter Process Communication (IPC) if it has been activated for the Pod[4]. Each Pod also contains an additional sleep container, which holds the networking resources like the Pod's IP when no other containers are running ([HJ20], pages 23-25).

As mentioned in section 1.2, any changes made to a container's file system are lost if it gets recreated or restarted. To keep data persistently across containers, volumes can be used. Figure 3.3 shows how containers can use shared volumes inside their Pod to communicate. Still, containers are short-lived and ephemeral, Carter found in his study from 2019 ([Car19]) that over 50 % of the containers lived less than five minutes. There can be different reasons for this,

[4] By default, the properties `hostIPC` and `hostPID` are not active in the Podspec.

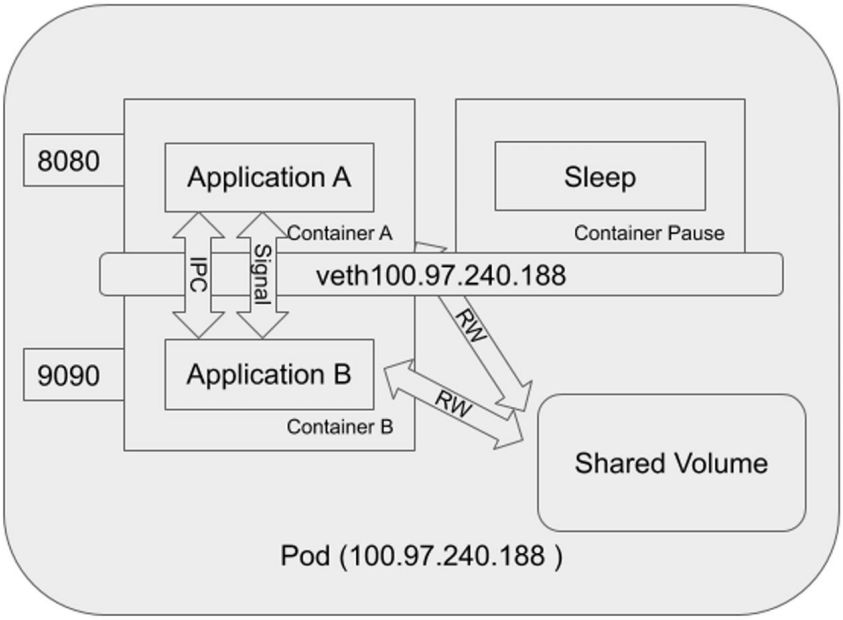


Figure 3.3: Communication inside a Pod with shared IP and network resources. From [HJ20], page 25

like developers spinning up lots of containers while testing. Nevertheless, the short lifespan of containers and Pods must be accounted for while securing them.

The following risks and consequences were identified if containers or Pods are compromised:

- **Arbitrary code execution**: Pods can do almost anything. Millions of images are available to base them on, and many companies develop their own. It can be hard to keep the overview, especially in big environments. To minimize the attack surface, the sources for pulling images publicly should be limited and chosen carefully.
- **Limited traceability**: The aforesaid short-livedness of containers complicates tracing them. According to Dao ([Dao20]), it can be hard to collect logs from all containers or backtrack exactly on which container something happened. Once a container has terminated, much valuable information from e.g. changes to its filesystem is unrecoverably lost.
- **Privilege escalation**: If a container manages to escape its isolation, it can get unauthorized access to resources like the node's OS or other containers.
- **Complex configuration**: Pod security can be hardened using e.g. Pod- and NetworkPolicies. However, according to Huang and Jumde ([HJ20], chapter 8), some of the default k8s settings can be "too open". Potentially, many privileges can be given to Pods like the ability to see all network traffic and processes on the node. Correspondingly, the maximal permissions Pods can obtain in a k8s cluster should be restricted.
- **Variety of goals**: Attacks on a k8s cluster can have different goals and are not limited to ransomware deployment or data exfiltration. For instance, Tesla's k8s console was hacked in 2018 with the sole goal of deploying containers for crypto mining ([Thu18]).

3.3 Components of master nodes

The k8s master, usually spread across several master nodes, controls and manages the workload running in Pods on the worker nodes. Figure 3.4 gives an overview of the most important components running on a k8s master node. Kubelet, kube-proxy, the OS and Container Runtime have been coloured yellow, as they also run on worker nodes. Considerations about these can be found in section 3.2. Master nodes don't run Pods with workload directly like worker nodes, but some master components like kube-dns also run in Pods. This is why for instance kubelet and a container runtime are needed in master nodes as well.

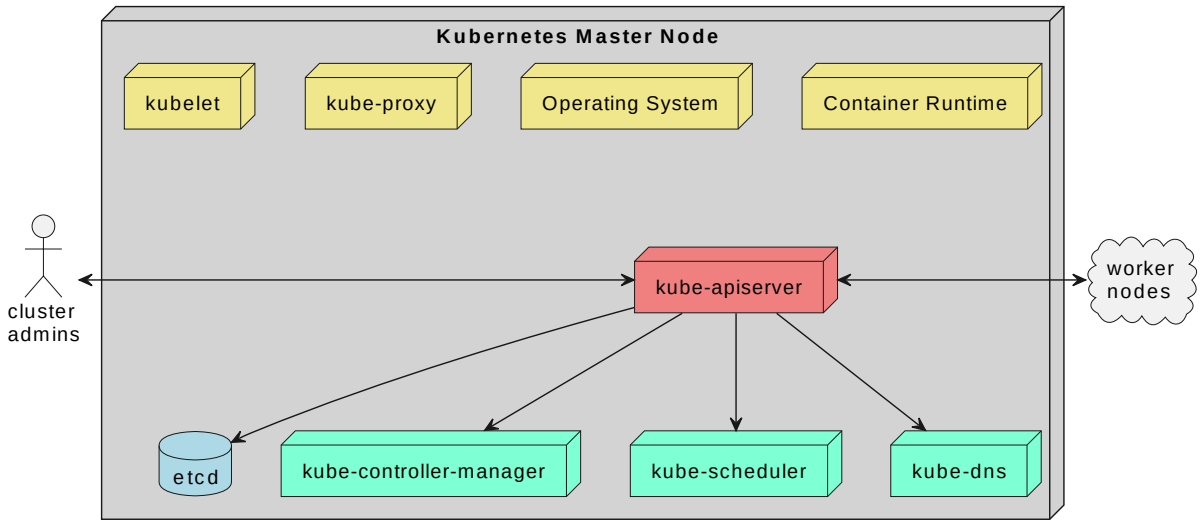


Figure 3.4: Important components to secure in a k8s master node. The yellow components are also found in worker nodes, more details about these can be found in section 3.2.

To better introduce the function and interaction of the other master components, the example from Figure 3.5 is used. It is inspired by Huang et al. ([HJ20], page 48), but it has been somewhat simplified for readability. Here, a k8s-administrator sends a request to the k8s master to increase the number of web application A's replicas from two to three. The request is sent to kube-apiserver, the single point of contact for administration of k8s clusters. Besides, the other master components mostly do not directly interact: They usually communicate through kube-apiserver ([HJ20], pages 7 and 48), the reasoning for this is given in subsection 3.3.2.

At first, kube-apiserver transmits a request to update web application A's configuration in etcd, a high-availability key-value store for configuration, state and metadata ([HJ20], page 7). After etcd has confirmed the update in the configuration, kube-apiserver informs the k8s-administrator that the update was registered. At this point, only the configuration is updated – to track when and how the additional Pod is created, the k8s-administrator would have to send supplementary requests to query the cluster state.

Etcd provides an API to watch configuration changes like the update of replicas, which kube-apiserver subscribes to. Based on that, kube-apiserver informs kube-controller-manager about the change. This component is a combination of core controllers watching for changes to the configuration, and then updating the cluster accordingly ([HJ20], page 8). Kube-controller-

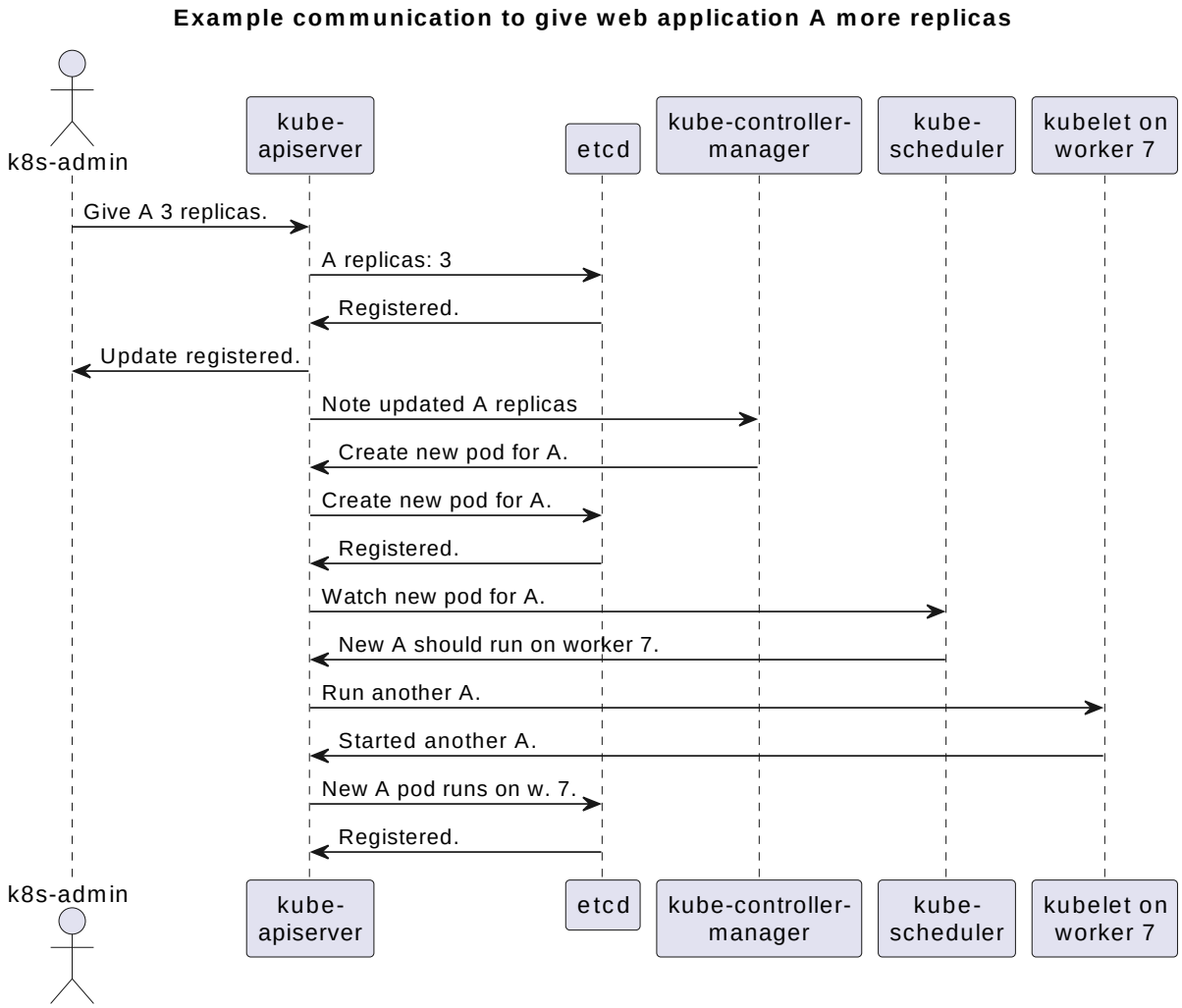


Figure 3.5: Example of communication between k8s master components: A k8s-admin increases the number of web application A's replicas to three. Based on [HJ20], page 48, but simplified for readability.

manager finds that an additional Pod is required for reaching three replicas. Consequently, kube-apiserver asks etcd to register an additional Pod definition.

Kube-scheduler watches for newly created Pod definitions and assigns a node to run the Pod based on different criteria like available resources or policies ([HJ20], page 8). In this case, kube-scheduler decides the new Pod for web application A should run on worker node seven. Kube-apiserver then contacts kubelet on worker node seven to run the Pod. Thereafter, the Pod definition in etcd is updated to include that it runs on worker seven. Finally, the state of three replicas for web application A is reached as requested by the k8s-administrator.

The Domain Name Service (DNS)-plugin is another master component, but did not fit into the example. It allows for instance a Service (see subsection 3.2.1) to be reached through a human-readable domain-name rather than its IP. The remaining subsections below analyse possible risks and consequences for the aforementioned master components in case they are compromised.

3.3.1 Etcd – etc distributed: Reliable storage for configuration and state

Most configuration files used for applications in Linux systems are stored in the etc directory. Originally, this meant et cetera, but nowadays "editable text configurations" better describes what it does. Etcd stands for "etc distributed" and is a High Availability (HA) key-value store for data such as the k8s cluster configuration, state and metadata ([HJ20], page 7). It is the only stateful k8s component, all the others use it to store their configuration and state. Etcd was initially developed by CoreOS[5] to facilitate its concurrent system for control of OS upgrades and the distribution and storage of configurations ([Fan20]). Like for k8s itself, the Cloud Native Computing Foundation (CNCF) has taken over the development.

In k8s, etcd can be deployed in two ways ([Gup21b]):

- **Stacked**: Etcd uses storage and runs directly on the master nodes as an application.
- **External**: Etcd runs on dedicated nodes in a separate cluster.

An external cluster adds some administration and hardware overhead but provides additional security: An attacker with access to a master node would not get immediate access to etcd.

To keep its data consistent, etcd uses the RAFT[6] consensus algorithm. At its basis, quorum needs to be achieved for any action like "add", "remove" or "update". According to Caban ([Cab19]), quorum is decided by having "a number of voting members greater than 50 % of the total number [N] of etcd instances", so for $N = 3$, at least two nodes need to vote. Consequently, RAFT enables fault-tolerance for losing up to $\frac{N-1}{2}$ nodes. For it to work, N must be odd, so the minimum number of nodes for a production cluster is three given that a single node provides no redundancy. One of the nodes is elected as a leader and responsible for data synchronisation and distribution. If it fails, another node is elected. Only one node needs to be contacted to read or write data: RAFT then fixes the synchronisation across the cluster ([Fan20]).

A compromise of etcd could have the following risks and consequences:

- **Loss of confidentiality**: In the default configuration, neither data travelling to and from nor data at rest in etcd is encrypted ([HJ20], page 49). As even secrets and access keys are stored here, not properly configuring encryption poses a substantial security risk.
- **Unauthorized access**: According to Huang and Jumde ([HJ20], page 52), etcd is the "brain" of the cluster. An attacker controlling it effectively controls the whole cluster. To minimise this risk, it is advisable to deploy etcd in a separate cluster. It should be protected by a firewall only granting access to kube-apiserver with enforced authentication ([HJ20], page 99). Regular and isolated backups are also advisable ([Kub24d]).

3.3.2 Kube-apiserver: Message broker and single point of contact

This component is the single point of contact into the k8s cluster – not only for k8s-administrators but also for the other k8s master components. In some way, it can be seen as a message relay

[5] More information about CoreOS can be found in subsection 3.2.3

[6] Acronym for Replicated And Fault-Tolerant

or message broker for k8s. At first glance, this might seem avoidable: Why cannot e.g. kube-scheduler contact etcd directly for updates?

In fact, it would be avoidable if all the k8s-components were running on the same machine. But as k8s is a distributed system which can scale over hundreds of nodes, using kube-apiserver as a message broker has the following benefits:

1. **Performance**: As etcd is the only stateful component, kube-apiserver can run in parallel and divide the load ([Kub24d]). In case of failure, another instance can take over.
2. **Efficient communication**: Instead of all components establishing connections between each other, they only connect to kube-apiserver ([HJ20], page 79).
3. **Security**: Instead of securing all components separately, the total attack surface is reduced to almost only kube-apiserver, from which communication can run well-defined, properly authenticated and encrypted.

On the worker nodes, kube-apiserver directly communicates with kube-proxy and kubelet, other worker components are configured through these or independently like the OS. As its name suggests, the communication with kube-apiserver runs through REST[7] Application Programming Interfaces (APIs). The aforementioned watch functionality in etcd allows components like kube-scheduler or kube-controller-manager to subscribe to specific changes. As only kube-apiserver communicates directly with etcd, it subscribes to the watch functionality on behalf of the other components and then forwards notifications about changes to them ([HJ20], page 47).

A compromise of kube-apiserver could have the following implications:

- **Limited Traceability**: By default, auditing on kube-apiserver is disabled, which prevents forensic analysis ([HJ20], page 52).
- **Unauthorized access**: Kube-apiserver is the "heart" of a k8s cluster ([HJ20], page 52). An attacker managing to compromise it gains control over the whole cluster.

  This risk can be minimized by restricting access to kube-apiserver and enforcing authentication and encryption ([HJ20], page 52).

3.3.3 Kube-controller-manager and cloud-controller-manager: Control loops

The core control loops shipped with k8s are running inside this component to regulate the state of the k8s cluster. Constantly, the control loops compare the administrator's desired state of the cluster with the actual state and make updates accordingly ([HJ20], page 8). From a logical perspective, the controllers each running one control loop are separate. Still, kube-controller-manager runs all controllers compiled into a single binary executable running in a single process ([Kub24d]). Some of the most important controllers are:

- **Node controller**: Responsible for noticing and responding when nodes go down or need to be added ([Kub24d]).

[7] Acronym for Representational State Transfer. Defines a couple of standards REST APIs should adhere to.

- **Replication controller**: Maintains the correct number of Pods for each microservice on the system ([HJ20], page 8).
- **Job controller**: Watches for Job objects representing one-off-tasks and creates Pods to run them until completion ([Kub24d]). A one-off task is only run once and then terminated.
- **Endpoint slice controller**: Populates EndpointSlice objects providing a link between Services and Pods ([Kub24d]).
- **Service accounts and tokens controller**: Creates default ServiceAccounts and API-tokens for new namespaces. ServiceAccounts are used by Pods interacting with kube-api-server for authentication. API-tokens are used for authentication to an API, and namespaces in k8s can be used to divide a k8s cluster into independent parts ([HJ20], page 8).

Cloud-controller-manager was introduced as a separate component to decouple cloud vendor code from k8s core code. It runs controllers to interact with the underlying cloud provider like Amazon Web Services (AWS), Azure Kubernetes Service (AKS) or Google Cloud Platform (GCP) ([HJ20], page 8). Cloud-controller-manager links the cluster into the cloud provider's API and only runs controllers necessary for the specific cloud provider. This component is not present in small testing environments or if k8s is deployed on-premises. Some of its controllers are ([Kub24d]):

- **Node controller**: Checks if a cloud node was deleted in case it stops answering.
- **Route controller**: Sets up routes in the cloud infrastructure.
- **Service controller**: Creates, updates and deletes cloud provider load balancers.

A compromise of one of the controller-managers can have the following risks and implications:

- **Loss of availability – Denial of Service (DoS)**: Spin up Pods until cluster breakdown.
- **Loss of confidentiality**: Access to the controller-manager allows to map out the cluster.

3.3.4 Kube-scheduler: Assigning Pods to nodes

The decisions of which node to run a workload on are taken by kube-scheduler. It is the default scheduler for k8s and watches for newly created Pods that have not been assigned a worker node to run on yet. To find the optimal node, different factors are taken into account ([Kub24d]):

- **Hardware constraints**: E.g. graphical processing might not be available everywhere.
- **Resource requirements**: Which nodes have enough resources left to run this Pod?
- **Data locality**: Data-heavy workloads run more efficiently close to their data.
- **Inter-workload interference**: Some workloads run more or less efficiently together.
- **Deadlines**: Workloads with a short deadline can require more hardware to finish in time.
- **Policies**: This criterion can override all the others. For instance, it allows manual specification to make a workload more or less affine to run on some nodes or restrict the execution to a specified set of nodes.

Compromising kube-scheduler can have the following risks and consequences ([HJ20], page 53):

- **Loss of availability (DoS)**: Disturbing the correct scheduling of Pods can be used to bring down nodes or make services unavailable.
- **Loss of confidentiality**: Access to the kube-scheduler allows to map out the cluster.

3.3.5 Domain Name Service (DNS): Identify for Kubernetes Services

K8s Services get their own virtual IP as mentioned in subsection 3.2.1, but often a human-readable descriptive domain name identifying each Service simplifies cluster administration significantly. This can be achieved by deploying a Domain Name Service (DNS)-plugin, which is recommended but not mandatory for k8s to work ([Kub24d]). Earlier, kube-dns deployed in three separate containers was the default DNS-plugin, but since v1.12, CoreDNS deployed in a single container is recommended in terms of security and performance ([HJ20], page 8).

A compromised k8s DNS-plugin could have the following risks and consequences:

- **Loss of availability (DoS)**: For instance, routing requests for Service domains to non-existing IP addresses or overloading Services by routing all traffic to a single Service.
- **Loss of confidentiality**: A compromised DNS-plugin can see all the services deployed in the cluster, but no other specific information about nodes, Pods et cetera. Thus it is less critical than for example kube-scheduler or kube-controller-manager.

3.4 High-level risk analysis of Kubernetes components

In the preceding sections of this chapter, the most important k8s components in worker and master nodes have been introduced to address RQ1 (see section 1.4). To prepare RQ2 in the following chapters, a qualitative high-level risk analysis is performed to determine which components to put the main focus on when intrusion detection tools for k8s are compared.

Risks of compromise for the components are defined as probability $\times$ impact, but the definition of compromise differs for worker and master components: For the former, that is the risk of them being compromised on a single worker node. For the latter, it is defined as an attacker gaining complete control over that component on all master nodes. If a worker component has a high impact of compromise, it can be used to fully control its node. A high impact for a master component corresponds to it being able to control the whole k8s cluster if it is compromised. Low impact only provides limited access and attack capabilities.

The probability of compromise has the same definition for worker and master components. Components that are highly configurable, extendable by plugins or allow running custom code get a high probability rating because it significantly increases their attack surface. Components which are not extendable and whose code base is entirely maintained by official entities like the Cloud Native Computing Foundation (CNCF) get a low probability rating.

The findings are summarised in Table 3.1 and 3.2. It is important to properly see the risks in Table 3.1 and 3.2 in relation to each other: A single compromised worker node has a far lower

impact than complete control over a master component. The reasoning for the probability and impact assessments is given in the next section 3.5.

Table 3.1: Summary of risks in worker components. The risks correspond to the component being compromised on a single node.

Component	Risks and Consequences	Probability	Impact
OS	Root access on the node gives full access to the node and all the k8s components running on it	Low	High
Kube-proxy & CNI	Can read (confidentiality) or alter (integrity) network traffic and limit service availability	Low	Medium
Container Runtime & kubelet	Allows to start and stop containers, affecting CIA triad properties. Almost like compromise of OS	Low	High
Container & Pod	Short-lived, hard to trace, variety of configurations, images and achievable attack goals → High risk. Impact like container runtime for escape	High	Low (Escape: High)

Table 3.2: Summary of risks in master components. The risks correspond to the entire component being controlled in the cluster, not only a single instance.

Component	Risks and Consequences	Probability	Impact
Kube-apiserver & etcd	Complete control of the whole cluster. Allows any attack on CIA triad and other assets	Low	High
Kube-scheduler	Allows to map out the whole cluster (confidentiality). Can be used for DoS (availability)	Low	Medium
Kube-controller-manager	Allows to map out the whole cluster (confidentiality). Can be used for DoS (availability)	Low	Medium
Kube-dns	Can only map out services in the cluster (confidentiality). Can be used for DoS (availability)	Low	Low

3.5 Prioritisation of assets and attack vectors

Analysing the risks has resulted in the following list of assets to protect, prioritised by the likelihood and impact of compromise:

1. **Pods and Containers**: Focus on traceability given their short-lived nature.
2. **OS on worker and master nodes**: Detect anomalous behaviour like container escapes.
3. **Master components kube-apiserver and etcd**: Detect anomalies / compromise.
4. **Other master and worker components**: Kubelet, kube-proxy, kube-scheduler etc.

Pods and containers are the most versatile component in k8s: Millions of images, versions and configurations exist. This makes them a very likely initial attack vector – in some way, attackers manage to smuggle malicious code into a container running in a Pod. It is sufficient to run specific containers on some worker nodes for some attacker goals like crypto mining. Compromising the

whole cluster is unnecessary and not even desirable because a full compromise is likely to be detected relatively fast, whereas attackers in a crypto mining attack would like to keep their containers undetected as long as possible.

The short-lived nature of Pods and containers introduced in subsection 3.2.4 makes them hard to trace after termination since all the changes made to the container's file system like log files are lost. Therefore, it is important to facilitate persistent auditing and logging for Pods and containers, enabling forensics to reconstruct how an attack was carried out initially. The second priority of keeping an eye on the OS on worker nodes is an extension of securing Pods and containers because attacks like container escapes to the OS are most likely detectable from the OS itself.

In the third and fourth priority on the list, other k8s-components whose code base is maintained by the Cloud Native Computing Foundation (CNCF) are found. Zero-day exploits are the most probable attack vector for these components to get compromised. The immense market share of k8s increases its attractivity for threat actors to look for new vulnerabilities. If they find one, many organisations can be attacked promising high profits. Consequently, zero-day exploits on k8s components have happened in the past and are going to happen in the future.

Nonetheless, new zero-day exploits are rare and thus hard and expensive to find. Previous zero-days like log4shell[8] have shown that they quickly become known to the public when countless targets are attacked. A highly skilled community of cyber security analysts in organisations like mnemonic or Mandiant often rapidly finds a way to detect the associated attack, a patch for the vulnerability usually follows shortly after. Complete protection against zero-days is impossible, but when they happen they can often quickly be detected and resolved given proper internal routines.

Misconfigurations leading to insufficient security hardening are another, more likely attack vector for k8s core components. Others would be the lack of proper updating routines for the cluster and its components, or wrong usage of policies. These are undoubtedly important but clearly fall under the proactive angle of k8s security, which is not the focus of this thesis (see section 1.4).

A compromise of kube-apiserver or etcd by far poses the highest risk, as it gives full control over the whole k8s cluster. Nevertheless, they appear no higher than third on the list simply because their compromise is less likely than a compromised Pod. Additionally, a significant part of the k8s deployments do not run on-premises, but in cloud environments provided by AWS, GCP or AKS. Cloud providers often administrate the k8s master components themselves, not allowing users direct access. Consequently, manually securing these components is impossible. This leaves no other way out than trusting the cloud providers to secure them properly.

While cloud providers do secure the master and worker components, Pods and containers still need to be verified and secured by the users themselves ([Kae24], page 29), confirming the need to first and foremost look at them. This reasoning concludes the chapter, giving a clear guideline of what to focus on for the rest of the thesis.

[8] A critical vulnerability in Java's widely used logging library log4j. It became known on December 10th, 2021.

Chapter 4
Lab environment to test tools

A lab environment was needed to get an understanding of the internal workings of k8s (RQ1) and to be able to test tools for intrusion detection in k8s against a baseline (RQ2). The different strategies and tools considered to build this environment are presented with their advantages and disadvantages, concluded by the lab environment that was finally used.

4.1 Kubernetes lab environments

This section shows the different deployment options that were considered for the lab environment.

4.1.1 Homelab on used machines

At first, a local implementation of a k8s cluster on old, physical machines was considered. On an x64 2010 Mac mini with eight GB of RAM, Ubuntu Server OS was set up, along with two Advanced RISC Machine (ARM)-based Raspberry Pis (rpis) running Raspbian OS with one GB of RAM each. Figure 4.1 gives an overview of this setup.

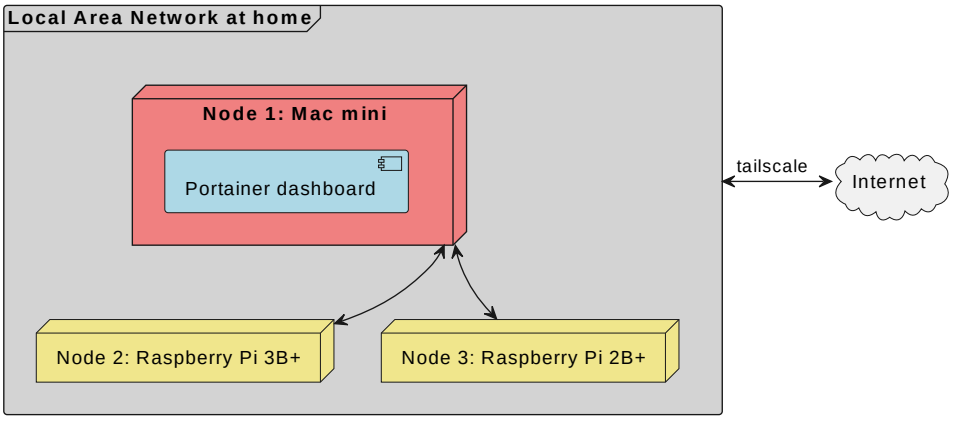


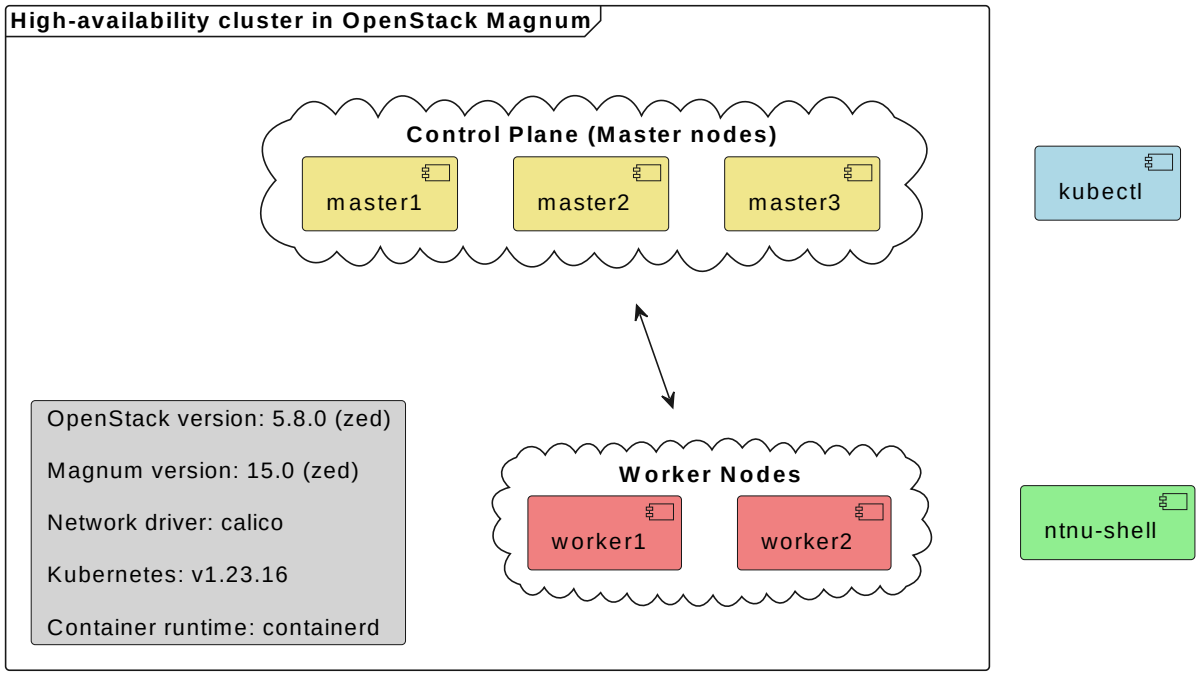
Figure 4.1: Setup of a k8s cluster with three nodes, a Mac mini and two rpis in the home network.

Portainer was deployed on the Mac mini to keep an overview of the nodes and the containers running on them. Additionally, Tailscale was installed on all nodes, allowing remote work on the cluster e.g. from the university campus in a securely authenticated way. However, this setup in the homelab on old hardware had some drawbacks:

1. **Incompatible processor architectures**: Most software today is optimized for the x86 / x64 architecture, which most servers today and the Mac mini in this setup are based on. Current trends indicate a transition towards more usage of the ARM-architecture in the future, but until now most software can't run on ARM. This makes a cluster with different architectures on its nodes challenging to use.
2. **Limited lab environment isolation**: Even though Tailscale ensures securely authenticated access from the internet, all the lab nodes could communicate freely with all the devices in the home network due to limitations in the available hardware. This can be dangerous when potential malware is tested in the lab.
3. **Limited applicability of test results**: This setup has little in common with k8s clusters used in production worldwide.

4.1.2 Managed Kubernetes cluster in OpenStack Magnum

Discussions with fellow students about problems in the home lab revealed the possibility of getting cloud computing resources from NTNU for master projects. Shortly after, a dedicated OpenStack project hosted in Gjøvik, Norway was created for this master's thesis, with 32 Virtual Central Processing Units (vCPUs), 64 GB RAM and 300 GB volume storage. Figure 4.2 shows an overview of the k8s cluster that was created using OpenStack Magnum:



Node type	OS	vCPUs	RAM [GB]
master nodes	fedora CoreOS 35	2	2
worker nodes	fedora CoreOS 35	4	4
kubectl	Ubuntu Server 22.04	4	4
ntnu-shell	Ubuntu Server 22.04	N/A	N/A

Figure 4.2: High Availability (HA) k8s cluster created with OpenStack Magnum. The two VMs kubectl and ntnu-shell outside the cluster are used for administration of k8s and OpenStack.

OpenStack Magnum allows to automatically deploy a fully functional k8s cluster in OpenStack, simply by defining some parameters like the intended number and type of master and worker nodes, the network driver, and the container runtime in a template. Magnum installs all the required components on the nodes and ensures proper networking and load balancers. To simulate a HA production cluster, three master nodes were deployed, being the minimum requirement to achieve quorum in a High Availability (HA)-etcd-cluster (see subsection 3.3.1). More worker nodes can easily be added at a later point in time if necessary. OpenStack Magnum can be configured from ntnu-shell, a general-purpose administration server available to all students and managed by NTNU. An additional node called kubectl was deployed outside the cluster to run investigation tools not provided by NTNU like nmap.

Advantages of OpenStack Magnum:

1. **Easy to set up**: After some time of getting used to it, new clusters are quickly created.
2. **Realistic testing scenario**: Master and worker nodes are separate. Like in real-world production, an OS optimized for containers is used for the nodes, here it is CoreOS.

OpenStack Magnum is very convenient to use, nevertheless it has some major limitations:

1. **Fixed OpenStack version**: As OpenStack is managed by NTNU, it is not always updated to the most recent release. The currently deployed version zed was originally released in October 2022 and has already reached its End Of Life (EOL).
2. **Fixed k8s version**: Each version of OpenStack comes with a particular version of Magnum, which again is only compatible with a specific k8s-version. This k8s-version usually lags at least a year behind the current k8s release, for zed it is k8s version 2.23 originally released in December 2021 (see [Ope24b]).
3. **Fixed node OS type and version**: Like for k8s itself, Magnum requires a specific version of CoreOS, for zed it is version 35 originally released in September 2021. CoreOS 35 usually updates itself to the most recent version within an hour[1], but Magnum prevents these automatic upgrades. Moreover, manual OS modifications are difficult in Magnum.
4. **Limited access to cluster components**: In Magnum, most cluster components cannot be configured freely, for instance etcd is always installed directly on the master nodes, and cannot be configured as an external cluster. Access to directly modify CNI-plugins like calico is also limited.
5. **Tedious cluster modifications**: All cluster property changes other than updating the number of worker nodes require a new template, which can be applied to the cluster. This applies for instance to updating to a higher OpenStack version or the OS on the nodes.

[1] This has been verified by deploying a VM with CoreOS 35 in OpenStack outside Magnum, it updated quickly.

4.1.3 Microk8s cluster in OpenStack

K8s and its components are evolving rapidly, with new releases and features constantly being added. To allow recent and reliable testing results, it was necessary to look into alternatives to OpenStack Magnum to enable cutting-edge testing on more recent k8s versions. At first, it was considered to manually deploy a more recent k8s cluster in OpenStack without relying on Magnum, simply configuring all components ourselves. However, the documentation on this topic turned out to be scattered and partly outdated, and given the complexity of the task it was decided not to invest more time in it.

Some complete k8s distributions mainly targeted for local testing and development were investigated to overcome these challenges, as they are easy to install and maintain compared to OpenStack Magnum. Another k8s cluster was finally deployed using microk8s from Canonical, the company behind the Ubuntu. It is shown in Figure 4.3:

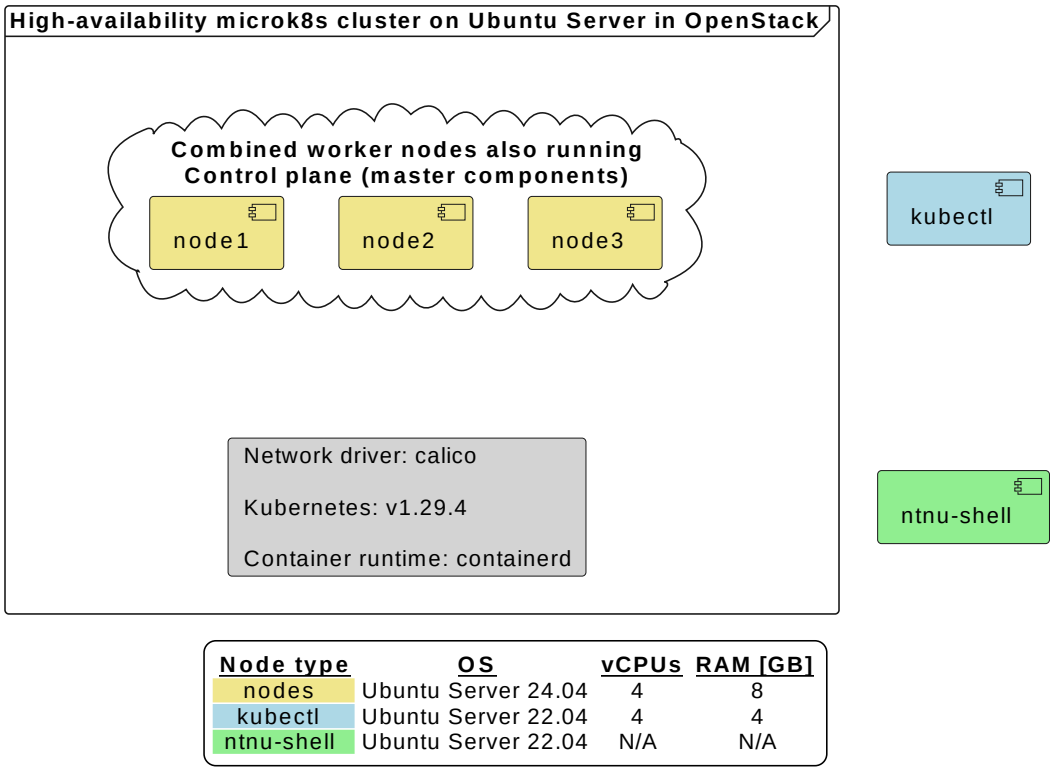


Figure 4.3: HA microk8s cluster on Ubuntu Server nodes in OpenStack. All the nodes in the cluster run master components and workloads simultaneously, no clean worker nodes have been deployed. The two VMs kubectl and ntnu-shell outside the cluster are used for administration of k8s and OpenStack.

Advantages of microk8s:

1. **Easy to deploy and maintain**: Microk8s can be installed with a single command, and it is just as simple to join multiple nodes in a HA cluster. It is also well-documented.
2. **Production-ready**: Due to its HA-abilities, microk8s can be and is being used in production environments according to its developer Alex Jones ([Jon22]). It is also extendable with add-ons and has a lot of capabilities that other smaller k8s distributions do not have.

