Hochschule Fulda
University of Applied Sciences

AI | Angewandte Informatik

# EFFEKTIVITÄT VON ZERO-TRUST-ARCHITEKTUREN IN KUBERNETES-CLUSTERN

MASTERARBEIT

zur Erlangung des Grades eines Master of Science
im Fachbereich Angewandte Informatik
der Hochschule Fulda


| | |
|---|---|
| Eingereicht von: | Christian Bernhard Dechant |
| Anschrift: | Langenbieberer Str. 14<br>36145 Hofbieber |
| Matrikelnummer: | 1161283 |
| E-Mail-Adresse: | christian-bernhard.dechant@informatik.hs-fulda.de |
| Gutachter: | Prof. Dr. Michael Zohner<br>Prof. Dr. Sebastian Rieger |
| Eingereicht am: | 18. Dezember 2024 |

# Zusammenfassung

Die Softwareentwicklung durchläuft kontinuierliche Veränderungen, von Bare-Metal-Servern über Container bis hin zu Containerorchestrierungsplattformen wie Kubernetes. Trotz der Einführung zahlreicher Best Practices zur Absicherung von Kubernetes bleibt die Frage offen, ob diese traditionellen Sicherheitsansätze ausreichend sind, um den dynamischen und verteilten Charakter moderner Kubernetes-Umgebungen zu schützen. In diesem Zusammenhang gewinnt das Zero-Trust-Konzept „Never Trust, Always Verify“ zunehmend an Bedeutung.

Diese Arbeit analysiert die Sicherheitsarchitektur von Kubernetes-Clustern, um die Effektivität der Zero-Trust-Architektur bei der Identifizierung und Behebung von Schwachstellen sowie der Risikominderung von Bedrohungen des Clusters zu bewerten. Dabei werden Security Best Practices und Komponenten der Zero-Trust-Architektur vorgestellt und in den Umgebungen Minikube und EKS anhand einer Beispielapplikation implementiert.

Die Analyse zur Effektivität umfasst eine detaillierte Messung des Ressourcenverbrauchs (CPU und Speicher), der Bedrohungsverhinderung anhand von Bedrohungstechniken, Angriffsvektoren und aktueller Schwachstellen für Kubernetes Cluster, zusätzlicher Schwachstellen, Angriffsvektoren und Herausforderungen und geht auf die Komplexität und den Aufwand der Integration einer Zero-Trust-Architektur ein.

So besteht eine Zero-Trust-Architektur für Kubernetes Cluster aus Network Policies, einem Service Mesh, Authorization Policies und zur Erweiterung der Authentifizierungsmöglichkeiten einem Open Policy Agent. Ebenso essenziell ist das Monitoring, dass durch Grafana, Prometheus und Kiali durchgeführt werden kann. Die Zero-Trust-Architektur erhöht den Ressourcenverbrauch; insbesondere im Public Cloud Cluster kann dieser zusätzliche Ressourcenverbrauch zu doppelt so hohen Kosten führen im Vergleich zu einem Standard-Setup ohne Zero-Trust. Weiterhin zeigt die Auswertung zur Bedrohungsverhinderung, dass Security Best Practices allein bereits einen umfassenden Schutz gegen die untersuchten Bedrohungen bieten können. Zusätzlich werden die Herausforderungen und Nachteile der Zero-Trust-Architektur, wie die erhöhte Komplexität und der erforderliche Zeitaufwand für die Implementierung, diskutiert. Während die Zero-Trust-Architektur den Schutz vor internen Bedrohungen verbessern kann, hat sie begrenzten Einfluss auf externe Bedrohungen und führt zu erhöhtem Verwaltungsaufwand und Datenaufkommen.

Für Unternehmen, die kosteneffiziente Lösungen suchen und den zusätzlichen Verwaltungsaufwand der Zero-Trust-Praktiken berücksichtigen müssen, könnte eine sorgfältige Kombination von Best Practices und gezielten Zero-Trust-Maßnahmen eine ausgewogene Lösung darstellen, die Sicherheit und Effizienz berücksichtigt.

# Abstract

Software development is constantly changing, from bare-metal servers to containers and container orchestration platforms such as Kubernetes. Despite the introduction of numerous best practices for securing Kubernetes, the question remains whether these traditional security approaches are sufficient to protect the dynamic and distributed nature of modern Kubernetes environments. In this context, the zero trust concept "Never Trust, Always Verify" is becoming increasingly important.
This thesis analyzes the security architecture of Kubernetes clusters to evaluate the effectiveness of the Zero Trust architecture in identifying and remediating vulnerabilities and mitigating threats to the cluster. Security best practices and components of the zero-trust architecture will be presented and implemented in the Minikube and EKS environments using a sample application.
The effectiveness analysis includes a detailed measurement of resource consumption (CPU and memory), threat prevention based on threat techniques, attack vectors and current vulnerabilities for Kubernetes clusters, additional vulnerabilities, attack vectors and challenges, and addresses the complexity and effort of integrating a zero trust architecture.
This paper analyzes the security architecture of Kubernetes clusters in order to demonstrate the effectiveness of the monitoring, which can be carried out by Grafana, Prometheus and Kiali, is just as essential. The Zero Trust architecture increases resource consumption; in the public cloud cluster in particular, this additional resource consumption can lead to costs that are twice as high compared to a standard setup without Zero Trust. Furthermore, the evaluation of threat prevention shows that security best practices alone can provide comprehensive protection against the threats examined. In addition, the challenges and disadvantages of the zero-trust architecture, such as the increased complexity and the time required for implementation, are discussed. While zero-trust architecture can improve protection against internal threats, it has limited impact on external threats and leads to increased administrative overhead and data volume.
For organizations seeking cost-effective solutions and needing to consider the additional administrative burden of zero trust practices, a careful combination of best practices and targeted zero trust measures could provide a balanced solution that considers security and efficiency.

# Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn Prof. Michael Zohner, der meine Masterarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Auch gebührt mein Dank Herrn Prof. Sebastian Rieger, der als Zweitgutachter meine Arbeit betreut und begutachtet hat.

Bei beiden möchte ich mich auch für die Möglichkeit die Masterarbeit über Zero-Trust und Kubernetes-Cluster schreiben zu dürfen, bedanken. Die Unterstützung bei der Findung des Themas weiß ich sehr zu schätzen.

Ebenso möchte ich mich bei der Reservix GmbH für die Unterstützung bei der Anfertigung dieser Masterarbeit bedanken. Reservix GmbH stellte die in dieser Arbeit verwendete AWS EKS-Instanz bereit, sodass die Ausarbeitung im Rahmen dieser Arbeit möglich war.

Außerdem möchte ich mich bei den vielen Korrekturlesern bedanken, die mit viel Zeiteinsatz meine Masterarbeit korrigiert und Anmerkungen geschrieben haben.

Abschließend möchte ich mich bei meinen Eltern, meinem Bruder und meiner Freundin bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Christian Dechant

Hofbieber, 18. Dezember 2024

Hochschule Fulda
University of Applied Sciences

| | |
|---|---|
| Christian Bernhard Dechant | 1161283 |
| Vor- und Nachname | Matrikelnummer |

Angewandte Informatik
Studiengang

# Versicherung gemäß § 25 Absatz 4 ABPO

Hiermit erkläre ich, dass ich die vorliegende Prüfungsleistung - bei einer Gruppenarbeit den entsprechend gekennzeichneten Anteil der Arbeit - mit dem

Titel Effektivität von Zero-Trust-Architekturen in Kubernetes Clustern

selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe.

Des Weiteren hat die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Hochschule oder Prüfungsstelle vorgelegen. Ich versichere, dass alle eingereichten Versionen dieser Prüfungsleistung einander entsprechen.

Mir ist bekannt, dass die Abgabe einer unwahren Versicherung als Täuschung gilt und die Arbeit in diesem Fall mit „nicht ausreichend“ (5,0) bewertet wird.

| | |
|---|---|
| 03.09.2024 | C. Dechant |
| Datum | Unterschrift |

# Inhaltsverzeichnis

# 1 Einleitung

Die Softwareentwicklung durchläuft einen ständigen Wandel, der von technologischen Fortschritten und sich verändernden Anforderungen an Anwendungen und Infrastrukturen geprägt ist. In den Anfangszeiten wurden Anwendungen direkt auf Bare-Metal-Servern bereitgestellt, wobei jeder Server einer spezifischen Anwendung gewidmet war. Mit zunehmender Komplexität der Anwendungen und steigenden Anforderungen gewann die Container-basierte Virtualisierung an Bedeutung. Container ermöglichen es, Anwendungen mitsamt ihrer Abhängigkeiten in einer leichten, portablen und konsistenten Weise zu verpacken. Doch je komplexer die Anwendungen wurden, desto größer wurden auch die Herausforderungen an deren Architektur. Traditionelle monolithische Anwendungen erwiesen sich als schwerfällig und wenig flexibel, um den Anforderungen an Skalierbarkeit gerecht zu werden. Um diese Herausforderungen zu bewältigen, setzte sich das Microservices-Konzept durch, das darauf abzielt, Anwendungen in kleinere, unabhängige Services zu zerlegen, die über APIs miteinander kommunizieren. Diese Architektur erlaubt es, einzelne Microservices unabhängig voneinander zu entwickeln, zu testen, bereitzustellen und zu skalieren, was die Flexibilität und Anpassungsfähigkeit erheblich steigert. [Kui]
Mit der zunehmenden Verbreitung von Microservices und Containern entstand jedoch ein neues Problem: die effiziente Verwaltung und Orchestrierung einer wachsenden Zahl von Containern. Schnell wurde klar, dass ein System erforderlich ist, das diese Aufgabe zuverlässig übernehmen kann. Die Veröffentlichung von Kubernetes als Open-Source-Projekt durch Google im Jahr 2015 markierte einen entscheidenden Meilenstein in der Softwareentwicklung. Kubernetes bot eine leistungsstarke und flexible Lösung zur Verwaltung von Container-Clustern und ermöglichte es, Anwendungen zuverlässig und skalierbar zu betreiben. In kurzer Zeit entwickelte sich Kubernetes zum de-facto-Standard für Container-Orchestrierung. Heute setzen Unternehmen Kubernetes nicht nur zur Verwaltung ihrer Anwendungen ein, sondern orchestrieren damit auch ihre gesamte Infrastruktur. Kubernetes-Cluster können dabei in verschiedenen Größen und Konfigurationen betrieben werden, von kleinen Entwicklungsumgebungen wie Minikube bis hin zu großen Produktionsumgebungen, die auf mehreren virtuellen Maschinen basieren. Große Cloud-Anbieter wie Amazon, Google und Microsoft bieten zudem verwaltete Kubernetes-Dienste wie Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE) und Azure Kubernetes Service (AKS) an, die den Betrieb von Kubernetes-Clustern weiter vereinfachen. Diese verteilte Architektur, die die Workloads innerhalb eines Clusters verwaltet und orchestriert, bringt jedoch erhebliche Sicherheitsherausforderungen mit sich. Diese Herausforderungen resultieren aus verschiedenen Angriffsvektoren, die sich auf die Architektur, das Netzwerk, das Cluster

und die eingesetzten Container beziehen. Dazu gehören potenzielle Schwachstellen in den Kubernetes-Komponenten selbst, Angriffe auf die Kommunikationswege zwischen den Nodes und nicht zuletzt Konfigurationsfehler bei der Einrichtung und dem Betrieb des Clusters. [Kui]
Um diese Sicherheitsrisiken zu minimieren, ist ein tiefes Verständnis der Bedrohungen und Schwachstellen erforderlich, die in einer Kubernetes-Umgebung existieren. In den letzten Jahren wurden zahlreiche Security Best Practices für Kubernetes entwickelt, um den Schutz solcher Systeme zu gewährleisten. Diese Best Practices umfassen Maßnahmen wie die Einschränkung der Netzwerkkonnektivität zwischen Containern, die Begrenzung der Rechte und Funktionen, die ein Container ausführen darf, und die regelmäßige Durchführung von Schwachstellen-Scans. Zudem ist die kontinuierliche Protokollierung aller Aktivitäten im Cluster sowie die Implementierung von Mechanismen zur frühzeitigen Erkennung von Anomalien von entscheidender Bedeutung. [owa]
Trotz dieser Maßnahmen bleibt jedoch die Frage offen, ob traditionelle Sicherheitsansätze ausreichen, um den dynamischen und verteilten Charakter moderner Kubernetes-Umgebungen ausreichend zu schützen. Ein Ansatz, der in den letzten Jahren zunehmend an Bedeutung gewonnen hat, ist das Zero-Trust-Konzept [Sta20]. Dieses Sicherheitsmodell, das ursprünglich im Jahr 2010 entwickelt wurde, hat seit 2021 an Popularität gewonnen und gilt als vielversprechender Weg, um die Sicherheit in verteilten Systemen zu erhöhen. Im Gegensatz zu traditionellen Sicherheitsmodellen, die auf einem Perimeterschutz basieren und implizites Vertrauen gewähren, verfolgt Zero-Trust das Prinzip „Never Trust, Always Verify“ [BOS+21]. Das bedeutet, dass jedes Subjekt, sei es eine Maschine, ein Nutzer oder eine Anwendung, sich kontinuierlich authentifizieren und autorisieren muss, bevor es Zugang zu Ressourcen erhält. Vertrauen wird in diesem Modell nicht dauerhaft gewährt, sondern muss bei jeder Interaktion neu bestätigt werden. [Sta20]
Vor dem Hintergrund der dynamischen und verteilten Natur von Kubernetes-Clustern stellt sich jedoch die Frage, wie effektiv das Zero-Trust-Prinzip in dieser Umgebung angewendet werden kann.

## 1.1 Ziele der Arbeit

Das Ziel dieser Arbeit ist es, die Wirksamkeit einer Zero-Trust-Architektur für Kubernetes-Cluster anhand einer konkreten Implementierung zu bewerten. Dazu werden zunächst durch eine umfassende Literaturrecherche die Bedrohungen, Herausforderungen und Schwachstellen (CVEs) in Kubernetes-Umgebungen identifiziert und analysiert. Darauf aufbauend werden die Security Best Practices für Kubernetes-Cluster detailliert erläutert und anschließend in Minikube sowie Amazon Elastic Kubernetes Service (EKS) implementiert. Anhand der erforderlichen Komponenten für eine Zero-Trust-Architektur innerhalb eines Kubernetes-Clusters sollen anschließend die folgenden Forschungsfragen dieser Arbeit untersucht und beantwortet werden:

**FF.1**: Wie kann eine Zero-Trust-Architektur in Kubernetes implementiert werden?

**FF.2**: Welche wirtschaftlichen Auswirkungen hat die Integration einer Zero-Trust-Architektur, insbesondere im Hinblick auf CPU- und Speicherressourcen?

**FF.3**: Welche Schwachstellen und Risiken können durch die Implementierung einer Zero-Trust-Architektur in Kubernetes behoben werden?

**FF.4**: Welche zusätzlichen Nachteile, Probleme und Herausforderungen bringt eine Zero-Trust-Architektur mit sich?

**FF.5**: Wie anspruchsvoll und zeitaufwändig ist die Umsetzung einer Zero-Trust-Architektur in Kubernetes?

## 1.2 Aufbau der Arbeit

Die Arbeit ist in sieben Kapitel gegliedert, die wie folgt strukturiert sind:

**Kapitel 2** vermittelt die Grundlagen der Containerisierung mit Docker und Kubernetes und erläutert die zentralen Komponenten eines Kubernetes-Clusters. Es geht außerdem auf die Konzepte von Zero-Trust und Service Meshes ein, um die sicherheitsrelevanten Aspekte und Kommunikationsstrukturen für Zero-Trust-Architekturen innerhalb von Kubernetes-Clustern zu erklären.

**Kapitel 3** widmet sich dem aktuellen Forschungsstand in den Bereichen Sicherheit in Kubernetes-Clustern sowie Zero-Trust-Modelle und -Architekturen. Besonderes Augenmerk liegt dabei auf der Anwendung dieser Konzepte im Kontext von Cloud-Computing und Kubernetes-Clustern.

**Kapitel 4** bietet eine detaillierte Sicherheitsanalyse, um ein umfassendes Verständnis der potenziellen Bedrohungen und Schwachstellen eines Kubernetes-Clusters zu erlangen. Diese Analyse beleuchtet architekturbedingte Angriffsvektoren und Herausforderungen sowie spezifische Schwachstellen einzelner Komponenten wie API-Server, Kubelet, etcd und der Container Runtime Docker. Die Schwachstellen werden dabei anhand von Common Vulnerabilities and Exposures (CVEs) aufgezeigt.

**Kapitel 5** stellt auf den Erkenntnissen der Sicherheitsanalyse aufbauend die Security Best Practices für Kubernetes-Cluster vor. Es werden die Zero-Trust-Empfehlungen der Cloud Native Computing Foundation (CNCF) erläutert und spezifische Schwerpunkte einer Zero-Trust-Architektur für Kubernetes dargelegt. Diese Best Practices und Konzepte werden in Minikube und in Amazon EKS in der Public Cloud implementiert.

**Kapitel 6** evaluiert die Effektivität der implementierten Zero-Trust-Architekturen, sowohl lokal als auch in der Public Cloud. Die Evaluierung umfasst Aspekte wie den Ressourcenverbrauch, die Wirksamkeit der Abwehr zuvor identifizierter Angriffe, die Komplexität der Architektur sowie die Identifizierung neuer Schwachstellen, offener Ports und möglicher Nachteile der eingesetzten Zero-Trust-Architektur.

**Kapitel 7** bildet den Abschluss der Arbeit. In diesem Kapitel werden die gewonnenen Erkenntnisse zur Effektivität der Zero-Trust-Architektur zusammengefasst und ein Ausblick auf mögliche zukünftige Forschungsansätze in diesem Bereich gegeben.

# 2 Grundlagen

Dieses Kapitel führt in die Grundlagen der containerbasierten Virtualisierung, Kubernetes (Kapitel 2.1), Zero-Trust (Kapitel 2.2) und Service Meshes (Kapitel 2.3) ein. Es bietet ein fundamentales Verständnis der Architektur und des Aufbaus von Docker und Kubernetes, das für die in Kapitel 4 durchgeführte Sicherheitsanalyse von wesentlicher Bedeutung ist. Zudem werden die Prinzipien und Grundsätze von Zero-Trust sowie die Grundlagen von Service Meshes erläutert, die einen wichtigen Bestandteil der in Kapitel 5 vorgestellten Zero-Trust-Architektur darstellen.

## 2.1 Containerisierung und Kubernetes

Containerisierung ist das grundlegende Konzept, auf dem sowohl Container als auch Kubernetes als Orchestrator aufbauen. Containerbasierte Virtualisierung, auch als Containerisierung bezeichnet, ist ein Softwarebereitstellungsprozess, bei dem der Code zusammen mit allen notwendigen Dateien und Bibliotheken, die für die Ausführung in einer bestimmten Infrastruktur benötigt werden, gebündelt wird. Dies macht eine Installation der passenden Version für das jeweilige Betriebssystem überflüssig. Mithilfe der Containerisierung lässt sich ein einzelnes Softwarepaket (auch Image genannt) erstellen, das auf einer Vielzahl von Geräten und Betriebssystemen, die eine entsprechende Runtime bereitstellen, ausgeführt werden kann. [amae]

Containerisierung reduziert den Ressourcen-Overhead und verbessert die Nutzung von Datenzentren, indem mehrere gekapselte Prozesse auf einem gemeinsamen OS-Kernel ausgeführt werden. Diese individuellen Instanzen werden als Container bezeichnet. [WRK+19] Im Gegensatz zur Virtualisierung durch einen Hypervisor ermöglicht Containerisierung das Betreiben mehrerer virtueller Umgebungen auf einem gemeinsamen Host-Kernel, was weniger CPU, Speicher und Netzwerkressourcen erfordert. [WRK+19]

Zu den Vorteilen der Containerisierung gehören Portierbarkeit, Skalierbarkeit, Fehlertoleranz und Agilität. Durch die Möglichkeit, Anwendungen in verschiedenen Umgebungen bereitzustellen, ohne den Programmcode ändern zu müssen, wird die Aktualisierung von Legacy-Anwendungscode auf moderne Versionen ermöglicht. [amae]
Skalierbarkeit ergibt sich aus der effizienten Ausführung der leichtgewichtigen Container,

die keine Betriebssystem-Instanz starten müssen. Dies ermöglicht es, mehrere Container für verschiedene Anwendungen auf einem einzigen Computer laufen zu lassen, wobei diese Container dieselben gemeinsamen Ressourcen des Betriebssystems nutzen, ohne einander zu beeinflussen. [amae]
Fehlertoleranz wird durch den Einsatz mehrerer Container, die jeweils Microservices ausführen, erreicht. Da Container in isolierten Benutzerbereichen betrieben werden, hat ein fehlerhafter Container keine Auswirkungen auf andere Container, was die Belastbarkeit und Verfügbarkeit der Anwendung erhöht [amae].
Aufgrund der isolierten Umgebungen, die Container bieten, können Änderungen am Anwendungscode ohne Beeinträchtigung des Betriebssystems, der Hardware oder anderer Anwendungen vorgenommen werden. Zudem ermöglicht diese Agilität verkürzte Software-Release-Zyklen und eine schnellere Implementierung von Updates. Docker, eine der beliebtesten Open-Source-Container-Laufzeitumgebungen, hat sich als de facto-Standard für Containerisierung etabliert. [amae]

### 2.1.1 Docker

Docker automatisiert das Deployment von Anwendungen in Containern, indem es eine Anwendungs-Deployment-Engine oberhalb der virtualisierten Container-Ausführungsumgebung einfügt. Dies ermöglicht es, Container ohne großen Aufwand von der Test- auf die Produktionsumgebung zu portieren. [WRK+19]
Docker unterstützt Service-orientierte und Microservice-Architekturen, wobei empfohlen wird, pro Applikation oder Prozess einen einzelnen Container zu verwenden. Dies unterstützt verteilte Anwendungen und gewährleistet Portabilität, Skalierbarkeit und Fehlertoleranz. [WRK+19]

Die Architektur von Docker folgt einem Client-Server-Modell, bestehend aus dem Client, dem Host und der Registry (siehe Abbildung 2.1). Der Docker Daemon des Docker Hosts verwaltet die Container, während der Docker Client über die CLI die Befehle bereitstellt. Container, die ausführbare Instanzen von Images sind, werden lokal auf dem Docker Host gespeichert und können von dort ausgeführt werden. [doc]
Ein Image ist ein Template, das die Instruktionen für das Erstellen eines Containers enthält. Diese Images basieren oft auf weiteren Images, die durch spezielle Funktionen erweitert werden. [doc]
Container sind die ausführbare Instanz eines Images und können über den Docker Daemon erstellt, gestartet, gestoppt, verschoben oder gelöscht werden. [doc]
Docker speichert sämtliche Images in einer Registry. Diese Registry kann öffentlich über das Internet zugänglich sein, wie zum Beispiel Docker Hub, oder lokal und somit privat auf eigenen Servern bereitgestellt werden [doc].

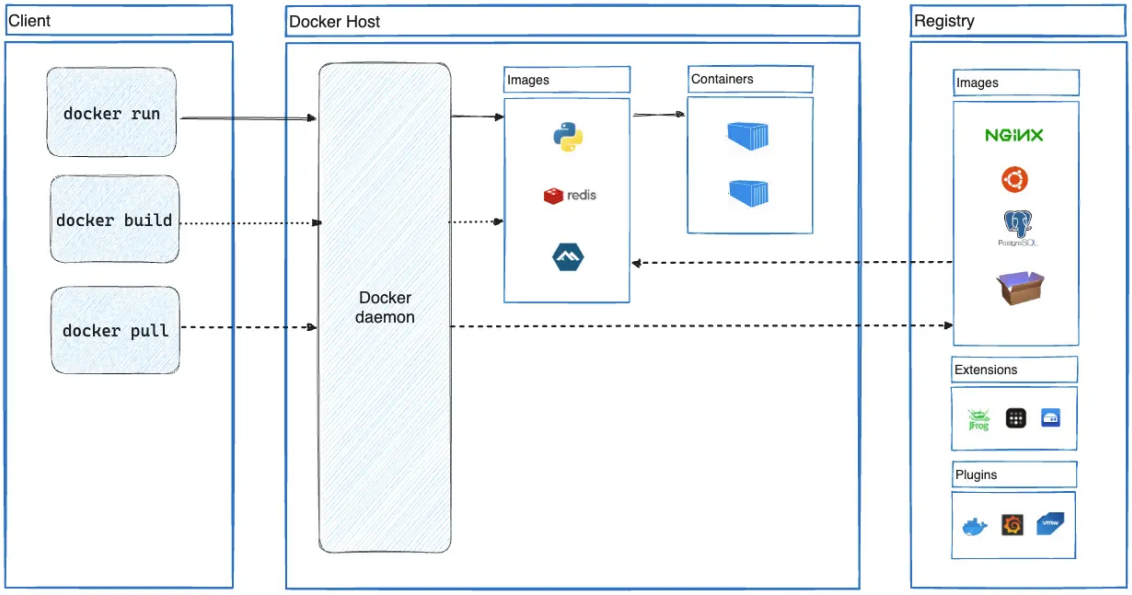



Abbildung 2.1: Darstellung der Docker-Architektur unterteilt in Client, Docker Host und Registry. Client stellt Kommandozeilen-Funktionen zur Verwaltung des Docker Daemon bereit, um Container auszuführen, Images herunterzuladen oder Container Images zu erstellen. Der Docker Daemon kommuniziert zum Herunterladen der Images mit der entfernten Container Image Registry. Auf dem Host verwaltet der Docker Daemon die Images und Container lokal. [doc]

Der Aufbau von Docker, wie in Abbildung 2.1 dargestellt, kann auch in Form von Diensten und Software-Komponenten beschrieben werden, wie in Abbildung 2.2 gezeigt. Auf der Client-Seite befindet sich die Docker CLI, die die Befehle im Nutzerkontext bereitstellt. Der Docker Daemon des Docker Hosts wird durch die *dockerd*-Komponente bereitgestellt, während die Container durch den Service-Daemon *containerd* verwaltet werden. Wie in der Abbildung 2.2 zu sehen ist, führt *dockerd* die Container nicht selbst aus, sondern überlässt die Ausführung der Apps den beiden Low-Level-Laufzeitumgebungen *runc* und *containerd-shim*. *runc* ist für die Interaktion mit dem unterliegenden Betriebssystem, die Isolation der Container durch Namensräume und Dateisysteme sowie die Initialisierung der Umgebung zuständig. *containerd-shim* dient als Hauptprozess für die Verwaltung und Überwachung der einzelnen Container. [ZKL+23]

### 2.1.2 Kubernetes

Kubernetes (k8s) ist ein Open-Source-Container-Orchestrierungssystem, das zur Standard-API für das Erstellen Cloud-nativer Anwendungen geworden ist. Fast jede öffentliche Cloud, wie Amazon Web Services (AWS), Google Cloud Provider und Microsoft Azure, bietet Kubernetes als Service an [BBHD20]. Kubernetes ermöglicht das Deployment, das Überwachen und das Skalieren von Containern über mehrere Rechner hinweg und bietet

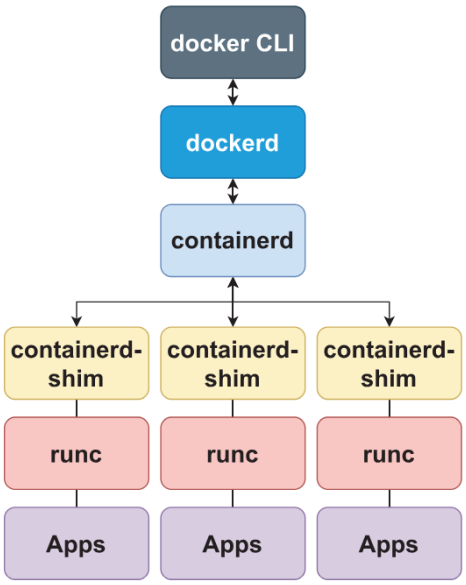



**Fig. 2.** Docker Architecture.

Abbildung 2.2: Docker Architektur unterteilt in die einzelnen Bestandteile basierend auf Abbildung 2.1. dockerCLI entspricht der Clientseitigen CLI, die direkt mit dem Docker Daemon (dockerd) kommuniziert. dockerd kommuniziert mit containerd. Der Service-Daemon containerd verwaltet die Container. runc und containerd-shim führen die Container aus. runc übernimmt die Interaktion mit dem unterliegenden Betriebssystem, die Isolation der Container und des Dateisystems und übernimmt die Initialisierung der Umgebung. containerd-shim dient der Verwaltung und Überwachung des einzelnen Containers. [ZKL+23]

Funktionen, um den gewünschten Zustand der Anwendungen zu definieren und zu steuern. [Kub19]

### Kubernetes Architektur

Die Architektur von Kubernetes basiert auf einer Client-Server-Architektur und besteht aus einer Menge von physischen oder virtuellen Maschinen und anderen Infrastruktur-Ressourcen, die zur Ausführung von Applikationen genutzt werden. Das Kubernetes Cluster ist eine Sammlung von Maschinen-, Speicher- und Netzwerkressourcen, die genutzt werden, um verschiedene Aufgabengrößen zu bewältigen. Die Architektur von Kubernetes ist in Abbildung 2.3 dargestellt. Das Cluster ist unterteilt in mehrere Nodes, die physische oder virtuelle Maschinen darstellen, dabei werden diese Nodes und das Cluster in zwei Ebenen unterteilt: die Control Plane (Verwaltungsebene) und die Data Plane (Ausführungsebene). [Say17]

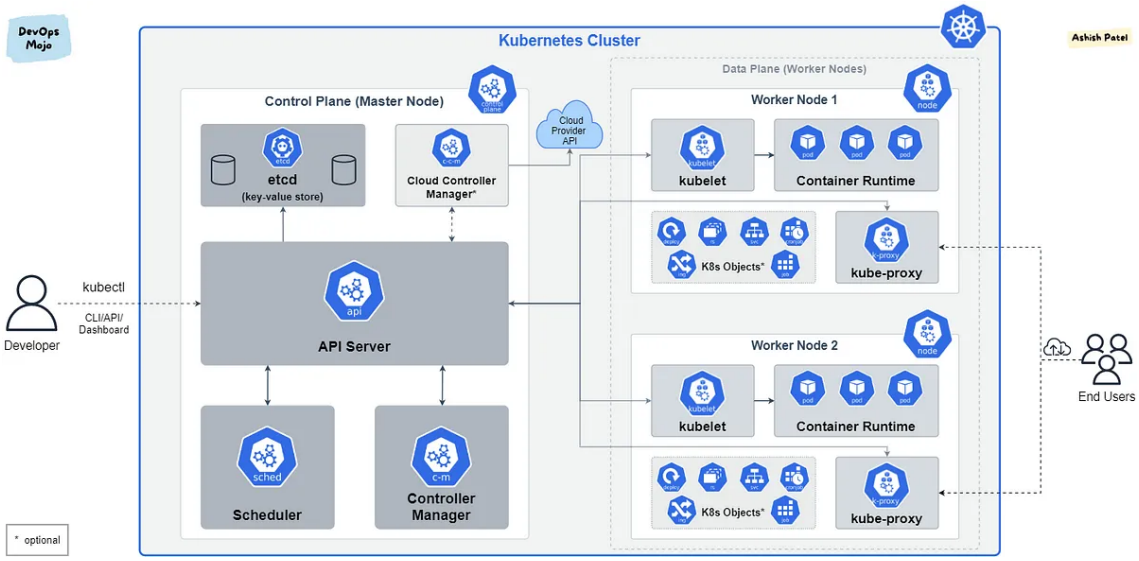



Abbildung 2.3: Darstellung der Kubernetes-Architektur unterteilt in Management-Ebene (Control Plane) und Datenebene (Data Plane) mit grundlegender Netzwerkkonnektivität der einzelnen Clusterbestandteile. Zu den Clusterbestandteilen gehören auf der Control-Plane, dem Master Node, das etcd ein Key-Value Speicher, der den Zustand des Clusters und einige Secrets verwaltet. Weiterhin finden sich auf der Control-Plane einige Controller und Scheduler, die für die Ausführung des Clusters und der Microservices gebraucht werden. Mithilfe der Netzwerkkonnektivität zum API-Server nimmt dieser die zentrale Verwaltung ein. Er kommuniziert mit den Nodes der Datenebene, die aus dem Kubelet, der für die Ausführung der Pods zuständig ist. Die Pods werden auf dem Worker in der Container-Runtime ausgeführt und bestehen aus einem oder mehreren Containern. Mithilfe des kube-proxy eines Worker nodes können Endnutzer auf die Anwendungen des Workers zugreifen. Mithilfe von Kubernetes Objekten, wie SVCs, Ingress-Gateways, Deploys und Jobs, werden die Pods bei der Ausführung unterstützt. [Fer]

**Control Plane:** Die Control Plane ist das zentrale Steuerungssystem des Clusters. Sie besteht aus mehreren Komponenten:

- **API-Server:** Der API-Server ist das zentrale Steuerungsmodul, das die Kommunikation zwischen den Komponenten und dem Cluster ermöglicht. Er nimmt Befehle von Nutzern (über kubectl oder Schnittstellen) und anderen Komponenten entgegen und verteilt diese an die entsprechenden Worker Nodes. [Say17]

- **etcd:** Ein verteilter Key-Value-Speicher, der den Cluster-Zustand speichert. etcd verwaltet Konfigurationsdaten, Secrets und andere Cluster-bezogene Informationen. [Say17]

- **Scheduler:** Der Scheduler ist für die Platzierung der Pods auf den verfügbaren Worker Nodes zuständig, basierend auf den definierten Ressourcenanforderungen und -verfügbarkeiten. [Say17]

- **Controller Manager:** Der Controller Manager führt verschiedene Aufgaben aus, die zur Aufrechterhaltung des gewünschten Zustands des Clusters erforderlich sind, wie das Skalieren von Replikationen und das Verwalten von Node-Zuständen. [Say17]

**Data Plane:** Die Data Plane besteht aus den Worker Nodes, auf denen die Container ausgeführt werden. Jeder Worker Node stellt die notwendigen Dienste und Umgebungen für die Ausführung von Containern bereit und wird durch die folgenden Komponenten unterstützt:

- **kubelet:** Das kubelet ist der primäre Agent auf jedem Node, der für das Herunterladen und Ausführen von Pods verantwortlich ist. Es überwacht den Zustand der Pods und kommuniziert regelmäßig mit der Control Plane, um den gewünschten Zustand sicherzustellen. [Say17]

- **Container Runtime:** Die Container Runtime ist die Laufzeitumgebung, in der die Container betrieben werden. Kubernetes unterstützt verschiedene Container-Runtimes wie Docker, containerd und CRI-O. [Say17]

- **kube-proxy:** Der kube-proxy ist für die Netzwerkkonnektivität innerhalb des Clusters verantwortlich und sorgt dafür, dass Pods sowohl untereinander als auch mit externen Diensten kommunizieren können. Er implementiert Netzwerkregeln und Load Balancing für den Datenverkehr zu den Services. [Say17]

#### Kubernetes Objekte und Ressourcen

Kubernetes verwaltet Anwendungen durch verschiedene Abstraktionen, die als Kubernetes-Objekte bezeichnet werden. Die wichtigsten davon sind:

- **Pods:** Ein Pod ist die kleinste und einfachste Kubernetes-Einheit, die eine Gruppe von einem oder mehreren Containern enthält, die sich Ressourcen wie Netzwerk und Speicher teilen. Pods sind kurzlebig und können jederzeit neu erstellt werden. [Say17]

- **Services:** Ein Service ist eine Abstraktion, die einen Satz von Pods als Netzwerkdienst definiert. Services ermöglichen eine dauerhafte IP-Adresse und DNS-Eintrag für eine Gruppe von Pods, die durch ein Label selektiert werden. Sie bieten eine stabile Endpoint-Verbindung, auch wenn sich die zugrunde liegenden Pods ändern. [Say17]

- **Ingress:** Ein Ingress ist ein API-Objekt, das HTTP- und HTTPS-Routen zu Services innerhalb des Clusters verwaltet. Es ermöglicht die Steuerung des externen Zugriffs auf die Dienste, typischerweise über HTTP. [Say17]

- **Secrets:** Ein Secret ist ein Kubernetes-Objekt, das sensible Daten wie Passwörter, OAuth-Tokens und SSH-Schlüssel enthält. Secrets werden unverschlüsselt gespeichert und können als Umgebungsvariablen oder Dateien in Pods eingebunden werden. [Say17]

#### Erweiterte Funktionen: Load Balancing und Service-Management

Kubernetes bietet integriertes Load Balancing für Services an. Wenn ein Service definiert ist, verteilt Kubernetes den eingehenden Datenverkehr automatisch auf die Pods, die dem Service zugeordnet sind. Dies geschieht durch Round-Robin-Verfahren oder benutzerdefinierte Regeln. Für erweiterte Anwendungsfälle können Entwickler jedoch auch eigene Load Balancing-Lösungen implementieren, um den Datenverkehr gezielt zu steuern.