3. **Always up to date**: According to Jones ([Jon22]), upstream changes to k8s quickly find their way into microk8s. Currently (July 2024), a microk8s release is available for the most recent stable version v1.30 of k8s.

Disadvantages of microk8s:

1. **No isolation between master and worker nodes**: Microk8s allows deploying worker nodes not running master components, but there is no way to set up clear master nodes not running workloads. In case of an attack, this shortens the way for an attacker coming from a worker pod to control master components.
2. **Limited configurability**: Many core k8s components are present but somewhat adapted and their configuration can often not be modified easily. Microk8s uses a dqlite, a modification of the database sqlite, instead of etcd. It is possible but discouraged to set up an external etcd cluster.
3. **Unrealistic testing scenario**: Microk8s nodes cannot run on container-OSs like CoreOS, and there are several other significant differences in the backend of microk8s and real-world on-premises or cloud production k8s clusters.

4.2 Setting up a baseline for testing intrusion detection tools

One of the key challenges in intrusion detection is telling the actual bad events or traffic from benign events and traffic coming from the legitimate users of a system. This usual benign traffic can be used to form a baseline, that is a definition of acceptable traffic and events. Deviations from this baseline can indicate malicious activity. An intrusion detection tool alerts on these deviations from the baseline preferably without alerting too much on benign traffic, these wrong alerts are called False Positives (FPs).

Consequently, it would be hard to properly test intrusion detection tools on an empty k8s cluster without anything running on it, there should be some application running that can be used as a baseline. Developing such an application from scratch would be out of the scope of this thesis. Fortunately, several cloud providers have published reference applications demonstrating how microservices work in their environment. AKS's "eShop on Containers" turned out to no longer be maintained for deployment in on-premises k8s clusters.

GCP's "microservices-demo" (see [Goo24]) however could be deployed in a modern on-premises k8s cluster. It incorporates online-boutique, a webshop where users can browse items, add them to their shopping cart and finally buy them. Figure 4.4 shows an example of how the checkout process can look in the online-boutique. Figure 4.5 gives an overview of the different microservices online-boutique consists of and their interactions. They all use distinct programming languages and technologies in the backend; as mentioned in section 1.1 the ability to do so is one of the benefits of using microservices.

The different backend technologies in the microservices allow different attack vectors to be tested. Finally, the loadgenerator microservice qualifies the online-boutique as a good baseline application, since it continuously generates traffic from which the real attacks need to be filtered.

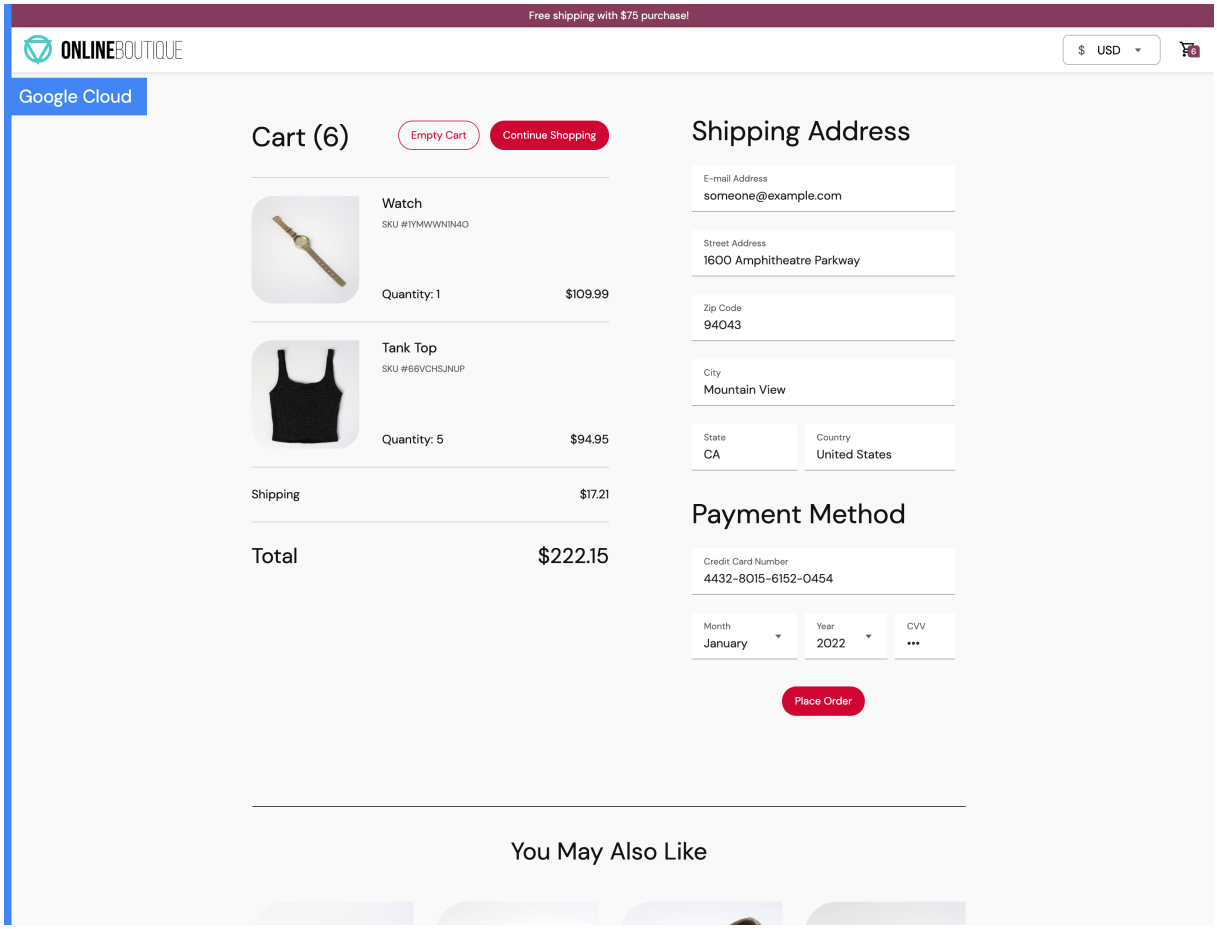


Figure 4.4: Example of the checkout process in the online-boutique. Screenshot from [Goo24].

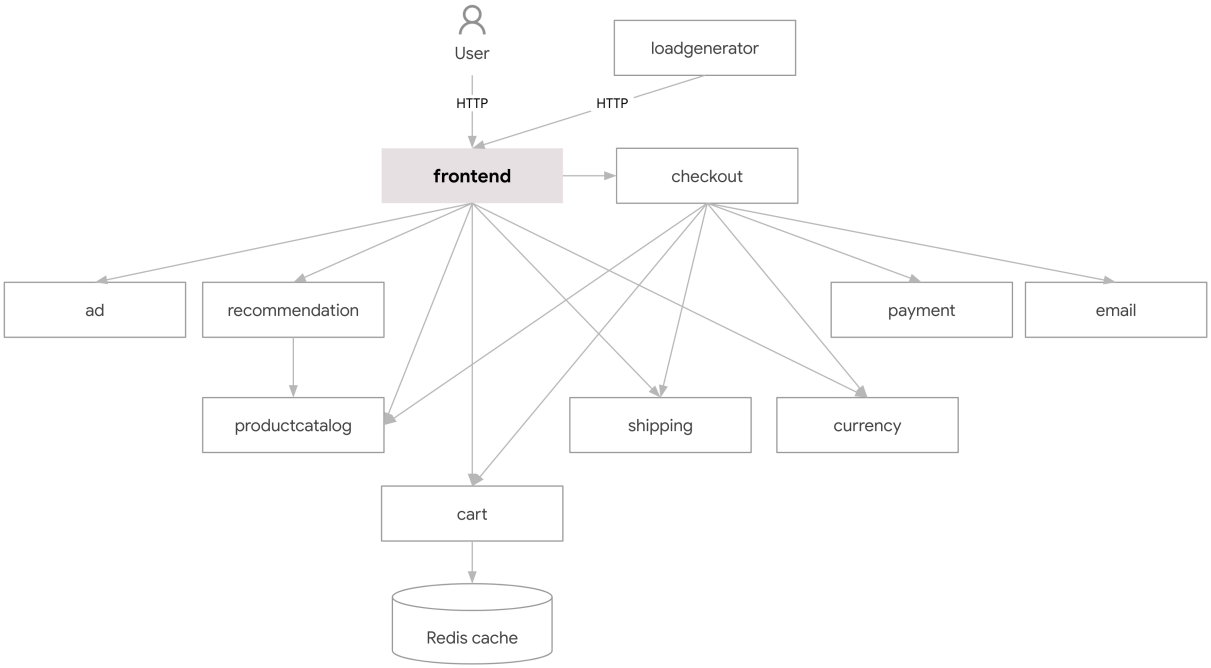


Figure 4.5: Overview of the different microservices that are part of the online-boutique in the microservices-demo from GCP. Figure from [Goo24].

Regardless of GCP being its primary target platform, the microservices-demo can also be installed to on-premises k8s clusters. Several practical tests revealed that the microservices-demo at its current state had become incompatible with k8s v1.23.16 running on the OpenStack Magnum-cluster, leaving the microk8s-cluster the only option to deploy it. An installation to microk8s was not officially documented but worked out in the end. The easiest installation via its Helm-chart was tried first, but unfortunately it resulted in errors as it is still experimental.

The recommended installation uses skaffold, an open-source tool allowing automatic building and deployment of applications in k8s. Not being part of any cluster, the kubectl-vm was used as a remote development machine. Here, Docker, kubectl, skaffold and kustomize[2] needed to be set up with their default installation routines after cloning the git-repository.

Skaffold also requires write access to a container-registry where it can push the container images it has built. Microk8s allows to enable a local, insecure container-registry add-on which runs as a Service in the cluster. In production environments, a properly authorized registry should be deployed, but the simple solution was found sufficient for this lab. The registry runs as a NodePort-service, exposing it at port 32000 on each node in the microk8s-cluster.

Two additional steps were necessary to configure the local insecure registry:

1. **Port forwarding**: The kubectl-vm is no node in the cluster, so the registry port must be forwarded (mirrored) from a cluster-node to the kubectl-vm, this was done using the following command: `ssh -fNT -L 32000:localhost:32000 microk8s-3`
2. **Set up the registry in Docker**: To make Docker aware of the insecure registry, it must be configured in `/etc/docker/daemon.json` with the content:

   ```
   {"insecure-registries":  ["127.0.0.1:32000"]}
   ```

   After creating that file, the docker-daemon must be restarted.

From the project's root directory, the microservices-demo can now be deployed into its own namespace microservices-demo on the microk8s-cluster using the following command:

```
skaffold run -n microservices-demo --default-repo localhost:32000
```

The default-repo argument (arg) ensures the local registry to be used. Otherwise, the setup fails while trying to push images to the default registry Docker Hub, to which it has no write access. Depending on the development machine and Internet connection, the first build takes 20 – 45 minutes; caching significantly accelerates subsequent builds. If everything runs flawlessly, the setup ends up deploying the microservices to the cluster and verifying that they are accessible.

According to its documentation, the demo requires at least 32 GB of disk space, four vCPUs and 4 GB of RAM. The microk8s-cluster is thus more than sufficient, but the full load took down some smaller local VMs during testing. A lot of the resource requirements are generated by the loadgenerator-microservice, which constantly generates work for all the other microservices, too.

[2] For some time now, kustomize has been part of kubectl and can be invoked with `kubectl apply -k`. Even though it is not listed as an official dependency, this setup fails without a separate kustomize binary.

4.3 Conclusion: Which lab environment was finally used?

After having established that the homelab in subsection 4.1.1 will not work out, the advantages and disadvantages of the clusters in OpenStack Magnum and microk8s were discussed in sections 4.1.2 and 4.1.3. Since the beginning, there has been an awareness of the Magnum-cluster running an outdated k8s version. Nonetheless, no action was taken until it turned out that the actively maintained GCP microservices-demo had become incompatible with k8s v1.23 from 2021, leaving it impossible to deploy on the Magnum-cluster.

The Magnum- and microk8s-cluster both have different strengths and weaknesses, but they complete each other well: Microk8s allows to run workloads and tools in an up-to-date k8s-cluster, whereas Magnum provides insights into the workings of the container-OS CoreOS and a real-world etcd.

Both Magnum and microk8s only allow limited access to and configuration of some core components, this is also the case for k8s hosted in cloud environments like GCP, AWS or AKS. Accordingly, these restrictions also apply to most real-world production environments. Only a completely self-configured k8s cluster on bare-metal-servers or OpenStack-VMs without Magnum would provide full access to configure all components. It would be a lot of work to configure such an environment, and based on the analysis from section 3.5, the main focus should be on protecting Pods rather than other cluster components.

This still leaves the question of how well these setups cover testing in cloud environments like GCP, AWS or AKS. At some point, it is all just k8s, configured in one way or another. Usage of cloud platforms incorporates additional costs, whereas microk8s and Magnum hosted in NTNU's OpenStack do not carry additional expenses from the student's perspective.

In the end, a combination of the following lab setups was used, ordered by priority:

1. **Microk8s cluster**: For applications that require k8s to be up to date. Also, tests that require the baseline from section 4.2 can only be carried out here.
2. **OpenStack Magnum cluster**: For tasks that require a realistic lab setup with a real etcd and separate master and worker nodes running a container-OS, but which can tolerate an outdated k8s version.
3. **Cloud providers**: Tests in managed cloud environments like AWS, AKS or GCP are avoided as much as possible and only carried out in limited, well-prepared scenarios to cut costs. For the most part, the two "free" OpenStack clusters at the top of this list suffice.

Chapter 5
Observation capabilities in Kubernetes

Vanilla k8s does not have any dedicated intrusion detection capabilities preinstalled, it just provides the components needed to orchestrate and manage containerized workloads. However, intrusion detection is not only about generating and receiving security alerts. To manually verify the alerts and exclude False Positives (FPs), it is necessary to get an overview of the relevant context, for instance, about the node a potentially infiltrated Pod is running on. The process of investigating that context is often referred to as digital forensics. In this chapter, several tools tools for digital forensics in k8s are evaluated based on a novel set of criteria. Each of these tools addresses at least one of the following reactive k8s security needs: Mapping out the resources and applications in a k8s cluster to get an overview of it, monitoring and logging, networking, remote access to k8s resources, and container imaging enabling analysis of containers in a sandbox. The results of these evaluations are summed up in Table 7.1 in chapter 7, which concludes with recommendations of tools for each of the aforementioned reactive k8s security needs in subsection 7.5.8.

5.1 Kubernetes security tools: Requirements and evaluation

This introduction starts the chapter by pointing out what kind of context information to focus primarily on, followed by sections about how the tools are retrieved and evaluated.

5.1.1 Desirable information for digital forensics in Kubernetes

When an attack on a digital system has been detected, or when there is suspicion of such an attack, digital forensics is conducted to figure out what happened. It seeks to answer questions like these: Which systems were affected by the attack? How did the attackers get in and establish a foothold? Who are the attackers? Which information did they manage to extract? The obtained data can be used to remove the attackers from the system and evaluate how systems can be protected better in the future. Sometimes, digital forensics end up inconclusive: Due to a lack of information far enough back in time, it can become impossible to draw a safe conclusion about who the attackers were or how they got in.

Mnemonic and other cyber security companies possess an Incident Response Team (IRT) which can conduct digital forensics after a cyber attack. The digital forensics industry is, just like Computer Science in general, still a relatively new field of research characterized by rapid change.

As a consequence, some of the most up-to-date knowledge about the field is easier to find in experienced people working with it daily than in literature.

Mats Christensen has worked in different sectors of the cybersecurity industry for the past 15 years and is now part of mnemonic's IRT. Recently, Christensen has played an important role in the team discovering another zero-day vulnerability CVE-2023-35081 in Ivanti's EPMM while working on CVE-2023-35078. Both vulnerabilities were actively exploited by Advanced Threat Actors (APTs) to hack several Norwegian authorities in the summer of 2023 ([Mne23], [Cyb23]). Furthermore, he has researched the digital forensics capabilities in AKS. Based on this, along with experience from digital forensics assignments in other systems, he came up with the following list of desirable information for digital forensics in k8s clusters:

- **Getting an overview of the cluster**: What kinds of services are running where, and which interdependencies do they have? This also includes performance monitoring back in time since performance bottoms or peaks can indicate suspicious activity.
- **Logs**: A central system that retains logs from Pods, containers, control plane (master nodes), apps and proxies. It must allow to retain logs from Pods that have terminated.
- **Networking information**: Collect data about who connected to whom and the amount and type of data being exchanged.
- **Remote Access**: To Pods, worker or master nodes to see what is running there.
- **Container disk and memory snapshots**: A Container disk snapshot allows to see all the files in its filesystem even after termination. A memory snapshot is a copy of its RAM and allows looking for attacks taking place in memory only without modifying files.

This list sets the priorities for selecting the tools to evaluate: For each list item, different tools are assessed to achieve the desired outcome.

5.1.2 Developing evaluation criteria for Kubernetes tools

Comparing different tools for different purposes is not always easy, and hard to do quantitatively and objectively because it quickly feels like comparing chalk and cheese. Nonetheless, a clear set of evaluation criteria can help compare software with a similar purpose.

Research revealed that there does not seem to be a standardised framework for software evaluation. A lot of the identified literature came from companies like Spendflo ([Ram23]) and Testsigma ([Jai24]) trying to sell their consulting services for software evaluation. They all propose different frameworks and strategies and use different names for apparently similar criteria.

In their paper, Jackson et al. propose a large set of 18 criteria ([JCB11]). It seemed overwhelming partly because some of the criteria overlap and can be summarized, some examples:

- **Usability**: Jackson et al. measure separately understandability, documentation and learnability. They can be summarised in one criterion: If the software is well-documented and easy to understand, it is also easy to learn and use.
- **Deployability**: Jackson et al. measure separately buildability and installability. This distinction is not necessary here, measuring the simplicity of installation should be sufficient.

Other criteria like the testability or analysability of the source code were not found to be relevant here, especially because commercial products with inaccessible proprietary source code are also evaluated. Jain ([Jai24]) proposes to measure scalability, however, given the limited test setup, this is hard to evaluate in the context of this thesis. If the documentation of a tool or other sources about it indicates that it scales badly, it will be commented on in the evaluation text instead. Other criteria which are hard to quantify follow the same approach.

Inspired by Jackson et al. ([JCB11]) and Ramaoorthy ([Ram23]), the following novel proposal of evaluation criteria presented in subsection 5.1.3 below has been composed using my own practical experience in the field. Inspired by a five-step Likert-scale as introduced by Joshi et al. ([Jos+15]), the rating scale in Table 5.1 is taken from Stiftung Warentest ([Sti24]), who use it to evaluate features in their tested products. In contrast to a rating of one to five stars, it distinguishes more clearly between "good", "bad" and "ok" and is more compact.

Table 5.1: Five-step Likert rating scale used by Stiftung Warentest to evaluate product features ([Sti24]).

++	+	○	−	−−
excellent	good	ok	poor	terrible

5.1.3 Novel proposal of evaluation criteria for Kubernetes tools

The tools evaluated in this report are rated according to Table 5.1 in the eight criteria below. If + and ++ are described together as +/++, it is implied that ++ is a stronger expression of the description than +. +/++ is also used if no meaningful distinction between + and ++ could be made. The same applies to combined −/ − − which stands for − and −−.

Adaptability: How easily can the tool be modified or extended?

+/++ Modifiable with settings or parameters and extendable with custom code or add-ons

○ Modifiable with some settings or parameters, but no custom code or add-ons

−/ − − By no means customizable, no settings. Generates static output based on input

Availability: What does it cost to use this tool?

+/++ Open source. ++: Freely available; +: Freemium, free features mostly sufficient

○ Commercial freemium: Partly free, but some important features require purchase

−/ − − Commercial software. −: Price low / unknown; −−: Expensive

Portability: How likely can this tool be used everywhere?

+/++ Works in any k8s deployment / cluster

○ Runs almost everywhere, some exceptions

− Many exceptions

−− Only for one specific k8s distribution or type of environment

Interoperability: How well does this tool work and integrate with other tools?

+/++ Easy to integrate. If it is a graphical tool, it has well-documented and complete APIs with well-structured input in common machine-readable formats. A command-line tool also allows export in machine-readable formats. Bonus if plug-ins or add-ons are available to integrate with other popular software.

○ Chaotic, outdated, incomplete and badly documented APIs or terminal inputs and outputs. No, or few outdated plug-ins or add-ons to integrate other software are available.

−/−− No APIs are available, to extract data it must be scraped from the website or from terminal output that is not machine-readable with manually maintained scripts.

Deployability: How easily is this tool installed?

++ Most likely it is already installed / available.

\+ Easy to install without modifying the cluster, e.g. locally on the analyst's machine.

○ Tool can be deployed easily in k8s, e.g. using Helm. Setup takes less than two hours.

− Requires some individual configuration, setup can take hours or days.

−− Deployment can take several people weeks or months and requires removal of other tools or a reconfiguration of the whole cluster. Setup can require specific training or skills.

Footprint: What additional operational costs come with running this tool?

++ Small resource consumption solely on the security analyst's machine.

\+ Big resource consumption solely on the analyst's machine.

○ Some additional Pods / resources in the cluster. Insignificant performance implications.

− Noticeable additional resource consumption in the cluster. For instance, one additional Pod on each node / in each namespace (virtual separation of resources in k8s cluster).

−− Significant additional Central Processing Unit (CPU) / RAM / disk space usage. For example, adding additional "sidecar" containers to every Pod in the cluster.

Popularity: Are there few or many others using this tool?

++ Big commercial or open-source product maintained by a foundation. Support or community help easily available if there are problems. Very low risk of product discontinuation.

\+ Smaller commercial product or open-source project with a big user base. Support or community help easily available if there are problems. Low risk of product discontinuation.

○ Smaller actively maintained open-source project with many users and several maintainers. Moderate risk of discontinuation.

− Smaller little, but regularly maintained open-source-project with a couple of users and one or very few maintainers. Even with a greater user base, these projects are subject to the risk of being discontinued at some point.

−− Open source projects being or appearing to be discontinued as they lack recent updates.

Usability: How difficult is learning to operate or use this tool?

+/++ The tool can be intuitively understood with the knowledge security analysts already have from working with other tools. Some complexity can be tolerated for tools combining many features, given consistency and complete and up-to-date documentation.

○ The tool is relatively easy to use, but requires some background from its documentation. If there is e.g. a new scripting language, it is relatively easy to learn and understand. Can also get this rating if the documentation is somewhat outdated or incomplete.

−/−− The tool requires specific training in a long time before it can be used. For instance, if a completely new and complex programming or scripting language must be learned or the program's output is hard to understand even for security professionals. Can also get this rating if the documentation for a complex tool is very incomplete or outdated.

5.1.4 Finding and evaluating Kubernetes tools

Given its significant market share and adoption, k8s has a lot of users and administrators and thus also a great open-source-community. It creates and maintains many smaller and bigger applications easing the use of k8s and extending its functionalities. Apurva Bhandari and Ajeet Singh Raina have created KubeTools ([Col24]), a website giving an overview of the available open-source and commercial tools for k8s where many of the tools evaluated below were found.

Many different types of applications and add-ons are available. Command-line (cmd)-tools are the most basic, but oftentimes also the most powerful tools for administration of infrastructure such as k8s. They are entirely text-based and have no Graphical User Interface (GUI), the only interface is a terminal in which users can issue text-based commands, whose reply or output again is text-based. Many of the available open-source tools only have a cmd interface, as it is easier to program and maintain than a full-fledged GUI application.

Graphical tools like a dashboard can in many cases provide an overview quickly and more pleasantly than cmd-tools. Their development is more laborious than cmd-tools, ergo fewer tools are available in this domain that is dominated by commercial actors or open-source projects with a bigger community. While offering some functionality that is not possible to realise in cmd-tools, graphical tools often only reimplement a subset of the functionalities of a cmd-tool with a similar purpose.

When tools are rated based on Table 5.1, the ratings should not be interpreted numerically in a way where a −−-rating for one criterion is equalised by a ++-rating for another. That would be like comparing apples and oranges, as the criteria measure entirely different aspects. Another factor that should be accounted for is that the importance of each criterion varies from case to case, which is why they also do not appear in a particular order in subsection 5.1.3. Even though a −−-rating often disqualifies a tool, it can be tolerable if it provides important functionality that cannot be achieved in another way, as the scenarios below illustrate:

- A free tool revealing information no other tool provides, but with an adaptability of −− due to only static output and no settings, can still be very useful[1].
- Given a unique functionality with a real business value, a high price (availability −−), operational cost (footprint −−) and training cost (usability −−) can be worth it.
- High popularity can in some cases equalize low usability: If there is a great community and the knowledge also can be applied elsewhere, it can be worth the training cost. Also, someone in the company probably already has the required skill set.

Note that there is no particular criterion measuring the utility of a tool because it is hard to measure quantitatively. However, utility is applied indirectly: Tools that are not found useful for any of the use cases in subsection 5.1.1 are not included in the report. Similarly, tools that appear unmaintained are usually not included, as adding them to the workflow is a "ticking time bomb" because unmaintained tools become incompatible with other up-to-date software at some point. On this note, a big company or foundation behind a project is no guarantee that it remains maintained. VMware's Octant ([Oct23]) is an example related to this thesis, Google's cemetery of projects it started and later killed is another prominent example of this ([Ogd24]).

The individual ratings following Table 5.1 are set based on installing and configuring each tool as a Proof of Concept (PoC) of using it in a production environment. This process and its results are evaluated, along with the documentation of the tools. These ratings are mostly meant as a coarse indication: Due to their subjective nature, other evaluators may disagree from time to time. A table like Table 5.2 is filled for the most important tools or groups of tools. Double ratings like +/○ mean that it depends from case to case if it is + or ○, these and other ratings not regarded as obvious are justified in advance. In cases where no in-depth evaluations were feasible for all possible use cases, the assessments were based on the principle of the presumption of innocence: If no practical proof or documentation states anything different, assume that it works and evaluate positively.

5.2 Tools for getting an overview of the cluster

This category contains dashboards or other tools to map out the cluster to see what kinds of services are running where, and with which interdependencies. This also includes performance monitoring back in time since performance bottoms or peaks can indicate suspicious activity.

5.2.1 Kubectl: Universal Kubernetes administration

Kubectl is the official Command Line Interface (CLI) for Kubernetes maintained by the CNCF ([Kub24a]), and often one of the first tools k8s-administrators need to learn. It is most likely already installed on the security analyst's machine, where it has a small footprint. According to k8s' documentation ([Kub24e]), it provides a "Swiss Army knife of functionality for working with Kubernetes clusters". Some of its capabilities are listed below:

[1] For some cmd-tools, a simple `tool-export | grep interesting_stuff` can already be sufficient to extract the data. This is still relatively easy to maintain, even though the output originally was not machine-readable.

- **Map out resources**: Commands like `get` or `describe` can be used to map out resources, for example, to get an overview of the services or Pods which are currently running. `Top` returns the resource consumption of nodes or Pods.
- **Deploy resources**: Commands like `create`, `edit` and `delete` can be used to modify resources in a cluster. `Patch` or `replace` can be used for batch modication e.g. in scripts.
- **Attach and run**: `Attach` allows to attach to the standard I/O streams stdin, stdout and stderr of an already running Pod or container. Like in Docker, `run` runs an image in a Pod using a single-line command and attaches to it.
- **Remote execution**: `Exec` allows to spawn a new process in an already running container. If the command is e.g. `/bin/bash`, `exec` starts a remote shell session on the container.
- **Port forwarding and Proxy**: `Port-forward` allows to forward traffic from a local port to a Service or specific Pod, or vice versa. With `proxy`, kubectl can act as a proxy, allowing to send requests to a remote kube-apiserver via a local port instead to e.g. bypass a firewall.
- **Retrieving files and logs**: With `cp`, files can be copied to or from Pods. `Logs` allows printing of the logs for a running container or Pod.

This shows that the "Swiss Army knife" is not only useful for mapping out a k8s cluster, but also for many other tasks.

Table 5.2: Rating for kubectl following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: ++	Footprint: ++
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.2.2 Kubernetes dashboard: A simple overview of the cluster

Kubernetes dashboard gives an overview of the resources deployed in k8s cluster along with some metrics. It is an official addon maintained by the CNCF ([Kub24c]), but no longer part of vanilla k8s. Still, it is well integrated and easy to install in the cluster via a Helm-chart. Figure A.1 in the Appendix shows the Kubernetes-dashboard running on the microk8s-cluster set up in subsection 4.1.3, with the twelve Services and Pods from the microservices-demo running.

The Kubernetes dashboard is a good starting point for an overview of what is running in a k8s-cluster. It shows information about all k8s resource types like nodes, Services, Pods, Secrets, NetworkPolicies or Roles. Containers do not appear directly in the dashboard, as they are managed through Pods and do not get their own k8s configuration. Resources can be deleted, or created and modified in JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML) files. For any action that modifies the cluster, the corresponding kubectl command is shown for verification before the change is executed.

Resources are aggregated and organised by namespace, Role-Based Access Control (RBAC) (see section 5.3) can be used to configure which namespaces and resources a user of the dashboard can see. Some resources like nodes can only be edited when the kube-system namespace is selected, as their YAML configuration belongs to that namespace. In other namespaces, nodes also appear showing which Pods from the namespace are running where, but the nodes' configuration cannot be edited. Some additional functionality is available for Pods: The logs can be viewed similarly

to kubectl's `logs`. Also, similar to kubectl's `exec`, it is possible to start a live terminal into a Pod directly from the dashboard to see what is happening inside it.

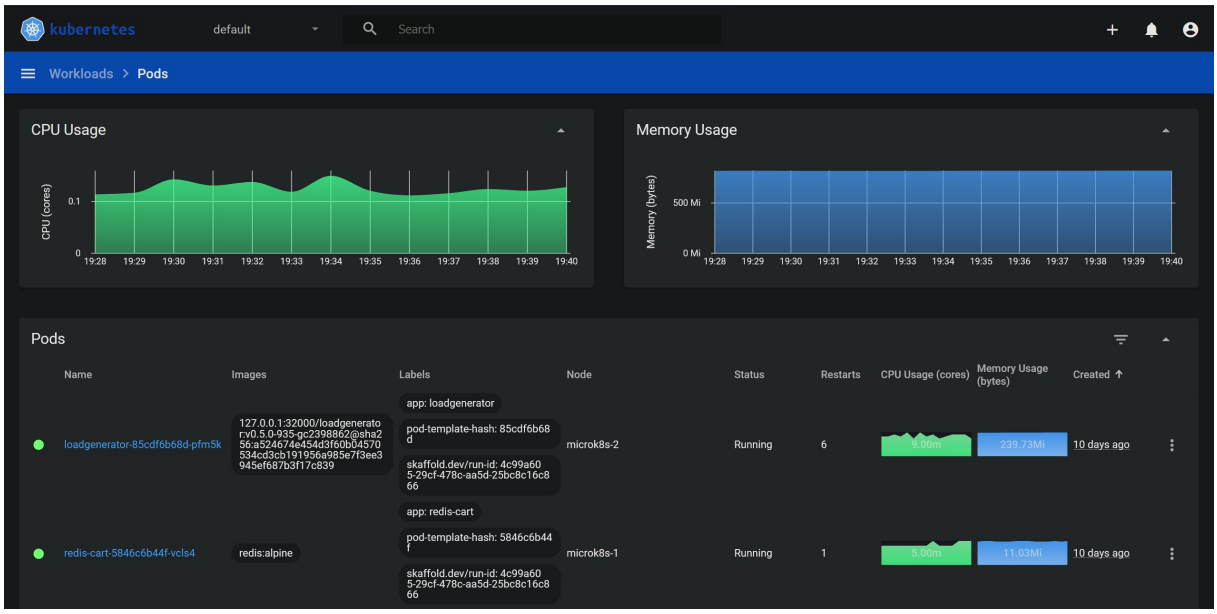


Figure 5.1: Pod metrics in Kubernetes dashboard, total for all and individually for each Pod

For some resources like nodes or Pods, the CPU and RAM consumption can be inspected, as illustrated in Figure 5.1. Kubernetes-dashboard's integrated metrics-scraper delivers these measurements. By default, it does one measurement per minute with a retention period of 15 minutes, configurable as cmd-args to the metrics-scraper Pod. If Helm is used for installation, the following can be added to the `upgrade` command, adding cmd-args to the Pod:

```
--set metricsScraper.containers.args={"--metric-duration=3h",
↪   "--metric-resolution=30s"}
```

It sets the retention (`metric-duration`) to three hours, with two measurements per minute (`metric-resolution`). However, experiments in the OpenStack Magnum cluster have revealed that the performance metrics displayed in the dashboard have some major limitations:

- Only four to fifteen data points are displayed, impossible to zoom in or out to see more.
- When a retention of three hours is configured with one measurement per minute, still only fifteen data points are shown with a distance of eleven to twelve minutes each.
- With a retention of three minutes and six measurements per minute, nothing is ever shown. It never shows more than one data point per minute, and never less than four totally.

Metrics-scraper or other dashboard components can be deactivated. This reduces the footprint, as the corresponding Pods are not deployed. Apart from metrics-scraper, kubernetes-dashboard provides no APIs. All of its other functionalities are also available in kubectl, which can be seen as kubernetes-dashboard's API. This still leads to an excellent interoperability. Only very few settings are available for the dashboard, it mostly comes as it is. For instance, it cannot be configured to show additional metrics and the order of widgets cannot be adapted.

Table 5.3: Rating of Kubernetes dashboard following Table 5.1, legend in subsection 5.1.3.

Adaptability:	○	Availability:	++	Deployability:	○	Footprint:	○
Interoperability:	+	Popularity:	++	Portability:	++	Usability:	++

5.2.3 Grafana dashboards with Prometheus operator: Display metrics

Grafana is an open-source analytics monitoring solution ([Gra24a]), developed and maintained by the company Grafana Labs and free even for commercial use. There are purchase options for premium features like expert support and enterprise plugins, but mostly the free features are sufficient. Grafana has become very popular in recent years. Accordingly, it is likely to already be deployed and used somewhere in bigger companies.

Figure 5.2: Official Grafana Labs Kubernetes dashboard deployed on the microk8s-cluster

There are many plugins and pre-defined dashboards like the official Grafana Labs Kubernetes dashboard shown in Figure 5.2 readily available for installation. They can also be modified or serve as a template for custom dashboards; in general, Grafana is very adaptable and customizable. Grafana depends on external data sources to fetch metrics data, it does not do measurements or directly connect to sensors itself. Data sources are integrated with plug-ins, which can be implemented for any data source. Many officially maintained plug-ins already exist for the most common data sources like Graphite, InfluxDB, Elasticsearch, different SQL-databases or the cloud platforms AKS, AWS and GCP.

Another important data source for Grafana is Prometheus, an open-source monitoring system which stores metrics it actively pulls from different sensors in time series. Originally developed at SoundCloud, Prometheus is still fully open source and joined the CNCF in 2016. It is widely used and has an active developer and user community ([Pro24a]).

If they have not been deployed elsewhere yet, a complete stack with Prometheus monitoring and Grafana for the cluster can be installed using a simple Helm-chart. It is called kube-prometheus-stack ([Pro24b]) and deploys among other things Prometheus Operator, a Prometheus-based monitoring stack for k8s. The stack also incorporates a set of pre-defined Grafana dashboards for k8s from the Prometheus Monitoring Mixin for Kubernetes, leveraging almost 30 dashboards with details about networking, nodes or different k8s components like the kubelet, kube-proxy, kube-scheduler, DNS or etcd. In terms of scalability, adding more metrics or increasing the measurement frequency also increases the footprint in terms of RAM, disk storage and CPU.

Some dashboards included in the kube-prometheus-stack like the ones for etcd and networking did not receive data in the vanilla setup, neither in OpenStack Magnum nor in the microk8s-cluster. Similarly, the Grafana Labs Kubernetes dashboard imported into the kube-prometheus-stack's Grafana showed no data about network utilisation. This demonstrates that even though initial deployment is relatively easy, not everything works out of the box: Some configurations depend on the k8s cluster deployment, raising the need for manual fine-tuning. All the Grafana dashboards are easy and intuitive to understand[2] once set up, but getting there can be cumbersome. At least good documentation and a great community are available to help.

Table 5.4: Rating of Grafana dashboards following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +	Deployability: ○	Footprint: ○
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.2.4 Integrated Development Environments (IDEs) for Kubernetes

Integrated Development Environments (IDEs) are used by many programmers to support their software development with features like version control and an integrated build and debugging pipeline. Numerous IDEs are available for different technology stacks and programming languages. There are open-source alternatives, but many have a commercial freemium licensing model, restricting some features to paying customers. Despite not necessarily being the first tool security analysts would think of to get an overview of a k8s cluster, IDEs have some features that are not found in cmd-apps or online dashboards. This section analyses the k8s integration for some of the most commonly used IDEs for developing applications that run on k8s. They are evaluated as a group to generally see how IDEs perform to get an overview of a k8s cluster.

Lens: The Kubernetes IDE

Lens is the biggest IDE dedicated to k8s development and can be installed in Windows, Mac and Linux. Started as an open-source project by the Finnish company Kontena, it became closed-source after being acquired by Mirantis, the company behind Docker. After the retirement of the open-source twin project OpenLens in 2023, community additions to the source code are only possible through plug-ins. The licensing model provides a free license with community support and basic features; premium features like the ability to use extensions require subscription on a per-user basis ([Mir24]).

[2] At least from the technical perspective of accessing and viewing them. There is full freedom to create chaotic, hardly understandable dashboards, but that does not fall into the responsibility of Grafana.

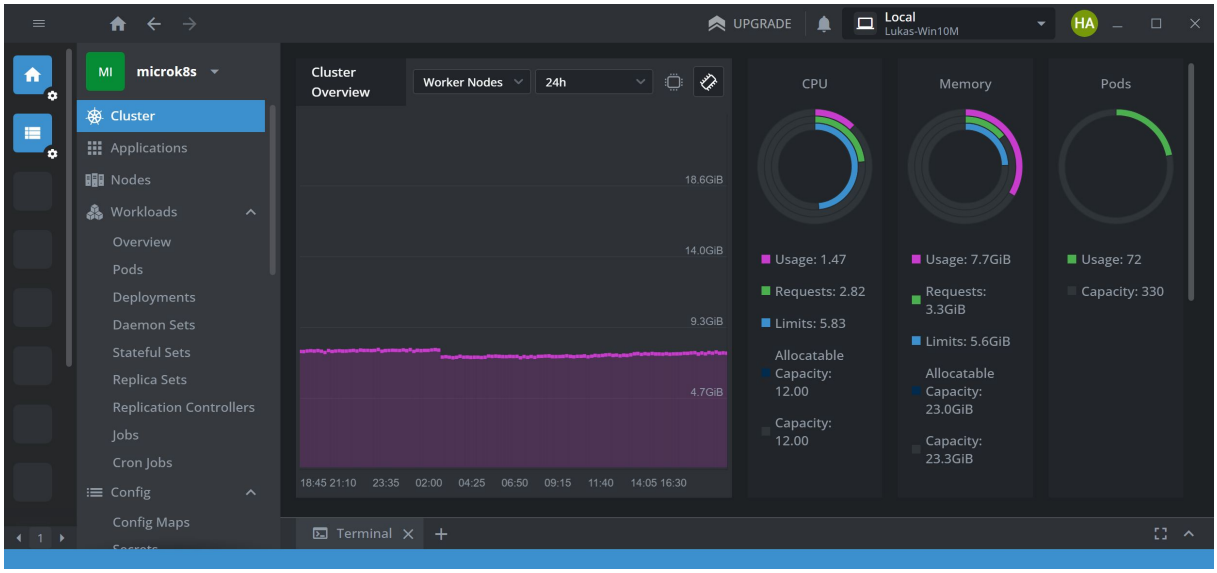


Figure 5.3: Lens dashboard on the microk8s cluster, with data from the kube-prometheus-stack

It is possible to display details for all the different resource types in k8s, even for ones that programmers rarely use like leases and autoscalers. There is also a Helm integration providing many details. However, tests revealed that it was not possible to upgrade a Helm-chart to its newest version through Lens since the buttons which are supposed to do it did not work.