Das Ingress-Objekt erweitert die Funktionen des Load Balancing, indem es HTTP- und HTTPS-Routen zu Services innerhalb des Clusters verwaltet. Es ermöglicht das Routing basierend auf verschiedenen Kriterien wie URL-Pfaden oder Hostnamen und kann in Kombination mit einem Ingress-Controller verwendet werden, um erweitertes Routing, SSL/TLS-Terminierung und Lastausgleich zu bieten.

### 2.1.3 Zusammenfassung

In diesem Kapitel wurden die Grundlagen der Containerisierung und die Architektur von Kubernetes behandelt. Es wurden Docker als Standard-Container-Laufzeitumgebung und Kubernetes als führendes Orchestrierungssystem vorgestellt. Wichtige Kubernetes-Konzepte wie Pods, Services, Ingress und Secrets sowie deren Rolle in der Verwaltung und Skalierung von Containern wurden erklärt. Darüber hinaus wurden die erweiterten Funktionen von Kubernetes im Bereich Load Balancing und Service-Management beleuchtet. Dieses Verständnis bildet die Grundlage für die weitere Diskussion über Zero-Trust und Service Meshes.

## 2.2 Zero-Trust

Zero-Trust ist ein Sicherheitskonzept, dass das traditionelle Modell des impliziten Vertrauens durch die Idee der Deperimeterisierung ersetzt. Dabei wird das Vertrauen in Systeme und Nutzer nicht mehr automatisch gewährt, sondern muss kontinuierlich überprüft werden, um Sicherheitsrisiken zu minimieren. [Sta20]

Das Zero-Trust-Paradigma konzentriert sich auf den Schutz von Ressourcen mit der grundlegenden Annahme, dass Vertrauen niemals implizit gegeben wird, sondern stets überprüft werden muss: „Never trust, always verify“ [BOS+21]. Dieser Ansatz verfolgt eine umfassende Sicherheitsstrategie für Unternehmensressourcen und Datensicherheit, die Identitätsmanagement, Zugangsdaten, Zugriffsverwaltung, Betrieb, Endpunkte, Hosting-Umgebungen und die zugrunde liegende Infrastruktur umfasst. [Sta20]

Die in diesem Kapitel erläuterten Grundlagen zu Zero-Trust und die vorgestellten Zero-Trust-Architektur-Variationen dienen dem Verständnis und als Grundlage für die Implementierung in Kapitel 5. Im Rahmen dieses Paradigmas werden drei zentrale Begriffe unterschieden:

**Zero-Trust (ZT)** ist eine Sammlung von Konzepten und Prinzipien, die darauf abzielen, Unsicherheiten bei der Durchsetzung präziser Zugriffsentscheidungen durch Minimierung von Berechtigungen pro Anfrage in Informationssystemen und -diensten zu verringern, insbesondere in Netzwerken, die als potenziell kompromittiert gelten. [Sta20]

**Zero-Trust-Architektur (ZTA)** beschreibt den Cybersicherheitsplan, der auf Zero-Trust-Prinzipien basiert und die Beziehungen zwischen Komponenten, die Workflow-Planung und die Festlegung von Zugriffsrichtlinien regelt. [Sta20]

**Zero-Trust-Unternehmen** bezeichnet die Netzwerk-Infrastruktur (physisch und virtuell) sowie die Betriebsrichtlinien eines Unternehmens, die als Ergebnis der Umsetzung eines Zero-Trust-Architekturplans entstehen. [Sta20]

Zusammenfassend lässt sich sagen, dass Zero-Trust ein Paradigma ist, das den Schwerpunkt auf Authentifizierung, Autorisierung und die Minimierung impliziter Vertrauenszonen legt, während es gleichzeitig die Verfügbarkeit sicherstellt und Verzögerungen bei Authentifizierungsmechanismen reduziert. Der Zugriff auf Ressourcen wird durch einen Policy Decision Point (PDP) und einen Policy Enforcement Point (PEP) gesteuert. [Sta20]

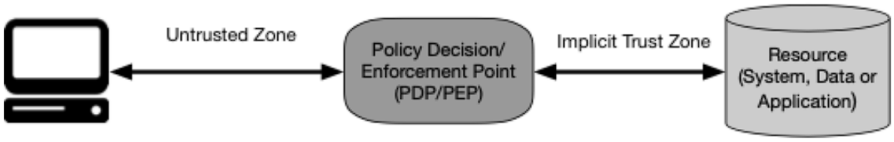



Abbildung 2.4: Darstellung des abstrakten Zugriffsmodells von Zero-Trust. Ein Computer (Subjekt) kommuniziert über die ungesicherte Zone mit dem Policy Decision Point und dem Policy Enforcement Point, die den Zugang zur Ressource in der impliziten Vertrauenszone steuern. Das PDP und PEP treffen Entscheidungen, um dem Subjekt den Zugriff zu gewähren. [Sta20]

Abbildung 2.4 illustriert das abstrakte Zugriffsmodell von Zero-Trust, bei dem ein *Subjekt* (dargestellt als Computer) aus der ungesicherten Zone mit dem PDP und PEP kommuniziert, die den Zugang zur geschützten Vertrauenszone kontrollieren. [Sta20]

Das System muss sicherstellen, dass das *Subjekt* authentisch ist und die Anfrage valide. Die Rolle von PDP und PEP besteht darin, angemessene Entscheidungen zu treffen, um dem Subjekt den Zugriff auf die Ressource zu ermöglichen. [Sta20]

#### Grundsätze

Das National Institute of Standards and Technology (NIST) hat 2020 sieben zentrale Grundsätze für Zero-Trust definiert, die wichtige Aspekte der Sicherheitsstrategie umfassen. Zunächst müssen *alle Geräte klassifiziert* werden, um unternehmenseigene von persönlichen Geräten zu unterscheiden. Jede *Kommunikation muss unabhängig von der Netzwerkadresse abgesichert* sein, da Vertrauen nicht auf der Netzwerkinfrastruktur basieren darf. Der Zugang zu Ressourcen sollte stets nach dem *Prinzip der minimalen Privilegien* erfolgen, wobei jede Authentifizierung und Autorisierung einzeln überprüft wird. Dynamische Richtlinien, die auf den Attributen des anfragenden Objekts basieren, sollen den Zugriff auf Ressourcen steuern. Ein *kontinuierliches Diagnose- und Schadensbegrenzungssystem* (CDM) überwacht die Sicherheitslage und ermöglicht bei Bedarf Anpassungen, wie etwa das Einspielen von Patches. Zur Erhöhung der Sicherheit empfiehlt das NIST den Einsatz von *Identitäts-, Berechtigungs- und Zugriffsmanagement*, einschließlich Multifaktor-Authentifizierung (MFA), sowie die *kontinuierliche Überwachung und Neubewertung* von Zugriffsrechten. Schließlich sollen *Daten über den Sicherheitsstatus, den Netzwerkverkehr und Zugriffsanfragen* gesammelt werden, um Richtlinien zu optimieren und anzupassen. [Sta20]

### 2.2.1 Zero-Trust-Architekturen

Die in den Zero-Trust-Grundsätzen definierten Sicherheitsansätze werden in der Praxis durch Zero-Trust-Architekturen (ZTA) umgesetzt. Diese Architekturen bestehen aus mehreren logischen Komponenten, die in Abbildung 2.5 dargestellt sind. Wesentlich ist dabei die Aufteilung des Policy Enforcement Point (PEP) und des Policy Decision Point (PDP) in zwei getrennte Ebenen: die Data Plane und die Control Plane.

Das PEP empfängt und leitet eingehende Anfragen an den Policy Administrator und die Policy Engine weiter. Die Policy Engine trifft Entscheidungen darüber, ob ein Subjekt den angeforderten Zugriff erhalten soll, während der Policy Administrator für den Aufbau oder Abbruch der Verbindung sorgt, indem er entsprechende Befehle an das PEP sendet. [Sta20] Zur Entscheidungsfindung greift die Policy Engine auf verschiedene Datenquellen zurück, darunter Identity Management Systeme, Continuous Diagnostics and Mitigation (CDM)

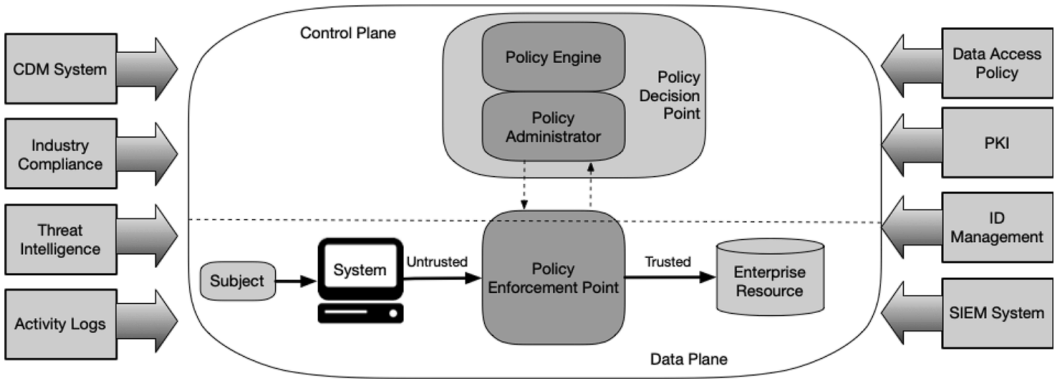



**Figure 2: Core Zero Trust Logical Components**

Abbildung 2.5: Unterteilung der Zero-Trust-Architektur in mehrere logische Komponenten. Der Policy Enforcement Point und der Policy Decision Point werden in zwei getrennten Ebenen dargestellt: Data und Control Plane. Weitere Datenquellen, wie ein Identity Management System, Continous Diagnostics und Mitigation (CDM) Systeme oder auch Access Logs. Dadurch ist es möglich, Entscheidungen auf einer breiteren Informationsgrundlage zu treffen. [Sta20]

Systeme und Access Logs. Diese zusätzlichen Datenquellen ermöglichen es, Entscheidungen auf einer breiteren Informationsgrundlage zu treffen. [Sta20] Die konkrete Implementierung einer Zero-Trust-Architektur hängt jedoch stark vom spezifischen Anwendungsfall und den Unternehmensanforderungen ab. Dies führt zu einer Vielzahl von Architekturvariationen, die in der Praxis Anwendung finden. [Sta20]

### 2.2.2 Variationen von Zero-Trust-Architektur-Ansätzen

Obwohl alle Zero-Trust-Ansätze die in Abschnitt 2.2 beschriebenen sieben Grundprinzipien befolgen, setzen verschiedene Ansätze unterschiedliche Schwerpunkte. Eine umfassende Zero-Trust-Lösung sollte Elemente aller folgenden Ansätze enthalten. Allerdings eignet sich nicht jeder Ansatz gleichermaßen für jeden Anwendungsfall. Bestehende Unternehmensrichtlinien und spezifische Anforderungen können dazu führen, dass ein Ansatz schwerer umzusetzen ist oder grundlegende Veränderungen in der Geschäftsabwicklung erfordert. [Sta20]

#### Erweiterte Identitätsverwaltung

Dieser Ansatz verwendet die Identität der Akteure als zentrale Komponente zur Erstellung von Zugriffsrichtlinien. Diese Richtlinien basieren in erster Linie auf der Identität und den zugewiesenen Attributen der Subjekte. Der Ressourcenzugang hängt daher primär von den Zugriffsrechten ab, die einem Subjekt gewährt werden. Weitere Faktoren wie das verwendete

Gerät, der Gerätestatus und die Umgebung können zusätzliche Anforderungen darstellen und das Ergebnis beeinflussen. [Sta20]

Ressourcen oder PEP-Komponenten, die den Ressourcenschutz gewährleisten, müssen die Fähigkeit besitzen, Anfragen an eine Policy Engine weiterzuleiten oder die Authentifizierung und Genehmigung eines Subjekts durchzuführen, bevor der Zugriff gewährt wird. [Sta20]

Dieser identitätsbasierte Ansatz eignet sich besonders für Cloud-basierte Anwendungen und Dienste, die möglicherweise keine eigenen Zero-Trust-Sicherheitskomponenten verwenden können, wie beispielsweise Software as a Service (SaaS)-Angebote. [Sta20]

#### Micro-Segmentation

Bei diesem Ansatz werden Infrastrukturgeräte wie intelligente Switches, Next-Generation Firewalls (NGFWs) oder spezielle Gateway-Geräte, die als PEPs fungieren, zum Schutz einzelner Ressourcen oder Ressourcengruppen eingesetzt. Die Schutzgeräte agieren als PEP, während ihre Verwaltung als Policy Engine und Policy Administrator fungiert. Dieser Ansatz ist vielseitig und eignet sich für verschiedene Anwendungsfälle und Einsatzmodelle. [Sta20]

Um vollständig zu funktionieren, erfordert dieser Ansatz ein Identity-Governance-Programm (IGP), stützt sich jedoch stark auf die Gateway-Komponenten, die als PEPs fungieren und die Ressourcen vor unbefugtem Zugriff oder Entdeckung schützen. Die Fähigkeit, Bedrohungen oder Änderungen im Workflow zu erkennen und die Verwaltung der PEP-Komponenten zu gewährleisten, ist hierbei entscheidend. [Sta20]

#### Netzwerk-Infrastruktur und Software Defined Perimeters

Dieser Ansatz nutzt die Netzwerkinfrastruktur zur Implementierung einer Zero-Trust-Architektur, oft durch den Einsatz eines Overlay-Netzwerks, das sowohl auf Layer 7 als auch darunter implementiert werden kann. Häufig wird in diesem Zusammenhang von Software Defined Perimeters (SDP) gesprochen, die Konzepte des Software Defined Networking (SDN) und des intent-based Networking (IBN) integrieren. [Sta20]

Der Policy Administrator fungiert hierbei als Netzwerk-Controller, der das Netzwerk basierend auf den Entscheidungen der Policy Engine konfiguriert und rekonstruiert. Der PEP, der weiterhin von Clients angefragt wird, wird von der PA-Komponente verwaltet. [Sta20]

Wenn dieser Ansatz auf der Anwendungsschicht (Layer 7) implementiert wird, ist das Agent/Gateway-Modell das gebräuchlichste Einsatzszenario. Hierbei etablieren der Agent und das Ressource-Gateway einen sicheren Kommunikationskanal zwischen Client und Ressource. [Sta20]

### 2.2.3 Variationen in der Praxis

Die oben beschriebenen Komponenten einer Zero-Trust-Architektur sind logische Komponenten. Dies bedeutet, dass sie nicht zwangsläufig durch einzelne Systeme repräsentiert werden müssen. Eine Ressource kann mehrere logische Komponenten gleichzeitig übernehmen, während eine einzelne logische Komponente aus mehreren Hardware- und Software-Elementen bestehen kann, die gemeinsam ihre Funktion erfüllen. [Sta20]

Im Folgenden werden einige praxisnahe Variationen der Implementierung ausgewählter Zero-Trust-Architektur-Komponenten beschrieben. Je nach Struktur des Unternehmensnetzwerks können mehrere Zero-Trust-Architektur-Einsatzmodelle gleichzeitig angewendet werden. [Sta20]

#### Geräte-Agent- oder Gateway-basierter Einsatz

Das Geräte-Agent/Gateway-basierte Modell unterteilt den PEP, wie in Abbildung 2.6 dargestellt, in zwei Komponenten. Der Agent ist eine Softwarekomponente, die auf einem Enterprise-System installiert ist, während das Gateway die Datenressource schützt. [Sta20]

Der Agent leitet Anfragen an den Policy Administrator weiter, der diese an die Policy Engine zur Evaluierung übermittelt. Wenn die Anfrage autorisiert wird, richtet der Policy Administrator einen Kommunikationskanal zwischen dem Geräte-Agenten und dem relevanten Ressourcen-Gateway über die Control Plane ein. Danach erfolgt die verschlüsselte Kommunikation zwischen dem Geräte-Agenten und dem Gateway, bis der Workflow abgeschlossen ist oder ein Sicherheitsereignis die Verbindung beendet. [Sta20]

#### Enklaven-basierter Einsatz

Eine Variante des Geräte-Agent/Gateway-Modells ist der Enklaven-basierte Einsatz, bei dem sich die Gateway-Komponente nicht direkt auf oder vor einzelnen Ressourcen befindet, sondern an der Grenze einer Ressourcenenklave, wie in Abbildung 2.7 dargestellt.

Dieses Modell eignet sich besonders für Unternehmen, die cloudbasierte Microservices für spezifische Geschäftsprozesse nutzen, wobei die gesamte private Cloud hinter einem Gateway

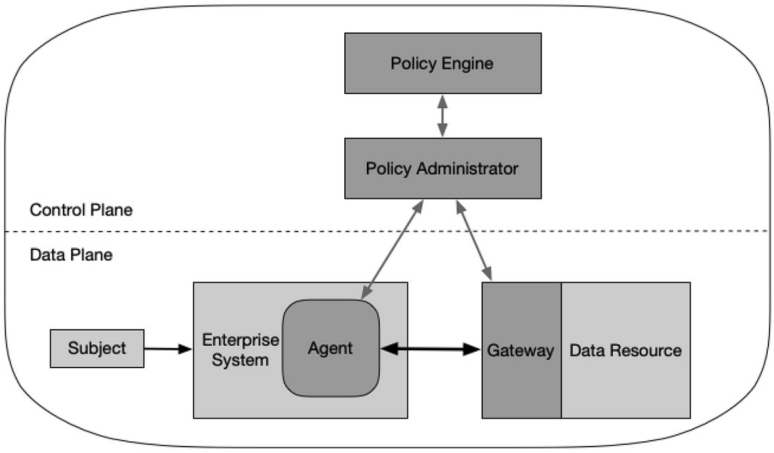



**Figure 3: Device Agent/Gateway Model**

Abbildung 2.6: Geräte-Agent/Gateway-basiertes Modell: Bei diesem Model wird das PEP in zwei Komponenten unterteilt: Agent, eine Software-Komponente, die auf dem Enterprise-System installiert ist, während das Gateway die Datenressource schützt. Beide kommunizieren mit dem Policy Decision Point (PDP), der in die beiden Bestandteile Policy Engine und Administrator zerlegt ist. [Sta20]

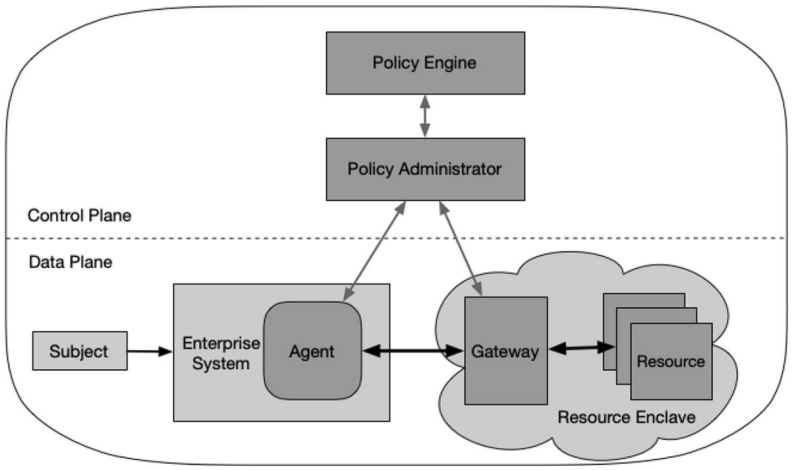



**Figure 4: Enclave Gateway Model**

Abbildung 2.7: Enklaven-basiertes Modell: Ist eine Variation des Geräte-Agent/Gateway-Modells. Die Gateway-Komponente befindet sich nicht direkt auf oder vor einzelnen Ressourcen, sondern an der Grenze einer Ressourcenenklave. Die Kommunikation findet überwiegend zwischen dem Agenten und dem Policy Decision Point (unterteilt in Policy Engine und Administrator) statt. [Sta20]

positioniert ist. Es bietet eine effektive Möglichkeit, Ressourcen mit einer gemeinsamen Geschäftsfunktion zu schützen, die nicht direkt mit dem Gateway kommunizieren können. [Sta20]

### Portalgestützte Bereitstellung von Ressourcen

In diesem Modell fungiert das PEP als Gateway, das als Portal für Subjekt-Anfragen dient. Dieses Gateway-Portal kann eine einzelne Ressource oder eine Sammlung von Ressourcen schützen, die für eine bestimmte Geschäftsfunktion benötigt werden, wie in Abbildung 2.8 dargestellt.

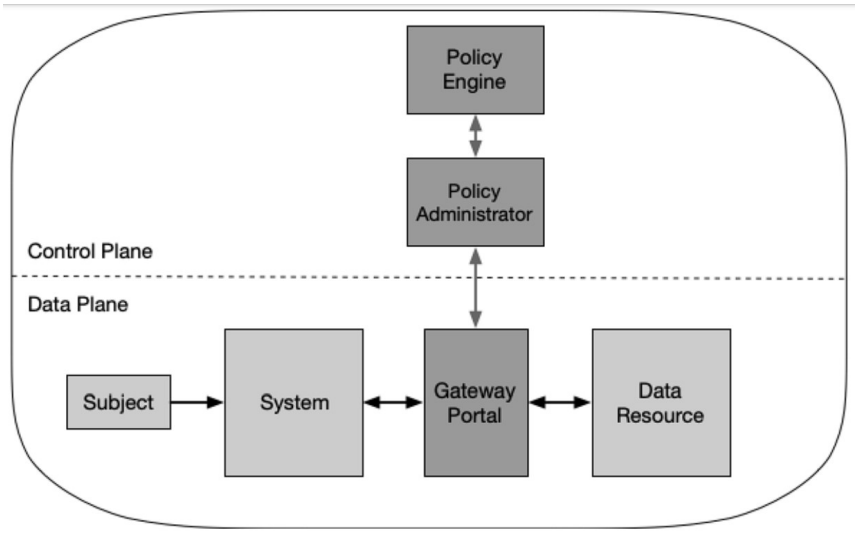



**Figure 5: Resource Portal Model**

Abbildung 2.8: Portalgestützte Bereitstellung von Ressourcen: Hierbei fungiert der Policy Enforcement Point als Gateway Portal. und kann so eine Sammlung von Ressourcen schützen, die für eine bestimmte Geschäftsfunktion benötigt werden. [Sta20]

Der Vorteil dieses Modells liegt in der zentralen Verwaltung und dem Schutz von Legacy-Anwendungen oder Sammlungen von Ressourcen, ohne dass clientseitige Agenten erforderlich sind. Allerdings fehlen dabei detaillierte Informationen über das Subjekt, das den Zugang anfordert, was die kontinuierliche Überwachung und Analyse des Clients erschwert. [Sta20]

### Sandboxing von Geräteanwendungen

Bei diesem Modell werden die Anwendungen in Sandboxes bereitgestellt, die die darunterliegende OS-Schicht abkapseln. Diese Sandbox führt Anwendungen isoliert aus, wodurch verhindert wird, dass schädliche Aktionen des Clients direkt auf die Produktionsumgebung zugreifen können. [Sta20] Die Architektur für das Sandboxing von Geräteanwendungen ist in Abbildung 2.9 dargestellt

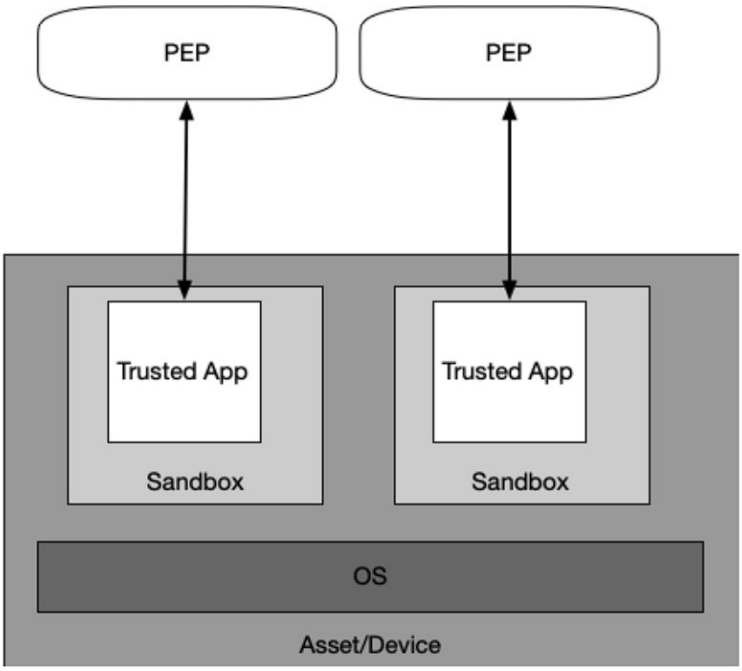



**Figure 6: Application Sandboxes**

Abbildung 2.9: Sandboxing von Geräteanwendungen: Die Anwendungen werden in Sandboxes bereitgestellt, die die darunterliegende OS-Schicht weiter abkapseln und kommunizieren direkt mit dem PEP. Durch die Sandbox wird verhindert, dass schädliche Aktionen des Clients direkt auf die Produktionsumgebungen zugreifen können. [Sta20]

### 2.2.4 Zusammenfassung der ZTA-Variationen

Die Zero-Trust-Architektur bietet zahlreiche Ansätze zur Implementierung. Die verschiedenen Modelle zeigen, wie flexibel Zero-Trust-Architekturen sein können und wie sie an unterschiedliche Netzwerkanforderungen und Sicherheitsniveaus angepasst werden können. Während alle Ansätze dasselbe Ziel verfolgen – den Schutz der Netzwerkressourcen und die Minimierung von Sicherheitsrisiken – unterscheiden sie sich in ihrer Implementierung und ihrem Anwendungsbereich. Eine Kombination mehrerer Ansätze, angepasst an die spezifischen Bedürfnisse und Strukturen eines Unternehmens, wird oft als die effektivste Methode zur Umsetzung einer Zero-Trust-Strategie betrachtet. [Sta20]
Die in diesem Kapitel beschriebenen Grundlagen und Variationen der Zero-Trust-Architekturen bieten eine Grundlage für die praktische Implementierung in Kapitel 5. Sie ermöglichen es, die verschiedenen Ansätze zu bewerten und an die spezifischen Anforderungen eines Unternehmens anzupassen.

## 2.3 Service Mesh

Ein Service Mesh ist eine spezialisierte Infrastrukturschicht, die für die Verwaltung und Absicherung der Kommunikation zwischen Microservices konzipiert wurde. Es unterstützt die Zero-Trust-Prinzipien, indem es die Interaktion zwischen Diensten in einer Cloud-nativen Umgebung zuverlässig und sicher gestaltet. Ein Service Mesh überwacht den Datenverkehr zwischen Diensten, authentifiziert die Identität der Dienste und setzt Richtlinien zur Zugriffskontrolle durch. Diese Eigenschaften bilden die Grundlage für die in Kapitel 5 implementierte und evaluierte Zero-Trust-Architektur, die auf den hier beschriebenen Mechanismen aufbaut, um eine sichere und flexible Microservice-Umgebung zu schaffen.

Die folgende Definition des Service Mesh von William Morgan beschreibt präzise, was ein Service Mesh ist:

> "A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware." [cnc]

Diese Definition verdeutlicht, dass ein Service Mesh eine spezielle Infrastrukturebene für die Kommunikation zwischen Diensten darstellt. In der Praxis besteht es aus einer Reihe von leichtgewichtigen Netzwerk-Proxies, die neben dem Anwendungscode bereitgestellt werden, ohne dass die Anwendungen selbst Kenntnis über die Proxies haben. Eine beispielhafte Darstellung einer solchen Service-Mesh-Architektur ist in Abbildung 2.10 zu sehen. [cnc]

### 2.3.1 Komponenten und Funktionen eines Service Meshes

Ein Service Mesh besteht aus zwei wesentlichen Komponenten: der Data Plane und der Control Plane.

- **Data Plane:** Diese Ebene umfasst eine Vielzahl intelligenter Proxys, die oft als Sidecars eingesetzt werden und den gesamten Netzwerkverkehr zwischen den Microservices kontrollieren und vermitteln. Sie übernimmt Aufgaben wie Service Discovery, Health Checking, Routing, Load Balancing, Authentifizierung, Autorisierung und Beobachtbarkeit. [LLG+19]

- **Control Plane:** Die Control Plane ist für die Verwaltung und Konfiguration dieser Proxys zuständig und kümmert sich um Routing sowie weitere Aufgaben. Sie erzwingt

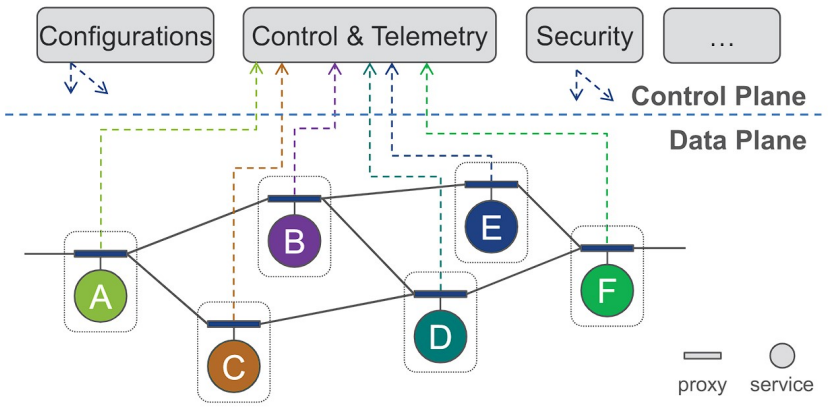



Fig. 1: An architectural overview example of the Service Mesh approach.

Abbildung 2.10: Beispielhafte Darstellung der Service-Mesh-Architektur bestehend aus Services A bis F und den dazugehörigen Sidecar-Proxies. Auf der Control Plane werden Funktionalitäten für Verwaltung, Telemetrie, Konfigurationen und Sicherheit bereitgestellt, die an die einzelnen Service-Proxys kommuniziert werden. Die Kommunikation zwischen den Services findet zwischen den entsprechenden Proxies statt. [LLG+19]

Richtlinien und sammelt Telemetriedaten, ohne selbst direkt in den Netzwerkverkehr einzugreifen. [LLG+19]

Durch die Trennung der Geschäftslogik von den Richtlinien für Sicherheit und Beobachtbarkeit bietet ein Service Mesh eine klare Struktur für die Verwaltung der Microservice-Kommunikation. Dies ermöglicht eine automatische Erkennung und Interaktion der Dienste untereinander und unterstützt fortschrittliche Bereitstellungsstrategien wie *Blue/Green-Deployments* und *Rolling Upgrades*.

Ein Service Mesh bietet folgende Kernfunktionen:

- **Service discovery:** Die Vielzahl an Service-Instanzen sowie sich dynamisch ändernde Zustände und Standorte innerhalb von Microservice-Anwendungen erfordern einen Mechanismus zur Diensterkennung, um es den Dienstnutzern zu ermöglichen, den Standort zu ermitteln und Anfragen an eine sich dynamisch verändernde Menge von Dienstinstanzen zu richten. [LLG+19]

- **Load Balancing:** Load Balancer in einem Service Mesh ermöglichen das Routing durch das Netzwerk. Im Vergleich zu einem einzelnen Routing-Mechanismus können moderne Service-Mesh-Load Balancer Latenz und den Status der Backend-Instanzen berücksichtigen. [LLG+19]

- **Fehlertoleranz:** Service Meshes sind meist oberhalb der TCP/IP-Schichten angesiedelt. Mit der Annahme, dass das Netzwerk auf Layer 3 und Layer 4 unzuverlässig ist, muss ein Service Mesh in der Lage sein, Fehler zu behandeln. Dies wird üblicherweise durch die Weiterleitung der Anfrage an eine funktionierende Service-Instanz erreicht. [LLG+19]

- **Verkehrsüberwachung:** Der gesamte Verkehr zwischen den Microservices wird überwacht, was die Erstellung von Berichten über Anfragen pro Zeiteinheit, Latenz-Metriken sowie Erfolgs- und Fehlerquoten ermöglicht. [LLG+19]

- **Unterbrechung der Verbindung:** Wird auf einen bereits überlasteten Service zugegriffen, hält Circuit Breaking die Anfragen zurück, anstatt den Dienst durch übermäßige Last komplett zum Ausfall zu bringen. [LLG+19]

- **Authentifizierung und Zugriffskontrolle:** Durch Richtlinien, die von der Control Plane durchgesetzt werden, kann ein Service Mesh festlegen, welche Dienste von welchen anderen Diensten angefragt werden können, sowie welche Art von Verkehr unauthentifiziert stattfindet und verhindert werden sollte. [LLG+19]

### 2.3.2 Funktionsweise und Anwendung

Die grundlegende Funktionsweise eines Service Mesh lässt sich wie folgt beschreiben:

1. Das Service Mesh nutzt dynamisches Routing, um herauszufinden, welchen Service der Anfragende aufrufen möchte. [cnc]

2. Hat das Service Mesh das richtige Ziel gefunden, zieht es alle Instanzen des relevanten Service-Endpunktes heran. [cnc]

3. Anschließend wird die Instanz ausgewählt, die am wahrscheinlichsten eine schnelle Antwort liefert. Diese Auswahl basiert auf einer Reihe von Faktoren, wie beispielsweise zuvor beobachteten Latenzzeiten der letzten Anfragen. [cnc]

4. Der Request wird an die ausgewählte Instanz gesendet, wobei die Latenz und die Antworttypen des Ergebnisses gespeichert werden. [cnc]

5. Ist die Instanz nicht erreichbar, antwortet sie nicht oder schlägt bei der Verarbeitung des Prozesses fehl, wird der Request zu einer anderen Instanz (sofern der Request idempotent ist) weitergeleitet. [cnc]

6. Zeigt eine Instanz regelmäßig Fehler, schließt das Service Mesh diese aus dem Load Balancing-Pool aus und versucht es später periodisch erneut. [cnc]

7. Läuft die Deadline für die Anfrage ab, lässt das Service Mesh diese proaktiv scheitern, anstatt zusätzliche Last durch weitere Versuche zu erzeugen. [cnc]

8. Alle Aspekte werden gespeichert, um Metriken und verteiltes Verfolgen von Anfragen zu ermöglichen. [cnc]

Die Auswahl an Service-Mesh-Implementierungen ist groß. Nach [LLG+19] sind für Cloud-Native-Projekte vor allem Istio, Linkerd, Amazon App Mesh und Airbnb Synapse attraktiv. Istio und Linkerd sind Cloud Native Computing Foundation Projekte [LLG+19], wobei Istio zunehmend das Tool der Wahl ist, um Beobachtbarkeit, betriebliche Agilität und richtliniengesteuerte Sicherheit in Cloud-nativen Kubernetes-Clustern zu gewährleisten [Iye]. Daher wird in dieser Arbeit das Service Mesh durch Istio implementiert.