As Figure 5.3 illustrates, Lens supports displaying simple metrics like the CPU, RAM, disk and network usage for different resources like nodes and Pods, too. A metrics-scraper must be deployed into the k8s-cluster which fetches these metrics. Lens can install its own metrics-scraper, or use the one deployed by the kube-prometheus-stack (see subsection 5.2.3). If the Prometheus operator is deployed already, Lens can autodetect it in many cases.

For this thesis, only the free version of Lens has been evaluated. It provides a wide range of features, and from the perspective of a security-professional who mostly wants to get an overview of the deployed resources, it does not lack substantial functionality. → availability: +/○

Visual Studio (VS) Code with Kubernetes plug-in

Visual Studio (VS) Code is Microsoft's universal code editor, which is "free[,] built on open source [and] runs everywhere", with "extensions to support just about any programming language" ([Mic24]). There are subscription-based purchase options for Visual Studio providing extended features but for the sake of getting an overview of Kubernetes Resources, VS Code being free even for commercial use appears sufficient. There are free plug-ins available for VS Code to ease development for Docker and k8s. → availability: +

The k8s plug-in allows listing the most common resources like Pods and nodes sorted by namespace, but it cannot display metrics for them. When a resource like a Pod is selected, its YAML-file is opened for editing with syntax highlighting; changes to these files can be pushed to the cluster. Configuration details also cannot be listed in tables and need to be fetched manually from the YAML-configuration files, making it harder to get an overview. A Helm integration is

available, however, the Helm binary needed to be updated manually to prevent a compatibility error, and already deployed Helm-charts are not displayed automatically.

JetBrains IDEs with Kubernetes plug-in

JetBrains is a Czech software development company releasing IDEs for different programming languages like PyCharm for Python or IDEA for Java. All IDEs are available in community versions with limited functionality, free even for commercial use, and purchased subscription-based Professional versions. Many important features like remote debugging, database integrations and the k8s plug-in are unavailable in the community additions. → availability: −

The k8s integrations in JetBrains IDEs have been tested in PyCharm Professional, which is accessible free of charge through a student license. The first tests were run on a local Ubuntu VM, where the available functionality was found to be comparable with VS Code, yet no Helm-integration could be found. Mainly, the plug-in allows applying local YAML-files with k8s configuration to the cluster.

A month later when the k8s plug-in was finally evaluated, it was no longer possible to configure it to connect to any of the deployed clusters. On Mac, the microk8s-cluster configuration could not even be added, and never appeared on the list of clusters. The OpenStack Magnum cluster appeared, but red exclamation marks indicated an error on the icons for resources like Pods. As a result of this not closer specified error, the resources could not be enumerated. Another test was run with a local microk8s test-cluster deployed on the Ubuntu test-VM together with PyCharm, but it resulted in the same exclamation marks and errors. All the configuration files used for the IDE were verified to work in Lens, VS Code and kubectl[3], and could thus be excluded as a source of error. No quick fix for these problems was found on the Internet, resulting in a deployability rating of −− for the k8s plug-in in JetBrains.

Wrapping up IDEs for Kubernetes

Most k8s-integrations are based on a kubeconfig YAML-file, which is also used to authenticate kubectl. Consequently, the integrations mostly work if kubectl can be used, resulting in a portability close to kubectl. The footprint is only on the analyst's machine, however there it can take gigabytes of disk storage and have significant CPU and RAM requirements. All the aforementioned IDEs are popular, well-documented and relatively easy to install and use. They also all have a plugin-system making them interoperable with other software. The availability varies and has been commented on individually above.

Table 5.5: Rating of IDEs for Kubernetes development following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +/○/−	Deployability: +/ − −	Footprint: +
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.2.5 K9s: Simplified command-line Kubernetes administration

Kubectl can be used for many k8s administration tasks from the terminal. Still, some of its commands can be verbose: Often, several `kubectl get` commands are required to fetch the

[3] Kubectl on the same machine was verified to be able to connect to the clusters.

necessary information to perform a change. The open-source CLI-based tool k9s addresses this issue and makes it easier to "navigate, observe and manage deployed [k8s] applications in the wild" ([Gal24a]). It is very popular, with over 250 contributors in its git-repository and several k8s-professionals like Anton Putra ([Put21]) recommending it as part of their k8s toolbelt.

Fernand Galiana is the main maintainer of k9s, promising that the project is open source and "will remain free for all" ([Gal24b]). Still, Galiana welcomes donations to keep the project up and running. Additionally, there is a paid version of k9s called k9sα containing the "next generation of features and enhancements" ([Gal24b]). All the tests and evaluations below are based on the free version, which appears to be sufficient for the use cases required in this context.

The k9s-dashboard with the Pods from the microservices-demo is shown in Figure 5.4, together with different status information about them. In the header, the current context, cluster, and user are displayed, along with the current CPU and RAM consumption in the cluster. Furthermore, recently used namespaces are displayed in purple, which can be selected by pressing zero, one, two or three. By default, the user interface is navigated by the keyboard only. Keybindings similar to the editors vim and emacs can be configured. Mouse navigation can be enabled, but tests revealed that it removes the ability to copy text from the k9s terminal.

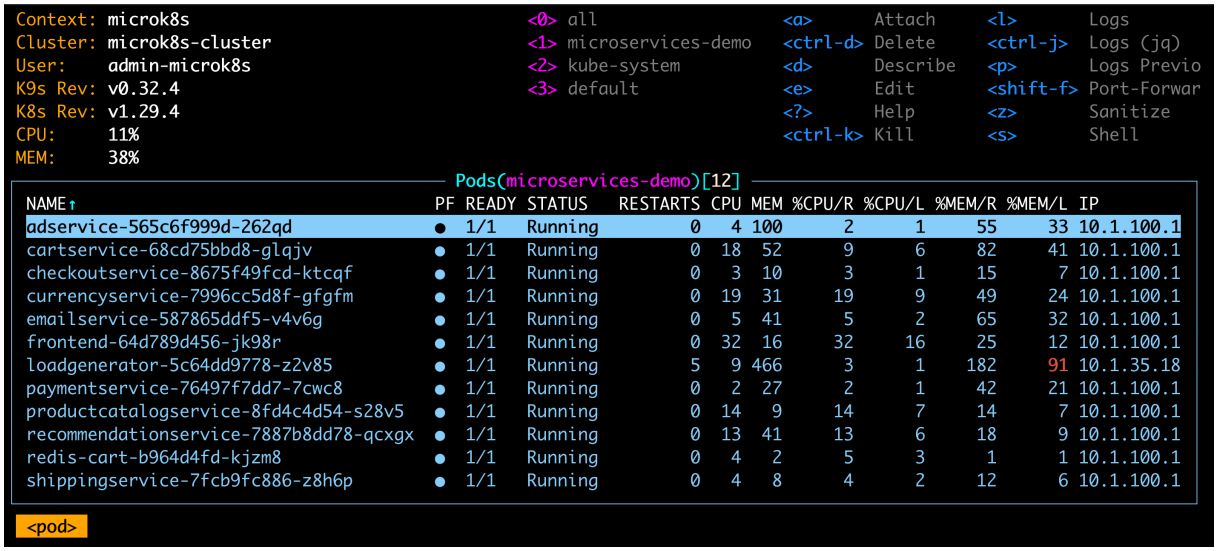


Figure 5.4: K9s dashboard displaying the Pods from the microservices-demo in their namespace

Tests of k9s in the different clusters have shown that navigation between different resources, following traces and displaying information is fast and intuitive[4]. Help is always quickly available using "?", ":help" and "Ctrl + a". Like in vim, pressing ":" allows to enter extended commands, for instance, to see other resource types like Services, Deployments and even Custom Resource Definitions (CRDs). Search results can be filtered with "/". Many other functionalities requiring a lot of typing in kubectl can be used quickly by pressing simple buttons or shortcuts, like "Shift + f" for port forwarding, "y" to see and "e" to edit the YAML for a resource, "d" to describe it, or "a" for attaching to a container or Pod.

[4] At least for security analysts familiar with cmd applications and some knowledge about k8s.

K9s provides a very good overview of the permissions of Roles and ServiceAccounts. Another extended functionality is xray revealing the interdependencies of k8s resources. Figure 5.5 shows xray for Services in the microk8s-cluster. Some functionality in k9s is not documented and YouTube tutorials do not cover every detail, so no documentation could be found for the symbols used in xray. Checking out the different resources' YAML with "y" showed that cardboard boxes are namespaces, lorries are Pods, credit cards are ServiceAccounts and whales are containers. Also here, results can be filtered with "/" and all the displayed resources can be extended or collapsed with space to narrow down on the relevant information.

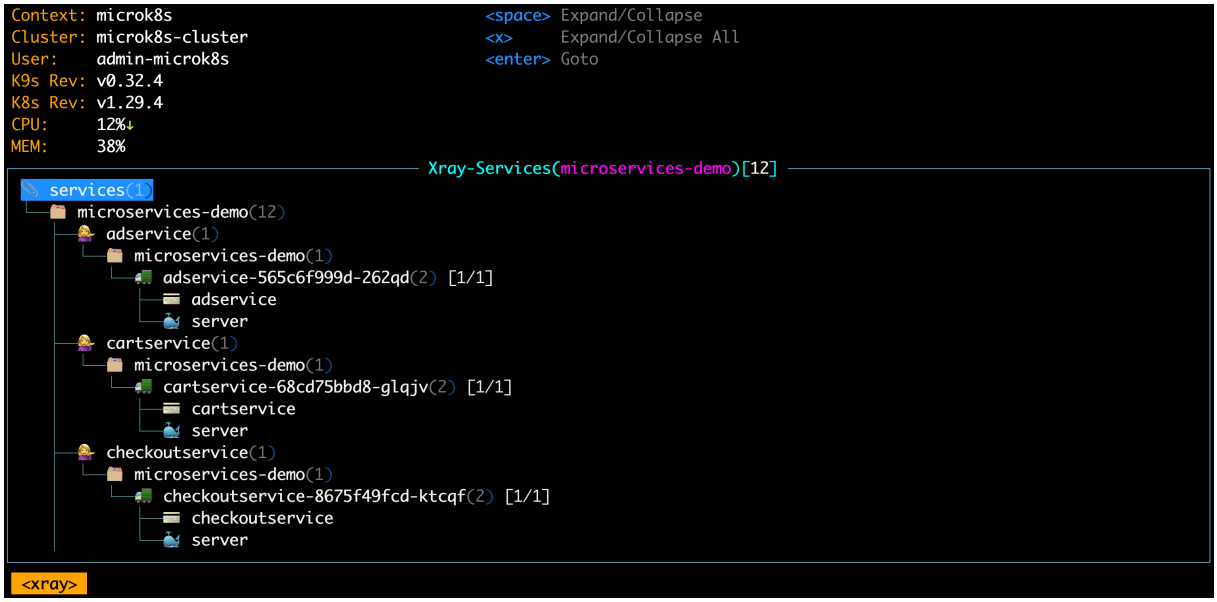


Figure 5.5: K9s xray for services in the microservices-demo on the microk8s-cluster

K9s's functionality can be easily extended by defining plug-ins, which can run and display the output from any cmd command available on the machine where k9s is installed. A list of community plug-ins can be found inside k9s's git-repository. For example, there are plug-ins available to view the events from a Pod, add a debug-container, or get Helm-values. The built-in "l" command is somewhat limited, so there also is a plug-in called "log-full" displaying all the logs since it was created from a Pod in the full-screen file-view application less.

Some of the logs from the microservices-demo are in JSON format, which can be tedious to read without additional formatting. The integrated community plug-in logs-jq intended to format JSON logs before displaying resulted in an error. However, it only took a few minutes to make the same functionality work by modifying and extending the log-full plug-in with jq for formatting, invoked by an additional shortcut "Ctrl + j". Figure 5.6 demonstrates the functionality of the novel plug-in, the source code can be found in Listing A.1 in the Appendix.

Question mark "?" can be pressed to see all available commands, including the ones added by plug-ins. Some of the most common shortcuts are also displayed in the header. In Figure 5.4, "Ctrl + j" can be seen, which activates the novel logs-jq plug-in. Some more resource consumption metrics can be seen using the `:pulses`-command, but in general, k9s' metrics capabilities are very limited. Some metrics-server e.g. from the kube-prometheus-stack (see subsection 5.2.3) must be deployed in the cluster for metrics to work. For all the other functionalities, k9s only

```
{"instant":{"epochSecond":1718100275,"nanoOfSecond":159286115},"thread":"grpc-default-executor-395","level":"INFO","l
oggerName":"hipstershop.AdService","message":"received ad request (context_words=[accessories])","endOfBatch":false,"
loggerFqcn":"org.apache.logging.log4j.spi.AbstractLogger","threadId":414,"threadPriority":5,"logging.googleapis.com/t
race":"${ctx:traceId}","logging.googleapis.com/spanId":"${ctx:spanId}","logging.googleapis.com/traceSampled":"${ctx:t
raceSampled}","time":"2024-06-11T10:04:35.159Z"}
```



```
{
  "instant": {
    "epochSecond": 1718100275,
    "nanoOfSecond": 159286115
  },
  "thread": "grpc-default-executor-395",
  "level": "INFO",
  "loggerName": "hipstershop.AdService",
  "message": "received ad request (context_words=[accessories])",
  "endOfBatch": false,
  "loggerFqcn": "org.apache.logging.log4j.spi.AbstractLogger",
  "threadId": 414,
  "threadPriority": 5,
  "logging.googleapis.com/trace": "${ctx:traceId}",
  "logging.googleapis.com/spanId": "${ctx:spanId}",
  "logging.googleapis.com/traceSampled": "${ctx:traceSampled}",
  "time": "2024-06-11T10:04:35.159Z"
}
```


Figure 5.6: Novel k9s plug-in reformatting JSON logs before displaying them using less, demonstrated by the means of the first log-line from adservice-565c6f999d-262qd selected in Figure 5.4. The unformatted log line is shown above; the same jq formatted and coloured line from the new plug-in is shown below. Listing A.1 in the Appendix shows the source code and how the plug-in can be installed.

needs to be installed on the security analyst's machine or a dedicated analysis VM. K9s has a small resource footprint, all common OSs are supported and the installation is easy since k9s uses the same configuration files and contexts as kubectl. Excellent interoperability is given through the plug-in mechanism, apart from that kubectl can be used for automation in scripts.

Table 5.6: Rating of k9s following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +	Deployability: ++	Footprint: ++
Interoperability: ++	Popularity: +	Portability: ++	Usability: +

5.2.6 KubeView: Kubernetes cluster visualiser and graphical explorer

Coleman released KubeView in 2019, a free open-source "k8s cluster visualiser and graphical explorer" ([Col22]). Similar to k9s's xray, it shows how the most common types of k8s resources are related in a graph view. Even though it only has five other contributors, the GitHub repository has currently been forked 104 times and it has 912 stars on GitHub, indicating some community activity. Even though the latest release is already three years old, the deployment via its Helm went through smoothly. Only one backend service and Pod are deployed into the cluster, directly serving the web interface and internally using APIs.

The graph with resources refreshes automatically with a configurable refresh interval. For stateful components like Pods, the colour indicates the state. For example, a ReplicaSet is red if its intended number of Pods is three, but only two Pods are running. Coleman demonstrates in this Youtube video[5] how the resources and their states are automatically updated in real-time for an application that is scaled up and down during varied load. Clicking on a resource shows more details about it and the full YAML file can also be displayed, see Figure 5.7 for an example.

[5] Coleman's Youtube video demonstrating live updates: https://www.youtube.com/watch?v=sRpCDx1rK00

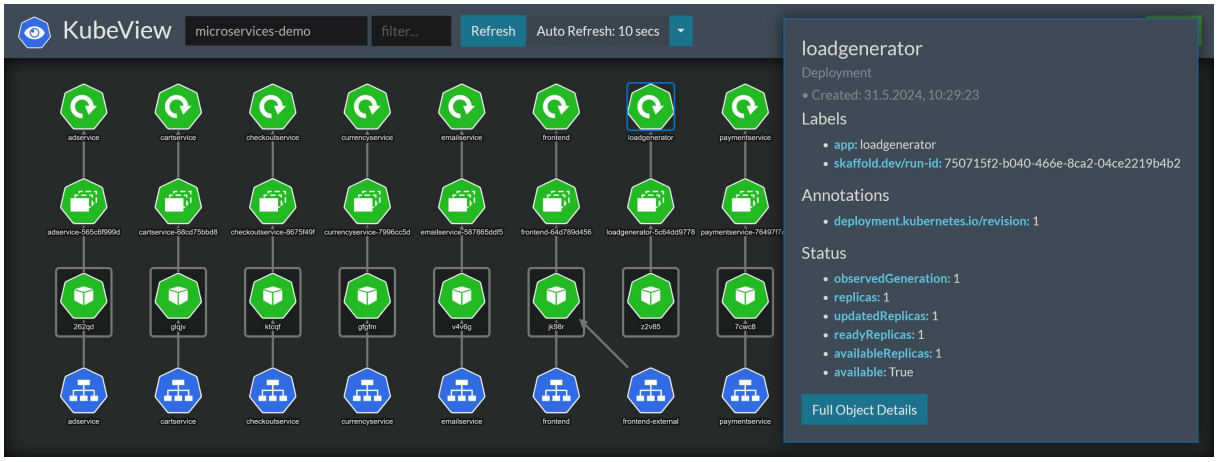


Figure 5.7: KubeView on the microservices-demo, showing details for the loadbalancer-service

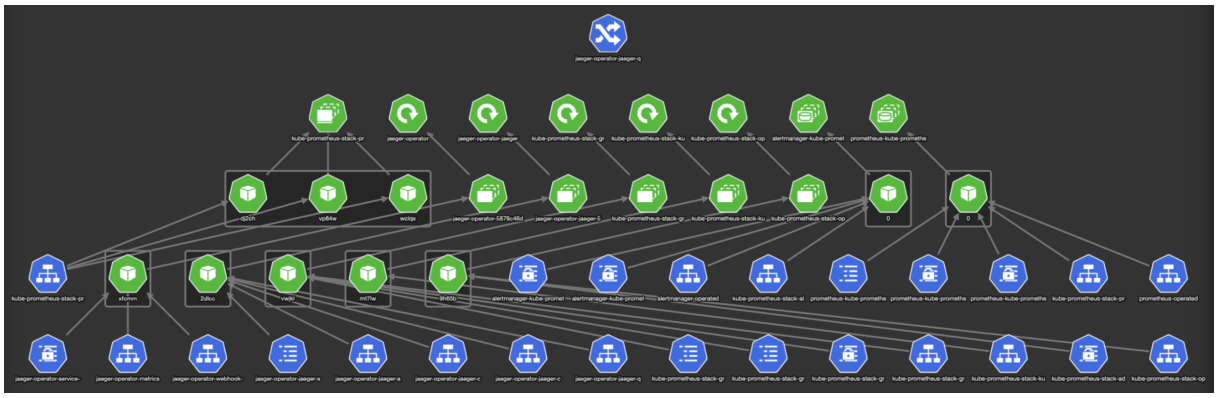


Figure 5.8: Kubeview with more resources on the kube-metrics namespace in the microk8s-cluster, with Pods from Jaeger and the kube-prometheus-stack like the three Prometheus Pods to the left. This is just to show how more complex layouts can look – the font is too small to see details. In the deployment, the GUI was easy to navigate by zooming in and out, but this is hard to reproduce on paper.

One or all namespaces can be selected and the view can be filtered with the filter field, then only showing resources matching the filter and their relatives. However, the filter is limited to the resource name and does not allow full-text filtering based on the YAML source of the resource. Another limitation is that only the most common resource types are displayed, Custom Resource Definitions (CRDs) are not shown, and neither are nodes. Also, there is no functionality to filter based on resource types, for instance, to only display Pods or only show resources running on particular nodes because no settings other than the namespace-, filter- and refresh-interval fields in the menu bar are available and there is no plug-in system to fine-tune KubeView either.

Figure 5.8 shows how a more complex graph of resources can look, here the kube-metrics namespace is shown in the microk8s-cluster, with the kube-prometheus-stack and the Jaeger Operator. It was accepted that the resource names are unreadable in figures 5.7 and 5.8, their main purpose is to illustrate how a graph can look. Practical tests have shown that this is not a problem becuase navigation inside the graph is easy by dragging it around with the mouse and zooming in and out. Manual reordering of resources by mouse is also possible by dragging them around.

Table 5.7: Rating of KubeView following Table 5.1, legend in subsection 5.1.3.

Adaptability: –	Availability: ++	Deployability: ○	Footprint: ○
Interoperability: +	Popularity: –	Portability: ++	Usability: ++

5.2.7 Jaeger: Tracing to reveal the interdependencies of services

Initially developed by Uber, Jaeger was made available open source and donated to the CNCF in 2019 ([Jae24]). It is a tracing application that allows to for example follow a user request from the front-end service to all its dependent services. An example from the microservices-demo would be that the front-end-service calls the checkout-service, which again calls the shipping-, product catalog-, and cart-services, and the cart-service calls the redis-cache (see Figure 4.5). These interdependencies can be mapped out with tracing, see Figure 5.9 for an example of a Jaeger dependency graph. Jaeger is also used to track how much time each of the involved microservices took to process the request, an example of these timespans for each microservice can be seen in Figure 5.10.

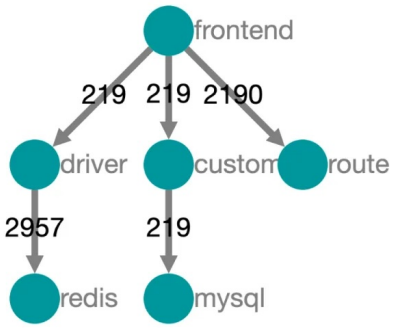


Figure 5.9: Dependency graph of microservices in Jaeger. Figure adapted from [Ana22]

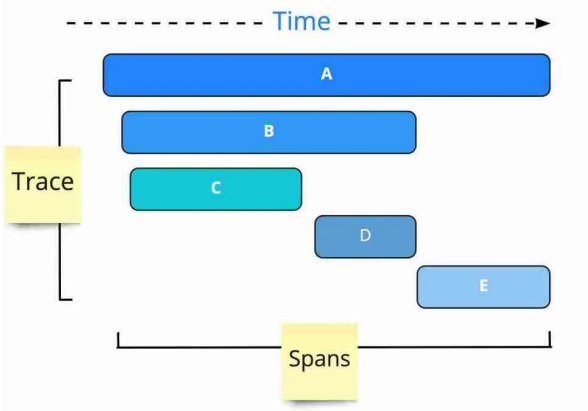


Figure 5.10: Timespans each involved microservice takes to process a request. Figure from [Zac22]

Notice that neither Figure 5.9 nor Figure 5.10 show the microservices-demo, both are taken from external sources. That is because installing and configuring Jaeger is not as simple as deploying a readily configured Helm-chart. Similar to Prometheus, there is an implementation of k8s's Operator Pattern using Helm to install the backend storing the traces. It is called Jaeger Operator and requires k8s's official cert-manager to be deployed in the cluster. However, the backend is not enough to use Jaeger since every microservice needs to be instrumented for the tracing to work. That means additional code needs to be added to each microservice, which enables following the workload from the frontend via driver to redis, see Figure 5.9.

There are different methods to instrument microservices, in some cases an additional container can be added instead of modifying the existing application's container. Nevertheless, instrumenting all the microservices can require significant work in a cluster that has not been instrumented yet. Due to this complexity, the deployability has been rated as $--$. In the context of this thesis, it was decided not to prioritise the time to instrument the microservices-demo, so Jaeger was not tested apart from deploying the Jaeger Operator. However, Kramer used Jaeger in his work comparing the performance of k8s network drivers due to its precise delay and performance measurement capabilities ([Kra24]).

Table 5.8: Rating of Jaeger following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: −−	Footprint: ○/−
Interoperability: ++	Popularity: ++	Portability: ++	Usability: +

5.2.8 Otterize Network Mapper: Map out relations between applications

The Otterize network mapper creates a service map that allows to study how the applications deployed in a k8s cluster are related. It "creates a map of in-cluster traffic by (1) capturing DNS traffic and (2) inspecting active connections" ([Ott24]). For that, the IP addresses participating in connections to the Pods are resolved until the root object is reached. This information is used to map out the different entities communicating inside the cluster ([Ott24]). Figure 5.11 shows a visual representation of this map for the microk8s-cluster. Cilium and Hubble introduced in subsection 5.5.2 and the Elastic logging stack introduced in subsection 5.4.1 appear at the top, while the microservices-demo appears at the bottom. With its initial commit being from 2022, Otterize's network mapper is still a relatively new open-source project. It is free but reports anonymous usage statistics back to Otterize unless they are explicitly disabled.

Figure 5.11: Full map of the services in the microk8s-cluster generated by the Otterize network mapper

To install the network mapper, the Otterize CLI must be installed on the analyst's machine. It requires a backend that can be installed using a Helm-chart. It deploys a DaemonSet with one sniffer Pod on each node and another Pod called network-mapper. It collects the information from the sniffers and builds the service map, which it keeps up to date as long as it is installed. There is no GUI, so all the interactions happen through CLI-commands. For example, Figure 5.11 was generated using `otterize mapper visualize -o <filename.jpg>`. In the figure,

all namespaces are included but it is possible to limit the graph to only a single namespace, for instance, the microservices-demo. A textual representation can be generated with `otterize mapper list`. It lists for each service which other services it is accessing.

Table 5.9: Rating of the Otterize network mapper following Table 5.1, legend in subsection 5.1.3.

Adaptability: +	Availability: ++	Deployability: ○	Footprint: −
Interoperability: ++	Popularity: +	Portability: ++	Usability: ++

5.2.9 NeuVector: Full lifecycle container security platform

Suse's NeuVector is an open-source full lifecycle container security platform with protection against vulnerabilities in the deployment process of k8s applications ([Neu24]). It has been included here for its reactive security features including an overview of the security policies and a network map visualising the interdependencies of the applications deployed in the k8s cluster. Most of its features are available free of charge, paid premium features include a pre-defined rule set, premium support and other services that help using NeuVector.

The tool provides vulnerability management by scanning container images for known vulnerabilities. Moreover, it helps to implement zero-trust network security in k8s clusters by providing a list of the implemented k8s network security policies and possible vulnerabilities in them, along with an overview of the deployed k8s resources. Figure 5.12 shows NeuVector's network activity graph listing all the communicating entities in the microservices-demo.

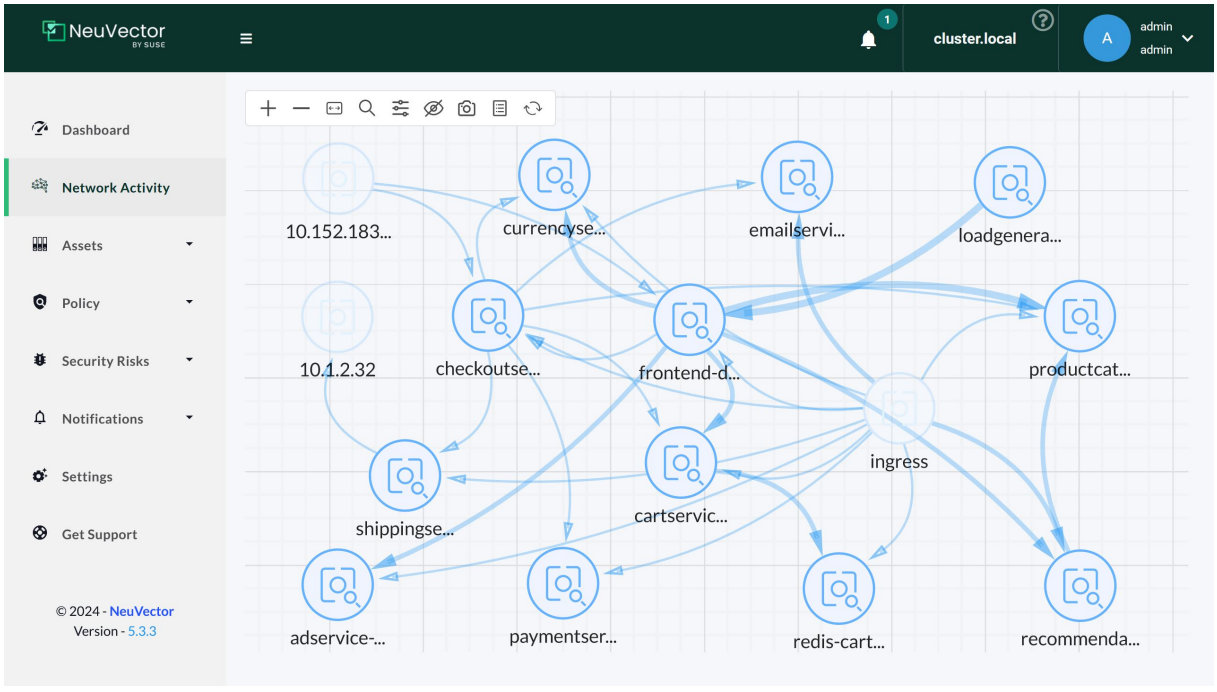


Figure 5.12: NeuVector network activity graph for the microservices-demo

A Helm-chart is available to deploy NeuVector, and no additional configuration was required to get it up and running. Using k9s, Figure 5.13 gives an overview of the k8s resources deployed in the cluster along with their CPU and RAM consumption. These k8s resources are:

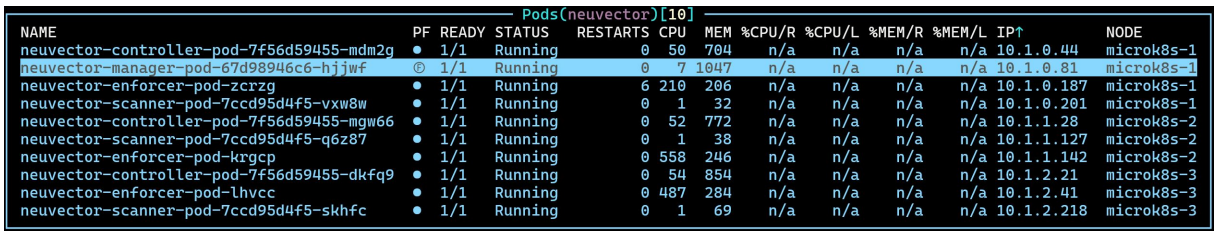


Figure 5.13: K9s overview of the Kubernetes resources deployed by NeuVector

- **Manager**: Among other things, this Pod serves the web GUI.
- **Enforcer**: A daemon set deploying a Pod that enforces the policies set in NeuVector on each node in the cluster.
- **Scanner**: A deployment consisting of three Pods for HA that fetch metrics data.
- **Controller**: A deployment consisting of three Pods for HA that controls the enforcers.

Figure 5.13 indicates a noticeable CPU consumption of about half a vCPU for the enforcers, and a considerable RAM consumption for all the resources. Nevertheless, there would only be more enforcers in a bigger cluster with more nodes, so the total cluster resource consumption would not increase significantly. NeuVector primarily provides proactive k8s security features like vulnerability scanning. After its deployment, it immediately detected that Secrets were built into the images Logstash containers. As the focus in this thesis is on the reactive features, only those capabilities like the network activity graph shown in Figure 5.12 were evaluated here.

Table 5.10: Rating of NeuVector following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +	Deployability: ○	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.3 Digression: Access control in Kubernetes

If kubectl is attached to a vanilla microk8s or minikube[6] cluster, it can perform all the actions listed in subsection 5.2.1. This corresponds to full k8s administrator rights, allowing anyone with access to kubectl to perform any change in the cluster. As mentioned in subsection 3.3.2, kube-apiserver has full access to read and modify any resource in etcd. With almost full control over kube-apiserver, any Secret or authentication certificate can be read using kubectl, see Figure 5.14:

```
ubuntu@kubectl:~$ kubectl get secret microk8s-dashboard-token -n kube-system
-o jsonpath={".data.token"} | base64 -d
eyJhbGciOiJSUzI1NiIsImtpZCI6InlRQUdoWUdmRWRIWHFJTmg2ZE9uMmFXclNJcWdhMU5fem96S
VBkLUVTN3cifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5
pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8v
c2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJtaWNyb2s4cy1kYXNoYm9hcmQtdG9rZW4iLCJrd
WJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiZGVmYXVsdC
```

Figure 5.14: Example of getting Secrets like the bearer token for the kubernetes-dashboard with kubectl.

[6] Both are k8s distributions mainly for development, which can deploy a whole cluster on a single machine.

Even though kubectl can do a lot in this configuration, access to modifying the underlying infrastructure is still limited. For instance, even the highest privileged kubectl user cannot alter the infrastructure-related cluster-parameters in OpenStack Magnum, see subsection 4.1.2.

User access to resources is usually limited in k8s production environments and not as open as the default in microk8s. In k8s, permissions and resource access are configured using Role-Based Access Control (RBAC). K8s administrators get user accounts to administrate the cluster, which can be assigned different roles granting access to resources. Kubectl requires login with a specific account, ensuring that users only get the permissions they are entitled to. RBAC is also used internally in k8s, where each Pod uses a so-called ServiceAccount to authenticate to kube-apiserver. RBAC with user or ServiceAccounts also determines and limits the capabilities each tool presented in this chapter actually gets ([HJ20], page 62).

Another separation of resources in k8s is done using namespaces, which can be used to divide a cluster into multiple virtual clusters. Each namespace can be assigned different resource quotas or specific nodes. Naturally, user accounts can be restricted to only have access to certain namespaces ([HJ20], pp. 64-65). With a group, several users can be assigned the same roles. A user account in the `system:master` group gets full access to the cluster, whereas accounts in the `system:kube-proxy` group only get access to the resources kube-proxy requires ([HJ20], p. 62).

It might sound tempting to give everyone working on the cluster full administrator permissions, but that can be dangerous not only from a security perspective. Usually, following the Principle of Least Privilege (POLP) giving everyone no more than exactly the permissions they need to do their job greatly limits the number of possible security backdoors in a system. My own experience from working in the field of cyber security has shown that security professionals investigating alerts in k8s using tools like kubectl mostly need reading permissions to map out the context around an alert. Consequently, I would recommend assigning their accounts to a group giving limited writing, but wide reading permissions on the cluster.

5.4 Tools for collecting and viewing logs

When a Pod terminates or dies, everything stored in its filesystem, including all log files, is lost. Pods may also rotate log files, that is deleting old data to free disk space for new data. In a nutshell, a system is needed that continuously fetches logs from the Pods, and stores them queriable for a retention period. This section focuses on the Elastic Stack, as it is widely adopted and the basic functionality required for this use case is obtainable free of charge. It also integrates well with other open-source logging tools, some of which are evaluated here. Presumably, some of the findings can be applied to other commercial logging stacks like Splunk, too.

5.4.1 Elastic Cloud on Kubernetes (ECK): Complete logging stack

Elastic NV is the American-Dutch company developing the Elastic stack formerly known as ELK stack. It consists of the data shippers Beats, the data processing pipeline Logstash, the distributed search and analytics engine Elasticsearch, the GUI and dashboard application Kibana, and many other logging infrastructure tools ([Ela24]). As a consequence of cloud providers like Amazon reselling their products “without contributing back”, Elastic NV moved its open-source

products under the copyleft Server Side Public License (SSPL) in 2021 ([Ban21]). When SSPL-licensed software is offered as a service to others, all the required software, APIs and dependencies to run that new service must be published under the SSPL-license, too ([Mon18]). Elastic offers paid versions of its services with enhanced features in the cloud, but the core services remain free as long as they are self-managed and not sold as a service to others ([Ban21]).

Despite some complaints from the open-source world about the change in licensing, the Elastic Stack remains a popular choice as a full-scale logging solution with a great user community. Its large amount of functionality and capabilities can make it somewhat complex to deploy and learn to use, but at least it is well-documented. Furthermore, the Elastic stack is highly customisable with settings, APIs, and plug-in systems.

Section A.4 in the Appendix shows the configuration that was required to get the Elastic Cloud on Kubernetes (ECK)-stack up and running in the microk8s-cluster, including enrichment with k8s metadata when the log files are imported in Filebeat and visualisations in the Kibana GUI. The setup parses container logs to address the first priority from section 3.5 and audit logs from the control plane to address the third priority of protecting kube-apiserver and etcd. On the one hand, a lot of configuration and fine-tuning are required to properly set up the ECK Stack, on the other hand, it provides a lot of functionality that is useful for forensics and intrusion detection in k8s. Accordingly, setup and usage are not simple, but at least well-documented. Any data can be sent to Elasticsearch and visualised in Kibana, and there are also capabilities for dashboards similar to the kube-prometheus-stack with Prometheus and Grafana.

Another issue that needs to be discussed is ECK's footprint in the cluster, which is significant. A longer retention period requires more disk space for Elasticsearch. 24 hours of logging in the current setup took up 3.5, 2 and 1.5 GB of disk storage on the nodes. Elasticsearch does not automatically delete old data when it runs out of disk space, it just rejects new data when that happens. Therefore, an Elasticsearch Index Lifecycle Management (ILM) policy was needed to ensure that enough disk space remains available to store new data, see subsection A.4.5.

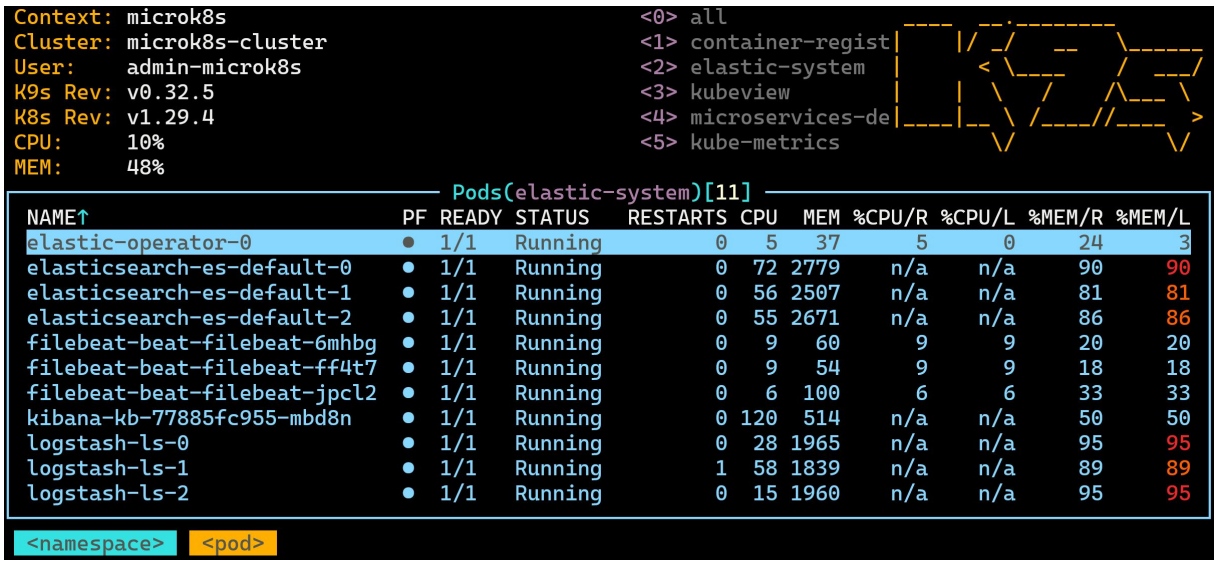


Figure 5.15: K9s view of the resource consumption of the ECK Stack in the microk8s-cluster

Many components in the ECK Stack make use of the Java Virtual Machine (JVM) internally, and especially Elasticsearch and Logstash have significant CPU and RAM requirements. The more resources Elasticsearch gets, the faster searches can be performed in Kibana: More RAM → more quickly accessible cached data → faster searches. The resource consumption of the ECK Stack components in k9s is shown in Figure 5.15. Reducing Logstash's RAM limit to one GB per instance resulted in the Pods restarting from time to time due to Out of Memory (OOM) errors. The Elasticsearch instances use > 80 % of their available RAM, this percentage remained unchanged even when the RAM limit per instance was increased from two to three GB. Elasticsearch always takes all the RAM it can get, and so does Logstash. Deploying ECK increased the average RAM usage in the cluster from about 20 % to close to 70 %. As a result, the microk8s-cluster nodes were upgraded from 8 to 16 GB of RAM per node. An average of 48 % of these upgraded 48 GB of RAM in the cluster is now used, as indicated in Figure 5.15.