### 2.3.3 Einordnung in die Zero-Trust-Architektur

Ein Service Mesh lässt sich am ehesten dem Enklaven-basierten Model innerhalb der Zero-Trust-Architektur zuordnen. Dieses Modell fokussiert sich darauf, eine geschützte Umgebung zu schaffen, in der Dienste sicher miteinander interagieren. Ein Service Mesh implementiert Funktionen wie Service Discovery, Load Balancing, Traffic Management, Sicherheit und Überwachung, die in vielerlei Hinsicht den Aufgaben eines Enklavengateway in einer Zero-Trust-Architektur entsprechen. [cnc, LLG+19, Sta20]

Das Device Agent/Gateway Model und das Ressource-Portal Model fokussieren sich hingegen auf die Sicherung und Überwachung von Endgeräten und den Zugang zu Ressourcen über zentrale Zugangspunkte und sind daher nicht direkt mit den Aufgaben eines Service Meshes vergleichbar. [cnc, LLG+19, Sta20]

### 2.3.4 Zusammenfassung

In diesem Kapitel wurde das Konzept des Service Meshes als wichtige Infrastrukturschicht zur Unterstützung der Zero-Trust-Prinzipien vorgestellt. Ein Service Mesh ermöglicht die sichere und zuverlässige Kommunikation zwischen Microservices in einer Cloud-nativen Umgebung. Es implementiert verschiedene Funktionen wie Service Discovery, Load Balancing, Fehlertoleranz, Verkehrsüberwachung, Circuit Breaking sowie Authentifizierung und Zugriffskontrolle.

Die Struktur eines Service Meshes, bestehend aus der Data Plane und der Control Plane, sowie seine Funktionsweise und Anwendung wurden detailliert erläutert. Ein Service Mesh erfüllt eine ähnliche Rolle wie das Enklaven-basierte Model in einer Zero-Trust-Architektur, indem es den Datenverkehr zwischen Diensten sichert und überwacht.

Diese Grundlagen bieten eine solide Basis für die Implementierung einer Zero-Trust-Architektur, die in Kapitel 5 weiter ausgeführt wird.

# 3 Stand der Wissenschaft und Technik

In diesem Kapitel wird die bestehende Forschung zur Effektivität von Zero-Trust-Architekturen in Kubernetes Clustern untersucht. Ziel ist es, einen umfassenden Überblick über die bisher erzielten Erkenntnisse zu geben, die Ansätze und Methoden zu vergleichen, sowie ungeklärte Fragen und Forschungslücken zu identifizieren. Diese Analyse bildet die Grundlage für die vorliegende Arbeit.

Die vorhandene Literatur wird in die folgenden vier Hauptbereiche unterteilt:

3.1 **Schwachstellen in Kubernetes Clustern**: Es werden Forschungsergebnisse vorgestellt, die sich mit den Schwachstellen in Kubernetes Clustern befassen. Zudem wird begründet, warum Zero-Trust in Kubernetes Clustern notwendig ist.

3.2 **Sicherheitsanalysen von Kubernetes-Clustern**: Studien und Ausarbeitungen zur Verbesserung der Sicherheit von Kubernetes Clustern werden betrachtet.

3.3 **Zero-Trust-Modelle und Architekturen**: Dieser Abschnitt untersucht die Literatur zu den Prinzipien und Implementierungen von Zero-Trust-Modellen in verschiedenen Kontexten, insbesondere im Cloud-Computing.

3.4 **Zero-Trust-Architekturen in Kubernetes Clustern**: Studien und wissenschaftliche Arbeiten zu Zero-Trust-Architekturen in Kubernetes Clustern werden vorgestellt und evaluiert. Außerdem wird ein Blick auf vorhandene Arbeiten zur Effektivität von Zero-Trust in Kubernetes Clustern geworfen.

## 3.1 Schwachstellen in Kubernetes Clustern

In diesem Bereich werden Forschungsergebnisse zu Schwachstellen in Kubernetes Clustern vorgestellt.

Mit der wissenschaftlichen Forschungsarbeit haben Zeng et al. in 2023 [ZKL+23] einen Beitrag zu den Sicherheitsrisiken des Container-Orchestrators Kubernetes geleistet. Die Autoren erläutern zunächst einige Sicherheitsbedenken und Schwachstellen im Zusammenhang

mit Containern. Die Autoren betonen wesentliche Sicherheitsempfehlungen, darunter die Vermeidung von Verbindungen zwischen Containern, um die Sicherheit zu gewährleisten. Sie heben außerdem hervor, dass der Schutz der Host-Ressourcen entscheidend ist, um das Risiko von Privilegienausweitung oder DoS-Angriffen zu minimieren. Um die Sicherheit von Containern und deren Orchestrierung zu erhöhen, diskutieren die Autoren verschiedene Lösungen und Werkzeuge. Sie schlagen unter anderem vor, sichere Netzwerke aufzubauen und darauf zu achten, dass die Container keine unnötigen Host-Ressourcen nutzen. Dabei zeigen sie auch spezifische Technologien wie Calico und Cilium, die dabei helfen können. Die Autoren bringen auch verschiedene Sicherheitsaspekte mit Docker, Kubernetes und Istio in Verbindung. Es werden potenzielle Schwachstellen dargelegt und darauf basierende Lösungsansätze aufgezeigt.

Minna und Chandasekaran (2021) beschäftigen sich in [MBR+21] mit den Sicherheitsrisiken von Kubernetes-basierten Netzwerkkonfigurationen. Die Autoren diskutieren verschiedene Bedrohungen, die sich aus der Implementierung von Kubernetes ergeben. Dabei gehen sie auf spezifische Angriffstechniken und mögliche Verteidigungsstrategien ein. Die Arbeit stellt auch den Zusammenhang zwischen Kubernetes-spezifischen Angriffen und dem allgemeineren ATT&CK-Framework dar, wobei die jeweiligen Angriffstechniken dargestellt und mögliche Lösungsansätze diskutiert werden.

Die Masterarbeit von Mytrilinakis aus dem Jahr 2020 [Myt20] untersucht verschiedene Angriffsmöglichkeiten auf Kubernetes und empfiehlt Abwehrmechanismen. Die Arbeit geht dabei auf die Stuktur von Kubernetes und dessen Objekte ein, die für die Beschreibung des gewünschten Zustands eines Clusters erforderlich sind. Im Rahmen dieser Arbeit wurden verschiedene Angriffe auf ein Kubernetes-Cluster durchgeführt, die in vier Kategorien eingeteilt werden können. Hierbei handelt es sich um Netzwerkangriffe, Scanning von Containern vor dem Hochladen in eine Registry, AppArmor- und Seccomp-Profile sowie Prozessisolation und Netzwerkkonfiguration in Containern. Es werden auch mögliche Gegenmaßnahmen vorgestellt, wie der Einsatz von Network Policies, Pod Security Policies und AppArmor-Profilen, um Angriffe auf das Kubernetes-Cluster zu verhindern. Diese Gegenmaßnahmen werden in Kapitel 5 aufgegriffen, wenn es um die Implementierung von Security Best Practices des Clusters geht.

Die vorgestellten Forschungen zu Schwachstellen in Kubernetes liefern wichtige Erkenntnisse über die Angriffsvektoren sowie die Schwachstellen in Design und Architektur von Kubernetes Clustern. Diese Erkenntnisse fließen in die vorliegende Arbeit ein, um eine Sicherheitsanalyse für Kubernetes Cluster durchzuführen und auf dieser Grundlage die Hauptrisikobereiche des Clusters herauszuarbeiten.

## 3.2 Sicherheitsanalysen von Kubernetes-Clustern

Die Basis für die Untersuchung von Zero-Trust-Architekturen in Kubernetes-Clustern bilden die umfangreichen Forschungen zu den Sicherheitsmechanismen in Kubernetes.

Eine Arbeit von Shamim et al. aus dem Jahr 2020 [SBR20] beantwortet die Forschungsfrage nach den Kubernetes Security Practices, die von Anwendern gemeldet werden. Die Autoren führen eine graue Literaturübersicht durch, bei der es sich um Literatur handelt, die nicht vom kommerziellen Verlagswesen kontrolliert und nicht in der Buchhandlung erhältlich ist, und wenden qualitative Analysen auf 104 Internet-Artefakte an, um Sicherheitspraktiken zu identifizieren. Dabei wurden elf Sicherheitspraktiken identifiziert, die die Sicherheit des Clusters weiter erhönen. Darunter finden sich Role Based Access Control, Schwachstellen-Scans, Netzwork Polcies Shahim et al. [SBR20] kommen letztlich zu dem Schluss, dass die effektive und sichere Nutzung von Kubernetes die Implementierung von Sicherheitspraktiken auf mehreren Ebenen des Clusters (Container, Pods, Datenbanken, Netzwerk) erfordert. Weiterhin kann die Systematisierung des Wissens über Sicherheitspraktiken als Benchmark für Praktiker und für zukünftige Forschungen im Bereich Kubernetes-Sicherheit dienen. Die Arbeit von Shahim et al. [SBR20] hebt die Bedeutung der Sicherung von Kubernetes-Installationen hervor und bietet eine Grundlage für weitere Untersuchungen zu Sicherheitskonfigurationen und deren Wirksamkeit.

Eine Arbeit von Darwesh et al. (2020) [DHA22] identifiziert zehn Sicherheits-Best-Practices, die in Kubernetes Clustern umgesetzt werden sollten. Die Autoren beleuchten dabei vier wesentliche Sicherheitsherausforderungen: die Kompromittierung der Control Plane, die Kompromittierung von Pods und Knoten, die Sicherung der Netzwerkverbindungen und die Container-Sicherheit. Um diesen Herausforderungen zu begegnen, empfehlen sie die Umsetzung der folgenden Sicherheitspraktiken: Authentifizierung und Autorisierung, Nutzung eines privaten Kubernetes API-Endpunkts, Netzwerk-Richtlinien, pod-spezifische Richtlinien, Audit-Logging, Trennung von Namespaces, Isolierung von Kubernetes-Knoten durch private Netzwerke und Beschränkung des SSH-Zugangs, regelmäßige Aktualisierung der Kubernetes-Version, Verschlüsselung und Zugangsbeschränkung zu etcd, Ressourcenbegrenzung sowie SSL/TLS-Verschlüsselung. Darwesh et al. [DHA22] betonen, dass die Standardkonfiguration häufig unsicher ist und daher die Sicherheit in Kubernetes Clustern von zentraler Bedeutung ist.

Budigiri et. al. (2021) [BBM+21] beschreiben in ihrer Arbeit Network Policies in Kubernetes: Performance Evaluation and Security Analysis die Funktionsweise von Netzwerk-Richtlinien (NetworkPolicies) in Kubernetes. Sie stellen in ihrer Arbeit zwei wesentliche Container Networking Interfaces (CNI) Plugins vor, deren Leistungs-overhead sie vergleichen und die Sicherheit der Policies bewerten. Hinsichtlich der Skalierbarkeit und des Overheads konnten Sie in ihrem Testmodell keinerlei signifikaten Einschränkungen feststellen, selbst bei einer hohen Anzahl an Policies. Calico, dass auch Bestandteil dieser Arbeit ist, arbeitet mit einer

dezentralen Implementierung, bei der nur relevante Policies an der veth-Schnittstelle des Pods ausgewertet werden. Bei der Bewertung der Sicherheit, stellen die Autoren Budigiri et. al. klar, dass NetworkPolicies für die Verringerung von Angriffsflächen sehr effektiv sind, jedoch nicht alle Angriffe abwehren können, da Angreifer weiterhin zugelassene Verbindungen missbrauchen können. Schwachstellen von NetzwerkPolicies sind Fehlkonfigurationen, schwache Policy-Design, Bedrohungen von Tenant-Administratoren, privilegierte Netzwerke und Schwachstellen in der Implementierung. Als Lösung zur Minimierung der Schwachstellen nennen die Autoren Lösungswege aus der Literatur. Die Arbeit hebt die Notwendigkeit weiterer Forschung zur Verringerung von Performance-Overheads und zur formalen Überprüfung von Netzwerkpolicies hervor.[BBM+21]

Die vorgestellten Forschungen stellen die wichtigsten Sicherheitsmechanismen für Kubernetes Cluster vor. Während Shamim et al. [SBR20] und Darwesh et al. [DHA22] zehn bzw. elf Sicherheitsmechanismen herausstellen, beschäftigen sich Budigiri et al. [BBM+21] konkret mit dem Einsatz von NetworkPolicies in Kubernetes und stellen heraus, dass diese als effektives Mittel zur Schließung von Angriffsvektoren genutzt werden können. Somit werden die Ergebnisse der vorgestellten Arbeiten sich in den Security Best Practices für Kubernetes Cluster wiederfinden.

## 3.3 Zero-Trust-Modelle und -Architekturen

Dieser Hauptbereich untersucht Literatur, die sich mit den Prinzipien und Implementierungen von Zero-Trust-Modellen beschäftigt.

Buck et al. (2021) [BOS+21] führten eine umfassende Literaturübersicht zum aktuellen Stand der Forschung im Bereich Zero-Trust-Sicherheitsmodelle durch und identifizierten dabei wesentliche Forschungslücken. Die Autoren analysierten sowohl akademische als auch praxisorientierte Veröffentlichungen im Rahmen einer multivokalen Literaturübersicht. Aus dieser Analyse entwickelten sie einen Forschungsrahmen für Zero-Trust, der die vorhandene Literatur strukturiert und zukünftige Forschungsrichtungen aufzeigt. Die Ergebnisse verdeutlichen, dass sich die akademische Literatur überwiegend auf die Architektur und Leistungsverbesserungen von Zero-Trust konzentriert, während praxisorientierte Veröffentlichungen eher organisatorische Vorteile und mögliche Migrationsstrategien thematisieren. Die Arbeit bietet eine solide Grundlage für zukünftige Forschungen im Bereich Zero-Trust und betont die Notwendigkeit einer umfassenden Untersuchung, die technische, wirtschaftliche und nutzerbezogene Perspektiven integriert. [BOS+21]

Eine vergleichende Review von Sarkar et al. [SCS+22] aus dem Jahr 2022 zur Sicherheit von Zero-Trust-Netzwerken in Cloud-Computing-Systemen stellt verschiedene Anforderungen von Forschungsmodellen für Zero-Trust-Cloud-Netzwerken dar und vergleicht diese miteinander. Hierfür wuerden neun unterschiedliche Parameter in drei Haupttypen eingeteilt.

Der Schwerpunkt der Rezension liegt auf der Unterscheidung von anforderungsspezifischen Merkmalen, die notwendig sind, um interne und externe Cyberbedrohungen zu hemmen, die Sichtbarkeit des Netzwerks zu erhöhen und Vertrauen innerhalb des Netzwerks zu generieren. Die dabei untersuchten Technologien zur Modellierung von Zero-Trust-Netzwerken in Cloud-Computing-Systemen umfassen Token-basierte Authentifizierung, automatisierte Bedrohungsabwehr, Identitätsmanagement, Vertrauensbewertungssysteme und Richtlinienimplementierungs-Frameworks wie das FURZE-System. Weiterhin geht die Arbeit auf Herausforderungen bei der Implementierung von Zero-Trust-Netzwerken ein. Darunter die Komplexität der Systeme, die Integration von Zero-Trust-Netzwerken in bestehende Netzwerke und die Herausforderungen bei der Verwaltung von Identitäten, Priorisierung der Datenflüsse, Segmentierung von Daten und der Verwaltung von Richtlinien. Eine weitere Herausforderung besteht darin, eine angemessene Sicherheitsstrategie und ein zuverlässiges Identitätsmanagement zu implementieren, das den Vertrauensstatus von Anwendungen, Benutzern und Geräten in einer dynamischen Umgebung zuverlässig bestimmen kann. [SCS+22]

Eine wesentliche Arbeit im Bereich Zero-Trust-Modelle und Architekturen leisteten Mehraj und Banday mit ihrer Arbeit aus dem Jahr 2020. [MB20] In ihrer Arbeit beschreiben die Autoren die autorisierte Verwendung von Cloud Computing-Ressourcen. Hierzu beschreiben die Autoren ein Zero-Trust Sicherheitsmodell, um die modernen Sicherheitsherausforderungen in Cloud-Infrastrukturen zu adressieren. Dazu wird das Vertrauen in Cloud-Services untersucht. Der Artikel stellt auch die Trust-Management-Definition für Cloud Computing vor, bevor das Zero-Trust-Modell ausführlich erläutert und diskutiert wird. Die Autoren kommen zu dem Ergebnis, dass die Zero-Trust-Strategie ein neues Sicherheitsmodell in der Cloud-Umgebung einführt, das dazu beitragen kann, das Vertrauen in Cloud-Systeme zu stärken, indem es die Sicherheit erhöht und Zugriffskontrollen implementiert. Schließlich zeigt die Arbeit auch einige Möglichkeiten für zukünftige Forschung auf, wie zum Beispiel die Untersuchung der Auswirkungen von Änderungen in Cloud-Services auf das Trust-Management-Modell.[MB20]

Die Arbeit von Tan Kee Hock aus dem Jahr 2023 [Tan23] untersucht, wie Zero-Trust-Architekturen in AWS eingesetzt werden können, um Insider-Bedrohungen effektiv zu bekämpfen. Tan Kee Hock beschreibt die für die Arbeit entworfene Architektur und die Emulation von Workloads auf AWS, um funktionsübergreifende Einheitlichkeit durch die Segmentierung von Zonen zu erreichen. Ebenso werden Empfehlungen für spezifische AWS-Services, die auf Zero-Trust-Prinzipien basieren, genannt. Darüber hinaus geht die Arbeit detailliert auf den Datenschutz und die Emulation von Insider-Bedrohungen in AWS ein. Insgesamt stellt diese Arbeit wichtige Erkenntnisse für die Implementierung von Zero-Trust-Architekturen in AWS dar. Insbesondere im Hinblick auf die Identifizierung von Insider-Bedrohungen und Maßnahmen zur Bekämpfung. [Tan23]

Die vorgestellten Arbeiten zu Zero-Trust-Modellen und -Architekturen fokussieren sich auf die Analyse und Verbesserung von Zero-Trust-Modellen, insbesondere in Bezug auf Cloud-Computing, Sicherheitsstrategien und organisatorische Implementierungen. Im Gegensatz dazu konzentriert sich die vorliegende Arbeit auf die praktische Implementierung einer Zero-Trust-Architektur zur Erhöhung der Anwendungssicherheit und Cluster-Sicherheit, einschließlich der Messung der Effektivität unter Berücksichtigung von Ressourcen, neuen Schwachstellen, auftretenden Problemen und der in dieser Arbeit herausgestellten Bedrohungsmatrix.

## 3.4 Zero-Trust-Architekturen in Kubernetes Clustern

Im Folgenden werden Arbeiten und Forschungen zu Zero-Trust-Architekturen in Kubernetes-Clustern vorgestellt.

D'Silva und Ambawade [DA21] haben einen Beitrag zu Zero-Trust-Architekturen in Kubernetes geleistet. Sie motivieren die Arbeit damit, dass traditionelle Sicherheitsansätze auf dem Konzept des Perimeterschutzes für dynamische und verteilte Systeme wie Kubernetes nicht mehr ausreichend sind und möchten mithilfe einer Zero-Trust-Architektur innerhalb von Kubernetes ein höheres Maß an Sicherheit gewährleisten. Die Autoren erläurtern, wie man durch eine Hybride-RBAC-ABAC-Zugriffskontrolle und spezifische Attribute innerhalb der Organisation den Zugriff steuern kann. Dabei wird insbesondere auf die Verwendung von Kubernetes eingegangen. Es wird gezeigt, wie durch die Nutzung von permanenten Bearer-Token eine sichere Verbindung zu den Anwendungen übern den Kubernetes-Masternode hergestellt wird. Hierbei werden OpenID Connect und weitere Tools und Dienste wie Docker, Keycloak und React.js erläutert, um die Zero-Trust-Architektur umzusetzen. Insgesamt bietet die Arbeit einen detailreichen Überblick über die Implementierung einer modernen Zero-Trust-Architektur mit Kubernetes für ein sicheres und stabiles IT-System in der heutigen technologiegetriebenen Welt. [DA21]
Im Gegensatz zur Arbeit von D'Silva und Ambawade [DA21], die sich auf die Implementierung einer Zero-Trust-Architektur innerhalb von Kubernetes durch eine hybride RBAC-ABAC-Zugriffskontrolle und die Nutzung von OpenID Connect konzentriert, liegt der Schwerpunkt der vorliegenden Arbeit auf der Implementierung einer Zero-Trust-Architektur direkt in Kubernetes. Dabei wird die Effizienz dieser Architektur anhand von CPU- und Speicherressourcen, der Verhinderung von Schwachstellen sowie möglichen Nachteilen untersucht.

Surantha und Felix (2020) [SI20] erläutern in ihrer Arbeit eine Fallstudie für eine Finanzdienstleistungsgesellschaft, die sich dafür entscheidet, Kubernetes-Technologie zu nutzen, um ihre Anwendungen zu betreiben. Ziel der Studie ist es, ein sicheres Kubernetes-Netzwerk-Design auf Grundlage eines Zero-Trust-Modells für dessen Kubernetes-Cluster zu schaffen. Es wird ein Überblick über die verschiedenen Prozesse und Methoden zur Anforderungs-

erfassung gegeben, um die Geschäftsziele, technischen Ziele und Einschränkungen des Unternehmens zu definieren, die in den Zielen eines Konzeptentwicklungsprozesses münden. Um den Zielen der Gesellschaft zu entsprechen, werden verschiedene Lösungen vorgestellt. Methoden zur Umsetzung eines sicheren Netzwerk-Designs sind hierbei: Netzwerkvirtualisierung, BGP-Routing, Firewalls, externe Syslog-Server und Kubernetes Technologien, wie Labels. Die Arbeit von Surantha und Ivan zeigt, wie das Netzwerk rund um Kubernetes gesichert werden kann.
Die vorliegende Arbeit fokussiert sich auf die Implementierung einer Zero-Trust-Architektur innerhalb eines Kubernetes-Clusters, um die einzelnen Cluster-Komponenten zu schützen. Während die von Surantha und Felix [SI20] beschriebene Fallstudie sich auf die Sicherung eines privat verwalteten Netzwerks konzentriert und die Umsetzung eines Zero-Trust-Modells auf Netzwerkebene in privaten Clustern thematisiert, bezieht diese Arbeit zusätzlich öffentliche Cloud-Anbieter mit Kubernetes ein. Der Schwerpunkt liegt daher auf der Integration von Zero-Trust-Prinzipien durch die internen Sicherheitsmechanismen von Kubernetes.

Der Artikel "Mitigating Insider Threats in Amazon Elastic Kubernetes Service (EKS) - A Zero Trust Perspective"von Tan Kee Hock (2024) [Tan24] untersucht Zero-Trust-Architekturen innerhalb von Amazon Elastic Kubernetes Service (EKS) mit dem Ziel, die verfügbaren Sicherheitskontrollen in Bezug auf Insider-Bedrohungen zu bewerten. Die Arbeit baut auf früheren Untersuchungen des Autors zur Abschwächung von Insider-Bedrohungen in AWS auf und hebt den Mangel an akademischer Forschung in der Schnittmenge von Cloud, Zero-Trust und Insider-Bedrohungen hervor. Mithilfe einer simulierten Arbeitslast werden Insider-Bedrohungsszenarien innerhalb der EKS-Umgebung emuliert, um das Verhalten der nativen AWS-Sicherheitsdienste zu beobachten. Darauf basierend werden Empfehlungen zur Verbesserung der Abwehrmöglichkeiten gegen Insider-Bedrohungen vorgeschlagen, darunter die Implementierung von Multi-Faktor-Authentifizierung, die Reduzierung des Umfangs von IAM-Rollen, die Erhöhung der Audit-Protokollierung sowie die Netzwerksegmentierung durch VPCs und Sicherheitsgruppen zur Durchsetzung des "Least-PrivilegePrinzips. Weitere Empfehlungen umfassen die Einführung von Ende-zu-Ende-Verschlüsselung für Daten im Ruhezustand und während der Übertragung, die Sicherung der Control Plane sowie die Bereitstellung von Mechanismen zur Bedrohungserkennung und -reaktion. Diese Arbeit bietet wertvolle Einblicke in die Sicherheitsfunktionen, die in Kubernetes-Clustern zur Verteidigung gegen Insider-Bedrohungen eingesetzt werden können.
Während Tan Kee Hock [Tan24] den Schwerpunkt auf die Bewertung und Verbesserung von Sicherheitskontrollen in Amazon EKS zur Abwehr von Insider-Bedrohungen legt, fokussiert sich die vorliegende Arbeit auf die Implementierung einer Zero-Trust-Architektur innerhalb eines Kubernetes-Clusters, unabhängig von spezifischen Cloud-Anbietern. Im Gegensatz zu Hock's Arbeit, die primär AWS-native Sicherheitsdienste und ihre Wirksamkeit untersucht, konzentriert sich diese Arbeit auf die Analyse der Effizienz von Zero-Trust-Prinzipien durch Kubernetes-interne Sicherheitsmechanismen unter Berücksichtigung von Ressourcenverbrauch, Schwachstellenprävention und potenziellen Nachteilen.

Rodigari et al. (2021) [ROM+21] untersuchten die Performance von Zero-Trust in Multi-Cloud-Umgebungen, indem sie die Latenz und Ressourcenauslastung (CPU, Speicher) analysierten. Dazu verwendeten sie zwei Kubernetes-Cluster von Google (GKE) und Amazon (EKS) sowie das Service Mesh Istio zur Implementierung von Zero-Trust. Ziel der Studie war es, festzustellen, ob Zero-Trust in einer Multi-Cloud-Architektur zu Leistungseinbußen im Datenverkehr führt. Die Ergebnisse zeigten, dass bei 1000 aufeinanderfolgenden HTTP-Anfragen die Auswirkungen auf die Cluster-CPU und den Speicherverbrauch minimal waren. Überraschenderweise führte der Einsatz von Istio sogar zu einer 50%igen Reduktion des Speicherverbrauchs in den EKS- und GKE-Clustern, was die Autoren auf die Auslagerung lokaler Pod-Prozesse auf die Steuerungsebene zurückführen. Allerdings war die Latenz im Vergleich zu einem Multi-Cloud-Deployment ohne Istio deutlich höher, was auf die effizientere Bearbeitung der Netzwerkanfragen durch Istio's Ingress-Gateway im Vergleich zum grundlegenden Kubernetes-Load Balancer zurückzuführen ist. [ROM+21]
Im Gegensatz zur Arbeit von Rodigari et al. [ROM+21], die die Performance von Zero-Trust in Multi-Cloud-Umgebungen untersucht, befasst sich diese Arbeit mit der Implementierung einer Zero-Trust-Architektur innerhalb eines einzelnen Kubernetes-Clusters. Während Rodigari et al. [ROM+21] die Auswirkungen auf Latenz und Ressourcenverbrauch in einer Multi-Cloud-Umgebung analysieren, liegt der Schwerpunkt dieser Arbeit auf der Effizienzbewertung der Zero-Trust-Architektur in Bezug auf Ressourcenauslastung, Schwachstellenprävention und potenzielle Nachteile.

De Weever und Andreou [WA20] implementieren in ihrer Arbeit ein Proof of Concept, um die Wirksamkeit von Google Kubernetes Engine (GKE) in Kombination mit Istio, Cilium (Netzwerk-Plugin) und Hubble zur Sicherung des Datenverkehrs zwischen Microservices zu demonstrieren. Sie nutzen diese Tools, um die Verschlüsselung und Absicherung des Datenverkehrs, insbesondere im „East-West-Verkehr“ zwischen Microservices, zu gewährleisten. Die Autoren stellen fest, dass die Verschlüsselung des Datenverkehrs durch mutual TLS (mTLS) ermöglicht wird, während Hubble detaillierte Einblicke in den Datenverkehr bietet und die Sichtbarkeit erhöht. Istio unterstützt mit seinen Autorisierungsrichtlinien die Mikrosegmentierung durch den Einsatz von Sidecar-Proxys, um den Datenverkehr effizient zu regulieren. Die Arbeit betont die Bedeutung operativer Kontrollen wie SSL-Verschlüsselung und Mikrosegmentierung zur Erhöhung der Sicherheit. Cilium wird als Mittel zur Minderung von Sicherheitsrisiken durch kompromittierte Sidecar-Proxys vorgeschlagen.
Während De Weever und Andreou [WA20] den Fokus auf die Implementierung von operativen Kontrollen zur Sicherung des „East-West-Datenverkehrs“ zwischen Microservices legen, konzentriert sich diese Arbeit auf die umfassende Implementierung einer Zero-Trust-Architektur innerhalb eines Kubernetes-Clusters. Dabei wird insbesondere die Effizienz der Architektur hinsichtlich Ressourcenverbrauch, Schwachstellenvermeidung und möglichen Nachteilen untersucht.

Pace (2021) [Pac21] untersucht in seiner Masterarbeit die Implementierung von Zero-Trust-Sicherheitsprinzipien in cloud-nativen Umgebungen unter Verwendung von Kubernetes und

Istio als Service Mesh. Die Arbeit befasst sich mit den Herausforderungen, die sich durch die Verlagerung von IT-Infrastrukturen in die Cloud und die Nutzung von Microservices-Architekturen ergeben, und schlägt vor, wie Zero-Trust-Prinzipien in diesen dynamischen und verteilten Umgebungen angewendet werden können. Dabei wird gezeigt, dass Zero-Trust-Architekturen, die keine Unterscheidung mehr zwischen vertrauenswürdigen und nicht vertrauenswürdigen Netzwerkgrenzen machen, eine kontinuierliche Überprüfung aller Zugriffe und Aktionen innerhalb eines Netzwerks erfordern. Pace [Pac21] verwendet Istio, um Zero-Trust-Prinzipien in Kubernetes-Clustern zu implementieren. Dabei ermöglicht Istio die Verwaltung der Kommunikation zwischen Diensten, die Durchsetzung von Authentifizierungs-, Autorisierungs- und Verschlüsselungsrichtlinien sowie die Bereitstellung von Überwachungsfunktionen für das Traffic-Management und die Sicherheitsüberwachung. Zusätzlich wird die Erweiterbarkeit von Istio durch WebAssembly (WASM)-Module untersucht, die zur Implementierung maßgeschneiderter Sicherheitskontrollen wie Web Application Firewalls (WAF) genutzt werden. Diese Erweiterungen erlauben eine granulare Steuerung der Sicherheitsrichtlinien, die speziell auf einzelne Microservices zugeschnitten sind. Während Pace (2021) [Pac21] sich auf die Implementierung von Zero-Trust-Sicherheitsprinzipien in Cloud-nativen Umgebungen mit einem speziellen Fokus auf Kubernetes und Istio als Service Mesh konzentriert, legt diese Arbeit ihren Schwerpunkt auf eine umfassende Evaluierung der Zero-Trust-Architektur innerhalb von Kubernetes-Clustern. Pace untersucht die Herausforderungen und Möglichkeiten, die durch die Nutzung von Istio und WebAssembly (WASM)-Modulen zur Unterstützung von Zero-Trust-Prinzipien entstehen, und bietet eine detaillierte Betrachtung der Implementierung in einem Service Mesh. Im Gegensatz dazu zielt diese Arbeit darauf ab, die Effektivität und wirtschaftlichen Auswirkungen einer Zero-Trust-Architektur zu bewerten, einschließlich der Analyse von Bedrohungen, Schwachstellen, und Best Practices sowie der Untersuchung der praktischen Herausforderungen und Komplexität bei der Umsetzung in verschiedenen Kubernetes-Umgebungen wie Minikube und Amazon Elastic Kubernetes Service (EKS).

# 4 Sicherheitsanalyse

In diesem Abschnitt der Arbeit werden zunächst die Bedrohungen anhand der Microsoft-Bedrohungsmatrix erläutert (Kapitel 4.1). Anschließend werden die Sicherheitsherausforderungen und Angriffsvektoren, die oft bereits im Architekturdesign von Kubernetes verborgen sind, in den Kontext dieser Matrix eingeordnet (Kapitel 4.2). Die Matrix wird dann durch aktuelle Schwachstellen, sogenannte Common Vulnerability Exposures (CVEs), erweitert, um eine umfassende Bedrohungsmatrix zu erstellen (Kapitel 4.3). Abschließend werden die Ergebnisse der Microsoft-Matrix sowie die identifizierten Bedrohungen und Schwachstellen der Cluster-Komponenten ausgewertet (Kapitel 4.4).

## 4.1 Microsoft Bedrohungsmatrix

Eine gute Übersicht über die Bedrohungen eines Kubernetes Clusters liefert die Bedrohungsmatrix, die Microsoft 2020 erstellt und 2021 aktualisiert hat. Die Matrix ist in Abbildung 4.1 dargestellt. Dargestellt werden die Techniken der folgenden zehn Bedrohungstaktiken, die jeweils aus verschiedenen Techniken bestehen: 1) Initial Access, 2) Execution, 3) Persistence, 4) Privilege Escalation, 5) Defense Evasion, 6) Credential Access, 7) Discovery, 8) Lateral Movement, 9) Collection und 10) Impact.

**Initial Access** beschreibt Angriffstechniken, um einen ersten Zugang auf die Cluster-Ressourcen zu erlangen. Dieser kann über die Management-Ebene oder über eine infizierte oder angreifbare Ressource geschehen. Folgende Techniken können innerhalb des Kubernetes Clusters für einen Initial Access von einem Angreifer ausgeübt werden:

**a)** Using Cloud Credentials: Mithilfe von Cloud-Zugangsdaten, die ein Angreifer erlangt, kann er Zugriff auf die Cluster-Management-Ebene bekommen. [Weib]

**b)** Kompromittierte Images in Registry: Kompromittierte Images in der Registry können das gesamte Cluster gefährden. Ein Angreifer könnte versuchen, Zugang zu einer privaten Container-Registry zu erlangen und dort infizierte Images bereitzustellen, die später im Cluster eingesetzt werden. Auch unbekannte Images aus öffentlichen Registries und infizierte Base-Images, auf denen viele Unternehmen ihre eigenen Images aufbauen, stellen ähnliche Risiken dar. [Weib]

| Taktiken / Techniken | Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion |
|---|---|---|---|---|---|
| Bedrohung | Using Cloud Credentials | Exec in einen Container | Backdoor Container | Privilegierter Container | Container Logs löschen |
| Bedrohung | Kompromitiertes Image in Registry | bash/cmd innerhalb Containers | Writable hostpath mount | Cluster-Admin Binding | Delete Kubernetes Events |
| Bedrohung | Kubeconfig Datei | Neuer Container | Kubernetes Cronjob | hostPath mount | Pod/container name similarity |
| Bedrohung | Schwachstelle in einer Anwendung | Anwendungs-Exploit (RCE | Malicious Admission Controller | Access Cloud Ressourcen | Verbindung von Proxy Server |
| Bedrohung | exponierte sensible Schnittstellen | SSH-Server innerhalb des Containers | | | |
| Bedrohung | | Sidecar Injection | | | |

| Taktiken / Techniken | Credential Access | Discovery | Lateral Movement | Collection | Impact |
|---|---|---|---|---|---|
| Bedrohung | Kubernetes Secrets auflisten | Zugriff auf Kubernetes API-Server | Zugriff auf Cloud-Ressourcen | Images einer privaten Registry | Data Desctruction |
| Bedrohung | Mount Service Principal | Zugriff auf die Kubelet API | Container Service Account | | Resource Hijacking |
| Bedrohung | Zugriff auf Container Service Account | Netzwerk-Mapping | Cluster-internes Networking | | Denial of Service |
| Bedrohung | Anwendungs-Zugangsdaten in Konfigurationsdateien | Zugriff auf das Kubernetes Dashboard | Anwendungs-Zugangdaten in Konfigurationsdateien | | |
| Bedrohung | Zugriff auf managed Identität-Zugangsdaten | Instance Metadata API | Schreibbare Volumen mounts auf dem Host | | |
| Bedrohung | Böswilliger AdmisionController | | CoreDNS Poisining | | |
| Bedrohung | | | ARP poisoning und IP spoofing | | |