Table 5.11: Rating of Elastic Cloud on Kubernetes (ECK) following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +	Deployability: −	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: +

5.4.2 Fluentd and Fluent Bit: More resource-efficient log-collectors

The log-collection with Filebeat and Logstash from the ECK stack presented in the last section provides a lot of features, however, it is also resource-hungry, especially Logstash. Fluentd was started as a more resource-efficient alternative by Treasure Data, and later became an open-source CNCF graduated project ([Flu24b], [Ber23]). It is one of the most popular Docker images and widely used ([Ber23]). Fluentd is implemented in C and Ruby with a memory footprint of around 60 MB. Being built as a Ruby Gem, it requires a certain number of other Gems to run. It is highly customisable, too, and extendable with more than 1000 external plug-ins ([Flu24a]).

Fluent Bit was started as another CNCF graduated project in 2015 by the same authors, and is built "on top of the best ideas of the Fluentd architecture and design" ([Flu24a]). It is implemented entirely in C, making it compatible to run efficiently even in embedded devices like microcontrollers. With a memory footprint of about one MB and no external dependencies, it is even more performant than Fluentd and regarded the "next generation solution" ([Flu24a]). Fluent Bit is also customisable with over 100 built-in plug-ins. Booth tools can export their parsed logs to a variety of outputs, in this case Elasticsearch and Kibana are reused from the ECK-deployment. The technical details of this setup are shown in section A.5 in the Appendix.

Fluent Bit provides functionality similar to Filebeat and Logstash combined. Figure 5.16 demonstrates how it uses similar CPU and less RAM than Filebeat alone, significantly reducing the footprint. The k8s metadata added by Fluent Bit and Filebeat contains similar information, but the naming is different. Comparing the number of container-logs per hour from Fluent Bit and Filebeat in Kibana like in Figure A.3 showed that each time interval had exactly the same number of logs from both sources, in this way proving that both tools are set up correctly.

In his example that was deployed and adapted as a Proof of Concept (PoC) in section A.5, Ernst used Fluent Bit because the environment variables for credentials and ConfigMaps for configuration files were better supported in its Docker image and due to its improved efficiency

Pods(elastic-system)[14]

NAME↑	PF	READY	STATUS	RESTARTS	CPU	MEM	%CPU/R	%CPU/L	%MEM/R	%MEM/L
elastic-operator-0	●	1/1	Running	4	3	38	3	0	25	3
elasticsearch-es-default-0	●	1/1	Running	3	87	2766	n/a	n/a	90	90
elasticsearch-es-default-1	●	1/1	Running	2	49	2492	n/a	n/a	81	81
elasticsearch-es-default-2	●	1/1	Running	2	71	2667	n/a	n/a	86	86
filebeat-beat-filebeat-rmltj	●	1/1	Running	1	12	124	12	12	41	41
filebeat-beat-filebeat-t6pwv	●	1/1	Running	1	3	67	3	3	22	22
filebeat-beat-filebeat-vczb2	●	1/1	Running	2	6	180	6	6	60	60
fluent-bit-h7w4s	●	1/1	Running	2	12	10	n/a	n/a	n/a	n/a
fluent-bit-mlqtk	●	1/1	Running	1	8	33	n/a	n/a	n/a	n/a
fluent-bit-xh6kg	●	1/1	Running	1	2	34	n/a	n/a	n/a	n/a
kibana-kb-5f5d87757d-sl7vf	Ⓕ	1/1	Running	2	46	578	n/a	n/a	56	56
logstash-ls-0	●	1/1	Running	4	16	1741	n/a	n/a	85	85
logstash-ls-1	●	1/1	Running	3	57	1701	n/a	n/a	83	83
logstash-ls-2	●	1/1	Running	5	13	1412	n/a	n/a	68	68

Figure 5.16: K9s resource consumption comparison of the ECK Stack and Fluent Bit

([Ern20]). As both tools are similar, no more time was spent on getting Fluentd to work after finding that the Fluentd microk8s-plug-in did not work out of the box. Moreover, only container logs were implemented here as a PoC, presumably it is also possible to set up audit-logs like they were for Logstash in the ECK Stack.

Table 5.12: Rating of Fluent Bit and Fluentd following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: ○	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.4.3 Kube-audit-rest: Retrieve audit-logs in cloud platforms

As mentioned earlier, k8s in cloud environments like AKS, AWS or GCP often either do not provide access to audit logs from the control plane, or access to audit logs is expensive with each API call to fetch the logs costing money. Therefore, Tweed developed the open-source tool kube-audit-rest enabling "Kubernetes audit logging when you don't control the control plane" ([Twe24]). It has been specifically designed with scenarios in mind where it is impossible to directly configure kube-apiserver, for instance, in k8s cloud environments. There are only three other maintainers apart from Tweed, which is understandable since the initial commit was only two years ago. Recent activity in the repository indicates that it is actively maintained.

Kube-audit-rest captures all mutation/creation API calls to disk before exporting them to logging infrastructure like the ECK Stack, which is cheaper than the Cloud Service Provider managed offerings charging per API call and not supporting ingestion filtering ([Twe24]). It adds a validation webhook capturing API calls, which are then forwarded via the log-aggregator Vector.

Tweed provides an example configuration with YAML-files that could be deployed to the microk8s-cluster and worked well. Some adaptations were made in the deployment YAML-files to configure Elasticsearch datastreams instead of individual indices and to integrate it with the existing ECK Stack deployment. In particular, `bulk.action:create` needed to be added, along with an additional input filter adding `data_stream.namespace` and `data_stream.dataset`-fields, the code with the modifications is available in the Appendix.

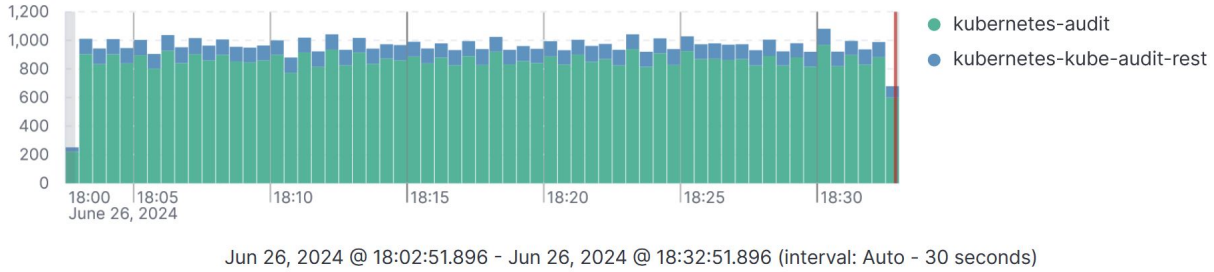


Figure 5.17: Kibana comparison of the volume of log messages coming in during the last five minutes from kube-audit-rest versus the full audit logs parsed from set up in Filebeat in subsection A.4.3.

The volume of audit logs coming in from kube-audit-rest is far less than the full audit logs that were set up in Filebeat in subsection A.4.3, as the tiny blue bars on top of the green ones in Figure 5.17 indicate. This is since the policy in Listing A.3 that was set up for Filebeat logs everything. Since kube-audit-rest is restricted to modification and creation requests in the k8s-API, read only requests cannot be captured in this setup.

During the testing, I also noticed that Vector regularly sent error messages about packets of data that could not be uploaded to Elasticsearch. I filed an issue about this on GitHub[7] and got a reply from Tweed few hours later with a fix. It was the same problem as in Logstash in subsection A.4.4: Unpacked k8s Hyper Text Transfer Protocol (HTTP) objects could not be indexed due to inconsistencies, so they had to remain JSON strings. Later, I contributed some of my fixes to the example deployment in Pull Requests (PRs) on GitHub. Even though the total volume and type of logs it can scrape is limited, kube-audit-rest scrapes more audit-logs than if there were none at all, and it is free.

Table 5.13: Rating of kube-audit-rest following Table 5.1, legend in subsection 5.1.3.

Adaptability: +	Availability: ++	Deployability: ○	Footprint: ○
Interoperability: ++	Popularity: −	Portability: ++	Usability: +

5.4.4 Other tools for viewing logs live in Kubernetes

These tools do not allow persistently storing and querying logs back in time like the full logging-stack tools shown previously. However, they can help to easily check out the logs directly from running Pods for debugging and getting an overview. Logs can for instance be viewed in the earlier introduced kubernetes-dashboard, kubectl using `kubectl logs` and k9s. Originally developed by Wercker and later forked and community-maintained, Stern is another open-source tool for viewing k8s logs that allows to stream the logs from multiple Pods simultaneously ([Ste24]). A link for an extended plug-in I developed and contributed to k9s can be found in the Appendix, it leverages Stern, jq and other capabilities for logs in k9s.

5.5 Tools for collecting and viewing networking information

Communication plays an integral role in the world of microservices; between the services inside the cluster as well as with the outside world. In case of a communication outage, almost no

[7] GitHub issue for kube-audit-rest: https://github.com/RichardoC/kube-audit-rest/issues/169

services can continue their operation unimpacted. Keeping an overview of everything that is communicating in bigger k8s clusters can be challenging. Moreover, network communication is important for many attack scenarios; proper mechanisms increase visibility and allow alerting and detection. This section first highlights some of the most common network drivers for k8s, with a special focus on their observability and detection capabilities. Thereafter, other tools facilitating observability in k8s networks like service meshes are evaluated.

Network drivers are the heartbeat of k8s clusters – no communication runs without them. They get a double rating like −/+ on deployability: The first rating is − or −− because exchanging the network driver for a running k8s cluster is a risky and complicated operation. The second rating evaluates how easy it is to configure the network driver in a new vanilla k8s cluster.

5.5.1 Calico: Kubernetes' initial network driver

Tigera's open-source Calico was the first k8s network driver and used during the initial development of k8s. Its "network policy engine formed the original reference implementation of Kubernetes network policy during the development of the API" ([Tig24], page 40). Calico has been the de-facto default network driver for k8s and is yet widely deployed. Microk8s still relies on Calico by default, so it is deployed in the microk8s-cluster. OpenStack Magnum used Calico as its default network driver until version yoga from 2022. Even though Calico is still supported, the default was changed to the simpler and less feature-rich network driver flannel from version zed. Many other providers change their defaults to the modern and feature-rich Cilium.

In a vanilla k8s cluster, Calico can be installed easily using a Helm-chart or its own CLI. Furthermore, it has a plug-in system and is highly configurable. For example, an extended Berkeley Packet Filter (eBPF) driver replacing kube-proxy can be configured for improved throughput. Moreover, Calico has extended support for network policies and other features relevant for the proactive side of k8s security. However, most of the features enabling extended observability and visibility into the network are limited to Calico Cloud, Tigera's paid enterprise version of Calico ([Tig24]). These premium features include the export of NetFlow logs or displaying a graph of the relations between the applications deployed in k8s. As it requires the k8s cluster to be exposed to the Internet which none of the lab k8s clusters are, these enterprise features could not be evaluated, more about this in section 6.3. Accordingly, the evaluations in Table 5.14 only account for the freely available features.

Table 5.14: Rating of Calico following Table 5.1, legend in subsection 5.1.3, deployability: section 5.5.

Adaptability: ++	Availability: ○	Deployability: −/○	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.5.2 Cilium: Kubernetes' new de-facto standard network driver

Isovalent, which later became part of Cisco, was founded in 2016 to build the new open-source k8s network driver Cilium ([Cil24a]) that reached the CNCF graduated maturity level in October 2023. Since the beginning, Cilium has been based on the relatively new Linux kernel technology eBPF, extending the kernel's functionality by deploying lightweight VMs inside it instead of installing kernel modules ([Kra24]). Using eBPF allows many additional features, including

higher throughput at lower CPU and RAM load, compared to classic network drivers relying on the Linux tool iptables. Due to its high performance, universality, adaptability and high amount of freely available features, Cilium's popularity as k8s network driver is rising, especially in cloud environments. GCP, AWS, AKS and many others have all standardized on Cilium for k8s networking and security ([Cil24a]).

If they are activated, Cilium can also provide many of the functionalities of a service mesh. In his master thesis comparing the performance of eBPF network drivers and service-meshes, Kramer ([Kra24]) found that it can do so with a lower resource footprint and delay than most service meshes, more about this in subsection 5.5.3. Cilium provides many of its security and observability capabilities in external tools. While many features are available open source and free of charge, some features, especially in the observability and runtime enforcement tool Tetragon evaluated in section 6.2, require the purchase of Cilium's enterprise version. Many observability features are also available in Hubble, a security observability tool that can be activated in Cilium and requires it on its base to operate. Here, Cilium and Hubble are evaluated together in one evaluation table 5.15 because they are so tightly coupled.

As indicated in section 5.5, exchanging the network driver in a running k8s cluster is no trivial task. I experienced this first-hand while trying to exchange the running Calico with Cilium in the microk8s-cluster. Microk8s provides an add-on that installs Cilium and removes Calico. Naïvely activating the add-on rendered the cluster unusable because the deployment of Cilium experienced some errors. Among other things, CoreDNS stopped working: No network – no communication – nothing works. At this time, backup tools like Velero shown in subsection 5.7.2 were not deployed yet, and deactivating the Cilium add-on also did not bring Calico back.

Consequently, a new microk8s cluster had to be set up. It took several trials to get the new microk8s-cluster to properly run on Cilium, mainly because the Cilium add-on must be activated on *all* microk8s nodes *before* they join the cluster. Given this self-learned hint that was not documented anywhere, it was no problem to set up the cluster which now again runs smoothly. Cilium can also be deployed using its own CLI or Helm-charts, this was tested in subsection 5.7.1. Usually, Cilium and Hubble can be configured by using the Cilium CLI to redeploy it with the new Helm-values. Due to some limitations of Cilium as a microk8s add-on, it can only be reconfigured using the Cilium CLI with the `cilium config set`-command. This command takes another syntax for the parameter names, with dashes instead of the dots that appear in the Helm-values that are usually the only thing that is documented. The translation of these different notations can be found in the template of Cilium's config on GitHub ([Cil24b]).

Hubble: Network observability for Cilium

Using the Cilium CLI-command `cilium hubble enable`, Hubble can be activated easily. Appending `--ui` to that command additionally deploys Hubble's web GUI. It is relatively simple, but provides a valuable overview of the deployed services in the cluster based the network communication between them. Figure 5.18 shows how the different services in the microservices-demo are related, it resembles Figure 4.5 pretty well. Unfortunately, the boxes representing the services cannot be dragged around in the Hubble GUI to improve the readability on paper. As zooming and moving the graph around is easily possible in the GUI, this is not a problem for

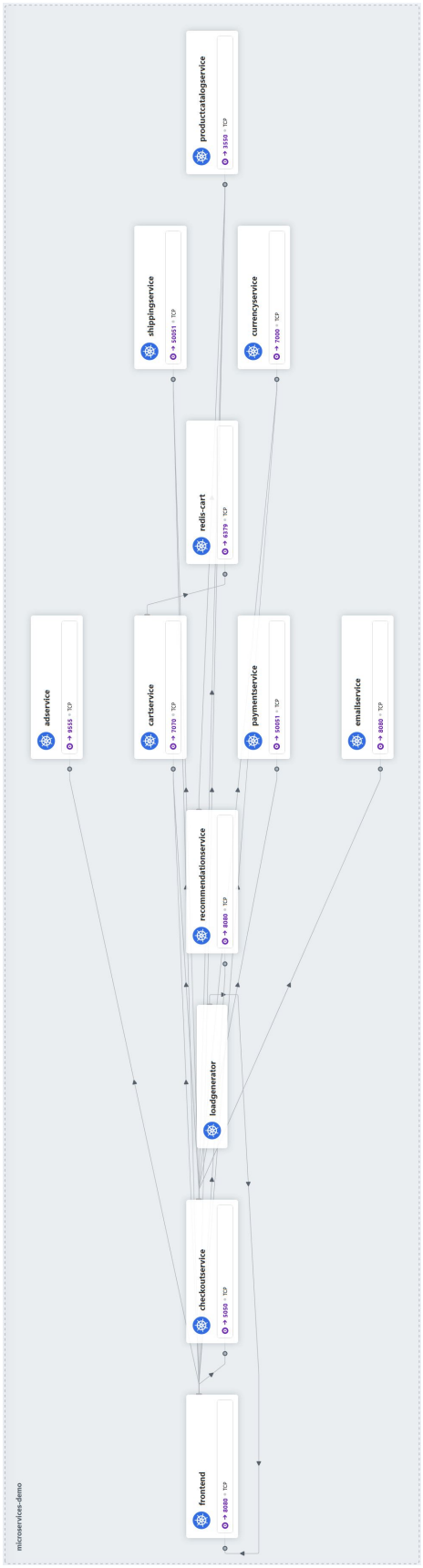


Figure 5.18: Hubble traffic graph for the microservices-demo

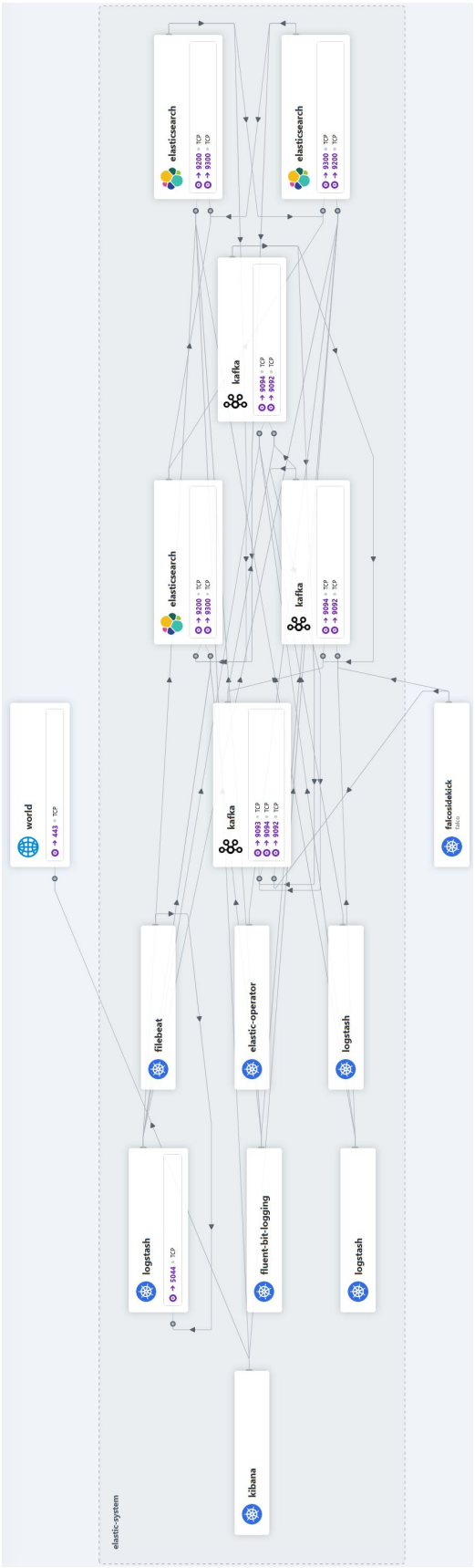


Figure 5.19: Hubble traffic graph for the Elastic Cloud on Kubernetes (ECK) Stack

practical usage. Figure 5.19 shows another network graph for the `elastic-system` namespace. It is noticeable that Kibana is fetching data from the Internet, while Falco sidekick from the `falco`-namespace is sending its data to Apache Kafka. At first, it said "Unknown" as application name for most of the applications in the ECK Stack because it turned out that Hubble fetches the application name from the "app" or "name"-label on the Pods. These labels are usually set for services to find their Pods – after adding them the problem was solved.

Below the service graph, the Hubble GUI displays the involved network flows. A network flow is a sequence of packets from a network port on a source machine to a network port on a destination machine. In the microservices-demo, all these sources and destinations are Pods, for which additional metadata can be displayed. These NetFlow logs can also be exported, this can be activated with the Cilium CLI command `cilium config set hubble-export-file-path /var/run/cilium/hubble/events.log`. As `/var/run` is mounted as a volume in the Cilium Pods, the NetFlow logs are stored directly on the nodes and not inside the Pods. From there, the JSON-formatted NetFlow logs could be imported into Elasticsearch with a Filebeat configuration similar to the one for the audit logs, see subsection A.4.3. Without any additional configuration, the NetFlow logs contain all the relevant NetFlow- and k8s metadata with among other things information about the involved Pods, namespaces and IPs.

Moreover, Hubble comes with its own CLI with extended functionality: It can display information about DNS requests and responses, extract and provide metrics about HTTP request rates, response-statuses and round-trip times, and provide insights into Network policies and their effects. All the metrics can be exported to, for example, Prometheus and Grafana.

Table 5.15: Rating of Cilium with Hubble following Table 5.1, legend in subsection 5.1.3. The double deployability-rating is since network drivers are hard to exchange but can be easy to deploy, see section 5.5.

Adaptability: ++	Availability: +	Deployability: −/○	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.5.3 Service meshes: Encryption and observability for microservices

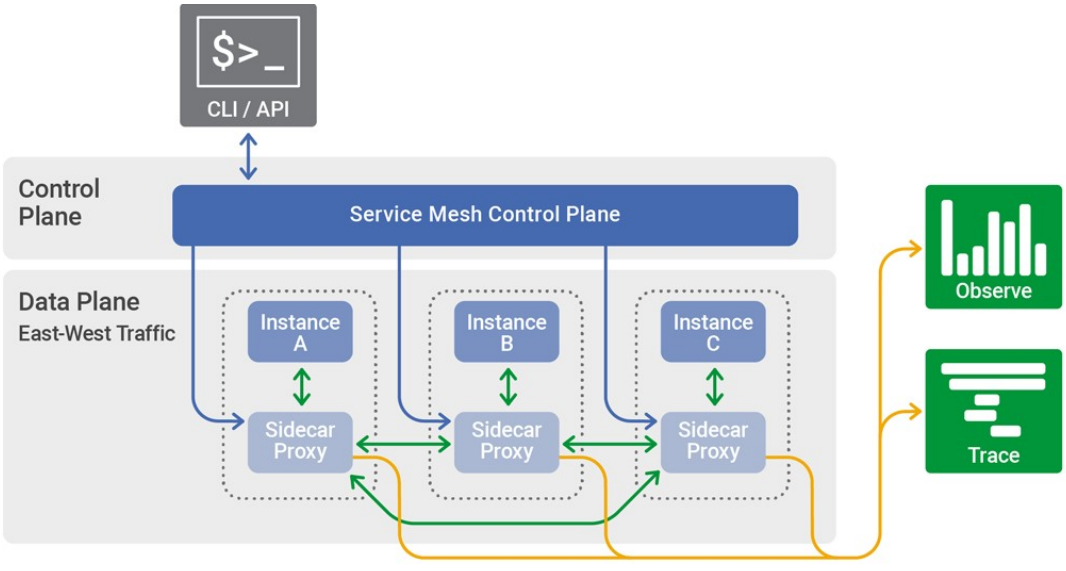


Figure 5.20: Architecture and communication flow for a service mesh with sidecars. Figure from [Aqu21]

In a vanilla k8s cluster with no network and Pod security policies in place, every Pod can communicate with any other Pod in the cluster. Moreover, this communication can be unencrypted without further configuration, leaving the transmitted data vulnerable to data exfiltration attacks that can be performed simply by inspecting the network packets. Service meshes seek to resolve this and other issues by adding another communication layer, typically implemented through an additional container called "sidecar" containing a proxy in each Pod ([Aqu21]). Figure 5.20 illustrates how all the network traffic from application instances A, B and C is redirected through their respective sidecar proxies. These proxies are configured from the service mesh's control plane and have, among other things, the following tasks ([Ama24], [Sgh22]):

1. **Service discovery**: A service registry dynamically discovers and keeps track of all services in the mesh, reducing the operational load of managing Service endpoints. Regardless of the underlying infrastructure, it allows services to find and communicate with each other.
2. **Encryption, Authentication and Authorization**: These properties can be achieved by activating mutual Transport Layer Security (mTLS) encryption for identity verification in service-to-service communication. Can also limit which services can talk to each other.
3. **Monitoring**: Observability and monitoring features allow gaining insights into the deployed services' health, performance and behaviour. Metrics like latency, error rates and resource utilisation can be collected, along with service events in audit logs. They can also include tracing abilities similar to Jaeger introduced in subsection 5.2.7.
4. **Traffic engineering**: Fine-tune how and where portions of the traffic are routed
   a) **Load balancing**: Advanced load balancing between different Pods on different nodes can be activated with various algorithms ensuring High Availability (HA).
   b) **Traffic splitting and canary deployments**: Incoming traffic can be divided between different service versions or configurations. For example, to test a new version referred to as the canary, 10 % of the traffic can be routed to the canary for real-world testing. The remainder of the traffic still goes to the existing stable version.
   c) **Logical meshes**: Several physically separated clusters, for instance, in different parts of the world, can be virtually connected in a logical mesh.
   d) **Request mirroring**: Traffic can be duplicated to test or monitor a service without impacting its primary request flow in production.

The two CNCF graduated projects LinkerD and Istio are the most widely used and known service meshes. Along with many others from different vendors, they all share a relatively common feature set, resource consumption, and architecture. Due to these similarities, the first and oldest service mesh Istio was deployed in the microk8s-cluster and is evaluated as a representative for service meshes in general. It was deployed using the get-started guide ([Ist24]), in addition the `microservices-demo` namespace was activated for Istio injection of sidecar proxies. I noticed that the already deployed Pods from the microservices-demo did not automatically get the sidecar proxy, it was only installed after manually killing the running Pods. Figure 5.21 shows the traffic graph in Istio's web GUI Kiali illustrating how the Pods in the microservices-demo communicate with each other, it resembles Figure 4.5 pretty well.

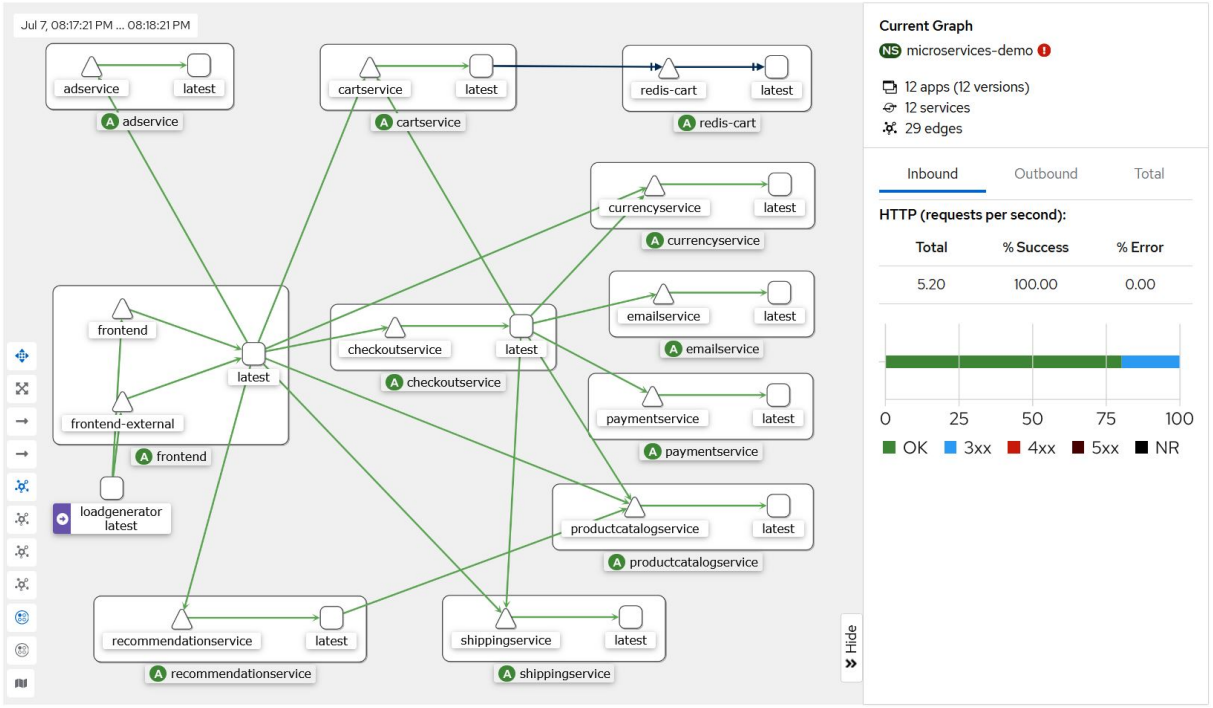


Figure 5.21: Istio's web GUI Kiali showing a traffic graph of the services in the microservices-demo

However, the benefits service meshes provide do not come for free, at least in the sidecar architecture the additional resource consumption is significant, resulting in a footprint-rating of −−. Furthermore, several blog posts from recent years like Sghiouar's ([Sgh22]) argument whether a service mesh is really necessary given its resource overhead and added complexity. A service mesh is only worth it if all of its capabilities are used and understood by everyone working on the k8s cluster. If not, developers can start reimplementing features the service mesh already provides in their own application code. Logical meshes connecting physically separate clusters together can also lead to problems: The developer only sees two intensely communicating Pods next to each other, while in reality, they are in different parts of the world, so the underlying cloud provider charges for this traffic. According to Sghiouar ([Sgh22]), service meshes are not the way to go if a company only wants easily configurable mTLS encryption in their clusters.

One of Istio's newest features is deploying it in "ambient mode", removing the need for sidecar proxies in each application Pod. Instead, additional cluster components are added once per namespace and once per node, significantly reducing the total resource consumption of the service mesh. In this case, the footprint-rating is −. It was tried to set up Istio in "ambient mode" in the microk8s-cluster but it did not work as easily out of the box as the classic sidecar-mode, such that the traffic graph shown in Figure 5.21 did not get populated. No more time was spent to fix it because "ambient" or not, another issue remains: Service meshes introduce additional delays in the order of milliseconds in the communication ([Kra24]).

Many of the additional features service meshes provide can be achieved more efficiently and without sidecars directly in the k8s network driver. For example, the traffic graph in Figure 5.21 can be created similarly as a premium feature in Calico or in Cilium's Hubble, see Figure 5.18. Kramer ([Kra24]) compared the performance implications of service meshes and Cilium and

concluded that Cilium is more efficient to achieve the basic features of service meshes. For more advanced features, Cilium can directly integrate with Istio to combine their strengths ([Kra24]).

Table 5.16: Rating of the service mesh Istio following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: −	Footprint: − − /−
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

5.5.4 Kubeshark: Live network traffic analyser for Kubernetes

Wireshark was the first analyser for raw network packets and protocols and has become the de-facto standard for these tasks. The open-source k8s network analyser Kubeshark ([Kub24f]) developed by KubeHQ is built on similar principles, but tailored specifically for k8s. Its main purpose is debugging of network issues and APIs in k8s applications. As only clusters with up to two nodes are covered by Kubeshark's free license, it usually requires a subscription-based license starting at 10 $ per month for production usage. Kubeshark comes with a CLI, from which the Helm-chart based installation can be initiated using `kubeshark tap`. It is deployed as a DaemonSet with one eBPF-based worker Pod running on each node. Additionally, a hub and frontend-Pod for the web GUI are deployed.

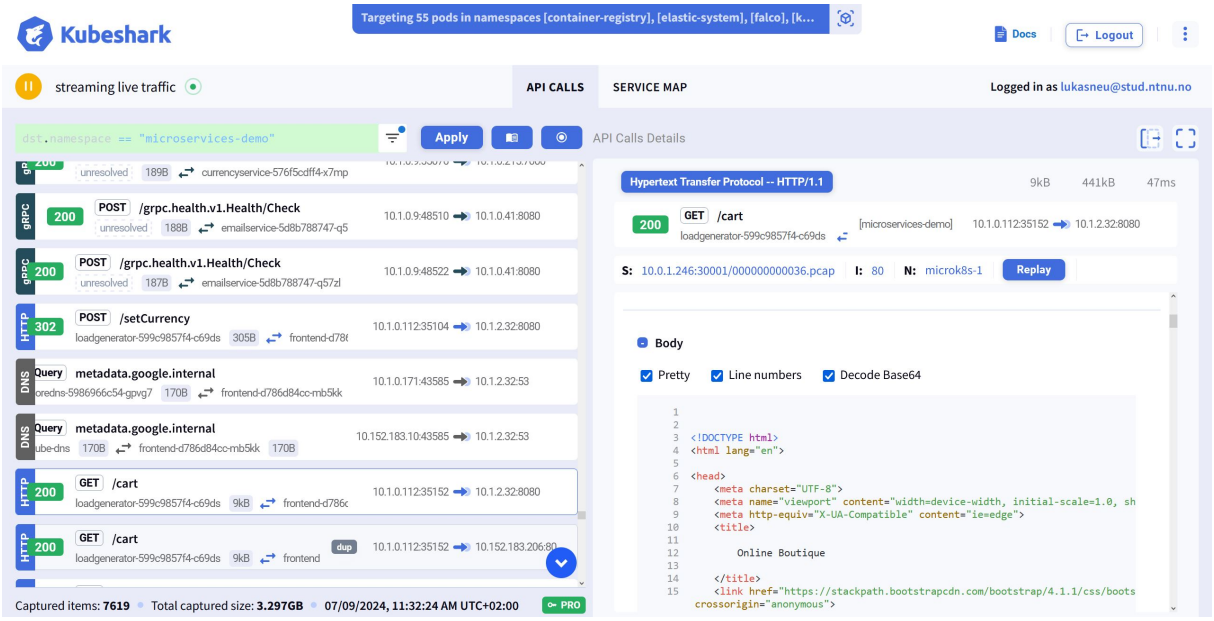


Figure 5.22: Kubeshark showing gRPC, DNS and HTTP network traffic in the microservices-demo

In Figure 5.22, Kubeshark's web GUI is shown for some network packets from the microservices-demo. Like in Wireshark, the involved network protocols like gRPC, HTTP or DNS are opened up and analysed, making it easier to track what is happening. On the right-hand side, the contents of a HTTP packet are shown. It is the response to loadgenerator's request to the frontend to see the contents of the shopping cart. When the `--tls`-flag is set during deployment of Kubeshark, the decrypted contents of Transport Layer Security (TLS) encrypted traffic can also be inspected. At least this worked out of the box in the microk8s-cluster with self-signed TLS certificates, it was not evaluated how this would work with a service mesh or mTLS.

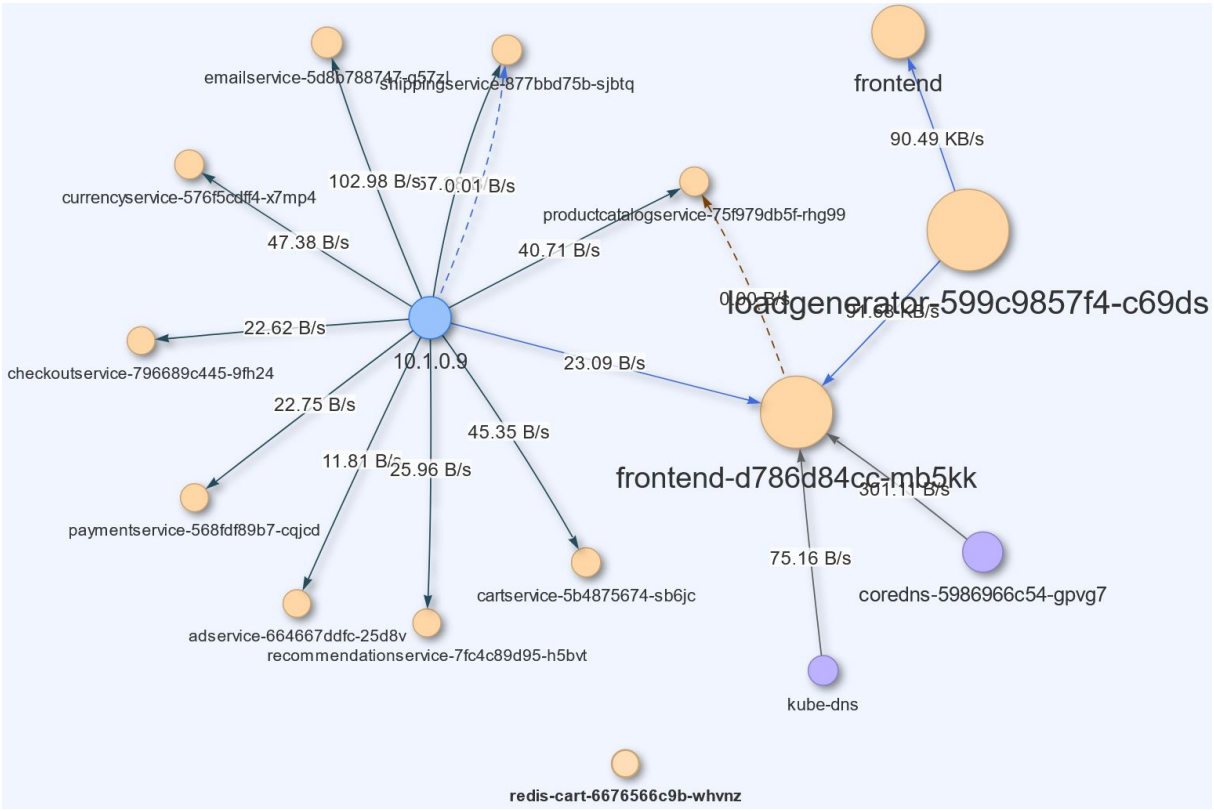


Figure 5.23: Kubeshark service map for the microservices-demo

Different filters can be applied in the GUI in a flexible query language with a syntax similar but not equal to Wireshark. This allows, for instance, to only see traffic to a certain node, Pod or Service. Such a filter was applied to only show traffic destined to the `microservices-demo` namespace in the service map shown in Figure 5.23. When it is started, the Service map continuously changes to account for updates and builds up gradually. Endpoints that receive or send more traffic are displayed bigger. Even though this is not very visible in the figure, bigger arrows indicate larger amounts of traffic.