Abbildung 4.1: Die Microsoft Bedrohungsmatrix aus dem Jahr 2021. Aktualisiert auf die aktuell relevanten Bedrohungstechniken eines Kubernetes Clusters, aufgeteilt in die 10 Bedrohungstaktiken Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection und Impact. Ein Angreifer kann die Techniken der einzelnen Taktiken (blau hervorgehoben) nutzen, um vom initialen Zugriff auf das Cluster über die Sammlung von Informationen und Anmeldedaten bis hin zur Zerstörung oder Ressourcen-Übernahme einen Angriff auf das Cluster durchzuführen. Die Matrix basiert auf den Taktiken des MITRE ATT&CK Frameworks. [Weia]

**c)** Kubeconfig Datei: Ein Angreifer kann durch die Kubeconfig-Datei Informationen über den Standort und die Zugangsdaten des Clusters erlangen und darüber mit kubectl auf das Cluster zugreifen. [Weib]

**d)** Schwachstelle einer Anwendung: Ein Angreifer kann durch eine solche Schwachstelle Zugriff auf das Cluster erlangen, indem er die Service Account Credentials des Containers nutzt, um Anfragen an den API-Server zu stellen. [Weib]

**e)** Ungeschützte sensitive Interfaces: Über öffentliche Ports können Angreifer Zugang zum Cluster erlangen, indem sie zum Beispiel das Kubernetes Dashboard manipulieren oder Authentifizierungsmechanismen im API-Server deaktivieren. [Weia, Weib]

Um Code innerhalb des Clusters auszuführen, wird die Taktik **Execution** genutzt. Hierzu gehören die folgenden Techniken:

**a)** Exec in einen Container: Angreifer können mithilfe des exec-Befehls Befehle in Containern ausführen und über eine Backdoor, wie ein legitimes Betriebssystem-Image, Schadcode einschleusen. [Weib]

**b)** Bash/CMD innerhalb eines Containers: Der Zugriff auf Container und die uneingeschränkte Ausführung von bash- und cmd-Befehlen ermöglichen eine Privilegien-Erweiterung und Zugriff auf den Host. [Weib]

**c)** Neuer Container: Angreifer, die über die erforderlichen Berechtigungen verfügen, können einen neuen Container im Cluster erstellen und deployen, weitere Elemente wie Pods oder Controller hinzufügen und zusätzliche Ressourcen erzeugen, um ihren Schadcode auszuführen. [Weib]

**d)** Anwendungs-Exploit: Ein Angreifer kann eine Anwendung ausnutzen, die für Remote Code Execution anfällig ist, um Schadcode im Cluster auszuführen. Wenn der Angreifer zudem aus der Anwendung heraus Zugriff auf einen Service-Account erhält, kann er Anfragen an den Kubernetes API-Server mit diesen Account-Details senden. [Weib]

**e)** SSH-Server in einem Container: Erhält ein Angreifer Zugang zum SSH-Server in einem Container, etwa durch Brute-Force-Angriffe auf die Zugangsdaten, kann er auch auf weitere Container über SSH zugreifen. [Weib]

**f)** Sidecar Injection: Der Angreifer fügt einen Sidecar-Container in einen legitimierten Pod ein, anstatt einen separaten Pod ins Cluster zu integrieren. Da der Sidecar-Container meist nebensächliche Aufgaben übernimmt (siehe Kapitel Sidecar Pattern), kann der Angreifer seinen Code unentdeckt ausführen. [Weia]

Falls der Angreifer seinen ersten Einstiegspunkt verliert, kann er durch **Persistenz**-Taktiken weiterhin Zugang zum Cluster behalten.

**a)** Backdoor Container: Durch eine Backdoor in einem Container kann ein Angreifer auf den Kubernetes Controller für DaemonSets oder Deployments zugreifen und sicherstellen, dass eine bestimmte Anzahl an Containern kontinuierlich auf einem oder allen Nodes des Clusters ausgeführt wird. [Weib]

**b)** Writable hostPath mount: Ein Angreifer kann durch ein beschreibbares Volume auf dem Host, das in einen Container gemountet wird, Persistenz auf dem zugrunde liegenden Host erreichen, beispielsweise durch einen Cronjob. [Weib]

**c)** Kubernetes CronJob: Der Kubernetes Cronjob-Controller erstellt und überwacht Pods für die Ausführung von Aufgaben. Angreifer können diesen Controller nutzen, um die Ausführung ihres bösartigen Codes zu planen, der dann als Pod im Cluster läuft. [Weib]

**d)** Infizierter Admission Controller: Der Admission Controller fängt Anfragen an den Kubernetes API-Server ab und kann diese bei Bedarf modifizieren. Angreifer können den AdmissionWebhook des Admission Controllers ausnutzen, um sich im Cluster zu verankern, indem sie beispielsweise Pod-Erstellungsvorgänge abfangen und modifizieren, um ihren bösartigen Container zu jedem erstellten Pod hinzuzufügen. [Weia]

**Privilege Escalation** beschreibt Techniken, um mehr Berechtigungen im Cluster zu erlangen. Hierfür können Angreifer die folgenden Privilegien einsetzen:

**a)** Privileged container: Ein privilegierter Container hat vollständigen Zugriff auf die Ressourcen des Host-Rechners, entfernt alle Einschränkungen regulärer Container und kann direkt Befehle und Aktionen auf dem Host ausführen. Angreifer, die Zugang zu einem privilegierten Container erhalten oder einen solchen erstellen, können somit auf die Ressourcen des Hosts zugreifen. [Weib]

**b)** Cluster-admin Binding: Role-based Access Control (RBAC) ermöglicht es, die Aktionen von Identitäten im Cluster zu regulieren. Die Rolle ClusterAdmin ist eine eingebaute, hoch privilegierte Rolle. Angreifer, die Berechtigungen zum Erstellen von Bindungen und Cluster-Bindungen haben, können Bindungen an die ClusterAdmin-Rolle oder andere hoch-privilegierte Rollen erstellen. [Weib]

**c)** hostPath Mount: Über einen hostPath-Mount des Containers kann ein Angreifer Zugriff auf den Host erlangen. [Weib]

**d)** Access Cloud Resources: Dieser Angriff betrifft Cluster, die in der Cloud gehostet werden. Angreifer können über die Cloud-Provider-Credentials, die zur Verwaltung des Clusters verwendet werden, auf weitere Cloud-Ressourcen zugreifen. Beispielsweise könnte ein Angreifer über einen hostPath-Mount Zugang zu den Service Principal Credentials erhalten. [Weib]

Techniken zur Vermeidung der Entdeckung, bekannt als **Defense Evasion**, zielen darauf ab, Aktivitäten innerhalb des Clusters zu verstecken und nicht erkannt zu werden [Weib]. Zu den Methoden der **Defense Evasion** zählen:

**a)** Clear Container Logs: Angreifer löschen Anwendungs- oder Betriebssystem-Logs auf einem kompromittierten Container, um ihre Aktivitäten zu verschleiern und zu verhindern, dass diese entdeckt werden. [Weib]

**b)** Delete Kubernetes Events: Kubernetes-Events protokollieren Änderungen und Ausfälle im Cluster, wie die Erstellung eines Containers, Image-Pulls oder das Pod-Scheduling. Angreifer können versuchen, diese Events zu löschen, um ihre Aktivitäten zu verbergen und eine Entdeckung zu verhindern. [Weib]

**c)** Pod/Container Name Similarity: Angreifer können den Namen ihres Backdoor-Containers so wählen, dass er ähnlich wie der Name von existierenden Controllern oder Services aussieht. Im kube-system Namespace, der administrative Container enthält, kann ein Angreifer seinen Container beispielsweise als Core-DNS-Service tarnen. [Weib]

**d)** Connect from Proxy Server: Angreifer nutzen Proxyserver oder anonyme Netzwerke wie TOR, um ihre IP-Adresse zu verbergen und ihre Aktivitäten zu verschleiern. [Weib]

Um Zugangsdaten, Identitäten und Passwörter von laufenden Anwendungen und dem Cluster zu stehlen, kann ein Angreifer Techniken der Taktik **Credential Access** anwenden. Dazu zählen folgende Techniken:

**a)** List Kubernetes Secrets: Kubernetes Secrets werden zur Speicherung und Verwaltung von Passwörtern und Verbindungszeichenfolgen im Cluster verwendet. Durch Verweise in der Pod-Konfiguration können diese Secrets in Pods verwendet werden. Angreifer können durch Zugriff auf die Pod-Konfiguration diese Secret-Daten einsehen und so Anmeldeinformationen für verschiedene Dienste erlangen. [Weib]

**b)** Mount Service Principal: Ähnlich wie beim Zugriff auf Cloud-Ressourcen kann ein Angreifer den Zugriff auf einen hostPath-Mount eines Containers nutzen, um die Anmeldeinformationen für den Service Principal zu erlangen. [Weib]

**c)** Access Container Service Account: In Kubernetes wird jedem Container standardmäßig ein Service Account zugewiesen, um mit dem API-Server zu kommunizieren. Erhält ein Angreifer Zugang zu einem Container, kann er das Service Account Token erlangen und damit weitere Aktionen im Cluster durchführen. [Weib]

**d)** Application Credentials in Configuration Files: Anwendungs-Zugangsdaten werden häufig in Konfigurationsdateien gespeichert. Ein Angreifer kann diese Daten ausnutzen, um Aktionen auf den betreffenden Anwendungen durchzuführen. [Weib]

**e)** Access Managed Identity Credential: Cloud-Anbieter verwalten bestimmte Identitäten des Clusters, die zur Authentifizierung bei Cloud-Diensten verwendet werden. Dies

erfolgt über einen Instance Metadata Service (IMDS), den Anwendungen nutzen, um auf die Identitätsgeheimnisse zuzugreifen. Ein Angreifer, der Zugang zu einem Pod erlangt, kann dieses Secret stehlen und damit Zugang zu weiteren Cloud-Ressourcen erhalten. [Weib]

**f)** Malicious Admission Controller: Ein kompromittierter Admission Controller kann über die standardmäßig mit Kubernetes ausgelieferte MutatingAdmissionWebhook Zugang zu Zugangsdaten erhalten. Durch die Nutzung dieser Webhook kann ein Angreifer Anfragen an den API-Server abfangen und dabei Zugangsdaten sowie andere sensible Informationen sammeln. [Weia]

Ein Angreifer setzt die Techniken der **Discovery** Taktik ein, um die Umgebung der kompromittierten Systeme zu erkunden. Die Techniken umfassen:

**a)** Access the Kubernetes API Server: Ein Angreifer mit Zugriff auf den API-Server kann den Status aller Komponenten abfragen sowie Informationen über Container, Zugangsdaten und andere Ressourcen im Cluster einsehen. [Weib]

**b)** Access Kubelet API: Durch API-Anfragen an die Kubelet API kann ein Angreifer Informationen über Pods sowie Details zum Node, wie CPU- und Speicherverbrauch, abrufen. [Weib]

**c)** Network Mapping: Durch Netzwerk-Mapping kann ein Angreifer Informationen über die laufenden Anwendungen im Cluster sammeln und nach bekannten Schwachstellen suchen. Da Kubernetes standardmäßig keine Einschränkungen für die Pod-zu-Pod-Kommunikation vorschreibt, kann bereits der Zugang zu einem einzelnen Container ausreichen, um das gesamte Cluster zu scannen. [Weib]

**d)** Access Kubernetes Dashboard: Das Kubernetes Dashboard ermöglicht es, Informationen über die verschiedenen Cluster-Ressourcen zu sammeln und Container sowie andere Ressourcen zu verwalten oder zu ändern. [Weib]

**e)** Instance Metadata API: Cloud-Anbieter bieten einen Instance Metadata Service, der umfassende Informationen über die virtuellen Maschinen des Clusters bereitstellt. Ein Angreifer kann über die API dieses Dienstes weitere Details zum System und zur Cloud-Umgebung erhalten. [Weib]

**Lateral Movement** beschreibt die Strategien eines Angreifers, sich innerhalb der Umgebung seines Opfers zu bewegen. [Weib] Dazu benötigt der Angreifer Zugang zu verschiedenen Ressourcen des Clusters und verwendet folgende Techniken:

**a)** Access Cloud Resources: Ein Angreifer, der Zugangsdaten des Cloud-Providers nutzt, die zur Verwaltung des Clusters verwendet werden, kann auf weitere Cloud-Ressourcen zugreifen. [Weib]

**b)** Container Service Account: Mit dem Container-Service-Account kann ein Angreifer Zugangsdaten und Anfragen an den API-Server abfangen sowie neue Ressourcen anlegen, um das Cluster zu unterwandern. [Weib]

**c)** Cluster internal networking: Ein Angreifer kann das interne Netzwerk des Clusters, das standardmäßig keine Einschränkungen hat, nutzen, um andere Container im Cluster zu erreichen. [Weib]

**d)** Application Credentials in Configuration Files: Zugangsdaten, die in Konfigurationsdateien von Anwendungen gespeichert sind, werden im Rahmen des Lateral Movement verwendet, um Zugriff auf andere Anwendungen im Cluster zu erhalten. [Weib]

**e)** Writable Volume Mounts on the Host: Ein Angreifer kann einen schreibbaren Host-Path-Mount nutzen, um direkt auf den Node zuzugreifen und innerhalb dessen weitere Anwendungen zu kompromittieren. [Weib]

**f)** CoreDNS Poisoning: Mit Berechtigungen auf Management-Ebene kann ein Angreifer die Konfiguration des CoreDNS-Dienstes im Cluster ändern, um dessen Verhalten zu manipulieren und die Netzwerkidentität anderer Dienste zu übernehmen. [Weia]

**g)** ARP Poisoning and IP Spoofing: Das Kubernetes-Netzwerk-Plugin Kubenet erstellt auf jedem Knoten ein Bridge-Netzwerk, das die Pods über $V_{eth}$-Paare verbindet. Dies ermöglicht ARP-Poisoning im Cluster, bei dem ein Angreifer die Anfragen anderer Pods verfälschen und Angriffe wie IP-Spoofing oder den Diebstahl von Cloud-Identitäten anderer Pods durchführen kann (CVE-2021-1677). [Weia]

Die Taktik **Collection** beschreibt die Technik des Zugriffs auf Container-Images aus einer privaten Registry, um Daten aus dem Cluster zu erlangen [Weia]. Für den Zugriff auf diese Registry, insbesondere bei Cloud-Anbietern wie Azure Container Registry (ACR) oder Amazon Elastic Container Registry (ECR), sind gültige Anmeldedaten erforderlich. Ein Angreifer, der Zugang zum Cluster hat, kann auch auf die private Registry zugreifen und deren Images abrufen. Hierzu kann er, wie unter Access Managed Identity Credential beschrieben, das Managed Identity Token nutzen. In EKS kann zudem die AmazonEC2ContainerRegistryReadOnly-Policy verwendet werden, die standardmäßig an die IAM-Rolle des Knotens gebunden ist [Weia].

Mithilfe der Techniken der **Impact** Taktik ist es einem Angreifer möglich das normale Verhalten des Clusters zu stören, auszunutzen oder zu zerstören. [Weib]

**a)** Data Destruction: Ein Angreifer kann Deployments, Konfigurationen, Speicher und andere Ressourcen im Cluster löschen, um das Cluster zu zerstören. [Weib]

**b)** Resource Hijacking: Beim Resource Hijacking missbraucht ein Angreifer eine kompromittierte Ressource für andere Zwecke. [Weib]

**c)** Denial of Service: Ein Angreifer kann versuchen, einen Denial-of-Service-Angriff durchzuführen, um die Verfügbarkeit der Container, der zugrunde liegenden Nodes oder des API-Servers zu stören oder diesen zum Absturz zu bringen. [Weib]

## 4.2 Sicherheitsherausforderungen und Angriffsvektoren

Kubernetes Cluster sind, wie in Kapitel 2.1.2 erläutert, komplizierte und aus mehreren interagierenden Komponenten bestehende Systeme. Gerade für kritische Anwendungen ist die Isolation und die Zerlegung dieser Schichten in mehrere Kompartments von enormer Bedeutung. Doch Kubernetes ist zunächst nur eine Orchestration-Plattform, die unbekannte Workloads ausführt. Aus diesem Grund ist es schwer, das System zu sichern und berechtigten Nutzern Zugang zu Ressourcen, Funktionen und Daten zu geben. In diesem Teil der Sicherheitsanalyse werden zunächst Sicherheitsherausforderungen im Cluster besprochen (Kapitel 4.2.1), um anschließend weitere Angriffsvektoren aufzuzeigen (Kapitel 4.2.2). Diese Ergebnisse dienen der Anreicherung der Bedrohungsmatrix aus Abbildung 4.1.

### 4.2.1 Sicherheitsherausforderungen

Kubernetes-Cluster sind aufgrund ihrer verteilten Architektur auf mehreren Ebenen Sicherheitsrisiken ausgesetzt. Auf der Host-Ebene stellt der Zugriff auf einen Node eine ernsthafte Bedrohung dar. Ein Angreifer könnte den gesamten Host und alle darauf laufenden Workloads kontrollieren. Zudem könnten modifizierte Kubelets verwendet werden, um bösartige Workloads einzuführen, Cluster-Informationen zu sammeln oder den API-Server zu stören. Besonders auf Bare-Metal-Maschinen sind Nodes anfällig für physische Kompromittierungen. Ein kompromittierter Node könnte auch als Teil eines Botnetzwerks agieren, was zu einer Ressourcenverknappung führen kann, da die Infrastruktur möglicherweise automatisch skaliert und mehr Ressourcen zuweist. Auch das etcd nimmt für die Speicherung von Zertifikaten, Zugangsdaten, Zuständen des Clusters und Token eine zentrale Rolle im Cluster ein. Schreib-Zugriff auf den API-Server des etcd kommt somit einem Root-Zugriff

auf dem ganzen Cluster gleich. Auch Lesezugriff kann zu Privilege Escalations führen. [owa].