Capturing raw network packets is a substantial feature for digital k8s forensics since it allows sharing the traffic with others and deep analysis in other tools. At first, it was tried to use Kubeshark CLI's built-in `export` command, but it did not work. A GitHub issue[8] was filed about it, in which the developers told me that the `export` command had been deprecated and replaced by others ([Kub24g]). Firstly, there is a traffic recorder in the GUI, which can run scheduled raw network traffic recording jobs. It stores its data on the worker Pods in the `/data/recordings` directory, from where it can be fetched using `kubectl cp`. Additionally, raw network traffic can be streamed directly from a worker Pod by using `kubectl exec` ([Kub24g]).

Table 5.17: Rating of Kubeshark following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: –	Deployability: ○	Footprint: –
Interoperability: ++	Popularity: +	Portability: ++	Usability: ++

[8] GitHub issue about `kubeshark export` not working: https://github.com/kubeshark/kubeshark/issues/1556

5.6 Tools for remote access to Kubernetes resources

Remote access to different resources plays an important role in k8s forensics to gather information and reconstruct how an attack took place. Pods and containers can be accessed remotely using the aforementioned tools kubectl with the commands `kubectl exec` or `kubectl attach` presented in subsection 5.2.1 or the Kubernetes dashboard evaluated in subsection 5.2.2. K9s featured in subsection 5.2.5 supports `attach` and `exec` for accessing containers and Pods, too.

The prioritisation of k8s components from section 3.5 based on the high-level risk analysis found that the nodes on which k8s is running also are important to protect. In self-hosted deployments where k8s is installed on bare-metal servers or VMs, the node OSs are usually accessible to administrators via SSH or other protocols for remote access. This is also the case for the two test-clusters used in this thesis, on OpenStack Magnum and microk8s. However, direct access to worker and master nodes is restricted in many cloud services like AKS, AWS or GCP.

5.6.1 Adapted vanilla Kubernetes Pods: Remotely access nodes

In some cases, specially configured Pods available in vanilla k8s can be used to circumvent this issue of inaccessible k8s nodes. This section presents two possible approaches and highlights and compares their implications.

Privileged Pods: Direct access to the node's resources

A YAML-deployment installing a privileged Pod based on Ubuntu Linux on each node in a cluster is shown in Listing A.8 in the Appendix. Additional permissions need to be set up in a ClusterRole (line 2) and its corresponding ClusterRoleBinding (line 17) to provide extended access to the privileged Pods, which are deployed on each node using a DaemonSet (line 30). An infinite while loop sleeping 30 seconds at a time is necessary to keep the Pod alive and prevent its immediate termination (line 53). Additionally, the following three settings have been enabled for the Pods (line 43):

1. **HostPID**: Ensures that the Pod shares the host's process namespace. Allows the Pod to enumerate all the host's processes, terminate them or create new ones.
2. **HostNetwork**: Ensures that the Pod shares the host's network namespace. Gives the Pod access to any Service running on localhost of the node and more.
3. **HostIPC**: Ensures that the Pod can use Inter-Process Communication (IPC) for direct interaction with all the processes on the node, for instance, through Linux pipes.

These settings provide almost full access to the node, quite some network diagnostics tools can also be run. Thus, the container isolation mechanisms that should be in place are completely circumvented: These Pods are no longer isolated and can remotely control their node.

Root access to a node by mounting its file system root as a volume

However, this approach requires quite some additional access. "Everything in Linux is a file" ([Kil23]), so mounting the host's root file system as a volume is already enough for quite some access to the host. Listing A.9 in the Appendix shows another DaemonSet, this time it does

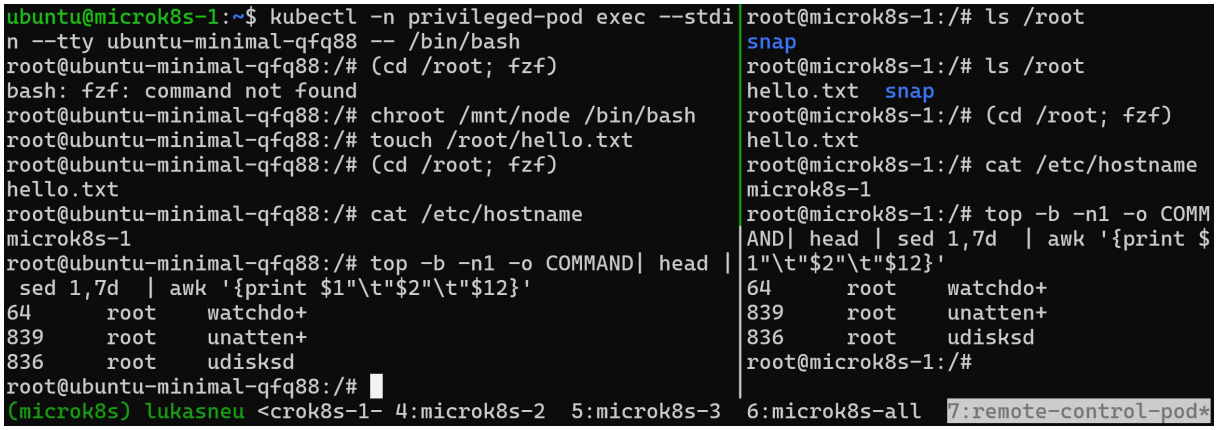


Figure 5.24: Demonstration of impersonating the node from a Pod when its root filesystem is mounted

not require an additional ServiceAccount or ClusterRole for additional access, it just mounts the node's file system root `/` (line 31) to `/mnt/node` (line 23).

In Figure 5.24, the terminal multiplexer tmux (see Figure 2.1) is used to demonstrate how a node can be remote controlled from a Pod only having its root file system mounted. The commands will be explained one by one in the order they were executed. Both terminals are run on the `microk8s-1` node in the microk8s-cluster. The left terminal window is used to impersonate the `microk8s-1` node from a Pod, the right terminal window shows commands on the `microk8s-1` node to verify the attack from the Pod on the left.

1. `ls /root` (right): Shows the contents of the node's root user's home directory `/root`, currently it only contains the `snap`-folder.
2. `kubectl -n privileged-pod exec -stdin -tty ubuntu-minimal-qfq88 - /bin/bash` (left): Use `kubectl exec` to execute a bash-shell on the `ubuntu-minimal-qfq88` Pod initialized in Listing A.9 and running on `microk8s-1`. The following commands on the left-hand side are thus executed in this Pod.
3. `(cd /root; fzf)` (left): Trying to execute the fuzzy-search (`fzf`) command in the Pod, but it is not found because it is not installed in the Pod.
4. `chroot /mnt/node /bin/bash` (left): The built-in Linux command `chroot` is used to change the root-directory of the current process, so now what earlier was `/mnt/node` becomes `/`. This command takes over control of the node, as all of the node's resources and files now appear as the Pod's files and resources, too. The following commands to the left run quasi on the node, limitations are listed at the end of this section.
5. `touch /root/hello.txt` (left): Create an empty file `hello.txt` in the node's root user's home directory `/root`.
6. `ls /root` (right): Rerun step 1 and voilà, `hello.txt` created in step 5 has appeared.
7. `(cd /root; fzf)` (left and right): Retry the fuzzy search command `fzf` from step 3. The command installed on the node is now found and works on both sides.
8. `cat /etc/hostname` (left and right): The hostname file shows `microk8s-1` on both sides.

9. `top -b -n1 -o COMMAND | head | sed 1,7d | awk '{print $1"\t"$2"\t"$12}'` (left and right): Due to limitations in this Pod, the typical `ps aux`-command fails to enumerate the processes here. Thus, the "table of processes" (`top`)-command is used. Its output is formatted for readability, showing only the process identifier (PID), username and executable for the top three processes sorted by executable (COMMAND). It is the same on both sides, showing that the Pod impersonating the node sees the same processes as the node.

As indicated in step 9, this setup still has some limitations. For instance, Ubuntu's snap programs like microk8s cannot be executed here as the privilege escalation is detected by snap's backend. Many built-in tools and other simpler commands like the text-editor Vim work though.

Wrapping up adapted vanilla Kubernetes Pods for remote access

This approach of remotely connecting to nodes via vanilla k8s Pods and other built-in resources is very versatile, precisely because no external tools are required. Privileged Pods and Pods mounting the node's root file system could be deployed in the OpenStack Magnum- and microk8s-cluster without requiring additional configuration. As these Pods can be used for privilege-escalation attacks as indicated in the previous section, this marks significant security vulnerabilities.

In most production k8s clusters, Privileged Pods sharing the node's resources cannot be deployed due to restrictions in Pod security policies, and the demo from the last section shows that they should be ([HJ20]). Still, by default, everything is allowed in OpenStack Magnum as well as microk8s. Microk8s's developers in Canonical recommend restricting the deployment of privileged containers ([Can24]) to Center of Internet Security (CIS) harden the cluster, but these settings are not applied by default. Consequently, there is a risk that there are k8s clusters in production that do not restrict this access. However, this discussion belongs to the proactive side of k8s security which is not the main focus of this thesis.

The second approach of mounting the node's root directory to the container is less likely to be restricted by policies than privileged Pods, so it is more likely to work. Nevertheless, the main purpose of these adapted vanilla Pods of accessing nodes in cloud environments most likely will not work, as the cloud providers most likely have thought of this[9]. Still, both approaches or their even more powerful combination of a privileged Pod with the node's root directory mounted can provide interesting insights. The abilities of the Pods themselves can also be extended by using different images and further customisation, Heinz shows this in his blog ([Hei23]).

Table 5.18: Rating of adapted vanilla k8s Pods following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: ○	Footprint: ○
Interoperability: ++	Popularity: ++	Portability: ++	Usability: +

5.7 Tools for container imaging and backup

Digital forensics often rely on snapshots of the machine's hard disk and RAM for analysis and to reconstruct how attacks might have taken place, see subsection 5.1.1. When attackers notice

[9] No practical tests on this were run in cloud environments, so there is a slight chance.

that they have been discovered, they sometimes pause their activities to make it harder to discover their entire foothold being all the machines they are controlling. Analysis attempts like establishing a remote SSH connection to the container could be detected by the attacker. Accordingly, the ability to create a snapshot of a running container can be valuable: The snapshot can be restored in a sandboxed environment for analysis without the attacker noticing ([Reb22]).

5.7.1 Kubelet Checkpoint API: Kubernetes native container snapshots

In his blog post from 2022 ([Reb22]), Red Hat's Reber introduces the kubelet Checkpoint API, which allows to create a stateful copy of a running container, including its RAM and file system. When the checkpointed container data is moved to another machine that is able to restore it, the restored container continues to run at the same point as it was checkpointed. Since k8s v1.25 from 2022, this feature has been in alpha stage and protected by a feature gate, making it only accessible after explicit activation as it is not ready for production usage yet ([Kub23]). The new feature extends kubelet's API and internally relies on the open-source tool CRIU, which stands for Checkpoint/Restore In Userspace ([CRI23b]).

It was attempted to test the checkpoint functionality based on the instructions given in Reber's blog ([Reb22]). At first, it was tried to make it work on the most commonly used k8s container runtime containerd ([Con24]) powering the microk8s- and OpenStack Magnum clusters. Reber mentions a PR on GitHub ([RK24]) that was supposed to make this feature available in containerd, it was merged on March 7th, 2024. Microk8s is currently running on containerd v1.6.28 which was released before that PR was merged, so it could not be used to evaluate this feature.

Accordingly, a new single-node test cluster was set up on another VM running Ubuntu Linux using the k8s native kubeadm-tool ([Kub24b]), as it allows a more fine-grained configuration of the underlying cluster components than microk8s. However, the kubelet API returned that the checkpoint feature was not implemented in containerd's current stable release v1.7.18 and neither in the release candidate of the new major version containerd v2.0.

K8s' dedicated container runtime cri-o ([CRI23a]) natively supports Checkpoint and Restore in Userspace (CRIU) of containers, the examples in Reber's blog ([Reb22]) are based on it. It was tested in a new kubeadm-cluster with cri-o as container runtime. After setting it up, the checkpoint function in the kubelet API is invoked with the following command:

```
sudo curl -X POST "https://localhost:10250/checkpoint/default/ubuntu/ubuntu"
↪ --insecure --cert /etc/kubernetes/pki/apiserver-kubelet-client.crt --key
↪ /etc/kubernetes/pki/apiserver-kubelet-client.key
```

The certificates used in this command mimic that kube-apiserver is authenticating at kubelet, `--insecure` ensures that `curl` accepts the self-signed certificates. In the last part of the Unified Resource Locator (URL), the container is specified with the pattern `<namespace>/<pod>/<container>`, here `default/ubuntu/ubuntu`. Changing these to a non-existent container leads to a reply from the API indicating that the container does not exist. If the container exists, kubelet's API complains that it was not able to load `libcriu.so.2`. The most recent version of CRIU ([CRI23b]) is installed and the library was found in `/lib/x86_64-linux-gnu/libcriu.so.2`. Therefore, it is not clear why the library cannot be loaded.

As it was impossible to resolve this error, the checkpoint functionality could not be evaluated as a PoC and thus no evaluation table concludes this section. However, recent activity from Reber and others in containerd and k8s on GitHub indicates that this is being worked on ([RK24]).

5.7.2 Velero: Open-source backup and restore of Kubernetes resources

Broadcom VMware hosts cloud native open-source projects in VMware Tanzu, of which the k8s backup and restore tool Velero ([Vel24]) is the biggest. It allows to "backup and restore, perform disaster recovery, and migrate Kubernetes cluster resources and persistent volumes" ([Vel24]). Regular full backups demand considerable resources in terms of storage, CPU and RAM consumption. Many configurations are available in the Velero CLI-tool, along with a plug-in system to extend the core functionality.

The "quickstart evaluation install" ([Vel24]) was applied in the microk8s-cluster. Velero only supports object storage[10] to store its backups, so it cannot be set up to store backup-files directly on a hard disk. Therefore, the example deploys minIO, an open-source object storage platform that mimics AWS S3 buckets. Inspecting the created backup in minIO showed that it did not include the backed-up Pod's file systems. Further research revealed that Velero only supports backing up the configuration of the selected k8s resources along with external volumes like the one for Elasticsearch in the ECK Stack in the microk8s-cluster.

This greatly limits the applicability of this and similar commercial tools like Veeam Kasten for digital forensics in k8s because neither the container's RAM nor file system can be included in the backup. Accordingly, a backed-up Pod cannot be restored in exactly the same state, for instance, in a sandbox for further analysis. Nevertheless, Velero has many useful capabilities for the administration of k8s clusters and its backups are a better starting point for digital forensics than nothing.

Table 5.19: Rating of Velero following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: ++	Deployability: ○	Footprint: ○
Interoperability: +	Popularity: ++	Portability: ++	Usability: ++

5.8 Tools that were not evaluated or disqualified

This section covers tools considered worth mentioning but later disqualified for some reason.

5.8.1 Tools that were not evaluated because there were alternatives

It was impossible to evaluate all available tools as time was limited for this project. Only the most popular tools were evaluated if several tools were found whose functionality and features looked similar. This does not disqualify other tools, in many cases they can be used instead of the ones evaluated here. Tools that were purposely skipped for these reasons are mentioned in this section.

[10] Object storage is optimized for unstructured data and often used for AI and machine learning applications.

Datadog's Vector ([Dat24]) is another more resource-efficient log-collector than the native tools in the Elastic Stack and which can be used for k8s. It is implemented in Rust and released under the copyleft open-source Mozilla license. For example, kube-audit-rest evaluated in subsection 5.4.3 uses it internally to forward its logs to Elasticsearch.

Grafana Loki is a "horizontally scalable, multi-tenant log aggregation system inspired by Prometheus" ([Gra24b]), developed and maintained by Grafana Labs. It does not index the contents of the logs, but only a set of labels on each log stream. This makes logging and indexing faster but does not allow full-text search on the logs. This makes it "simpler to operate and cheaper to run" ([Gra24b]). It was tried to deploy it, but the documentation of this tool turned out to be scattered, incomplete and complicated. Many different types of deployments are possible via Helm-charts. The documentation also covers many different ways to achieve persistent storage, for instance, in AKS, AWS and GCP. However, the documentation did not show which of the many Loki components require persistence, and also not how a custom storage class like the one used for Elasticsearch in Listing A.6, line 189 could be used. As a consequence, Grafana Loki was given up being too complex to deploy and adapt.

Apart from Calico and Cilium, there are also many other network drivers available for k8s like flannel, Weave Net or Kube-router. They were not evaluated in detail because Calico and Cilium are more common and have a richer feature set in terms of security.

KubeSkoop from Alibaba Cloud ([Ali24]) is a network diagnosis tool for Kubernetes that can model kernel events and display network anomalies similar to Cilium Hubble. A sample YAML deployment script is available on GitHub. KubeSkoop deploys additional Grafana and Prometheus instances, not allowing to reuse already existing instances in the cluster, for example, from the kube-prometheus-stack. Even though all Pods are visible in the dashboard, only the default namespace is visible in the network graph, indicating a missing ServiceAccount in the default Deployment for this. As there were alternatives, this tool was not evaluated further.

Showing an overview of resources along with metrics information like CPU and RAM for the resources in a k8s-cluster, ktop is "A top-like tool for your Kubernetes cluster". It is an open-source project mainly developed by Vladimir Vivien ([Viv23]). It must be installed via the plug-in-manager krew for kubectl. Navigation inside ktop is only possible with tab and arrow keys allowing to scroll through the list of resources, the namespace or displayed resource type cannot be switched inside the tool and the results cannot be filtered. To see another namespace, ktop must be restarted with the cmd-arg `--namespace`. Moreover, only current and thus no historical metrics can be displayed. As k9s has a more intuitive user interface and provides more functionality, ktop was not evaluated further.

5.8.2 Tools that are no longer maintained or deprecated

The open-source world is characterised by the fact that many projects are started, live for some time, and then die again because nobody maintains them anymore.

Many sources mention Octant as a good dashboard, it was described as an "open source developer-centric web interface for Kubernetes that lets you inspect a Kubernetes cluster and its applications" ([Oct23]). However, VMware discontinued its development in January 2023.

Weaveworks' Weave Scope was an open-source tool that "automatically generates a map of your application, enabling you to intuitively understand, monitor, and control your containerized, microservices-based application" ([Wea23]), its deprecation notice was added in July 2023.

5.8.3 Tools that are regarded immature or not ready for production usage

Tools in this category come from the open-source world and look like a good idea at first glance but apparently have not left the alpha stage of development. Deployment is poorly documented and does not run smoothly. Also, important features are still missing or do not work yet.

KlusterView by Open Source labs beta was started in 2023 and aims to provide an easy-to-use plug-and-play metrics-stack for k8s ([Ope23]). Tests of the deployment process did not run smoothly. As it exposes all of its Services as NodePorts, some of the requested ports were already taken and needed to be adapted, and that could not even be done the usual way via Helm-values. It also deploys into a fixed namespace that cannot be changed and redeploys Grafana and Prometheus even if they already exist in the cluster. Nothing could be found in the dashboard that the kube-prometheus-stack (see subsection 5.2.3) cannot show. Additionally, there is a graph view showing which Pods run in a node, but it is impractical to use and does not add any unique value justifying the usage of this tool. There are only three maintainers at the moment and the newest commit is from last year. Several features still wait for contributions from the open-source world.

Ksniff, a kubectl plug-in for remote network packet capture from Pods hat its last release of this open-source in 2022 ([Rud22]). Similar to Ktop, it is deployed as a kubectl plug-in that is installed via the plug-in manager krew. If it cannot find it, it tries to install tcpdump on the Pod. Then, the traffic is streamed back to the analyst's machine where it can be analysed in Wireshark or stored in a packet capture (PCAP) file. It also supports another mode where it deploys a privileged Pod into the namespace in case tcpdump cannot be installed. Both approaches are intrusive and could easily be detected by attackers. None of the tests performed on different Pods and in both modes worked out and produced a valid PCAP file, leaving this tool a good idea that is not production-ready yet.

5.8.4 Tools mainly focused on the proactive side of Kubernetes security

This category lists tools that are widely known for k8s security, but which are mainly focused on the proactive side of k8s security and do not provide important reactive features, see section 1.4. For example, static code analysis and checking container images for known vulnerabilities are primarily proactive k8s security features.

Palo Alto Prisma Cloud's open-source tool for checkov provides static code analysis for infrastructure as code (IaC) and software composition analysis (SCA). It scans cloud infrastructure and detects security and compliance misconfiguration using graph-based scanning ([Bri24]). Over 750 predefined policies are included to check for common misconfiguration issues. The project is very actively maintained, with 362 contributors and the last release today on July 9th, 2024.

Collabnix ([Col24]) also lists several other tools, for example, Aqua Security's vulnerability scanner Trivy and Center for Internet Security (CIS) benchmark tool kube-bench.

Chapter 6
Intrusion detection tools for Kubernetes

The previous chapter highlighted the forensic side of k8s security: How can information be extracted to investigate how an attack was performed? Here, the focus is on detecting the attack itself, this is often referred to as intrusion detection. Attacks are usually detected based on deviations from a baseline of normal behaviour. For the context of this thesis, GCP's microservices-demo introduced in section 4.2 and deployed in the microk8s-cluster serves as a baseline, its integrated loadgenerator continuously generates user activity in the webshop. Deviations from this "normal" behaviour can indicate attacks. An emphasis must be put on the word "can": These anomalies in application-, system-, or user behaviour *can* indicate an attack, but they *can* also be completely benign simply because the world is dynamic and things change.

One of the biggest challenges Intrusion Detection Systems (IDSs) face is the trade-off between False Positives (FPs) and False Negatives (FNs): When the IDS alerts on every little deviation or anomaly, there will be a lot of FPs. Axelsson mathematically concludes in his study of IDSs ([Axe00]) that an acceptable True Positive (TP) rate where the IDS correctly alerts on real alarms cannot be achieved together with a low FP rate where the IDS alerts without there being a real intrusion. This is due to the base-rate fallacy ([Axe00]), according to which only a negligible proportion of the deviations in a system are real intrusions. Thus, an IDS claiming to have zero FPs also has an increased risk of FNs, real attacks that are not detected by the IDS.

In an ideal world, there are no FPs or FNs but in reality, some FPs are usually tolerated to avoid the much worse FNs. Fine-tuning is required to filter the alerts from an IDS and separate the wheat from the chaff. The context around an alert also plays an important role. For example, imagine that the alarm is that a new privileged Pod has been started. Without context that is a bad thing, but when it turns out that a system administrator is legitimately setting up a new application it can be ok. Some alerts clearly indicating an attack might be worth keeping even when they generate a lot of FPs because the FPs can be verified with tools like the ones shown in chapter 5. If the alert cannot be verified false, it is likely a TP, a real attack. This continuous fine-tuning and trade-off is the story of cyber security service providers like mnemonic in a nutshell.

In the microservices-demo baseline, activities are relatively steady and do not increase in the evening or when Christmas is approaching. Consequently, a real-world baseline is much more dynamic and less clear. This kind of real-world testing falls outside the scope of this thesis, as it would take too long and require trials, for instance, in a real webshop environment that has real customers and is accessible via the Internet.

Therefore, the tools for intrusion detection in k8s mentioned here are evaluated similarly to the forensics tools from chapter 5. As a consequence, the main focus of the evaluations based on the criteria in subsection 5.1.3 is how well the system could be integrated into the workflows of a cybersecurity company. The real quality of the alerts can only be evaluated qualitatively based on information and research available about the IDS, it cannot be practically verified due to the limitations of the lab setup and scope of this thesis. Still, to the extent possible it will be evaluated if and how additional detection rules can be added and the IDS can be fine-tuned. The results are summarised and discussed in chapter 7.

6.1 Falco: Open-source intrusion detection for Kubernetes

Falco is an open-source "cloud-native runtime security tool [and IDS] for Linux operating systems" ([Fal24b]) implemented in C++ that can run on any physical or virtual Linux machine or container including devices based on ARM. It is built on top of sysdig, a system exploration tool released in 2014 by Degioanni ([Deg14]), who founded a company of the same name the year before. After its initial release in 2016, the company Sysdig donated Falco to the CNCF where it was adopted in 2018 and reached the graduated maturity level in February 2024. The original idea was to create an IDS that works similar to the network IDS Snort, but for system calls in the Linux OS kernel ([DG22], pages 12 and 13).

System calls are "what a running program uses to interface with its external world" ([DG22], page 4), some examples are reading and writing data to or from the disk or network, executing commands and communicating with other processes through IPC like Linux pipes. They are a "rich data source, even richer than [network] packets" because they include much more than just network communication ([DG22], page 12). Falco instruments the Linux OS kernel to collect the system calls with a driver, either by extending its functionality with kernel modules, or by installing a small sandboxed VM into the kernel with the eBPF technology. Both allow Falco to detect for instance privilege escalation, access to sensitive data, data exfiltration, and unwanted program executions or network connections ([DG22], page 5). Section A.7 in the Appendix shows how a PoC of Falco was set up in the microk8s-cluster for system calls and audit logs and with export of the events to Elasticsearch from the ECK-stack for further analysis.

Falco's capabilities and limitations

Degioanni et al. introduce some principles that influenced Falco's design ([DG22], pages 7 – 10). Their conclusions are condensed here for readability, illustrating what Falco can and cannot do:

1. **Streaming engine**: Data is processed quickly as it arrives. Moreover, data is analysed close to the detection avoiding data transports to other locations before analysis. Furthermore, stream processing does not allow correlation of different events like "alert if B follows A", detection is only possible on single, independent events.
2. **Simple stateless rule engine**: Falco is not designed to correlate different events for alerts and does not have the abilities of a general-purpose policy engine, complex operations like inspecting the contents of network packets are not supported. Instead, it is optimised for simple, stateless rules with a compact syntax that is easy to learn and understand.

3. **Adaptable detection rules**: Falco comes with a "robust" default ruleset that can be extended and customised based on the individual use case.
4. **Universal detection engine**: Falco has a flexible plug-in system conforming to a documented API that allows to add a plug-in for any data source. Registered plug-ins are available for, among others, k8s audit logs, Docker, Apache Kafka topics, Linux syslog and Linux journald, and logs from the cloud environments GCP and AWS.
5. **Scalable and little footprint**: Falco is optimised to have a low CPU and RAM overhead, such that it is scalable even to the biggest infrastructures and runs on any architecture. Only one Falco sensor per node is required because each node only has one Linux kernel.
6. **Uninvasive instrumentation**: Falco runs once on each node and does not require updating application code, installing libraries, or rebuilding containers with monitoring hooks.
7. **Stable and tested**: Both syscall drivers have undergone many iterations and years of testing, guaranteeing their performance and stability. Falco does not directly instrument application containers (see point 5), so they will not crash in case of bugs in Falco.
8. **Hard to evade**: "System calls never lie" ([DG22], page 9). The data collection mechanism is hard to disable or circumvent, and trying to do so leaves traces that Falco can detect.
9. **Highly interoperable**: Via the additional daemon Falcosidekick, Falco can export its alerts to many different applications including messengers like Mattermost, log collectors like Elasticsearch, message brokers like Apache Kafka, or a simple webhook.

Extending Falco's rule set

The output from a Falco rule in text format looks like this, JSON format is also configurable:

```
13:39:51.779518000: Error K8s Secret Get Successfully
↪  (user=system:serviceaccount:kube-system:kubernetes-dashboard
↪  secret=kubernetes-dashboard-key-holder ns=kube-system resource=secrets resp=200
↪  decision=allow reason=)
```

This example event comes from the k8saudit-plug-in and starts with the timestamp of the event, followed by its severity being "Error" in this case. Ordered from the most to the least severe, these are the severities defined in Falco ([DG22], page 21): Emergency, Alert, Critical, Error, Warning, Notice, Informational and Debug. The remainder of the output is the Message explaining what has happened. Here, a Secret has been retrieved which can indicate malicious activity. However, here it is most likely an FP because it is a recurring event initiated by kubernetes-dashboard. Other parameters that give more context about the event are added in parenthesis in the remainder of the Message. Note that the "reason"-field is empty, most likely because it was not set in the request kubernetes-dashboard sent to kubectl.

In addition to the microservices-demo, quite some applications are running in the microk8s-cluster because most of the earlier evaluated tools also still run in the background. Their background activity leads to a constant stream of events Falco alerts on. The majority of the events Falco alerts on are "Notice Unexpected connection to K8s API Server from container" with 350 to 400 occurrences per hour, originating from Grafana accessing kube-apiserver to fetch metrics. Similarly, there are alerts about ServiceAccounts being created and Secrets being accessed by different applications. Of course, my own activity of constantly modifying the cluster to add or update new tools also generates alerts.

The risk of any of these alerts being real intrusions executed by external hackers is low, as none of the lab clusters are directly exposed to the Internet. From my own working experience at mnemonic's Security Operation Center (SOC), I know that this "background noise" of constant activity is usually tuned down, so these alerts do not appear on a human analyst's screen for each detection. In this tuning process, it is important not to tune away too much, as the same alerts can be TPs in the right context. These low-severity alerts should still be kept available because they can give security professionals valuable context in analysing more severe alerts.

Falco offers rulesets with different stability for each detection source like system calls or audit logs. By default, only the currently 25 stable rules are loaded for system calls ([Fal24a]). Two more rulesets are available based on CNCF's maturity level: Currently 29 "sandbox" rules that are very new and not yet deeply tested, and 31 "incubating" rules for candidates to be added to the default "stable" rules ([Fal24a]). This process of testing and continuous evaluation to gradually increase the rule's maturity indicates a high maturity of the Falco project, as new rules are not simply put out in production. The commit history of the Falco rules GitHub repository reveals several updates of the rule set each month ([Fal24a]). By default, Falco only alerts on the first rule that matches an event, so the order in which rule files are evaluated is important. In the current setup, only the stable rules for syscalls and audit logs are used. They are loaded after custom rules in `/etc/falco/rules.d` (line 21 in Listing A.10).

Listing 6.1: Custom rule for Falco detecting when `/etc/hosts` has been accessed

```
1  customRules:
2    rules-poc.yaml: |-
3      - rule: Read Etc Hosts File
4        desc: Detect any attempt to read /etc/hosts file from within a Pod. The
         ↪  /etc/hosts file is crucial for network configuration and its unauthorized
         ↪  access or modification could indicate reconnaissance or malicious activity
         ↪  within the Pod.
5        condition: >
6          evt.type=open and evt.dir=<
7          and fd.name=/etc/hosts
8          and container.id != host
9        output: >
10         Detected read access to /etc/hosts (command=%proc.cmdline
           ↪  container_id=%container.id container_name=%container.name
           ↪  k8s_pod_name=%k8s.pod.name k8s_ns=%k8s.ns.name evt_type=%evt.type
           ↪  fd=%fd.name)
11       priority: INFORMATIONAL
12       tags: [filesystem, container, k8s, mitre_discovery]
```

Listing 6.1 defines a custom Falco rule that detects when the `/etc/hosts` file is read inside a Pod. This activity does not necessarily indicate a cyber attack, it was mostly chosen to create an easily testable PoC of adding a custom rule in Falco. Degioanni et al. give a detailed explanation of how to write and test Falco rules using sysdig in chapter 13 of their book([DG22]), so that process and the rule syntax will not be repeated here. As for the evaluation, I found the syntax to be logical and easily understandable, so it did not take long to write and test the rule in Listing 6.1. Interestingly, it did not trigger when it was first tested from the privileged Pod created in section 5.6.1. This is because for the privileged Pod, the condition in line 8 was not fulfilled since it shares the process namespace of the host, so `container.id == host`.

Running the same `cat /etc/hosts` command from the non-privileged `adservice` Pod in the microservices-demo correctly produced an alert from Falco.

```
17:11:29.613505069: Notice Unexpected connection to K8s API Server from
container (connection=10.1.100.143:50076->10.152.183.1:443 lport=50076
rport=443 fd_type=ipv4 fd_proto=fd.l4proto evt_type=connect user=<NA> …
```

```
17:11:23.014810528: Informational Detected read access to /etc/hosts
(command=cat /etc/hosts container_id=ca96eba3ae0f container_name=server
k8s_pod_name=adservice-77d9f7d7f8-hw4l8 k8s_ns=microservices-demo …
```

```
17:11:14.650248000: Warning Service account created in kube namespace
(user=system:node:microk8s-3 serviceaccount=calico-node
resource=serviceaccounts ns=kube-system)
```

```
17:11:13.070350000: Notice Attach/Exec to pod (user=admin
pod=adservice-77d9f7d7f8-hw4l8 resource=pods ns=microservices-demo action=exec
command=sh)
```

Figure 6.1: Falco alerts in Kibana, including an alert from the custom rule in Listing 6.1

Alerts from different Falco rules in Kibana are shown in Figure 6.1. The first detection belongs to the aforementioned frequent alerts from Grafana polling metrics from kube-apiserver. The second shows how the custom rule from Listing 6.1 looks when it is triggered from the `adserver` Pod in the microservices-demo. In the third alert, the k8s network driver Calico creates a new ServiceAccount. Finally, the fourth event shows how the remote shell to `adservice` was started, from which the custom Falco rule was triggered.

Table 6.1: Rating of Falco following Table 5.1, legend in subsection 5.1.3.

Adaptability: ++	Availability: +	Deployability: ○	Footprint: −
Interoperability: ++	Popularity: ++	Portability: ++	Usability: ++

6.2 Cilium Tetragon: Observability and runtime enforcement

Developed by Cisco's Isovalent, the open-source k8s security observability tool Tetragon is part of the Cilium universe introduced in subsection 5.5.2, but does not require Cilium itself to be deployed ([Tet24b]). It is also based on eBPF and delivers observability and detection capabilities based on Linux kernel events. The project itself is still relatively new and in active development, with the initial commit in the GitHub repository being from March 23$^{\text{rd}}$, 2022. Moreover, it is part of the CNCF at the sandbox maturity level.

All of Tetragon is k8s-aware, so rules and detections automatically know the context of a container, Pod, or service without the need of adding additional k8s metadata. In his blog post from April 29$^{\text{th}}$, 2024, Colvin ([Jer24]) introduces the new features of Tetragon v1.13, including HTTP visibility with data about the request and response, and TLS visibility monitoring TLS versions and weak encryption ciphers. Moreover, the default rule set was now promoted to a stable feature status and includes privilege escalation and kernel protection, network observability, file integrity, OS integrity and much more. However, some of these features including the default ruleset are limited to Tetragon's enterprise version. Nevertheless, many features are available open source as well.

Tetragon was deployed via its Helm-chart in the microk8s-cluster, where it comes up as a DaemonSet with one instance per node similar to Falco. Additionally, its CLI-tool tetra was installed. Contrary to Falco, Tetragon does not provide its own GUI for viewing its events. Therefore, Tetragon was set up to log its events to a folder on the node that is mounted in its Pods. Similar to Cilium Hubble shown in section 5.5.2, these logs are picked up by Filebeat and sent to Elasticsearch via Logstash. For direct comparison with the custom Falco rule in Listing 6.1, a Tetragon Tracing Policy was added that detects when `/etc/hosts` is being accessed. It is a modification of a quickstart example ([Tet23]) and shown in Listing 6.2. As Tetragon's Tracing Policies are k8s resources based CRDs, they are easier to track and update than Falco's rules that are managed in ConfigMaps from which the rules files are generated.

Listing 6.2: Custom rule for Tetragon detecting when `/etc/hosts` has been accessed. Based on [Tet23].

```
apiVersion: cilium.io/v1alpha1
    kind: TracingPolicy
    metadata:
      name: "etc-hosts-read-detection"
    spec:
      kprobes:
      - call: "security_file_permission"
        syscall: false
        return: true
        args:
        - index: 0
          type: "file"
        - index: 1
          type: "int"
        returnArg:
          index: 0
          type: "int"
        returnArgAction: "Post"
        selectors:
        - matchArgs:
          - index: 0
            operator: "Equal"
            values:
            - "/etc/hosts"      # Focus on /etc/hosts
```

In Figure 6.2, the output from both detection rules triggered by running `cat /etc/hosts` in the adservice-Pod from the microservices-demo is shown in Kibana. The Tetragon Tracing Policy triggers twice because it is not restricted to only read-access like the Falco rule. For this reason, it is also triggered by static events on other Pods that do not trigger the Falco rule. Due to timing limitations, no additional time was spent to narrow down the Tetragon Tracing Policy. This PoC of adding custom detection rules for the same event indicates that similar detections are possible in both IDSs.

Just like it was observed for Falco, deploying Tetragon automatically triggers some log events based on static activity in the cluster. Most of these events originate from Elasticsearch processes. Some others come from Velero running checks in the background. Even though they contain a lot of k8s metadata, it is not as clear from the logs why exactly these events trigger. For the custom Tetragon Tracing Policy, a `policy` field is automatically set, which contains the

@timestamp	data_stream.dataset	message
Jul 10, 2024 @ 11:33:36.993	kubernetes-ids-falco	{"uuid":"f31eff92-7054-454f-8d57-90f367cee26e","output":"09:33:35.980328845: Informational Detected read access to /etc/hosts (command=cat /etc/hosts container_id=b556b267f763 container_name=server k8s_pod_name=adservice-664667ddfc-25d8v k8s_ns=microservices-demo evt_type=open fd=/etc/hosts) …
Jul 10, 2024 @ 11:33:36.899	kubernetes-ids-tetragon	{"process_kprobe":{"process":{"exec_id":"bWljcm9rOHMtMzoxNDQ5OTI1NTMyNzgwMjk6MjA1ODA3OQ==", "pid":2058079, "uid":1000, "cwd":"/app", "binary":"/bin/cat", "arguments":"/etc/hosts", "flags":"execve clone", "start_time":"2024-07-10T09:33:35.979530714Z", "auid":4294967295, "pod":{"namespace":"microservices-demo", …
Jul 10, 2024 @ 11:33:36.899	kubernetes-ids-tetragon	{"process_kprobe":{"process":{"exec_id":"bWljcm9rOHMtMzoxNDQ5OTI1NTMyNzgwMjk6MjA1ODA3OQ==", "pid":2058079, "uid":1000, "cwd":"/app", "binary":"/bin/cat", "arguments":"/etc/hosts", "flags":"execve clone", "start_time":"2024-07-10T09:33:35.979530714Z", "auid":4294967295, "pod":{"namespace":"microservices-demo", …

Figure 6.2: Events from Tetragon and Falco triggering on read access to `/etc/hosts` in Kibana

descriptive rule name `etc-hosts-read-detection`. However, this field is not set for the other events Tetragon triggers on by default. In that sense, Falco's logs were easier to understand.

Isovalent's free online "Getting Started with Tetragon" course ([Tet24a]), released on April 8th, 2024, was conducted to examine more of Tetragon's capabilities. Similar to subsection 5.6.1, the course ([Tet24a]) makes use of privileged Pods to simulate intrusions and demonstrate how Tetragon can detect attacks. In their example, even a malicious Python script entirely running in the RAM and never creating a file in the Pod's file system can be detected ([Tet24a]). Contrary to Falco, Tetragon supports enforcing and blocking certain system call events, so in the next step it is demonstrated how the script's execution can be blocked with a specific policy. According to the documentation ([Tet24b]), Tetragon can also be configured for network intrusion detection, but due to timing limitations, this could no longer be evaluated.

Table 6.2: Rating of Cilium Tetragon following Table 5.1, legend in subsection 5.1.3.

Adaptability:	++	Availability:	+	Deployability:	○	Footprint:	−
Interoperability:	++	Popularity:	++	Portability:	++	Usability:	++

6.3 SNORT and Suricata: Network Intrusion Detection

As mentioned in point 2 of Falco's capabilities, it is not natively capable of inspecting the contents of network packets, greatly limiting its capabilities specifically as a k8s network IDS. No up-to-date open-source project was found that is specifically targeted towards intrusion detection on network communication in k8s clusters. Therefore, this section discusses whether the two most widely used open-source network IDSs SNORT and Suricata could be feasible to use and other ideas for this specific scenario.

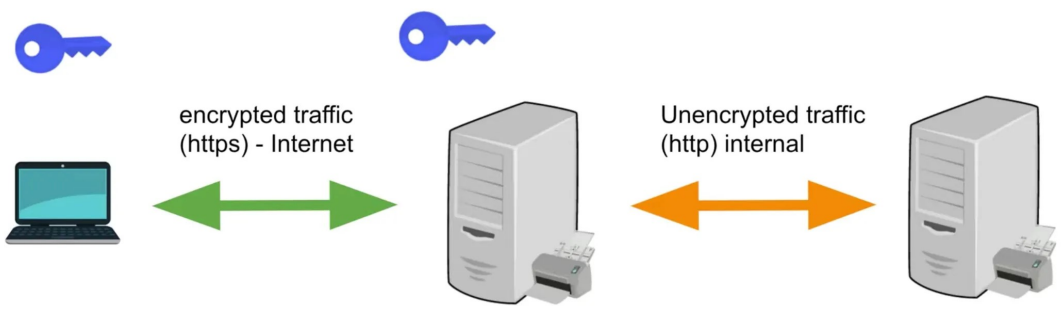


Figure 6.3: At the terminating proxy in the middle, the unencrypted network traffic can be inspected by an IDS. Figure from https://harshityadav95.medium.com/tls-termination-proxy-3733b69680cb

Originally started by Roesch in 1998, SNORT ([Cis24]) was the first open-source IDSs for inspection of network traffic in on-premises networks. In 2013 it was acquired by the networking company Cisco. It is based on a terminating proxy, allowing it to inspect external Transport Layer Security (TLS) encrypted network traffic between the on-premises servers and the Internet in clear text and alert on its network packet contents, see Figure 6.3. It can be deployed without this terminating proxy, too. However, the intrusion detection capabilities are very limited in this case because encrypted traffic cannot be decrypted, so network packet contents cannot be inspected for detection.

Suricata ([Ope24a]) was started as an alternative to SNORT by the Open Information Security Foundation (OSIF) and is deployed in a similar way with a terminating proxy. Both SNORT and Suricata follow a similar syntax for their signatures and have many other resemblances, too. Nonetheless, SNORT is limited to a single thread and thus can only make use of one CPU core at a time. Suricata was designed for multi-threading using multiple CPU cores simultaneously from the beginning, making it more performant when the volume of traffic is high.

Intrusion detection on the network traffic in k8s clusters encounters some specific challenges:

1. **More internal traffic**: In a monolithic application, the different components mostly communicate internally inside the application using different mechanisms without involving the network. When it is split up into microservices, all this internal communication between the components goes via the network internally in the cluster.
2. **Less external traffic**: Contrary to a monolith application, a smaller portion of the total network traffic in a k8s cluster goes in and out of it to the external world on the Internet.
3. **Encrypted internal traffic with mutual Transport Layer Security (mTLS)**: Different solutions like a service-mesh or extended capabilities of k8s network drivers like Calico and Cilium can be used to activate mTLS encrypting the internal traffic between microservices. To allow the decryption of this traffic to analyse its contents, it must be redirected through a terminating proxy server or analysed in the endpoints, here mostly Pods, before encryption or after decryption.
4. **Specific k8s metadata for granular detection**: Internal traffic has different characteristics than external traffic and thus requires specific detection rules. Like in Falco, the events must be enriched with k8s metadata like the associated container, Pod, and Service name. Similar traffic can be aggregated based on this metadata so that the IDS can see the "greater picture".

Points 1 and 2 illustrate that a great proportion of the network traffic in a k8s cluster is internal. Only relying on classic detection techniques focussing on external traffic to and from the Internet would consequently not suffice. Redirecting this internal traffic through a central proxy with an IDS like SNORT or Suricata would greatly limit the network throughput and add delays that limit the performance of all applications running in the cluster. One possible solution could be tapping on the k8s network driver like Calico or Cilium and make it forward a copy of its traffic to a network IDS. Yet, this tapping is not documented as a feature in common k8s network drivers. Furthermore, this approach would only work well as long as mTLS mentioned in point 3 is not activated to encrypt and protect the internal network communication.

Suricata and SNORT aggregate traffic into so-called flows, such that signatures can detect patterns on a combination of related packets, too. For similar detection to work in k8s, the internal network traffic would have to be enriched with k8s metadata as demonstrated in Falco and the ECK Stack. The traffic could then be aggregated by Pod, node, service, or other criteria. According to point item 4, a specific set of signatures would be required to allow targeted detection on internal traffic that makes use of metadata from the k8s context.

Another approach could be small IDS instances that run distributed in nodes or Pods similar to the service mesh architecture. It would at least solve point 3 as it could see the unencrypted traffic in the endpoints and get k8s metadata from the service-mesh. Nevertheless, as of now, no open-source project has been found that specifically addresses intrusion detection for k8s environment and specifically takes points 1 until 4 into account. Consequently, there also is no public set of signatures to rely on. A significant amount of work would be necessary to adapt an existing IDS like Suricata or a service-mesh to have the required functionality with a dedicated set of signatures. As no open-source tools are available for this use case at the moment, they also could not be evaluated as a PoC. Hence, no evaluation table concludes this section.

6.4 Endpoint Detection & Response (EDR): Node protection

Zero-day vulnerabilities have repeatedly shown that any component of a system can fall victim to a cyber attack, even the most hardened container OS like Fedora CoreOS. The high-level risk analysis in section 3.5 found the OS on k8s master and worker nodes to be the second most important asset to protect in a k8s cluster after containers and Pods. container escape attacks where a container leaves its isolation mechanisms to take control of its node's OS are most likely detectable from the node itself, subsection 5.6.1 demonstrates such a container escape. Other privilege-escalation attacks can also be detected, along with, for instance, Denial of Service (DoS), data exfiltration or insider threats from disloyal or blackmailed employees.

In the last few years, the field of protecting endpoints like servers, workstations, mobile phones, laptops or k8s nodes against cyber-attacks has evolved rapidly: From traditional antivirus detection engines relying on static signatures and file hashes towards full-scale Endpoint Detection and Response (EDR) continually collecting data from the endpoint and examining it for malicious or anomalous patterns ([Tol24]). Modern Endpoint Detection and Response (EDR) solutions still partly use elements from classic antivirus like file hashes and static signatures, but they can run much more complex detection strategies as well.

While they still can suffice for home usage, traditional antivirus engines can no longer alone cope with today's advanced and dedicated attacks on large business networks ([Tol24]). Even though there are some free antivirus engines available for home usage like Avast or AVG, there are almost no good free open-source alternatives. EDR on the other hand is so complicated and needs to cover so many different aspects and attack vectors that there are only commercial, closed-source alternatives from big software companies, some examples: Microsoft Defender for Endpoint, Broadcom VMware Carbon Black, Crowdstrike Falcon and Palo Alto Cortex XDR.

There are no free and open-source EDR alternatives, partly because publishing the signatures and detection mechanisms could make it easier for attackers to find a way to evade them. Getting

access to an EDR for protection is expensive, and the pricetag for a managed service including analysis by security professionals is even higher. All the aforementioned EDR solutions support detection on Linux hosts like k8s nodes. Other than that, they are not specifically targeted for k8s environments since they are designed to only protect the node's OS. If an EDR is already present in a company network, I would advise activating it for the k8s nodes, too. As EDR solutions are expensive and not specifically targeted on k8s, none of them have been installed and evaluated as a PoC. Hence, no evaluation table concluding this section either.

6.5 Kubernetes security as a commercial service

Intrusion detection and other security services specifically tailored for k8s are available from different vendors, for example Wiz, Sysdig, Aqua Security, Palo Alto Prisma Cloud and Tigera. In many cases, these services include the deployment of a k8s security and observability tool in the k8s cluster along with the security professionals analysing the alerts as part of the service. Often, it is impossible to just purchase a license for the tool and internally hire security professionals who analyse the alerts in their own company.

The following challenges were met while trying to evaluate these tools:

1. **Intransparent licensing model**: In general, prices and licensing models are not published, making a direct comparison of different vendors harder. Often, the licensing page only shows a contact form directly redirecting to the sales department.
2. **Request a demo**: Many vendors only offer a button where it says "Request a demo". For Tigera, this button created a test account allowing to evaluate "Calico Cloud" in my own cluster for two weeks. As the microk8s- and OpenStack Magnum clusters are not exposed to the Internet, they could not be connected to "Calico Cloud". At Wiz, the button promises to schedule an appointment with a security professional for a live demo.
3. **Closed source**: Apart from Sysdig publishing Falco with a limited set of open-source signatures, all the other tools these companies offer are closed source. How and what they detect, and how often they update their signatures is thus hard to evaluate.
4. **Varying amount of context**: From my own practical working experience, I remember that the level of information these tools give about the reasoning and context behind an alert varies: Some tools only say that there is a problem without providing any context, whereas others explain the reasoning and context of their alerts, sometimes even the triggering signature is shown.

As a consequence of these challenges, none of the commercial tools were evaluated. The demo request from my student-e-mail at Wiz has not gotten any reply yet. Moreover, the question is if such a demo would help answer this thesis's problem: A demo from a seller would not open the black box of closed source code mentioned in point 3, making it impossible to evaluate how and if cyber attack detections are performed. Consequently, there is also no evaluation table concluding this section because no PoC of a commercial tool could be set up.

Chapter 7
Results, discussion, and conclusion

After the sustainability contribution of this work, this chapter sums up the results and lists their limitations, followed by a note on how this research could be taken further, a discussion of the results including recommendations, and a conclusion. Many aspects have been discussed earlier already, the corresponding sections are linked to not repeat the argumentation from there.

7.1 Contribution to sustainability

This section addresses the contribution of this thesis to reaching the United Nations' Sustainable Development Goals (SDGs) ([Uni24]).

SDG 9 **Build resilient infrastructure**: This thesis focuses on detecting and preventing cyberattacks, which increases the resilience of public infrastructure relying on k8s.

SDG 12 **Responsible consumption and production**: The ratings of tools in this thesis include their footprint, which reflects their consumption of energy and other resources. This allows for choosing more resource-efficient tools. Moreover, the recommendations encourage not to deploy unnecessary tools whose function could be covered elsewhere.

SDG 13 **Climate Action**: Reducing the energy consumption and use of other resources also lowers the climate footprint of the deployed k8s security tools.

SDG 16 **Peace, justice, and strong institutions**: Detecting and preventing cyberattacks in k8s environments helps to combat organised crime as stated in SDG 16.4.

7.2 Summary of Results

In this section, the results obtained in this thesis are summed up.

7.2.1 High-level risk analysis of Kubernetes components

At first, a literature review was carried out to get an overview of the different k8s components and their interactions. The findings from it were summarised in chapter 3, where components in k8s worker nodes were presented in section 3.2 and the k8s control plane components running in master nodes were highlighted in section 3.3. Based on this, a high-level risk analysis of the k8s components was conducted in section 3.4, with the goal of identifying the most important components to protect in a k8s cluster as a guideline for further work on the project.

In section 3.5, the findings from the risk analysis were used to set up a prioritised list of components for further work. It identified containers and Pods as the first priority due to their high variety of possible workloads and because not all of their code is maintained by an organisation like the CNCF. As privilege escalation attacks from containers and Pods can affect and be detected from the node's OS, it has the second priority. The master components kube-apiserver and etcd got the third priority since a compromise of them corresponds to a compromise of the whole k8s cluster. All the remaining k8s components have the fourth and last priority.

7.2.2 Lab environments and a baseline for testing Kubernetes tools

Different ways to deploy a lab environment for testing k8s tools are examined in chapter 4. Section 4.2 shows how GCP's microservices-demo emulating an online shop ([Goo24]) was deployed as a baseline facilitating more realistic testing through simulated user activity. Finally, two lab environments were chosen for further testing in section 4.3, both were running on OpenStack in NTNU's cloud. Primarily, an up-to-date cluster based on Ubuntu Linux's microk8s was used that could run the baseline microservices-demo. It was completed by another cluster based on OpenStack Magnum that simulated a more realistic production k8s cluster but had become incompatible with the baseline microservices-demo due to its outdated k8s version. Cloud environments were mentioned as a third option for testing, however, due to their cost they ended up not being used.

7.2.3 Novel proposal of evaluation criteria for Kubernetes security tools

A novel proposal of an evaluation system for k8s security tools was created in section 5.1 since no fitting standardised evaluation system could be found. It is based on these criteria:

- **Adaptability**: How easily can the tool be modified or extended?
- **Availability**: What does it cost to use this tool?
- **Deployability**: How easily is this tool installed?
- **Footprint**: What additional operational costs come with running this tool?
- **Interoperability**: How well does this tool work and integrate with other tools?
- **Popularity**: Are there few or many others using this tool?
- **Portability**: How likely can this tool be used everywhere?
- **Usability**: How difficult is learning to operate or use this tool?

These eight criteria are evaluated on a non-numeric Likert-based rating scale in five steps shown in Table 5.1 and are independent as they measure different things. Whether a tool is useful was only assessed in a way that tools not regarded useful for reactive k8s security were not evaluated further, other than that this is up to the reader. The same applies to the weighting determining how much importance is given to each criterion, it is up to the reader and varies by use case.

7.2.4 Evaluation of reactive Kubernetes security tools

After an attack has taken place, digital forensics is carried out to figure out what happened and which systems were affected. The initial step required for digital forensics is gathering

Table 7.1: Summary of the ratings for all evaluated tools following Table 5.1, legend in subsection 5.1.3.

Tool \ Criterion	Adaptability	Availability	Deployability	Footprint	Interoperability	Popularity	Portability	Usability
Kubectl	++	++	++	++	++	++	++	++
Kubernetes dashboard	○	++	○	○	+	++	++	++
Grafana and Prometheus	++	+	○	○	++	++	++	++
IDEs for Kubernetes	++	+/○/−	+/− −	+	++	++	++	++
K9s	++	+	++	++	++	+	++	+
KubeView	−	++	○	○	+	−	++	++
Jaeger	++	++	−−	○/−	++	++	++	+
Otterize network mapper	+	++	○	−	++	+	++	++
NeuVector	++	+	○	−	++	++	++	++
Elastic Cloud Stack	++	+	−	−	++	++	++	+
FluentD and Fluent Bit	++	++	○	−	++	++	++	++
Kube-audit-rest	+	++	○	○	++	−	++	+
Calico	++	○	−/○	−	++	++	++	++
Cilium and Hubble	++	+	−/○	−	++	++	++	++
Service meshes / Istio	++	++	−	−−/−	++	++	++	++
Kubeshark	++	−	○	−	++	+	++	++
Adapted vanilla Pods	++	++	○	○	++	++	++	+
Velero	++	++	○	○	+	++	++	++
Falco	++	+	○	−	++	++	++	++
Tetragon	++	+	○	−	++	++	++	++

information from the system. An experienced coworker was interviewed in subsection 5.1.1 to determine which kinds of information are required for digital forensics in k8s environments. He pointed out the following five scenarios based on which k8s security tools were evaluated:

1. **Getting an overview of the cluster** (section 5.2): What kinds of services are running where, and which interdependencies do they have?
2. **Logs** (section 5.4): A central system retaining logs from containers, Pods, and the k8s control plane for a specified retention period.
3. **Networking information** (section 5.5): Collect data about who connected to whom along with the amount and type of data being exchanged.
4. **Remote Access** (section 5.6): To Pods, worker or master nodes to see what is running.
5. **Container disk and memory snapshots** (section 5.7): Back up a container's file system and RAM to allow analysis e.g. in a sandbox.

Additionally, IDSs that can detect and alert on cyberattacks in k8s environments were evaluated in chapter 6. The real quality of the alerts from the IDSs was mostly evaluated qualitatively based on research about them because of limitations of the lab environment and the scope of the thesis. Accordingly, the IDSs were evaluated after the same criteria as the digital forensics tools from chapter 5. Table 7.1 lists all evaluations for the 20 tools that were investigated in detail in this thesis. Some of the other tools not investigated in detail are listed in section 5.8. In section 7.5, these results are discussed including considerations of when to use which tools. Scripts to deploy a PoC of the recommmended tools are attached in the Appendix.

7.3 Limitations of the results

The following limitations (Ls) must be considered when studying the results of this thesis:

L1: **Limited variety of testing environments**: Only two different k8s environments were studied, one based on Ubuntu microk8s and another based on OpenStack Magnum. Almost no tests were run in other clusters based on kubeadm or Rancher, and no tests were run in k8s cloud services like AKS, AWS, or GCP.

L2: **Full cluster-admin permissions**: Both lab clusters were administrated with full administrative permissions in the tests, with no restrictive SecurityPolicies in place. Usually, these permissions are restricted in production clusters for security reasons, see section 5.3.

L3: **No testing on real-world data**: A real-world k8s environment has a very dynamic behaviour e.g. running a web store with real customers. The microservices-demo as baseline is more static in its behaviour and cannot fully simulate these dynamics.

L4: **Impossible to evaluate every possibly relevant tool**: Open-source projects come and go, especially if they are small, making them hard to keep track of even with the assistance of a website like Collabnix's "kubetools" ([Col24]). Hence, there is a risk of not discovering all the relevant tools. There also is no time to evaluate all possible tools.

L5: **Limited access to commercial tools**: Due to budget limitations, no licenses for commercial tools or cloud environments could be purchased. Several providers of commercial tools also did not react on the requests for a demonstration that were sent to them.

L6: **Not all evaluations verified**: Given the limitations of the lab, some criteria like portability could not be evaluated practically other than stating that it worked in the lab. They were assessed based on the principle of the presumption of innocence: If no practical proof or documentation states anything different, assume that it works and evaluate positively.

L7: **No in-depth evaluation of Intrusion Detection Systems (IDSs)**: Due to limitations L3 and in the scope and available time for the thesis, the particular detection capabilities of the assessed IDSs could not be evaluated in depth. Only one additional detection rule was added to the IDSs as a PoC allowing a rough evaluation of how rules are added.

7.4 Future work

In this section, several potential areas for further research and development (Fs) are proposed to build upon the findings and address the limitations identified in section 7.3:

F1: **Increase variety of test environments including cloud providers**: Further tests should be carried out especially in k8s cloud environments like AKS, AWS, and GCP. This would limit the effects of limitations L1, L2, and L6 and strengthen the ratings of e.g. portability and interoperability. Moreover, assumptions about these cloud environments could be verified, like remote access through Pods shown in subsection 5.6.1 and the correct function of the audit log scraper kube-audit-rest from subsection 5.4.3.

F2: **Test on real-world data**: To address limitations L2 and L3, it would be beneficial to test the detection and alerting capabilities of different tools in real-world environments.

F3: **Test commercial tools**: For a complete picture, tests of commercial solutions should be run to address limitation L5 and compare them with their open-source alternatives.

F4: **Properly evaluate the detection capabilities of IDSs**: To address limitation L7, the detection capabilities of the assessed IDSs should be assessed in depth, preferably in combination with future work F2 and F3 to evaluate scenarios like the following:

a) **Alerts on aggregated Falco events**: Falco's detection does not support alerting on series of related events like "Alert if B follows A". However, these kinds of detection could be possible in other IDSs using Falco's logs as input.

b) **Tetragon as k8s network IDS**: Tetragon is documented to be capable of network IDS functionalities, but do they work well for detections on network packet contents?

c) **Combining Falco and Tetragon**: For Figure 6.2, both IDSs were deployed in the same cluster, would it make sense to deploy both for maximum visibility?

Conducting tests to evaluate scenarios like these would reveal the limitations of the IDSs' detection capabilities and the real quality of their detection rulesets.

F5: **Develop detection signatures for more data sources**: Both evaluated IDSs only cover a subset of the possible data sources. For instance, none of them analyses container logs as a source of alerting despite them containing potentially relevant data. More research should be conducted to see what kind of detections are possible based on these data sources. Another data source could be EDR on k8s nodes as discussed in section 6.4.

F6: **Continuously re-evaluate the market**: The world of cyber security is moving fast and continuously evolving. IDSs and other tools require continuous updates and maintenance to detect the most recent attacks and prevent evasion. Tetragon only being two years old demonstrates that new tools are continuously developed whereas others are discontinued, making a continuous re-evaluation of the market necessary to keep up-to-date.

7.5 Discussion of the results

This section discusses the results obtained in this thesis given their limitations.

7.5.1 High-level risk analysis, lab environment and evaluation criteria

High-level risk analysis

Looking back, it helped to start by getting an overview of the different k8s components and prioritising what to focus on, since it also laid the technical foundation for the research that followed. The high-level risk analysis in section 3.4 proposed containers in Pods as the primary

resource to investigate protection for in k8s. These findings strengthen hypothesis H1 from section 1.4 about there being a k8s component standing out as the most important to protect.

Choosing a lab environment

After having established that the homelab cluster on old machines proposed in subsection 4.1.1 would not suffice, further possibilities to establish a lab environment in NTNU's cloud environment OpenStack were investigated. At first, a k8s cluster was deployed OpenStack's built-in solution Magnum, it is presented in subsection 4.1.2. It turned out that Magnum fixes the deployed k8s version depending on the deployed OpenStack-version, and the OpenStack version could not be modified since it is administered by NTNU. Moreover, Magnum locks down the OS on the cluster nodes such that it cannot be upgraded and almost not be modified. When it turned out that it was not possible to install the microservices-demo because of the outdated k8s version, it became clear that the OpenStack Magnum cluster would not suffice either.

Several options like Rancher and kubeadm were considered to set up such a cluster but were found too complicated because I did not have much experience in the area of k8s administration yet. In the end, Canonical's microk8s for Ubuntu was chosen because it was easy to configure for multiple nodes, well documented, and provided an up-to-date k8s version, see subsection 4.1.3. Its limitations became apparent much later: Playing around with the network drivers in section 5.5 irrecoverably destroyed the microk8s-cluster multiple times, requiring a new setup each time. Another problem is that microk8s deploys its own container runtime in a non-default location, requiring a fix documented in section A.7 also making it available in the default location.

Investigating the Kubelet Checkpoint API in subsection 5.7.1 revealed that kubeadm also allows to set up k8s clusters with nodes relatively easily and with more flexibility. Nevertheless, the microk8s-cluster was kept for convenience. As limitation L6 states, no cloud environments were used for testing in the end not only because of the additional costs but also to keep the scope of this thesis manageable. This choice of lab environment also implied limitation L2 given no restrictions in the cluster administration and L3 as there was no real-world data available.

Evaluating Kubernetes security tools with a novel set of criteria

In Table 7.1, the ratings following the evaluation criteria in section 5.1 for all the evaluated k8s security tools are summarised. Altogether, they measure how well a tool can be integrated into a company's workflows. During the process, the criteria only underwent minor adjustments without requiring major changes. Considerations about and explanations of how to interpret these criteria are also given in subsection 5.1.4. Adaptability, availability, deployability, footprint, and popularity were straightforward to evaluate because they were easy to test, measure, or obtain from the documentation. Either the tool is freely available, or it is not. Either the tool has the necessary settings and APIs to use it efficiently and integrate it with others, or it does not, and so on.

It was harder to quantify the remaining criteria interoperability, portability, and usability. The installation worked fine in the lab environment and no incompatible k8s distributions were documented, but does that imply it runs everywhere? Is the documentation really complete, or did it just randomly contain what was needed during the evaluation? Given the limited

timeframe, it was not possible perform in-depth evaluations of these criteria. Instead, as stated in limitation L6, the principle of the presumption of innocence was applied: If no practical proof or documentation states anything different, assume that it works and evaluate positively.

The weighting of the different criteria varies by use case and is, just like the tool's usefulness, up to the reader to decide. Accordingly, the ratings should not be regarded as numeric values and averaged since the criteria measure very different aspects. For each criterion, five different states were assigned to the rating scale, but a ○ or − rating does not necessarily imply that a tool is bad. For example, a −-footprint implies that one Pod is added to each node or each namespace. But that does not necessarily mean that the operational costs increase significantly if these Pods are resource-efficient. Nevertheless, the additional resource consumption in the cluster will be higher if a sidecar is added to each Pod in the cluster (−−), and lower if only a total of one Pod is added to the cluster (○) or no Pod is added to the cluster at all (+/++). Furthermore, the rating scale does not easily reflect small differences. For instance, Fluent Bit evaluated in subsection 5.4.2 gets the same −-rating on footprint as the ECK logging stack evaluated in subsection 5.4.1 because both are deployed at least once per node. Still, Fluent Bit uses more than ten times less resources than the combination of Filebeat and Logstash.

7.5.2 From the degrees of open source to commercial offerings

For many people, the concept of open source implies that volunteers who work free of charge except for donations are the only contributors. This is undoubtedly the case for many smaller open-source projects driven by volunteers in their free time. It is however a widespread misconception that open source necessarily implies non-commercial and that it is impossible to generate profits from. The evaluated IDSs Falco and Tetragon evaluated in chapter 6 are examples of this. Both are released under a Freemium-model, making features available under the Apache-2.0 license that does not even restrict commercial usage like selling on services including the product. The companies behind them, Sysdig and Isovalent, earn money among other things on commercial support and other paid premium features like proprietary detection signatures that are not available open source.

Another common misconception is that open source always incorporates all the freedoms of editing, reuse and republishing. For instance, the Elastic Stack evaluated in subsection 5.4.1 uses a copyleft licensing model. Among other things, it incorporates that the software can be used and installed freely, but if someone wants to resell services based on it to others, *all* the source code required to run that new product must also be published under the same copyleft license. If that is not an option, it is usually possible to pay a company like Elastic Inc. to be allowed to resell their services without publishing all the source code.

Discussing open-source licensing models is not the primary goal of this thesis. Nonetheless, the limitations of open-source or other licenses are important to consider while evaluating k8s security tools for specific use cases. As discussed in section 6.5, several factors made evaluating commercial offerings for k8s security difficult. "Scheduling a demo" usually does not allow an in-depth evaluation, at least when it is a video call not allowing to play around with it. Also, some companies like Wiz did not even react to the request, probably because it was sent from a student e-mail. Still, not getting any reply did not contribute to a good first impression.

A strategy that some, but not all commercial vendors follow to retain their customers long term is vendor lock-in. Apple's universe is a good example of this: All of Apple's devices are automatically connected and synchronised with each other without requiring much setup. Integrating devices from other vendors is often hard or impossible. Consequently, many customers keep purchasing more products from the same vendor that easily fit into their existing portfolio. Vendor lock-in endangers the principles of the free market: Changing the vendor often incorporates significant costs and work, forcing customers to stay even when there are better offerings from competitors and despite being dissatisfied with their current vendor.

Commercial offerings are also often proprietary and thus closed source apart from some exceptions. Accordingly, what they detect or do internally remains a black box and is not easily verifiable. Especially IDSs often require deep access to sensitive data. Knowing which information is accessed and how it is used could contribute to more peace of mind, but this is hard to retrace in proprietary software. Open source solves this challenge, even though some companies still keep parts of their open-source products closed as paid, proprietary add-ons.

Finally, an observation that was made evaluating tools for this thesis are the three GitHub issues that were filed for Falco, kube-audit-rest, and KubeView. Open-source tools can have errors and support is usually community-based only. Consequently, getting support and help for issues can take time. For bigger projects like Falco, premium-support is available as a paid add-on feature.

7.5.3 Tools for Kubernetes administration and remote access

A lot of k8s' inner workings are configured in YAML representations of its resources like Pods or Services. To analyse them and look around, k8s administration tools can be used. The following k8s administration tools were evaluated in this report:

- **Kubectl** (subsection 5.2.1): A very universal tool that can be used for almost any operation on k8s resources to e.g. create, delete or modify them. Moreover, much of the online documentation about k8s relies on it. It is easy to install and very likely to already be present on the k8s security analyst's machine.
- **K9s** (subsection 5.2.5): Kubectl commands can be verbose, and it can take a lot of typing to simply enumerate some resources or forward a port from a Service. The CLI-tool has most of the same capabilities while still being easier to use and requiring much less typing. Navigation is intuitive and additional tools like log-viewers can be added through plug-ins. Custom Resource Definitions (CRDs) are supported natively without requiring additional configuration – few other tools support CRDs so easily. After its discovery, it has been the most used tool by far to get an overview of what is happening in the cluster, debug issues and dig into k8s' inner workings.

Both kubectl and k9s can also be used for remote access to Pods. Remote access to worker and master nodes is easiest to achieve via Linux's built-in SSH-tool. If that is not possible, for instance, due to restricted access in cloud environments, section 5.6 shows how a node could be accessed through a privileged Pod or Pod with root volume mounts to its node. Note that this access through Pods only has a limited subset of the capabilities of full SSH access to a node.

7.5.4 Tools for networking and mapping out the cluster

Kubernetes network drivers

Network drivers interacting with the k8s CNI play an integral role in facilitating the inner workings of k8s, as almost all communication in k8s is network-based. For this reason, exchanging the network driver for a cluster with thousands of intertwined workloads running is a risky and time-consuming task that needs to be properly tested beforehand. When different k8s network drivers were evaluated during the work on this thesis, the microk8s-cluster crashed and needed to be rebuilt multiple times when misconfigurations interrupted the flow of network traffic.

For this report, two of the most widely used and feature-rich k8s network drivers were evaluated in section 5.5: Calico and Cilium. Both are feasible choices for high performance in large-scale k8s clusters. The key difference is that almost all of Calico's monitoring and other advanced capabilities are only accessible in its subscription-based Calico Cloud-service, which could not be evaluated in the lab clusters not being exposed to the Internet. Cilium offers a most of these features for free. Calico was the default k8s network driver for a long time, but in recent years Cilium's popularity has been increasing, in most k8s cloud environments it is now the default.

Mapping out the cluster

Especially in bigger k8s clusters, getting an overview of the deployed resources is a necessary first step in digital forensics to retrace how an attack has taken place. Two of the evaluated tools can visualise the relations between k8s resources, e.g. which Pods belong to which Service and so on. KubeView presented in subsection 5.2.6 can show graphical representations of the k8s resources, see Figure 5.7 and Figure 5.8. However, it can only show the most common resource types and has very limited filtering capabilities. K9s assessed in subsection 5.2.5 can provide similar information with its x-ray functionality shown in Figure 5.5. It also supports displaying ServiceAccounts, Secrets, CRDs, and many other resources.

This overview of resources, however, does not reveal the interdependencies of the applications deployed in a k8s cluster. Network graphs can visualise these relations between applications in the cluster. Six of the evaluated tools can display such a graph. For comparability, the referred figures all show the applications in the microservices-demo as shown in Figure 4.5:

- **Jaeger** (subsection 5.2.7): ~ can display network graphs, but requires all applications in the cluster to be instrumented and thus modified to work. Can be used if it is already present, but it must be noticed that uninstrumented services will not show up in the graph.
- **Neuvector** (subsection 5.2.9, Figure 5.12): Many filtering capabilities but for some IPs, the graph does not reveal what they are. Already in this little demo it can be hard to keep the overview in this figure. The thickness of the arrows roughly indicates the traffic amount. After going to its tab, loading the graph is slow and takes multiple seconds. The footprint is significant if this is the only purpose of this tool in the cluster.
- **Kubeshark** (subsection 5.5.4, Figure 5.23): The only network graph that also includes the amounts of traffic being transferred. Footprint is alright, but requires a paid license.
- **Cilum Hubble**: (section 5.5.2, Figure 5.18): As the figure indicates, it is not possible to drag around the entities in Hubble's network graph. Good filtering capabilities.

- **Otterize network mapper** (subsection 5.2.8, Figure 5.11): ~ was designed for this task, and follows another approach: Instead of showing all replicas of communicating Pods, each Service only comes up once. This significantly increases the readability when many replicas are doing the same thing. Also, the footprint is comparably small. The only disadvantage is that there is no GUI, so the graphs must be generated as images from the CLI.
- **Istio service mesh** (subsection 5.5.3, Figure 5.21): This graph is also focused on Services and not showing Pod replicas. Good filtering capabilities, but has the biggest footprint.

Istio and the Otterize network mapper provide the best readability in their graphs when there are many Pod replicas, which is important in big production clusters. Kubeshark includes traffic amounts, which can be handy if the investigation is focused on that.

7.5.5 Tools for monitoring, logging and information gathering

Monitoring and collection of metrics data

The Kubernetes dashboard evaluated in subsection 5.2.2 allows viewing the CPU- and RAM-consumption for nodes and Pods without further customisation at a very limited retention period. Grafana and Prometheus evaluated in subsection 5.2.3 provide highly adaptable and extendable dashboards that can display any sort of metrics for e.g. k8s components. Given sufficient storage capacity, data can be retained for any period of time. Many pre-configured dashboards are available, some are included in the freely available kube-prometheus-stack.

Logging and information gathering

Many different tools are available for gathering and aggregating information from logs and other data sources and retain them for a specified retention period. This period is limited either by the available storage in which the oldest data is deleted when it is filling up, or a timespan like 30 days. Some of these tools also index the data, that is making it quickly searchable in customised query languages. This thesis analysed the freely available Elastic Cloud on Kubernetes (ECK)-stack in subsection 5.4.1 as a representative PoC demonstrating the feasibility of exporting data into such a tool. Similarly, the data could be fed, for instance, into a Security information and event management (SIEM) system that allows alerting on it based on detection rules.

For this thesis, the ECK-stack's ability to split up JSON logs into separate fields greatly helped the analysis of the other tools producing these logs. One of the ECK-stack's major limitations is its high footprint in the pre-processing of logs using Filebeat and Logstash. However, these tools can be replaced with others like Fluent Bit evaluated in subsection 5.4.2 for more efficient processing while still using Elasticsearch for storing and indexing of the log data. Its copyleft license is another limitation since it does not allow to freely resell services relying on it. However, most alternatives like Cisco's Splunk are proprietary with even stricter limitations.

Another observation that was made during the evaluations in this thesis is that all of the different logs did not seem to follow a common standard. Falco's events and k8s metadata looks different than Tetragon's and Hubble's. They all contain information like the namespace and Pod name, but it always gets different names. For example the namespace-field is called `kubernetes.namespace` in Filebeat's container logs, `kubernetes.audit.objectRef.namespace`

in its audit logs, `falco.output_fields.k8s.ns.name` in Falco's event logs, and in Hubble's NetFlow logs it is `hubble.flow.destination.namespace`. For further event aggregation, normalisation of these fields to a common format would be necessary such that all `namespace`-fields have the same name no matter which log or event source they come from. ArcSight's Common Event Format (CEF) ([Arc06]) is an example of a widely used format for this purpose.

7.5.6 Tools for container imaging and backup

The main purpose of tools in this section in the context of reactive k8s security is the possibility to take snapshots of running containers including their RAM and file system. This allows deeper analysis in a sandbox without the attacker noticing. Currently, this functionality is developed for native k8s in the kubelet checkpoint API analysed in subsection 5.7.1. However, yet this feature is in the alpha development stage and thus not ready for production usage. Tests of this feature for this thesis did not manage to produce a working PoC of retrieving a container snapshot. There are backup tools for k8s resources like Velero assessed in subsection 5.7.2 but they do not allow full snapshots of containers including their RAM.

7.5.7 Tools for intrusion detection in Kubernetes

The market of intrusion detection tools alerting on suspicious behaviour in k8s environments is dominated by commercial actors selling k8s security as a service, this was discussed in section 6.5. Consequently, in many cases, it is not possible to only buy a license to the IDS itself without the service of security professionals analysing its alerts. Section 6.3 found the classic network IDSs SNORT and Suricata to most likely not be capable of network intrusion detection in k8s environments because they have not been adapted for the changed base conditions like mostly internal traffic and k8s context awareness yet. There are only two bigger free and open-source IDSs specifically for k8s: Sysdig's Falco assessed in section 6.1 and Isovalent's Tetragon evaluated in section 6.2. Both are based on the eBPF technology keeping their footprints relatively small.

Falco originally only focused on system calls as an event source, later k8s audit logs were added. Natively, Falco is not suitable as an IDS for network events because it cannot inspect their contents. Event correlation where alerts are generated based on a series of events also is not possible by design as events are analysed one by one. Default rulesets are available for both data sources delivering basic detection, access to extended premium rulesets can be purchased. Tetragon is still relatively new and has extended k8s context aware detection capabilities on among other things system calls from Pods and network traffic. Compared to Falco, the features that are included open source are very limited, for example, there is no free detection ruleset.

As mentioned in limitation L7, the particular detection capabilities of these IDSs could not be evaluated in depth. To test the feasibility of this process as a PoC, a new detection rule was added to both IDSs that managed to detect read access to the `/etc/hosts`-file. My personal impression was that the syntax for the Falco-rules was easier to understand and modify than Tetragon's. The log lines from Falco's alerts were also easier to categorise and put in context, even though both contain a lot of relevant k8s metadata.

7.5.8 Wrapping up Kubernetes security tools: Recommendations

A multitude of k8s security tools are available, differing in function, quality and licensing model. Apart from the question of how useful a tool can be, companies wanting to improve their reactive k8s security posture would also ask practical questions like these: Which capabilities do we already have in our current software portfolio? Which kind of technology stack do the employees already know? Which synergy effects can there be from combining some new products with existing ones? All these questions address the basic conditions for improving the reactive k8s security posture and the answers greatly vary from company to company. This weakens hypothesis H2 from section 1.4 since there most likely is no one-size-fits-all technology stack for reactive k8s security that would work equally well for everyone.

It often makes sense to keep tools if they are already present and the employees know and like them, so a cost-benefit analysis should be conducted before replacing them with others. Each additional product adds overhead at least in terms of maintenance and an increased footprint, so they should be chosen carefully. Each subsection below represents a reactive k8s security need. My first recommendation is that for each need, a tool that can address it should be made available to security professionals. The recommendations of tools below should be interpreted as: "If no other tool is already present that can provide similar functionality, I, the author of this thesis, would recommend tool X based on observations from this thesis". These recommendations do not neglect the possibility of achieving similar or better results with other tools.

Tools for Kubernetes administration and remote access

Kubectl (subsection 5.2.1) is most likely already installed and recommended because a lot of documentation is based on it and it has a wide range of features. Additionally, I would recommend k9s (subsection 5.2.5) because of its small footprint and wide range of features. It helps to get an overview quickly and see how k8s resources are related to each other. Both tools can be used for remote access to Pods. Of the available IDEs, I would recommend Lens. However, I would not recommend deploying an IDE only for a security analyst's k8s administration and remote access needs because of their significant footprint, but it can be used if it is already present.

Tools for Kubernetes networking

I would recommend the network driver Cilium with its diagnosis tool Hubble (subsection 5.5.2) because of its performance and free observability capabilities like the export of NetFlow logs. This does not imply the necessity to exchange an existing network driver since it probably can provide similar features. If it is already present, a service mesh (subsection 5.5.3) can and should be used for reactive k8s security features to unfold its full potential. However, due to its significant footprint and added complexity, I would not recommend deploying it only for reactive security features. Finally, I would recommend Kubeshark (subsection 5.5.4) for network traffic analysis and forensics as its additional costs are considered low enough for many applications.

Tools to map out relations between applications deployed in Kubernetes

In many cases, already deployed tools suffice to draw a network graph. If a new tool was to be installed, I would recommend the Otterize network mapper (subsection 5.2.8) because it is

free, has a small footprint, and gives a clean graph focused on Services and not including Pod replicas. As Figure 5.11 indicates, the graph remains readable even for the whole microk8s-cluster. Service meshes like Istio (subsection 5.5.3) can provide similar graphs with even more fine-grained control, but have a significantly larger footprint. As it is mostly focused on proactive k8s security features and has a significant footprint, I would not recommend deploying Neuvector (subsection 5.2.9) only for network graphs. If the focus is on analysing where how big amounts of traffic go, I would recommend the paid Kubeshark (subsection 5.5.4) having a relatively small footprint and including a network graph with exactly that information.

Tools for monitoring, logging and gathering of information

I would recommend using Prometheus and Grafana shown in subsection 5.2.3 to administrate and display metrics because of their high popularity and customizability. The pre-configured kube-prometheus-stack can be used as a simple basis of configuration if Prometheus and Grafana are not present elsewhere yet. If further research following future work F1 confirms its function in k8s cloud environments like AKS, AWS, or GCP, kube-audit-rest presented in subsection 5.4.3 can be recommended to fetch k8s audit logs more cheaply than by using cloud provider API calls. Moreover, I would recommend getting a central information gathering and retention system like the ECK-Stack. However, as there are so many competitors and the requirements greatly vary, no particular tool is recommended over another in this case.

Tools for container imaging and backup

No ready-to-use tool is available to take snapshots of running containers including their RAM and file system. Nonetheless, it is important for reactive k8s security and digital forensics, so I recommend keeping track of changes in the area and evaluate the kubelet checkpoint API presented in subsection 5.7.1 once ready for production or similar new tools coming up.

Tools for intrusion detection in Kubernetes

As a consequence of limitation L7, the assessed IDSs were mostly evaluated based on how well they can be integrated into a company's workflows, practical in-depth evaluations of their detection capabilities were not conducted. From the impressions gained by adding a new detection rule and evaluating the built-in detection capabilities based on the static activity from the microservices-demo baseline, Falco evaluated in section 6.1 has a much wider built-in set of detection rules than Tetragon assessed in section 6.2 and seems easier to learn. However, in its premium version Tetragon promises to have more detection capabilities, for example, on network packet contents which are not possible with Falco.

Future work F4 in section 7.4 suggests conducting further research allowing to evaluate the detection capabilities of Falco and Tetragon properly. Since this research has not been carried out yet, no clear recommendation in favour of Falco, Tetragon, or a commercial tool is given. Nevertheless, based on the limited research conducted here, Falco appears to be a good starting point since it incorporates the most capabilities for free. Even if the premium rulesets for Falco and Tetragon were purchased, it is likely that some additional signatures would have to be developed that specifically target vulnerabilities in a given k8s environment. To facilitate that, the difficulty of learning the respective signature syntaxes would have to be assessed.

Furthermore, future work F5 suggests evaluating how other data sources like container logs could be integrated into an IDS for k8s to facilitate more detections and context to the alerts.

7.6 Conclusion

This thesis aimed to identify how intrusion detection can be done in Kubernetes (k8s). In the beginning, the different k8s components were analysed to answer the first research question RQ1 about identifying the most important assets to protect in k8s. Based on a high-level risk analysis of the k8s components, it was concluded to prioritise the protection of Pods closely followed by the operating system on k8s nodes for further work. These findings strengthened hypothesis H1 about there being a k8s component standing out as the most important one to protect.

The second research question RQ2 asked to evaluate if there is a tool available for intrusion detection in k8s and if not to evaluate what would be necessary to implement it. It quickly turned out that there were multiple tools available, so lab environments and a set of evaluation criteria were set up enabling a comparison of these tools. In this framework, 20 tools providing different reactive k8s security functionalities were evaluated. Initially, tools were evaluated that support digital forensics and analysis of the context of security alerts in k8s. Finally, the two available open-source IDSs Falco and Tetragon were assessed theoretically based on the principles of cybersecurity, but their detection capabilities could not be evaluated in-depth. Conducting large-scale tests using real-world data offers exciting opportunities for further research.

For each of the reactive k8s security needs k8s administration, mapping out the resources and applications in the cluster, monitoring and logging, networking, container imaging and backup, and IDS, one or multiple tools have been recommended in the end. Since the base conditions determining which tools fit best into a company's existing portfolio greatly vary from company to company, these recommendations are only meant as a coarse indication in case no already deployed tool can provide the same functionality equally well. This finding weakens hypothesis H2 because there most likely is no one-size-fits-all optimal technology stack for reactive k8s security, every company will have to find their own based on their individual needs. Given the complexity and number of involved developers for the two evaluated open-source IDSs, completely developing a full new k8s IDS from scratch as suggested in RQ2 appears to be a giant task that is not easily feasible. Moreover, reactive k8s security needs to cover so diverse aspects that no single freely available tool can cover them all as RQ2 suggests. In any case, multiple tools are required.

To build further on this research, more tests should be conducted on real-world data and in other k8s environments like the cloud providers AKS, AWS, and GCP to improve the universality of the results. Moreover, due to budget limitations almost no commercial offerings could be evaluated, so they should be assessed in the future for a more complete picture. Finally, the detection capabilities of Falco and Tetragon should be evaluated in-depth to allow a proper comparison of their potentials. All in all, this thesis answers the question of how intrusion detection could be done in k8s environments in a well-founded manner, also including many other relevant aspects concerning reactive k8s security. Many of the results are reproducable by deploying the attached scripts for a PoC installation of all the recommended tools in the Appendix.

References

[Ali24] Alibaba Open Source. *KubeSkoop: Network monitoring & diagnosis suite for Kubernetes*. URLs: https://kubeskoop.io/ and https://github.com/alibaba/kubeskoop. Apr. 2024. (Last visited: July 9, 2024).

[Aly+24] Abdelrahman Aly, Mahmoud Fayez, et al. "Multi-Class Threat Detection Using Neural Network and Machine Learning Approaches in Kubernetes Environments". In: *2024 6th International Conference on Computing and Informatics (ICCI)*. Mar. 2024, pp. 103–108.

[AM18] Omar Al-Debagy and Peter Martinek. "A Comparative Review of Microservices and Monolithic Architectures". In: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. Nov. 2018, pp. 149–154.

[Ama24] Amazon Web Services (AWS). *What is a Service Mesh?* 2024. URL: https://aws.amazon.com/what-is/service-mesh/ (last visited: July 7, 2024).

[Ana22] Ankit Anand. *Jaeger vs SigNoz - Taking distributed tracing to the next level*. June 2022. URL: https://signoz.io/blog/jaeger-vs-signoz/ (last visited: June 14, 2024).

[Aqu21] Aqua Security. *Service Mesh: Architecture, Concepts, and Top 4 Frameworks*. Sept. 2021. URL: https://www.aquasec.com/cloud-native-academy/container-security/service-mesh/ (last visited: July 7, 2024).

[Arc06] ArcSight. *Common Event Format: Event Interoperability Standard*. 2006. URL: https://raffy.ch/blog/wp-content/uploads/2007/06/CEF.pdf (last visited: July 19, 2024).

[Axe00] Stefan Axelsson. "The base-rate fallacy and the difficulty of intrusion detection". In: *ACM Trans. Inf. Syst. Secur.* 3.3 (Aug. 2000), pp. 186–205. ISSN: 1094-9224. URL: https://doi.org/10.1145/357830.357849.

[Ban21] Shay Banon. *Doubling down on open, part II*. Jan. 2021. URL: https://www.elastic.co/blog/licensing-change (last visited: June 22, 2024).

[Ber23] Daniel Berman. *Fluentd vs. fluent bit: Side by side comparison*. Sept. 2023. URL: https://logz.io/blog/fluentd-vs-fluent-bit/ (last visited: June 25, 2024).

[BG23] Stephen J. Bigelow and Alexander S. Gillis. *What are microservices?: Definition and guide from TechTarget*. Oct. 2023. URL: https://www.techtarget.com/searchapparchitecture/definition/microservices (last visited: Mar. 13, 2024).

[Bri24] Bridgecrew. *Checkov: Policy-as-code for everyone*. URLs: https://www.checkov.io/ and https://github.com/bridgecrewio/checkov. July 2024. (Last visited: July 9, 2024).

[Bru20] Luca Bruno. *Orchestrating and monitoring OS auto-updates - DevConf.CZ 2020*. Mar. 2020. URL: https://www.youtube.com/watch?v=0jW1CTt1zdQ (last visited: Mar. 28, 2024).

[Cab19] William Caban. "High Availability". In: *Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams*. Berkeley, CA: Apress, 2019, pp. 31–54. ISBN: 978-1-4842-4985-7. URL: https://doi.org/10.1007/978-1-4842-4985-7_2.

[Can24] Canonical. *Microk8s - cis hardening and assesment: Microk8s*. 2024. URL: https://microk8s.io/docs/how-to-cis-harden (last visited: June 27, 2024).

[Car19] Eric Carter. *Sysdig 2019 container usage report: New kubernetes and security insights*. Oct. 2019. URL: https://sysdig.com/blog/sysdig-2019-container-usage-report/ (last visited: Mar. 20, 2024).

[Cil24a] Cilium Authors. *Cilium: eBPF-based Networking, Observability, Security*. URLs: https://cilium.io and https://github.com/cilium/cilium. 2024. (Last visited: July 7, 2024).

[Cil24b] Cilium Authors. *ConfigMap showing how Helm values are translated into Cilium config parameter names*. July 2024. URL: https://github.com/cilium/cilium/blob/main/install/kubernetes/cilium/templates/cilium-configmap.yaml (last visited: July 8, 2024).

[Cis24] Cisco. *Snort – Network Intrusion Detection and Prevention System*. 2024. URL: https://snort.org/ (last visited: July 4, 2024).

[Col22] Ben Coleman. *Benc-UK/kubeview: Kubernetes Cluster Visualiser and Graphical Explorer*. Nov. 2022. URL: https://github.com/benc-uk/kubeview (last visited: June 19, 2024).

[Col24] Collabnix. *Kubetools - curated list of Kubernetes Tools*. May 2024. URL: https://github.com/collabnix/kubetools (last visited: May 29, 2024).

[Con24] Containerd Authors. *Containerd: An industry-standard container runtime with an emphasis on simplicity, robustness and portability*. 2024. URL: https://containerd.io/ (last visited: July 6, 2024).

[CRI23a] CRI-O Project Authors. *cri-o – Lightweight container runtime for Kubernetes*. 2023. URL: https://cri-o.io/ (last visited: July 6, 2024).

[CRI23b] CRIU Authors. *CRIU: Checkpoint/Restore In Userspace*. Nov. 2023. URL: https://criu.org (last visited: July 6, 2024).

[CSW20] Nacha Chondamrongkul, Jing Sun, and Ian Warren. "Automated Security Analysis for Microservice Architecture". In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. Mar. 2020, pp. 79–82.

[Cyb23] Cybersecurity and infrastructure security agency (CISA). *Cybersecurity advisory – Threat actors exploiting Ivanti EPMM vulnerabilities: CISA*. Aug. 2023. URL: https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-213a (last visited: June 13, 2024).

[Dao20] Minh Dao. *How container lifespan affects observability*. Apr. 2020. URL: https://thenewstack.io/how-container-lifespan-affects-observability/ (last visited: Apr. 3, 2024).

[Dat24] Datadog. 2024. URL: https://vector.dev/ (last visited: June 26, 2024).

[Deg14] Loris Degioanni. *Announcing Sysdig: a system exploration tool*. Apr. 2014. URL: https://sysdig.com/blog/announcing-sysdig/ (last visited: June 30, 2024).

[DG22] Loris Degioanni and Leonardo Grasso. *Practical cloud native security with Falco*. eng. Sebastopol, CA: O'Reilly Media, Aug. 2022.

[DHV23] G. Darwesh, J. Hammoud, and A.A. Vorobeva. "A novel approach to feature collection for anomaly detection in Kubernetes environment and agent for metrics collection from Kubernetes nodes". eng. In: *Nauchno-tekhnicheskiĭ vestnik informatsionnykh tekhnologiĭ, mekhaniki i optiki* 23.3 (2023), pp. 538–546. ISSN: 2226-1494.

[Doc24] Docker Inc. *Manage data in Docker*. Feb. 2024. URL: https://docs.docker.com/storage/ (last visited: Mar. 14, 2024).

[Dod02] Gordana Dodig-Crnkovic. "Scientific methods in computer science". In: *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*. sn. 2002, pp. 126–130.

[DSB23] Giorgio Dell'Immagine, Jacopo Soldani, and Antonio Brogi. "KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments". eng. In: *Future internet* 15.7 (2023), p. 228. ISSN: 1999-5903.

[Ear23] Alan R. Earls. *What is a container image?: Definition from TechTarget*. Jan. 2023. URL: https://www.techtarget.com/searchitoperations/definition/container-image (last visited: Mar. 14, 2024).

[Ela24] Elastic NV. *ECK: Elastic Cloud on Kubernetes*. June 2024. URL: https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-overview.html (last visited: June 22, 2024).

[Ern20] Eric Ernst. *EFK using fluent-bit and the Elastic Operator*. June 2020. URL: https://gist.github.com/egernst/d8f20021db724ba831a2552ba02027fe (last visited: June 24, 2024).

[Fal24a] Falco Authors. *Falco Rules Overview*. URLs: https://falcosecurity.github.io/rules/ and https://github.com/falcosecurity/rules/. June 2024. (Last visited: July 4, 2024).

[Fal24b] Falco Authors. *Falco: Cloud Native Runtime Security*. URLs: https://falco.org/ and https://github.com/falcosecurity/falco. June 2024. (Last visited: June 30, 2024).

[Fan20] Zeng Fansong. *Getting started with kubernetes | etcd*. June 2020. URL: https://www.alibabacloud.com/blog/getting-started-with-kubernetes-%7C-etcd_596292 (last visited: Apr. 10, 2024).

[Fas24] Alexander Fashakin. *Kubernetes alternatives 2024: Top 8 container orchestration tools*. Jan. 2024. URL: https://www.servertribe.com/kubernetes-alternatives/ (last visited: Mar. 14, 2024).

[Fed24] Fedora Project. *Fedora CoreOS - The container optimized OS*. Mar. 2024. URL: https://fedoraproject.org/coreos/ (last visited: Mar. 28, 2024).

[Flu24a] Fluent Bit Authors. *Fluentd & Fluent bit*. Mar. 2024. URL: https://docs.fluentbit.io/manual/about/fluentd-and-fluent-bit (last visited: June 25, 2024).

[Flu24b] Fluentd Project. *Fluentd: Open source data collector for unified logging layer*. 2024. URL: https://www.fluentd.org/ (last visited: June 25, 2024).

[Gal24a] Fernand Galiana. *k9s: Kubernetes CLI To Manage Your Clusters In Style!* 2024. URL: https://k9scli.io/ (last visited: June 11, 2024).

[Gal24b] Fernand Galiana. *k9sα Kubernetes CLI To Manage Your Clusters In Style!* 2024. URL: https://k9salpha.io/ (last visited: June 11, 2024).

[Goo24] Google Cloud Platform. *GoogleCloudPlatform/microservices-demo: Sample Cloud-first application with 10 microservices showcasing Kubernetes, Istio, and grpc.* May 2024. URL: https://github.com/GoogleCloudPlatform/microservices-demo (last visited: May 19, 2024).

[Gra24a] Grafana Labs. *Grafana: The Open Observability Platform*. May 2024. URL: https://grafana.com/ (last visited: June 6, 2024).

[Gra24b] Grafana Labs. *Grafana/Loki: Like Prometheus, but for logs.* URLs: https://github.com/grafana/loki/tree/main and https://grafana.com/oss/loki/. June 2024. (Last visited: June 24, 2024).

[Gup21a] Amit Gupta. *Kubernetes Security and Observability – A Holistic Approach to Securing Containers and Cloud Native Applications*. 1st Ed. O'Reilly, Nov. 2021. ISBN: 978-1-098-10711-6.

[Gup21b] Oshi Gupta. *ETCD in Kubernetes*. Dec. 2021. URL: https://cloudyuga.guru/blogs/etcdk8s/# (last visited: Apr. 10, 2024).

[HAA22] Muhammad Faraz Hyder, Waqas Ahmed, and Maaz Ahmed. "Toward deceiving the intrusion attacks in containerized cloud environment using virtual private cloud-based moving target defense". eng. In: *Concurrency and computation* 35.5 (Jan. 2022). ISSN: 1532-0626.

[Hei23] Martin Heinz. *A collection of Docker images to solve all your debugging needs*. Sept. 2023. URL: https://martinheinz.dev/blog/104 (last visited: June 27, 2024).

[HJ20] Kaizhe Huang and Pranjal Jumde. *Learn Kubernetes Security: Securely orchestrate, scale, and manage your microservices in Kubernetes deployments*. eng. 1st ed. Birmingham: Packt Publishing, 2020. ISBN: 978-1-83921-650-3.

[Ist24] Istio Authors. *Istio: Getting Started*. Version 1.22.2. May 2024. URL: https://istio.io/latest/docs/setup/getting-started/ (last visited: July 7, 2024).

[Jae24] Jaeger Authors. *Open source, distributed tracing platform*. 2024. URL: https://www.jaegertracing.io/ (last visited: June 14, 2024).

[Jai24] Raunaik Jain. *What is software evaluation, & how to do it effectively?* May 2024. URL: https://testsigma.com/blog/software-evaluation/ (last visited: June 5, 2024).

[Jan20] Nana Janashia. *Master DevOps & Cloud with Nana*. 2020. URL: https://www.techworld-with-nana.com/ (last visited: Feb. 23, 2024).

[JCB11] Mike Jackson, Steve Crouch, and Rob Baxter. "Software evaluation: criteria-based assessment". In: *Software Sustainability Institute* 1 (2011).

[Jer24] Jeremy Colvin. *Isovalent Enterprise for Tetragon 1.13: Kubernetes Identity Aware Policies, Default Rulesets, HTTP and TLS Visibility, and More!* May 2024. URL: https://isovalent.com/blog/post/isovalent-enterprise-for-tetragon-1-13/ (last visited: July 8, 2024).

[Jon22] Alex Jones. *What is Microk8s?* Feb. 2022. URL: https://www.youtube.com/watch?v=GsS38peaM64 (last visited: May 19, 2024).

[Jos+15] Ankur Joshi, Saket Kale, et al. "Likert scale: Explored and explained". In: *British journal of applied science & technology* 7.4 (2015), pp. 396–403.

[Kae24] Steve Kaelble. *Kubernetes Security For Dummies, Wiz Special Edition*. For Dummies. John Wiley & Sons, Inc., 2024. ISBN: 978-1-394-24588-8.

[Kar+23] Buket Karakaş et al. "Enhancing Security in Communication Applications Deployed on Kubernetes: Best Practices and Service Mesh Analysis". MA thesis. 2023.

[Kil23] Aaron Kili. *Explanation of "Everything is a File" and Types of Files in Linux*. Apr. 2023. URL: https://www.tecmint.com/everything-is-file-and-types-of-files-linux/ (last visited: June 26, 2024).

[Kra24] Diderik Kramer. "Service mesh and eBPF security services in cluster networks". In: *NTNU TTM4905 Master Thesis* (June 2024).

[Kre20] John Kreisa. *Docker index: Dramatic growth in Docker usage affirms the continued rising power of developers*. July 2020. URL: https://www.docker.com/blog/docker-index-dramatic-growth-in-docker-usage-affirms-the-continued-rising-power-of-developers/ (last visited: Mar. 14, 2024).

[Kub23] Kubernetes Authors. *Kubelet Checkpoint API*. URLs: https://kubernetes.io/docs/reference/node/kubelet-checkpoint-api/ and https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/. Dec. 2023. (Last visited: July 6, 2024).

[Kub24a] Kubernetes Authors. *Command line tool (kubectl)*. Jan. 2024. URL: https://kubernetes.io/docs/reference/kubectl/ (last visited: June 3, 2024).

[Kub24b] Kubernetes Authors. *Creating a cluster with kubeadm*. URLs: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/ and https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/. Feb. 2024. (Last visited: July 6, 2024).

[Kub24c] Kubernetes Authors. *Deploy and access the kubernetes dashboard*. Apr. 2024. URL: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/ (last visited: June 3, 2024).

[Kub24d] Kubernetes Authors. *Kubernetes Components*. Jan. 2024. URL: https://kubernetes.io/docs/concepts/overview/components/ (last visited: Apr. 11, 2024).

[Kub24e] Kubernetes Authors. *SIG CLI: Guides and API References for Kubectl and Kustomize*. 2024. URL: https://kubectl.docs.kubernetes.io/ (last visited: May 27, 2024).

[Kub24f] Kubeshark Authors. *Kubeshark – API traffic analyser for Kubernetes*. 2024. (Last visited: July 9, 2024).

[Kub24g] Kubeshark Authors. *Kubeshark PCAP dumper (Capturing raw traffic) and Traffic recorder*. URLs: https://docs.kubeshark.co/en/raw_traffic_capture and https://docs.kubeshark.co/en/traffic_recorder. 2024. (Last visited: July 11, 2024).

[Kum22] Atul Kumar. *Container (Docker) vs Virtual Machines (VM): What Is The Difference?* Sept. 2022. URL: https://k21academy.com/docker-kubernetes/docker-vs-virtual-machine/ (last visited: Mar. 14, 2024).

[Kum23] Suraj Kumar. *What is kubernetes? discover the top 3 kubernetes trends on the rise in 2023*. Mar. 2023. URL: https://www.kellton.com/kellton-tech-blog/what-is-kubernetes-discover-top-kubernetes-trends (last visited: June 17, 2024).

[Lum18] Jake Lumetta. *Should you start with a monolith or microservices?: Nordic apis*. Jan. 2018. URL: https://nordicapis.com/should-you-start-with-a-monolith-or-microservices/ (last visited: Mar. 7, 2024).

[Mic24] Microsoft. *Visual studio code - Code editing. Redefined*. 2024. URL: https://code.visualstudio.com/ (last visited: June 10, 2024).

[Mir24] Mirantis. *Lens: The Kubernetes IDE*. 2024. URL: https://k8slens.dev/ (last visited: June 6, 2024).

[Mne23] Mnemonic. *Threat advisory: Ivanti Endpoint manager mobile (EPMM) authentication bypass vulnerability (CVE-2023-35078)*. URLs: https://www.mnemonic.io/resources/blog/ivanti-endpoint-manager-mobile-epmm-authentication-bypass-vulnerability/ and https://www.mnemonic.io/resources/blog/threat-advisory-remote-file-write-vulnerability-in-ivanti-epmm/. July 2023. (Last visited: June 18, 2024).

[Mon18] MongoDB. *Server Side Public License (SSPL)*. Version 1. Oct. 2018. URL: https://www.mongodb.com/legal/licensing/server-side-public-license (last visited: June 22, 2024).

[Neu23] Lukas Neuenschwander. "Intrusion Detection in Kubernetes – A comprehensive study of tools and techniques". In: *NTNU TTM4502 Specialization Project* (Nov. 2023).

[Neu24] NeuVector Authors. *NeuVector: Full Lifecycle Container Security*. URLs: https://www.suse.com/products/neuvector/ and https://github.com/neuvector/neuvector. June 2024. (Last visited: July 9, 2024).

[Ngu+20] Xuan Nguyen et al. "Network isolation for Kubernetes hard multi-tenancy". MA thesis. 2020.

[Oct23] Octant Authors. *Octant: Visualize your Kubernetes workloads*. 2023. URL: https://octant.dev/ (last visited: June 14, 2024).

[Ogd24] Cody Ogden. *Google Graveyard - Killed by Google*. May 2024. URL: https://killedbygoogle.com/ (last visited: June 1, 2024).

[Ope23] Open Source labs beta. *Oslabs-beta/klusterview: Get instant insights on your kubernetes clusters with our lightweight, plug-and-play performance monitoring tool*. 2023. URL: https://github.com/oslabs-beta/KlusterView (last visited: June 14, 2024).

[Ope24a] Open Information Security Foundation (OSIF). *Suricata – Observe. Protect. Adapt.* 2024. URL: https://suricata.io/ (last visited: July 4, 2024).

[Ope24b] OpenStack Foundation. *Magnum User Guide*. Mar. 2024. URL: https://docs.openstack.org/magnum/latest/user/index.html#supported-versions (last visited: May 19, 2024).

[Ott24] Otterize. *Ottorize network mapper*. URLs: https://docs.otterize.com/reference/configuration/network-mapper, https://github.com/otterize/network-mapper and https://github.com/otterize/otterize-cli. June 2024. (Last visited: July 9, 2024).

[Pal23] Palo Alto Networks. *2023 State of Cloud Native Security Report*. Mar. 2023. URL: https://www.paloaltonetworks.com/content/dam/pan/en_US/assets/pdf/reports/state-of-cloud-native-security-2023.pdf (last visited: Jan. 29, 2024).

[Pro24a] Prometheus Authors. *Overview: Prometheus*. 2024. URL: https://prometheus.io/docs/introduction/overview/ (last visited: June 6, 2024).

[Pro24b] Prometheus-Community. *Helm-Charts for kube-prometheus-stack*. May 2024. URL: https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack (last visited: June 6, 2024).

[Put21] Anton Putra. *You must use these 6 kubernetes tools!!! (kubectx, Kubens, Kube-PS1, K9s, Popeye, stern)*. Nov. 2021. URL: https://www.youtube.com/watch?v=xw3j4aNbHgQ (last visited: June 11, 2024).

[Ram23] Ajay Ramamoorthy. *Software evaluation checklist: How to evaluate software requirements*. July 2023. URL: https://www.spendflo.com/blog/software-assessment-checklist (last visited: June 5, 2024).

[Reb22] Adrian Reber. *Forensic container checkpointing in Kubernetes*. Dec. 2022. URL: https://kubernetes.io/blog/2022/12/05/forensic-container-checkpointing-alpha/ (last visited: July 6, 2024).

[Ree+21] Michael Reeves, Dave Jing Tian, et al. "Towards Improving Container Security by Preventing Runtime Escapes". In: *2021 IEEE Secure Development Conference (SecDev)*. Oct. 2021, pp. 38–46.

[RK24] Adrian Reber and Kubernetes Authors. *GitHub activity around container checkpointing in the Kubernetes and containerd repositories*. URLs: https://github.com/containerd/containerd/pull/10365, https://github.com/kubernetes/enhancements/issues/2008 and https://github.com/containerd/containerd/pull/6965. June 2024. (Last visited: July 6, 2024).

[RPW19] Akond Rahman, Chris Parnin, and Laurie Williams. "The Seven Sins: Security Smells in Infrastructure as Code Scripts". eng. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 164–175. ISBN: 9781728108698.

[Rud22] Eldad Rudich. Feb. 2022. URL: https://github.com/eldadru/ksniff (last visited: July 9, 2024).

[Sgh22] Abdellfetah Sghiouar. *Why you should NOT use Service Mesh*. URLs: https://medium.com/google-cloud/when-not-to-use-service-mesh-1a44abdeea31 and https://www.youtube.com/watch?v=qwSGI89RHt8. Jan. 2022. (Last visited: July 7, 2024).

[Ste24] Stern. *Stern/stern: Multi pod and container log tailing for Kubernetes – Friendly Fork of https://github.com/wercker/stern*. May 2024. URL: https://github.com/stern/stern (last visited: June 25, 2024).

[Sti24] Stiftung Warentest. *Stiftung Warentest: An introduction to Stiftung Warentest*. May 2024. URL: https://www.test.de/aboutus/ (last visited: June 2, 2024).

[Sys21] Sysdig. *Kubernetes Security Guide*. Guide-001 Rev. C. Sysdig, 2021.

[Tet23] Tetragon Authors. Oct. 2023. URL: https://github.com/cilium/tetragon/blob/main/examples/quickstart/file_monitoring.yaml (last visited: July 10, 2024).

[Tet24a] Tetragon Authors. *Getting started with Tetragon*. Apr. 2024. URL: https://isovalent.com/labs/tetragon-getting-started/ (last visited: July 8, 2024).

[Tet24b] Tetragon Authors. *Tetragon: eBPF-based Security Observability and Runtime Enforcement*. URLs: https://tetragon.io and https://github.com/cilium/tetragon. 2024. (Last visited: July 8, 2024).

[Thu18] Rob Thubron. *Tesla's cloud account hacked to mine cryptocurrency*. Feb. 2018. URL: https://www.techspot.com/news/73376-tesla-cloud-account-hacked-mine-cryptocurrency.html (last visited: Apr. 3, 2024).

[Tig24] Tigera. *Introduction to Kubernetes Networking and Security*. San Francisco, CA, USA, 2024. URL: https://www.tigera.io/lp/kubernetes-networking-ebook/ (last visited: May 9, 2024).

[Tin20] Macdara Tinney. "Intrusion Detection for Kubernetes Based Cloud Deployments". In: *Master's thesis at the University of Dublin* (May 2020).

[Tol24] Shila Toledano. *The Evolution Of Cyber Security: From Antivirus To EDR*. Apr. 2024. URL: https://redentry.co/en/blog/evolution-of-cyber-security/ (last visited: July 4, 2024).

[Twe24] Richard Tweed. *RichardoC/kube-audit-rest: Kubernetes Audit Logging, when you don't control the control plane*. June 2024. URL: https://github.com/RichardoC/kube-audit-rest (last visited: June 26, 2024).

[Uni24] United Nations (UN). *Sustainability Goals*. 2024. URL: https://sdgs.un.org/goals (last visited: July 17, 2024).

[Vai23] Lionel Sujay Vailshery. *Container Orchestration Platform Share Worldwide 2022*. Sept. 2023. URL: https://www.statista.com/statistics/1224681/container-orchestration-usage-share-worldwide-by-platform/ (last visited: Apr. 5, 2024).

[Vel24] Velero Authors. *Velero: Backup and migrate Kubernetes resources and persistent volumes*. 2024. URL: https://velero.io/docs/main/contributions/minio/ (last visited: July 6, 2024).

[Viv23] Vladimir Vivien. *Vladimirvivien/KTOP: A top-like tool for your Kubernetes Clusters*. Sept. 2023. URL: https://github.com/vladimirvivien/ktop (last visited: June 19, 2024).

[Vou+22] Konstantinos Voulgaris, Athanasios Kiourtis, et al. "A Comparison of Container Systems for Machine Learning Scenarios: Docker and Podman". In: *2022 2nd International Conference on Computers and Automation (CompAuto)*. Aug. 2022, pp. 114–118.

[Wea23] Weaveworks. *Weaveworks/scope: Monitoring, Visualisation & Management for Docker & Kubernetes*. 2023. URL: https://github.com/weaveworks/scope (last visited: June 14, 2024).

[WHG21] Katrine Wist, Malene Helsem, and Danilo Gligoroski. "Vulnerability analysis of 2500 docker hub images". In: *Advances in Security, Networks, and Internet of Things: Proceedings from SAM'20, ICWN'20, ICOMP'20, and ESCS'20*. Springer. 2021, pp. 307–327.

[Won22] Terence Wong. *Monoliths versus microservices*. July 2022. URL: https://octopus.com/blog/monoliths-vs-microservices (last visited: Mar. 8, 2024).

[Zac22] Tom Zach. *Jaeger tracing: The ultimate guide*. Feb. 2022. URL: https://www.aspecto.io/blog/jaeger-tracing-the-ultimate-guide/ (last visited: June 14, 2024).

[Zah+23] Ehtesham Zahoor, Maryam Chaudhary, et al. "A formal approach for the identification of redundant authorization policies in Kubernetes". eng. In: *Computers & security* 135 (2023), p. 103473. ISSN: 0167-4048.

[Zdu+23] Uwe Zdun, Pierre-Jean Queval, et al. "Microservice Security Metrics for Secure Communication, Identity Management, and Observability". eng. In: *ACM transactions on software engineering and methodology* 32.1 (2023), pp. 1–34. ISSN: 1049-331X.

Appendix A

Additional figures and technical documentation

A.1 Additional figures

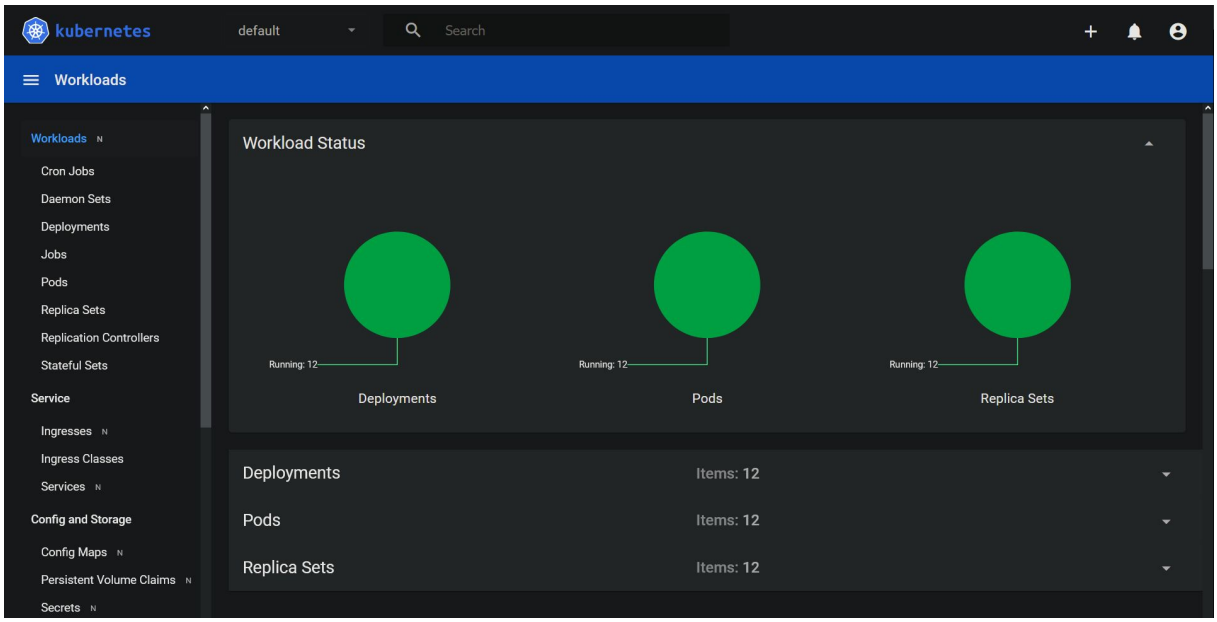


Figure A.1: Workload overview in the Kubernetes dashboard deployed in the microk8s-cluster.

A.2 Code and scripts on GitHub and attached to this file

A description of a PoC showing how to deploy all the recommended tools in a clean microk8s cluster can be found on GitHub: https://github.com/fjellvannet/intrusion-detection-in-kubernetes.

The same code-fragments can also be extracted from this PDF: `.zip`: `.7z`:

If you are not allowed to extract the zip-file, try 7z.

A.3 K9s plugin to improve the readability of JSON logs

K9s plugin to improve the readability of JSON logs. This plugin requires jq to be installed along with k9s. To use it, the following code must be added to `plugins.yaml`, which must be placed in k9s's config directory. The location of the config directory depends on the OS and

is documented here: https://k9scli.io/topics/config/. The logging-functionalities in k9s were further examined after writing this plug-in, the code for a plug-in combining a lot of other extended logging functionality for k9s can be found on GitHub where I suggest to add a new plug-in `logging-ultimate` https://github.com/derailed/k9s/issues/2758 and in the zip-file.

Listing A.1: K9s plug-in to improve the readability of JSON logs

```
plugins:
  jqlogs:
    shortCut: Ctrl-J
    confirm: false
    description: "Logs (jq)"
    scopes:
      - po
    command: bash
    background: false
    args:
    - -c
    - '"$@" | jq --color-output | less -r '
    - dummy-arg
    - kubectl
    - logs
    - $NAME
    - -n
    - $NAMESPACE
    - --context
    - $CONTEXT
    - --kubeconfig
    - $KUBECONFIG
```

A.4 Deployment of the Elastic Cloud on Kubernetes (ECK)

This section demonstrates how the complete ECK stack evaluated in subsection 5.4.1 was deployed in the microk8s-cluster, including the ECK operator, Filebeat fetching audit and container logs and enriching them with k8s metadata, Logstash processing and reformmatting the logs, Elasticsearch storing the logs persistently and Kibana for visualisation. This setup fetches and stores container and audit logs. The YAML-script that is deploying all these resources and referred in the descriptions can be found in Listing A.6.

A.4.1 Deployment of the ECK Operator

Earlier, there were Helm-charts available to deploy the Elastic Stack on k8s, but they have been deprecated[1] in favour of the new ECK Operator. Its Helm-chart deploys CRDs for all the Elastic components. These CRDs simplify the process of setting up the stack: Instead of having to manually set up Pods, Services, and Deployments and configure their interaction, a new Kibana-resource defined in a CRD automatically deploys its Pods and Services and automatically knows how to interact with a Logstash-resource that is also defined in a CRD. Moreover, the Helm-chart deploys the ECK Operator, a Pod that reads the CRDs and ensures that the corresponding Services, Pods and other k8s resources are deployed for e.g. Kibana. It acts similarly to kube-scheduler and kube-controller-manager but specifically for the Elastic CRDs, ensuring that the desired state from the YAML-configuration-files is deployed in the cluster.

[1] Found out that after having spent quite some time trying to set up everything the deprecated way...

Helm-charts are also available to deploy a whole ECK-stack in k8s in different configurations, however, they were found not to satisfy the needs specified in subsection 5.1.1 without further configuration. Therefore, the ECK stack was set up and adapted by manually defining and fine-tuning the required resources in a YAML-file that can be found in Listing A.6 in the Appendix. Section 3.5 concluded with containers and Pods being a likely attack vector, therefore the pipeline is set up to fetch and retain all container logs. Furthermore, audit-logs from the control-plane are gathered according to the recommendations from subsection 5.1.1.

A.4.2 Enable audit logs in microk8s

One requirement that needed to be configured specifically were audit logs about activity in the control plane. In clusters with full access to the control plane like the OpenStack Magnum-cluster or the microk8s-cluster, audit logs can be activated by specifying additional cmd-args to the kube-apiserver. To ensure compatibility and see logs from the microservices-demo, the ECK-stack was set up in the microk8s-cluster. For microk8s, the following additional arguments need to be appended to `/var/snap/microk8s/current/args/kube-apiserver` on all the nodes:

Listing A.2: Additional cmd-args for kube-apiserver in microk8s to activate audit logs from the control plane, must be appended to `/var/snap/microk8s/current/args/kube-apiserver` on all nodes.

```
1 --audit-log-path=/var/log/apiserver/audit.log
2 --audit-log-maxbackup=3
3 --audit-log-maxsize=128
4 --audit-log-maxage=3
5 --audit-policy-file=/var/snap/microk8s/current/args/kube-api-audit-policy-all.yaml
```

Microk8s must be restarted on all nodes to apply the changes. These settings configure `audit.log` as the log-file with up to three old files being kept when the log-file has reached a maximum size of 128 MB and a retention period of three days. Finally, `kube-api-audit-policy-all.yaml` is set as the policy-file, in which the level and kind of audit events to log is configured. It is shown in Listing A.3 and placed in the same directory as the `args/kube-apiserver`-file in which all Pods in microk8s can access it without further configuration.

Listing A.3: Audit policy file `kube-api-audit-policy-all.yaml` enabling logging of all audit-events.

```
1 apiVersion: audit.k8s.io/v1
2 kind: Policy
3 rules:
4   - level: RequestResponse # log everything for all resources
```

A.4.3 Configure Filebeat to fetch and enrich log-files

Filebeat is the first step in the ECK logging pipeline reading the log-files and doing some initial preparations. It is deployed as a DaemonSet, ensuring that one Filebeat-pod is running on each node in the cluster (Listing A.6, line 59). Volume mounts exposing folders from the node on which microk8s is running are required for Filebeat to be able to access the logs. In microk8s these folders were `/var/log/containers`, `/var/log/pods` and `/var/lib/docker/containers`, additionally `/var/log/apiserver` for the auditlogs (Listing A.6, lines 70 to 96).

In the default configuration, the logs only get very little metadata. Additional processors can be added to Filebeat's configuration to enrich the logs with context information from k8s, for

instance, to see which node and Pod a container was running in (from line 35). Filebeat needs to access the k8s-API for the `add_kubernetes_metadata`-processor to work, this access must be configured in an additional ServiceAccount (line 3) with corresponding ClusterRoleBinding (line 8). Moreover, the `add_fields`-processor is used to set the `data_stream.dataset`-field to be able to distinguish where the logs originate from: `kubernetes-audit` or `kubernetes-container` (line 44). As the `add_kubernetes_metadata`-processor only works for container-logs, `kubernetes.node.name` is manually set on audit-logs to backtrack which node they originate from (line 56).

A.4.4 Setting up Logstash for further processing

Even though Filebeat can send its data directly to Elasticsearch, it was decided to deploy Logstash in between for further processing. For redundancy, three instances of Logstash were deployed for redundancy: One on each node (line 103). Just like Filebeat, Logstash needed to explicitly run as root: Not optimal from a security perspective (line 111). Datastreams in Elasticsearch are named based on the following pattern of fields: `<data_stream.type>-<data_stream.dataset>-<data_stream.namespace>`. `Data_stream.type` is automatically set to `logs` and `data_stream.dataset` was already set by Filebeat. Logstash sets `data_stream.namespace` to the k8s-namespace if it is known from the processor, or "unknown" if it is not (line 129).

Container-logs get a lot of metadata from the `kubernetes_add_metadata`-processor, all of which are put in separate fields to be easily searchable. As they do not possess such a processor, audit-logs have little metadata and mostly consist of one big JSON-formatted `message`-field. This `message`-field contains interesting metadata, however, it is not easily searchable deeply inside the JSON. Accordingly, it was decided to use Logstash to unpack the `message`-field into separate fields under `kubernetes.audit` to make its contents more searchable (line 140). Visual examples of the effects of having fields separately accessible and searchable are given in subsection A.4.6.

Unfortunately, it was not as easy as just unpacking the `message`-JSON into separate fields in one step: Sometimes it contains keys like `"."`, which are fine for Logstash, but cannot be indexed by Elasticsearch and are thus rejected. After spending days trying to fix this issue with the assistance of GitHub Copilot and Google, it turned out that recursively renaming all `"."`-keys into `"_."` was not enough: Now Elasticsearch complained about inconsistent data, for instance, trying to put a subkey in a column that had earlier been defined to take strings only[2].

A deeper analysis of the data causing the error revealed that only subkeys of `requestObject` and `responseObject` were inconsistent and had fields like `"."`. That makes sense – there are no standards for how the data in HTTP-requests and responses sent to APIs must look like. Both problems, the `"."` and inconsistency, could be resolved by collecting the fields under `requestObject` and `responseObject` back into a JSON-string. Like that, no matter what it contains, the logs remain consistent because `requestObject` and `responseObject` are always assigned a String if they are present. A custom Ruby filter was constructed that takes all the subkeys of `requestObject` or `responseObject`, and collects them in a JSON-string (line 142). As this only affects the subkeys of these two fields, most metadata remains accessible and searchable in separate fields.

[2] Elasticsearch is strict about the data type in each column. Usually, it is set automatically when the first log-line is parsed. A value with a different, incompatible type from another log line will not be accepted.

A.4.5 Persistent storage in volumes for Elasticsearch

Elasticsearch is the "heart of the free and open Elastic Stack" and centrally stores data for "fast search, fine-tuned relevancy and powerful analytics that scale with ease" ([Ela24]). For all the other tools evaluated until now, the volatility of Pods was accepted: Once a Pod is terminated, all of its data is gone, and a reboot leads to a completely blank slate. For Elasticsearch, this makes little sense: To be able to retain data, it must be able to store it persistently without filling up the disk of the node VM. Just like for etcd (see subsection 3.3.1), at least three Elasticsearch-nodes are necessary for High Availability (HA) (line 207).

NTNU provided access to particularly fast and responsive Solid State Drive (SSD)-volumes referred to as SSD-4K, which allow up to 4000 Input/Output (IO) operations per second. Three separate 20 GB SSD-4K-volumes were created in OpenStack and initialized with an EXT4-filesystem, one for each node in the microk8s-cluster. Native OpenStack-volumes are only available in the OpenStack Magnum-cluster, microk8s does not support them directly. Therefore, the volumes were mounted at `/mnt/elasticsearch` on the node-VMs, and then they were made accessible for Elasticsearch via volume claims (line 208). A specific storage class called `elasticsearch` was created for these volumes based on the microk8s hostpath class (line 189).

As only 20 GB of storage is available for each node, an aggressive Index Lifecycle Management (ILM) policy was defined in Listing A.4. New data is stored in a separate new index when the current one is at least three days old (line 8) or exceeds four GB (line 9), this process is called "rolling over" the index. Old indices are deleted when they are at least four days old (line 14). After that, the ILM policy needs to be applied to the indices, this is shown in Listing A.5. The code from Listing A.4 and Listing A.5 needs to be run in the Elasticsearch API, this can be done using the "Dev Tools"-pane in Kibana.

Listing A.4: Elasticsearch Index Lifecycle Management (ILM) policy

```
1  PUT _ilm/policy/4_days_delete_policy
2  {
3    "policy": {
4      "phases": {
5        "hot": {
6          "actions": {
7            "rollover": {
8              "max_age": "3d",
9              "max_size": "4GB"
10           }
11         }
12       },
13       "delete": {
14         "min_age": "4d",
15         "actions": {
16           "delete": {}
17         }
18       }
19     }
20   }
21 }
```

Listing A.5: Applying the Elasticsearch ILM template to indices starting with `logs-*`

```
1  PUT _index_template/logs
2  {
3    "index_patterns": ["logs-*"],
4    "data_stream": {},
5    "template": {
6      "settings": {
7        "number_of_shards": 1
8      },
9      "mappings": {
10       "properties": {
11         "@timestamp": {
12           "type": "date"
13         },
14         "message": {
15           "type": "text"
16         }
17       }
18     }
19   },
20   "priority": 200
21 }
```

A.4.6 Visualisation and data access with Kibana

Kibana is the GUI frontend for the Elastic Stack allowing to "run data analytics and speed and scale for observability, security, and search" ([Ela24]). With the help of the CRDs, the YAML-configuration is simple (Listing A.6, line 227). Elasticsearch is referenced simply by its name and it is not necessary to manually specify ports, usernames, passwords and Secrets to connect it. Figure A.2 shows the container-logs from the last hour in Kibana:

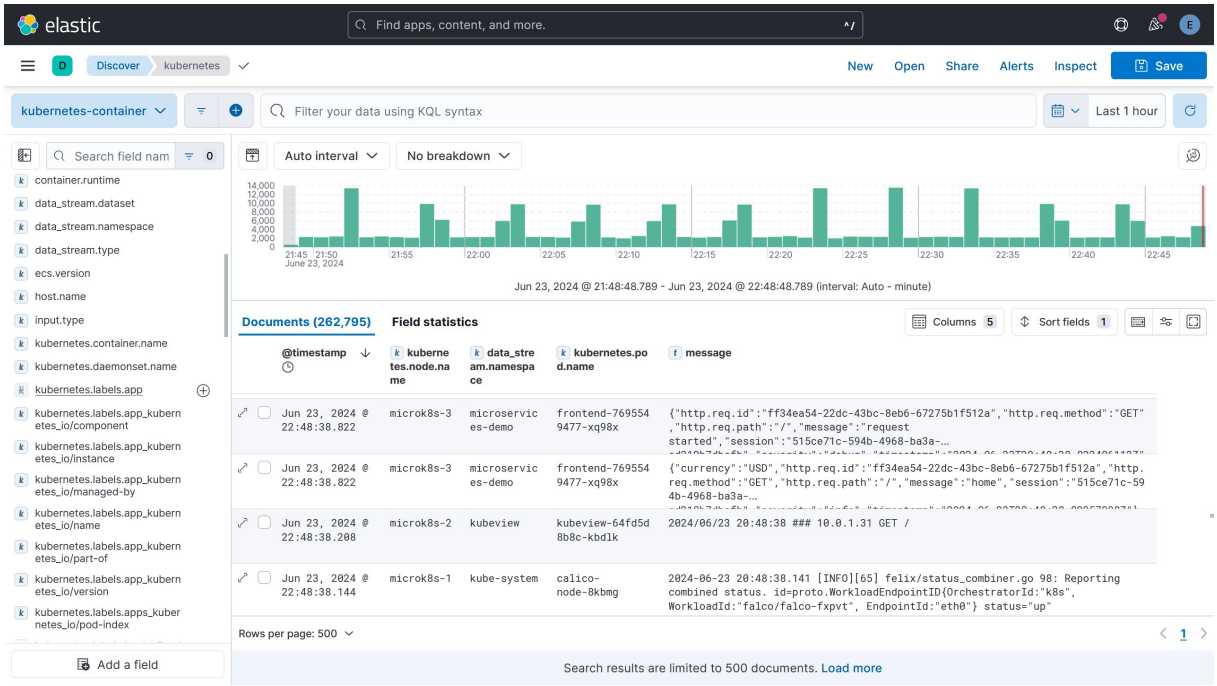


Figure A.2: Kibana on the microk8s-cluster showing container-logs from the last hour

On the left-hand side of Figure A.2, the `data_stream`-fields determining which datastream the logs are sent to can be seen. Note also the large number of fields starting with `kubernetes.`, they all originate from the `add_kubernetes_metadata`-processor in Filebeat and contain information like the Pod's name or k8s namespace which would not be accessible without it. The figure has been zoomed in to keep the font readable, therefore only four log-lines are visible: Two from the front-end in the microservices-demo, one from KubeView and one from Calico, the network driver in microk8s. Unlike the audit-logs, the container-logs originate from independent applications returning logs in different formats: The first two log-lines are in JSON-format, but the other two are not. Consequently, the message-field was only unpacked into subfields for the audit-logs, resulting in fields similar to the `kubernetes.`-fields in Figure A.2. Separate fields can be more easily queried using Kibana's query language KQL and shown as separate table-columns.

A dataview must be created in the Discovery tab in Kibana to view the logs like in Figure A.2. It is a selection of datastreams being named after the schema indicated in the Logstash section. Datastreams are an abstraction of multiple indices, which Elasticsearch uses internally to store its data. Three dataviews were created: `kubernetes-audit` selecting `logs-kubernetes-audit-*`, `kubernetes-container` selecting `logs-kubernetes-container-*` and `kubernetes-all` for audit and container-logs together with the selection `logs-kubernetes-*`.

The timeline in Figure A.2 indicates some peaks in the number of logs coming in in the last hour, but over a longer period of time like 24 hours, the number of logs coming in remains pretty stable as shown in Figure A.3. That makes sense, as there is no real user activity in the cluster, and all the logs originate from applications running in the background like the microservices-demo.

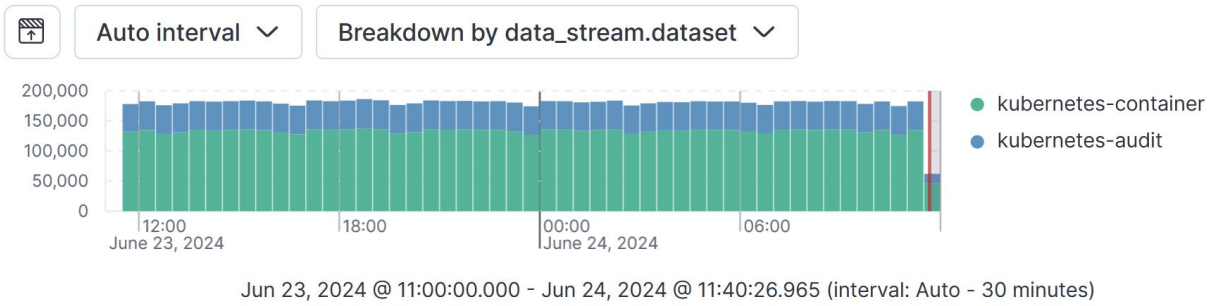


Figure A.3: Kibana on the microk8s-cluster showing that the total volume of audit- and container-logs remained relatively constant during the last 24 hours.

A.4.7 Deployment script for Elastic Cloud on Kubernetes

Deployment YAML-script for the Elastic Cloud on Kubernetes (ECK) stack, with Filebeat fetching logs from all the containers and Logstash enrichment with k8s metadata. A ServiceAccount is included allowing the enrichment with k8s metadata in Filebeat. At first, the ECK operator must be installed using Helm. The script can then be applied using `kubectl apply -f`. Note that the persistent volume mounts must be prepared on the microk8s nodes, so make sure they are mounted in `/mnt/elasticsearch`.

Listing A.6: Deployment YAML-script for the Elastic Cloud on Kubernetes (ECK)

```
# the service accocunt and cluster role binding for the filebeat are required, otherwise the
↪  filebeat will not be able to access the kubernetes API and the enrichment
↪  add_kubernetes_metadata will not work
apiVersion: v1
kind: ServiceAccount
metadata:
  name: elastic-beat-filebeat
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: elastic-beat-autodiscover-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: elastic-beat-autodiscover
subjects:
- kind: ServiceAccount
  name: elastic-beat-filebeat
  namespace: elastic-system
---
apiVersion: beat.k8s.elastic.co/v1beta1
kind: Beat
metadata:
  name: filebeat
```

```yaml
spec:
  type: filebeat
  version: 8.14.1
  config:
    incluster: true
    filebeat.inputs:
      - type: container
        paths:
          - /var/log/containers/*.log
        data_stream:
          dataset: kubernetes-container
        processors:
        - add_kubernetes_metadata:
            host: ${NODE_NAME}
            matchers:
            - logs_path:
                logs_path: '/var/log/containers/'
        - add_fields:
            target: data_stream
            fields:
              dataset: kubernetes-container
      - type: log
        paths:
          - /var/log/apiserver/audit.log
        processors:
        - add_fields:
            target: ""
            fields:
              data_stream:
                dataset: kubernetes-audit
              kubernetes:
                node:
                  name: ${NODE_NAME}
    output.logstash:
      hosts: ["logstash-ls-beats:5044"]
  daemonSet:
    podTemplate:
      spec:
        dnsPolicy: ClusterFirstWithHostNet
        serviceAccount: elastic-beat-filebeat
        automountServiceAccountToken: true
        hostNetwork: false
        securityContext:
          runAsUser: 0
        containers:
        - name: filebeat
          volumeMounts:
          - name: varlogcontainers
            mountPath: /var/log/containers
          - name: varlogpods
            mountPath: /var/log/pods
          - name: varlibdockercontainers
            mountPath: /var/lib/docker/containers
          - name: varlogapiserver
            mountPath: /var/log/apiserver
          env:
```

```
          - name: NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
        volumes:
        - name: varlogcontainers
          hostPath:
            path: /var/log/containers
        - name: varlogpods
          hostPath:
            path: /var/log/pods
        - name: varlibdockercontainers
          hostPath:
            path: /var/lib/docker/containers
        - name: varlogapiserver
          hostPath:
            path: /var/log/apiserver
---
apiVersion: logstash.k8s.elastic.co/v1alpha1
kind: Logstash
metadata:
  name: logstash
spec:
  count: 3
  version: 8.14.1
  elasticsearchRefs:
    - clusterName: eck
      name: elasticsearch
  podTemplate:
    spec:
      securityContext:
        runAsUser: 0
      containers:
        - name: logstash
          resources:
            limits:
              memory: 2Gi
  pipelines:
    - pipeline.id: main
      config.string: |
        input {
          beats {
            port => 5044
          }
        }
        filter {
          if [data_stream][dataset] == "kubernetes-container" {
            if [kubernetes][namespace] {
              mutate {
                replace => { "[data_stream][namespace]" => "%{[kubernetes][namespace]}" }
              }
            } else {
              mutate {
                replace => { "[data_stream][namespace]" => "unknown" }
              }
            }
```

```
          }
          else if [data_stream][dataset] == "kubernetes-audit" {
            json {
              source => "message"
              target => "[kubernetes][audit]"
            }
            ruby {
              init => "
                require 'json'
                def process_audit_data(event)
                  ['requestObject', 'responseObject'].each do |field|
                    obj = event.get('[kubernetes][audit][' + field + ']')
                    if obj.is_a?(Hash) || obj.is_a?(Array)
                      json_str = obj.to_json
                      event.set('[kubernetes][audit][' + field + ']', json_str)
                    end
                  end
                end
              "
              code => "
                process_audit_data(event)
              "
            }
            if [kubernetes][audit][objectRef][namespace] {
              mutate {
                add_field => { "[data_stream][namespace]" =>
                ↪  "%{[kubernetes][audit][objectRef][namespace]}" }
              }
            } else {
              mutate {
                add_field => { "[data_stream][namespace]" => "unknown" }
              }
            }
          }
        }
        output {
          elasticsearch {
            hosts => [ "${ECK_ES_HOSTS}" ]
            user => "${ECK_ES_USER}"
            password => "${ECK_ES_PASSWORD}"
            ssl_certificate_authorities => "${ECK_ES_SSL_CERTIFICATE_AUTHORITY}"
          }
        }
  services:
    - name: beats
      service:
        spec:
          type: ClusterIP
          ports:
            - port: 5044
              name: "filebeat"
              protocol: TCP
              targetPort: 5044
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```yaml
metadata:
  name: elasticsearch
provisioner: microk8s.io/hostpath
reclaimPolicy: Retain
parameters:
  pvDir: /mnt/elasticsearch
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
  name: elasticsearch
spec:
  version: 8.14.1
  nodeSets:
    - name: default
      count: 3
      volumeClaimTemplates:
        - metadata:
            name: elasticsearch-data
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 20Gi
            storageClassName: elasticsearch
      podTemplate:
        spec:
          containers:
            - name: elasticsearch
              resources:
                limits:
                  memory: 3Gi
---
apiVersion: kibana.k8s.elastic.co/v1
kind: Kibana
metadata:
  name: kibana
spec:
  version: 8.14.1
  count: 1
  elasticsearchRef:
    name: elasticsearch
```

A.5 Deployment of Fluent Bit logging into ECK

This section shows how Fluent Bit presented in subsection 5.4.2 was deployed in the microk8s-cluster to parse container logs and store them in Elasticsearch from the ECK-stack shown in section A.4. At first it was tried to deploy Fluentd which is available as a community microk8s plug-in that can be enabled using `microk8s enable fluentd`. This plug-in also deploys Elasticsearch and Kibana-instances, but it did not work as expected: Fluentd went into a loop of crashing in its container and being restarted, and the Kibana-instance was not accessible via

port forwarding. There are also Helm-charts available for both tools, but they do not interact with the ECK-deployment as easily as its native components with CRDs. Thus, access to Elasticsearchfigurations must be set up manually.

Ernst's GitHub Gist ([Ern20]) provided a good starting point for setting up Fluent Bit with the ECK Stack with manual component definitions and no Helm-chart. Being four years old, some modifications were necessary to adapt the code to the current setup and requirements, the updated code can be found in Listing A.7 below. Like for Filebeat, a ServiceAccount (line 3), cluster role (line 7), and ClusterRoleBinding (line 18) are set up to provide Fluent Bit access to the k8s API for enrichment of the logs with k8s-metadata. Fluent Bit's Pods are set up as a DaemonSet, ensuring one Pod on each node in the cluster (line 31). The connection details for Elasticsearchas environment variables which are set in the Pod (line 60), the password is extracted directly from the Secret that stores it for the ECK deployment (line 69). Additionally, the directories containing logs are mounted as volumes, together with ConfigMaps that are converted into filter configuration files (line 91) and Lua scripts (line 94).

Firstly, the general configuration file for Fluent Bit is generated, where the log-level etc. are set up (line 116). Then, the container-log-files are set up as input (line 131) along with a filter that adds k8s-metadata (line 142). For consistency with the other logs from the ECK Stack, a custom filter file (line 155) was added to rename the field containing the log-message from "log" to "message" (line 159). Also, the `data_stream` fields (see subsection A.4.3) were not set automatically. Setting `data_stream.dataset` in the filter resulted in an additional field `data_stream_dataset`, because Ernst's deployment ([Ern20]) contains an additional output filter replacing dots in field names with underscores (line 177). Removing `Replace_Dots` in the output was not an option, as it resulted in other errors sending data to Elasticsearch.

Thus, the built-in filters could not be used to set the `data_stream` fields. However, more complex filter operations can be defined in Lua scripts similar to Logstash's Ruby filters, so a Lua filter was added (line 160). It runs the script `set_dataset_and_namespace()` defined in another ConfigMap (line 235). The Elasticsearch output also needed to be adjusted. To get datastreams in which Elasticsearch handles indices, `Logstash_Format` and `Logstash_Prefix` were deactivated, and the `Index` property was set to `logs-kubernetes-fluent-bit` (line 166). According to GitHub Copilot, dynamically setting the index-name based on the `kubernetes.namespace_name` field is possible, but complicated to implement. Therefore, all the container logs from Fluent Bit arrive in the same index, instead of separate indices per namespace as configured in Logstash in the ECK Stack.

Listing A.7: Deployment YAML-script for Fluent Bit logging into ECK's Elasticsearch

```
1  apiVersion: v1
2  kind: ServiceAccount
3  metadata:
4    name: fluent-bit
5  ---
6  apiVersion: rbac.authorization.k8s.io/v1
7  kind: ClusterRole
8  metadata:
9    name: fluent-bit-read
10 rules:
11 - apiGroups: [""]
```

```
  resources:
  - namespaces
  - pods
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: fluent-bit-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: fluent-bit-read
subjects:
- kind: ServiceAccount
  name: fluent-bit
  namespace: default
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  labels:
    k8s-app: fluent-bit-logging
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit-logging
      version: v1
  template:
    metadata:
      labels:
        k8s-app: fluent-bit-logging
        version: v1
        kubernetes.io/cluster-service: "true"
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "2020"
        prometheus.io/path: /api/v1/metrics/prometheus
    spec:
      containers:
      - name: fluent-bit
        image: fluent/fluent-bit:latest
        imagePullPolicy: Always
        ports:
          - containerPort: 2020
        env:
        - name: FLUENT_ELASTICSEARCH_HOST
          value: "elasticsearch-es-http"
        - name: FLUENT_ELASTICSEARCH_PORT
          value: "9200"
        - name: FLUENT_ELASTICSEARCH_USER
          value: "elastic"
        - name: FLUENT_ELASTICSEARCH_PASSWORD
```

```yaml
          valueFrom:
            secretKeyRef:
              name: elasticsearch-es-elastic-user
              key: elastic
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: fluent-bit-config
          mountPath: /fluent-bit/etc/
        - name: lua-scripts
          mountPath: /fluent-bit/etc/lua
          readOnly: true
      terminationGracePeriodSeconds: 10
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
      - name: fluent-bit-config
        configMap:
          name: fluent-bit-config
      - name: lua-scripts
        configMap:
          name: fluent-bit-lua-scripts
      serviceAccountName: fluent-bit
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      - operator: "Exists"
        effect: "NoExecute"
      - operator: "Exists"
        effect: "NoSchedule"
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  labels:
    k8s-app: fluent-bit
data:
  # Configuration files: server, input, filters and output
  # ======================================================
  fluent-bit.conf: |
    [SERVICE]
        Flush               1
        Log_Level           info
        Daemon              off
        Parsers_File        parsers.conf
        HTTP_Server         On
        HTTP_Listen         0.0.0.0
```

```
124         HTTP_Port           2020
125
126       @INCLUDE input-kubernetes.conf
127       @INCLUDE filter-kubernetes.conf
128       @INCLUDE output-elasticsearch.conf
129       @INCLUDE filters.conf
130
131     input-kubernetes.conf: |
132       [INPUT]
133           Name                tail
134           Tag                 kube.*
135           Path                /var/log/containers/*.log
136           Parser              docker
137           DB                  /var/log/flb_kube.db
138           Mem_Buf_Limit       5MB
139           Skip_Long_Lines     On
140           Refresh_Interval    10
141
142     filter-kubernetes.conf: |
143       [FILTER]
144           Name                kubernetes
145           Match               kube.*
146           Kube_URL            https://kubernetes.default.svc:443
147           Kube_CA_File        /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
148           Kube_Token_File     /var/run/secrets/kubernetes.io/serviceaccount/token
149           Kube_Tag_Prefix     kube.var.log.containers.
150           Merge_Log           On
151           Merge_Log_Key       log_processed
152           K8S-Logging.Parser  On
153           K8S-Logging.Exclude Off
154
155     filters.conf: |
156       [FILTER]
157           Name                modify
158           Match               kube.*
159           Rename              log message
160       [FILTER]
161           Name                lua
162           Match               kube.*
163           script              /fluent-bit/etc/lua/set_dataset_and_namespace.lua
164           call                set_dataset_and_namespace
165
166     output-elasticsearch.conf: |
167       [OUTPUT]
168           Name                es
169           Match               *
170           Host                ${FLUENT_ELASTICSEARCH_HOST}
171           Port                ${FLUENT_ELASTICSEARCH_PORT}
172           HTTP_User           ${FLUENT_ELASTICSEARCH_USER}
173           HTTP_Passwd         ${FLUENT_ELASTICSEARCH_PASSWORD}
174           #Logstash_Format    On
175           #Logstash_Prefix    logs-kubernetes-fluent-bit
176           Index               logs-kubernetes-fluent-bit
177           Replace_Dots        On
178           Retry_Limit         False
179           TLS                 On
```

```
180         TLS.verify          Off
181         Suppress_Type_Name  On
182
183   parsers.conf: |
184     [PARSER]
185         Name   apache
186         Format regex
187         Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?:
            ↪  +(?<path>[^\"]*?)(?: +\S*)?)?" (?<code>[^ ]*) (?<size>[^ ]*)(?:
            ↪  "(?<referer>[^\"]*)" "(?<agent>[^\"]*)")?$
188         Time_Key time
189         Time_Format %d/%b/%Y:%H:%M:%S %z
190
191     [PARSER]
192         Name   apache2
193         Format regex
194         Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?:
            ↪  +(?<path>[^ ]*) +\S*)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)"
            ↪  "(?<agent>[^\"]*)")?$
195         Time_Key time
196         Time_Format %d/%b/%Y:%H:%M:%S %z
197
198     [PARSER]
199         Name   apache_error
200         Format regex
201         Regex  ^\[[^ ]* (?<time>[^\]]*)\] \[(?<level>[^\]]*)\](?: \[pid (?<pid>[^\]]*)\])?(
            ↪  \[client (?<client>[^\]]*)\])? (?<message>.*)$
202
203     [PARSER]
204         Name   nginx
205         Format regex
206         Regex ^(?<remote>[^ ]*) (?<host>[^ ]*) (?<user>[^ ]*) \[(?<time>[^\]]*)\]
            ↪  "(?<method>\S+)(?: +(?<path>[^\"]*?)(?: +\S*)?)?" (?<code>[^ ]*) (?<size>[^ ]*)(?:
            ↪  "(?<referer>[^\"]*)" "(?<agent>[^\"]*)")?$
207         Time_Key time
208         Time_Format %d/%b/%Y:%H:%M:%S %z
209
210     [PARSER]
211         Name   json
212         Format json
213         Time_Key time
214         Time_Format %d/%b/%Y:%H:%M:%S %z
215
216     [PARSER]
217         Name        docker
218         Format      json
219         Time_Key    time
220         Time_Format %Y-%m-%dT%H:%M:%S.%L
221         Time_Keep   On
222
223     [PARSER]
224         Name        syslog
225         Format      regex
226         Regex       ^\<(?<pri>[0-9]+)\>(?<time>[^ ]* {1,2}[^ ]* [^ ]*) (?<host>[^ ]*)
            ↪  (?<ident>[a-zA-Z0-9_\/\.\-]*)(?:\[(?<pid>[0-9]+)\])?(?:[^\:]*\:)? *(?<message>.*)$
227         Time_Key    time
```

```
        Time_Format %b %d %H:%M:%S
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-lua-scripts
data:
  set_dataset_and_namespace.lua: |
    function set_dataset_and_namespace(tag, timestamp, record)
        local new_record = record
        -- Initialize data_stream table if it doesn't exist
        new_record["data_stream"] = new_record["data_stream"] or {}
        -- Set data_stream.namespace to "all"
        new_record["data_stream"]["namespace"] = "all"
        -- Set data_stream.dataset to "kubernetes-fluentbit"
        new_record["data_stream"]["dataset"] = "kubernetes-fluentbit"
        -- print("Namespace set to: " .. new_record["data_stream"]["namespace"])
        -- print("Dataset set to: " .. new_record["data_stream"]["dataset"])
        return 1, timestamp, new_record
    end
```

A.6 Deployment script for privileged Pods

Applying this YAML-script on a k8s cluster deploys a privileged Pod on each node in the cluster, which can be used for forensics on that node. The privileged Pod in Listing A.8 shares the network- and process namespace with its node, so running `ps aux` shows all the processes running on the node. Privileged Pods sharing the network namespace etc. need to be allowed in the cluster and the service account deploying these Pods through kubectl. Listing A.9 shows a minimal configuration for Pods mounting the node's file system root `/` mounted to `/mnt/node`, also providing quasi root access to the node.

Listing A.8: Deployment YAML-script for a privileged Pod on each node in the cluster

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: privileged-pod
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create", "get", "list", "watch", "delete"]
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["create", "get", "list", "watch", "delete"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create", "get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-privileged-binding
subjects:
- kind: ServiceAccount
```

```
  name: default
  namespace: default
roleRef:
  kind: ClusterRole
  name: privileged-pod
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ubuntu-daemonset
  labels:
    app: ubuntu
spec:
  selector:
    matchLabels:
      app: ubuntu
  template:
    metadata:
      labels:
        app: ubuntu
    spec:
      hostPID: true  # share the node's process namespace
      hostNetwork: true # share the node's network namespace
      hostIPC: true  # allow Inter Process Communication (IPC) with any process on the node
      containers:
      - name: ubuntu
        image: ubuntu:latest # could use any other image like alpine as well
        securityContext:
          privileged: true
        command: ["/bin/bash", "-c", "--"]
        args: ["while true; do sleep 30; done;"]
      tolerations:
      - operator: "Exists"
      nodeSelector:
        kubernetes.io/os: linux
```

Listing A.9: Deployment YAML-script for Pods mounting the node's file system root `/` to `/mnt/node`

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ubuntu-minimal
  labels:
    app: ubuntu-minimal
spec:
  selector:
    matchLabels:
      app: ubuntu-minimal
  template:
    metadata:
      labels:
        app: ubuntu-minimal
    spec:
      containers:
      - name: ubuntu-minimal
```

```
18          image: ubuntu:latest
19          command: ["/bin/sh", "-c", "--"]
20          args: ["while true; do sleep 30; done;"]
21          volumeMounts:  # Add this section
22          - name: host-root
23            mountPath: /mnt/node
24        tolerations:
25        - operator: "Exists"
26        nodeSelector:
27          kubernetes.io/os: linux
28        volumes:
29        - name: host-root
30          hostPath:
31            path: /
32            type: Directory
```

A.7 Deployment of Falco with audit logs export of events

This section shows how Falco evaluated in section 6.1 was set up in the microk8s-cluster and configured for audit logs and exporting of the event logs to Elasticsearch in the ECK-Stack set up in section A.4. Falco provides an official Helm-chart for easier deployment in k8s. There is a Falco-addon available for microk8s, but the deployed Pods came up in an Error-state which could not be resolved as microk8s plug-ins are not configurable. Therefore, the Helm installation was chosen for deployment in the microk8s-cluster. A Helm-values file was created to adapt the installation, it can be found in Listing A.10 below along with the corresponding Helm-command.

The eBPF-driver is chosen (line 2). Its installation to the kernel goes automatic without further intervention but delays the initial startup of the Falco-Pods by ten to fifteen minutes. Moreover, the k8s metadata enrichment is enabled using `collectors.kubernetes.enabled=true` (line 4). It deploys an additional Pod called falco-k8s-metacollector that fetches metadata from kube-apiserver and sends it to the Falco-plug-in k8smeta. Both components are deployed and set up to communicate correctly, but practical tests with individual rules revealed that the fields from the k8smeta plug-in are not populated correctly. Falco recently changed the way k8s metadata are fetched, so this is likely a bug. After verifying the installation and function of both components several times and not finding any errors, I filed a GitHub issue[3] about this.

Another source for alerting and enrichment with k8s metadata is the container-runtime socket. At first, this enrichment did not work because Falco could not access the microk8s containerd socket. This is because microk8s is deployed as a Ubuntu snap running programs in a sandbox, so it actually runs its own instance of containerd. Consequently, microk8s and Docker can run independently from each other. However, Falco does not find the containerd socket inside the snap as it is not at the default Linux location. This was fixed by creating an empty file in the default location `/run/containerd/containerd.sock`, and using `mount` with the `--bind`-parameter to make the microk8s containerd socket file available in the default location, too.

Only a few outputs like a file, an HTTP endpoint or Linux syslog are available in vanilla Falco. An additional k8s Service called Falcosidekick and acting as a proxy allows exporting Falco's

[3] GitHub issue about k8smeta fields not being available: https://github.com/falcosecurity/plugins/issues/514

alerts to many other sources, see section 6.1, point 9 for some examples. It also comes with a built-in dashboard and event-viewer called Falcosidekick-UI that was activated (Listing A.10, line 9). It stores its data with a configurable retention period in a Redis database. Figure A.4 shows the top part of its dashboard after 48 hours of data from the cluster itself and Falco's event-generator, that was installed to evaluate Falco's built-in detection capabilities.

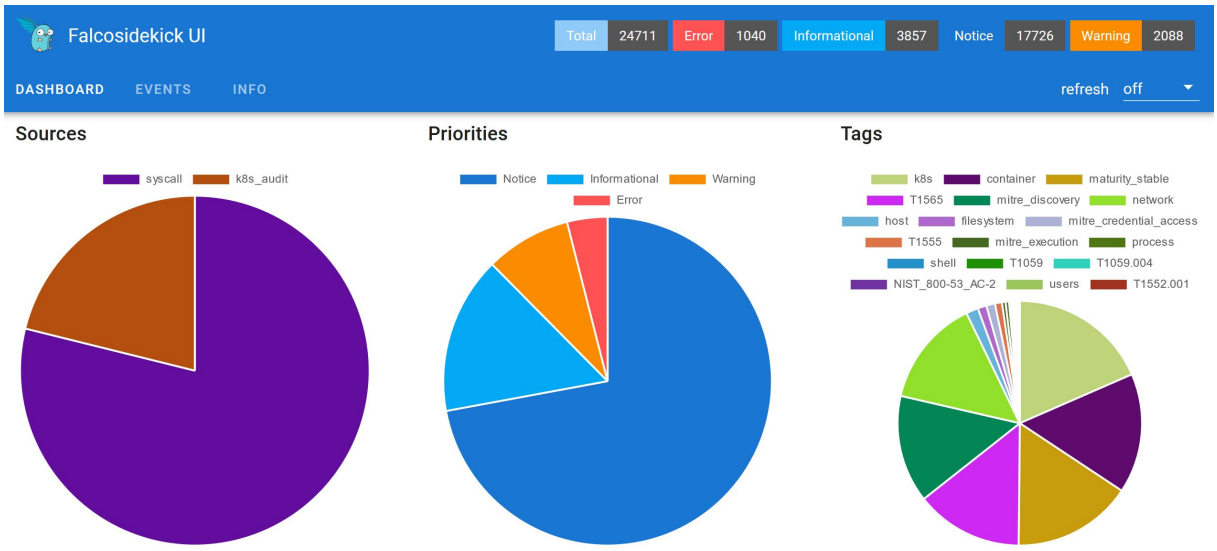


Figure A.4: Falcosidekick-UI dashboard showing statistics over the alerts from the last 48 hours

Furthermore, an output to Elasticsearch in the ECK Stack was configured to facilitate filtering and further analysis of the events from Falco. According to its documentation, Falcosidekick only supports either plain unencrypted HTTP or properly signed certificates, but no self-signed certificates. Everything in the microk8s-cluster is set up with self-signed certificates as no root certificate was configured. Therefore, Falcosidekick's Elasticsearch output did not work as it could not verify the self-signed certificate. Apache Kafka, a message broker, was used as a proxy to overcome this issue. Falcosidekick streams its messages to a Kafka topic (line 11), from where another Logstash pipeline fetches them to export them to Elasticsearch. Falco's Helm-chart does not support fetching the Kafka password from its secret, therefore it must be fetched with kubectl and set in the Helm-command, this is shown in the first part of Listing A.10. In a production environment, direct export from Falcosidekick to Elasticsearch would work without a proxy like Kafka as there would be a root certificate and thus no self-signed certificates.

Audit-logs can be added to Falco as an additional data source using the k8saudit-plug-in which depends on the json plug-in (line 19). Similar to kube-audit-rest, it uses a webhook to fetch the audit logs. However, it is configured as an additional cmd-arg for kube-apiserver and requires Falco to be reachable via a NodePort-service exposing the Falco-service on a high port on each node (line 29). Additionally, `falcoctl` is configured to automatically fetch and update the k8saudit- and default Falco-rules every sixth hour (line 40).

Falco can be deployed with the k8saudit-plug-in for audit-logs and outputs to Apache Kafka and the Falcosidekick-UI using the Helm-command below. Note that the password for Apache Kafka is set in the command, so Apache Kafka should be installed beforehand. To use the custom rule defined in Listing 6.1, append `-f falco-rules.yaml` to the Helm-command.

Listing A.10: Helm-command and -values for Falco

Helm-command to install Falco and set the Kafka-password

```
helm upgrade --install falco falcosecurity/falco --namespace falco -f falco-values.yaml --set
↪  falcosidekick.config.kafka.password="$(kubectl get secret kafka-user-passwords --namespace
↪  elastic-system -o jsonpath='{.data.client-passwords}' | base64 -d | cut -d , -f 1)"
```

Helm-values falco-values.yaml

```yaml
driver:
  kind: ebpf
collectors:
  kubernetes:
    enabled: true
falcosidekick:
  enabled: true
  webui:
    enabled: true
  config:
    kafka:
      hostport: "kafka.elastic-system.svc.cluster.local"  # Adjust with your Kafka broker
      topic: "falco-events"  # Your Kafka topic
      username: "user1"  # Kafka username
      topiccreation: true
      sasl: "PLAIN" # Kafka SASL/PLAIN authentication
      #password: # Kafka password set via command line
falco:
  load_plugins: [k8saudit, json]
  rules_file:
    - /etc/falco/rules.d
    - /etc/falco/falco_rules.yaml
    - /etc/falco/k8s_audit_rules.yaml
services:
  - name: k8saudit-webhook
    type: NodePort
    ports:
      - port: 9765 # See plugin open_params
        nodePort: 30007
        protocol: TCP
falcoctl:
  config:
    artifact:
      install:
        resolveDeps: true
        refs:
          - k8saudit-rules:0.7
          - falco-rules:3.1.0
      follow:
        refs:
          - k8saudit-rules:0.7
          - falco-rules:3.1.0
  artifact:
    install:
      enabled: true
    follow:
      enabled: true
```