Auf der Netzwerk-Ebene ist ein tiefes Verständnis der Kommunikation zwischen Systemkomponenten essenziell. Es muss sichergestellt werden, dass sowohl die interne Kommunikation im Cluster als auch die Verbindung zur Außenwelt gesichert sind. Die Entdeckung von Cluster-Komponenten durch DNS, Discovery-Services oder Load Balancer kann unterschiedliche Sicherheitsrisiken bergen. Zudem stellt das Schlüsselmanagement für die Verschlüsselung von Daten eine weitere Herausforderung dar. [Say17, owa]

Die Nutzung von Container Images bringt zusätzliche Risiken mit sich, da das Cluster nicht direkt weiß, welche Aufgaben ein Container übernimmt. Obwohl Quotas die Ressourcennutzung begrenzen und der Zugriff auf andere Komponenten eingeschränkt werden kann, bleibt der Schutz der Build-Pipeline und der Repositorys kritisch. Sicherheitsüberprüfungen der Basis-Images, die oft von Drittanbietern stammen, können vernachlässigt werden. [Say17, owa]

Die Fernverwaltung von Kubernetes-Clustern erhöht das Risiko, da Konfigurationen und Bereitstellungen möglicherweise nicht gründlich getestet werden. Mitarbeiter, die von unterwegs arbeiten, sind zusätzlichen Sicherheitsrisiken ausgesetzt, da ein Angriff auf ihre Geräte zu einer vollständigen Kompromittierung des Clusters führen kann. [Say17, owa]

In Pods teilen Container dasselbe Netzwerk und manchmal auch gemountete Volumes vom Host-Dateisystem, was Sicherheitsprobleme zwischen Containern und auf den gesamten Node ausweiten kann. Experimentelle Add-ons und DaemonSets, die auf allen Knoten laufen, können ebenfalls zusätzliche Risiken darstellen. [Say17, owa]

Schließlich können in der Organisation und den Prozessen Sicherheitsaspekte in der schnellen Entwicklung und kontinuierlichen Bereitstellung (Continuous Deployment) in den Hintergrund gedrängt werden. Dies kann dazu führen, dass Sicherheitsprobleme nicht rechtzeitig erkannt oder behoben werden. Kleinere Organisationen könnten zudem nicht über die notwendige Expertise verfügen, um Sicherheitsaspekte in Kubernetes-Clustern effektiv zu verwalten. [Say17, owa]

### 4.2.2 Angriffsvektoren

Aus den zuvor dargestellten Angriffsvektoren können die folgenden Bedrohungstaktiken in die Bedrohungsmatrix aufgenommen werden.

### Node-Ebene

Ein besonders kritischer Angriffsvektor auf der Node-Ebene ist die Übernahme eines Hosts, wodurch ein Angreifer die Kontrolle über alle darauf laufenden Workloads erlangt. [Say17, owa]
Dies betrifft mehrere Taktiken in der Bedrohungsmatrix. Der erste Schritt ist der Initial Access, bei dem der Angreifer Zugriff auf den Node erhält. Anschließend kann er mithilfe der **Execution**-Taktik das Kubelet durch eine modifizierte Version ersetzen, um bösartigen Code auszuführen und Informationen zu sammeln. Durch diesen Austausch erlangt der Angreifer **Persistence**, da er das modifizierte Kubelet als dauerhaften Zugangspunkt zum Kubernetes-Cluster nutzen kann. Gleichzeitig kann er über das manipulierte Kubelet seine Berechtigungen erweitern, indem er direkt mit dem API-Server kommuniziert und weitere Workloads ausführt. Diese Manipulation des Kubelets, die auch Sicherheitsmechanismen umgeht, wird der **Defense Evasion**-Taktik zugeordnet. Darüber hinaus kann der Angreifer den kompromittierten Node für Resource Hijacking verwenden, was der **Impact**-Taktik zugeordnet wird. Ein weiterer relevanter Angriffsvektor auf der Node-Ebene ist der physische Zugang zu Bare-Metal-Maschinen, der zu Manipulation und Datenabgriff führen kann, was ebenfalls der **Impact**-Taktik zugeordnet wird.

### Netzwerk-Ebene

Die Netzwerkkonnektivität des Clusters bringt zusätzliche Angriffsvektoren mit sich, wie die öffentliche Zugänglichkeit von Ressourcen und Endpunkten, unverschlüsselte sensible Daten bei Übertragung und Speicherung sowie Schwächen in der Netzwerkisolation und -trennung. [Say17, owa]
Die Risiken durch öffentliche Zugänglichkeit ermöglichen es einem Angreifer, die **Discovery**-Taktik zu nutzen, um das Netzwerk zu erkunden und Schwachstellen zu identifizieren. Außerdem kann der Angreifer durch **Lateral Movement** (zum Beispiel über das Cluster-Internal-Networking) innerhalb des Clusters voranschreiten. Zudem ermöglicht eine öffentlich zugängliche Schnittstelle das Abfangen von Anmeldeinformationen, was ebenfalls in der Bedrohungsmatrix vermerkt ist.

Der API-Server, der die zentrale Stelle des Clusters darstellt, stellt zwei API-Endpunkte unter Port 8080 und 6443 zur Verfügung. Während Port 6443 eine Authentifizierung ermöglicht, ist Port 8080 nicht authentifiziert. Auch das etcd (2379–2380) stellt öffentliche Ports zur Verfügung, die geschützt werden müssen. Hinzukommen Ports der Kubelet API (10250), des kube-scheduler (10251), dem kube-controller-manager (10252) und dem Read-only Zugang der Kubelet API auf Port 10255. Die Ports der Control Plane Komponenten sind übersichtlich in Tabelle 4.1 dargestellt. [owa] Auf Seiten der Worker Nodes finden sich ebenfalls drei Komponenten, die öffentliche Ports bereitstellen. Hierzu zählen die API des Kubeletes auf Port 10250 und dessen Read-only API auf Port 10255, sowie eine Port-Range

Tabelle 4.1: Öffentliche Ports der Control-Plane-Node

| Protokoll | Port Range | Komponente |
|---|---|---|
| TCP | 6443 & 8080 | Kubernetes API Server |
| TCP | 2379–2380 | etcd Server Client API |
| TCP | 10250 | Kubelet API |
| TCP | 10251 | kube-scheduler |
| TCP | 10252 | kube-controller-manager |
| TCP | 10255 | Read-only Kubelet API |

von 30000 bis 32767 für NodePort Services. Die Ports der Worker Nodes Komponenten sind in Tabelle 4.2 dargestellt. [owa]

Tabelle 4.2: Öffentliche Ports der Worker Nodes

| Protokoll | Port Range | Komponente |
|---|---|---|
| TCP | 10250 | Kubelet API |
| TCP | 10255 | Read-only Kubelet API |
| TCP | 30000-32767 | NodePort Services |

Sowohl Worker als auch Control Plane Nodes enthalten die Kubelet APIs, über die es möglich ist, Zugriff auf den Node und die Container des Clusters zu erhalten. Standardmäßig erlauben die Kubeletes nicht authentifizierten Zugriff zu dieser API. [owa].

Unverschlüsselte, sensible Daten sind anfällig für **Collection** und **Exfiltration**. **Exfiltration**, eine neuere Taktik in der MITRE ATT&CK-Matrix, umfasst Techniken, die Angreifer einsetzen, um Daten aus dem Netzwerk zu stehlen. Zunächst sammelt der Angreifer unverschlüsselte Daten (**Collection**) und überträgt sie dann aus dem Netzwerk (**Exfiltration**). Schwächen in der Netzwerkisolation bieten Angriffsmöglichkeiten für **Defense Evasion**, indem der Angreifer Netzwerkregeln manipuliert, sowie für **Lateral Movement**, indem er sich durch Schwachstellen im Netzwerk weiter ausbreitet.

#### Image-Ebene

Die Angriffsvektoren auf der Image-Ebene lassen sich gut mit der Microsoft-Matrix abbilden. [Say17, owa]
Bösartige Images, die Schadcode enthalten, betreffen die Taktiken **Execution** (durch Application Exploit (RCE)) und **Persistence** (durch Backdoor Container). Ein Angreifer kann somit Code aus einem infizierten Image ausführen und durch den Backdoor Container dauerhaft im System verbleiben. Zusätzlich kann der Angreifer den Schadcode innerhalb des Images tarnen, um nicht entdeckt zu werden, was der **Defense Evasion**-Taktik zugeordnet wird.

#### Konfigurations- und Bereitstellungsebene

Auf dieser Ebene zählen Remote-Administration von Kubernetes-Clustern, Schwächen in der Konfiguration und Fehler sowie die Gefahr durch Remote-Mitarbeiter zu den Angriffsvektoren. [Say17, owa]
Die Remote-Administration ermöglicht einem Angreifer den gleichen Zugang zum Cluster wie einem legitimen Mitarbeiter, etwa durch die Nutzung von Cloud-Zugangsdaten und der kubeconfig-Datei (**Initial Access**). Während einige Angriffstechniken hier bereits in der Matrix vermerkt sind, können Schwächen in der Konfiguration zu einer Manipulation von Konfigurationsdateien führen, was **Defense Evasion** unterstützt. Solche Schwächen können auch ungewollt zu **Privilege Escalation** beitragen.

#### Pod- und Container-Ebene

Auf dieser Ebene sind Angriffsvektoren wie die Nutzung gemeinsamer Netzwerk- und Speicherressourcen in einem Pod relevant. Ein Angreifer kann Schwachstellen in Pods ausnutzen, um sich auf andere Container oder den Node auszubreiten, was **Lateral Movement** ermöglicht. Außerdem kann er die Ressourcen in einem Pod nutzen, um höhere Privilegien zu erlangen (**Privilege Escalation**). Beide Techniken werden neu in die Matrix aufgenommen. Ein weiterer Vektor ist der schadhafte Container, der es dem Angreifer ermöglicht, Code auszuführen (**Execution**) und durch Nutzung eines kompromittierten Containers dauerhaft im Cluster zu verbleiben (**Persistence**). Zudem können Schwachstellen in Add-ons oder Plugins ausgenutzt werden, um Code auszuführen (**Execution**) oder Sicherheitsmechanismen zu umgehen (**Defense Evasion**). Diese Bedrohungen sind bereits in der Bedrohungsmatrix enthalten. [Say17, owa]

#### Organisatorische und kulturelle Herausforderungen

Obwohl diese Herausforderungen keine direkten Angriffsvektoren darstellen, verdeutlichen sie die Notwendigkeit, diese Ebene zu berücksichtigen. Ein fehlender Sicherheitsfokus bei Entwicklern kann zu unvorsichtigen Konfigurationen führen, die wiederum **Privilege Escalation** ermöglichen. Auch die schnelle Entwicklung begünstigt die Umgehung von Sicherheitsmaßnahmen und Standards, um Zeit zu sparen, was Risiken in der **Defense Evasion**-Taktik erhöht. [Say17, owa]

## 4.3 Schwachstellenanalyse

In diesem Kapitel werden Schwachstellen und deren Auswirkungen auf Kubernetes-Cluster und die Container-Runtime Docker detailliert untersucht. Zunächst werden aktuelle Schwachstellen der Kubernetes-Komponenten (Kapitel 4.3.1) sowie des Container Runtime Interfaces Docker (Kapitel 4.3.1) analysiert. Daraufhin werden vergangene Schwachstellen dieser Komponenten (Kapitel 4.3.3 und Kapitel 4.3.4) vorgestellt.
Die Betrachtung aktueller Schwachstellen liefert Informationen über die gegenwärtigen Bedrohungen, die zum Zeitpunkt der Erstellung der Arbeit bekannt sind. Obwohl bei diesen zu erwarten ist, dass diese Schwachstellen in Kürze behoben werden, geben diese dennoch einen Einblick in die historische Entwicklung. Diese Schwachstellen werden in die Bedrohungsmatrix (Abbildung 4.1) eingeordnet, um deren Sicherheitsrelevanz und potenzielle Auswirkungen zu bewerten. Vergangene Schwachstellen dienen als Referenz und helfen dabei, historische Sicherheitsprobleme nachzuvollziehen, um ihre Relevanz für die heutige Sicherheitslage zu verstehen. Diese Informationen fließen auch in die Erstellung der Heatmap ein, die in Kapitel 4.4 behandelt wird.

### 4.3.1 Aktuelle Schwachstellen: Kubernetes-Komponenten

Im Folgenden wurden Schwachstellen in den Komponenten des Kubernetes Clusters untersucht. Zum Zeitpunkt dieser Arbeit stehen die Kubernetes Versionen 1.28. bis 1.31. noch unter aktivem Support [endb]. Eine Betrachtung der Schwachstellen bezieht sich somit auf die genannten Versionen.

CVE-2020-8554 ermöglicht es einem Angreifer, mit Berechtigungen zum Erstellen oder Aktualisieren von Diensten und Pods, das Feld für die externe IP in einer Dienstkonfiguration zu manipulieren. Auf diese Weise können sie dem kontrollierten Dienst eine beliebige IP zuweisen. Auf diese Weise kann der Angreifer Datenverkehr, der für andere Dienste oder Knoten bestimmt war, auf seinen eigenen Dient umleiten und so Man-in-the-Middle-Angriffe (MitM), das Abfangen von Daten und eine mögliche Unterbrechung des Dienstes ermöglichen.[cvea]
Diese Schwachstelle kann in der Bedrohungsmatrix in den Taktiken Persistence, Collection, Exfiltration, Impact eingeordnet werden. Da für die Ausnutzung der Schwachstelle ein Zugriff und die Berechtigungen vorhanden sein müssen, wird diese nicht in der **Initial Access**-Taktik eingeordnet. Die Veränderung von Services, ermöglicht einem Angreifer einen Zugang zu Cluster zu behalten (**Persistence**), durch einen Man-in-the-Middle-Angriff Daten zu sammeln (**Collection**), diese Daten außerhalb des Clusters zu versenden (**Exfiltration**) und durch die Umleitung des Traffics auch Ressourcen für sich zu beanspruchen oder den Dienst lahm zu legen.(**Impact**).

Mit CVE-2024-3177 wurde eine Schwachstelle entdeckt, bei der es Benutzern möglich sein, kann Container zu starten, die die vom Service Account Admission Plugin durchgesetzte Richtlinie umgehen, wenn sie Container, Init-Container und ephemeral Containers mit dem envFrom-Feld verwenden. Die Richtlinie des Serviceaccounts Admission Plugin stellt sicher, dass Pods, die mit einem Service-Account laufen, nur auf Secrets verweisen können, die im Secrets-Feld des Service-Accounts angegeben sind. [cvec]
Durch diese Schwachstellen haben Angreifer die Möglichkeit, die Secrets-Richtlinie zu umgehen, was es ermöglicht, dass sie Zugriff auf vertrauliche Informationen bekommen (**Credential Access**). Mithilfe dieser Informationen und Secrets können Sie höhere Privilegien oder erweiterten Zugriff auf Ressourcen erhalten (**Privilege Escalation**).

### 4.3.2 Aktuelle Schwachstellen: Container-Runtime Docker

Im Gegensatz zu Kubernetes hat Docker keine End Of Life Daten festgelegt. Jedoch wird für die letzten 4 Versionen ein Security-Support bereitgestellt. Dies betrifft zum Zeitpunkt der Erstellung dieser Arbeit die Versionen 23. bis 27.1. [enda].

Eine Schwachstelle der runc-Komponente wurde im Januar 2024 entdeckt. Dabei handelt es sich um CVE-2024-21626, einen Dateideskriptor Leak, der es einem Angreifer ermöglicht einen neuen Container-Prozess im Arbeitsverzeichnis des Host-Dateisystems erlaubt. Dadurch hat der Angreifer Zugriff auf das Host-Dateisystem. Der Exploit dieser Schwachstelle existiert und die Wahrscheinlichkeit, dass diese Schwachstelle ausgenutzt werden kann, wird im August 2024 mit 5 % angegeben. CVE-2024-21626 kann in der Bedrohungsmatrix durch die beiden Angriffe, die mit dieser Schwachstelle einhergehen, in der **Privilege Escalation**-Taktik eingeordnet werden, da ein Zugriff auf den Host durch diese ermöglicht wird. [cveb]

### 4.3.3 Vergangene Schwachstellen: Kubernetes-Komponenten

Obwohl zum Zeitpunkt der Arbeit nur wenige Schwachstellen in den aktuellen Kubernetes- und Docker-Versionen existieren, lohnt es sich, einen Blick auf die Schwachstellen vergangener Versionen zu werfen, um zu verstehen, welche Komponenten häufig Schwachstellen ausgeliefert sind.

Die CVEs der Kubernetes-Komponenten sind in Tabelle A.1 dargestellt. Insgesamt wurden 16 Schwachstellen im API-Server entdeckt, die von Server-side Request Forgery über (Proxy-)Umgehungen von Admission-Richtlinien bis hin zu Schwachstellen des Typs Denial of Service reichen. Diese hohe Anzahl von Schwachstellen reflektiert die zentrale Rolle des API-Servers innerhalb der Kubernetes-Architektur. Für die Kubectl-Komponente, die mit dem API-Server kommuniziert, wurden in der Vergangenheit sieben Schwachstellen

identifiziert. Diese Schwachstellen umfassen fehlerhafte Neutralisierungen von Escape-, Meta- oder Control-Sequenzen sowie Remote Code Execution und Denial of Service-Angriffe. Auch die Kubelet-API und ihre Interaktion mit dem kube-proxy wurden in den letzten Versionen mit sieben CVEs bedacht. Diese reichten von unauthentifizierten Read-only-HTTP-Anfragen, die zu Denial of Service-Angriffen führten, bis hin zu Umgehungen von Sicherheitsmaßnahmen wie dem Seccomp-Profil. Das etcd des Master Nodes wies in der Vergangenheit insgesamt neun Schwachstellen auf. Da das etcd sämtliche Informationen über das Cluster speichert und verwaltet, wie den Zustand, Secrets und Zugangsdaten, beziehen sich die Schwachstellen häufig auf den Zugriff auf diese Schlüssel. Darüber hinaus gehören auch Denial of Service Angriffe aufgrund fehlender Eingabevalidierung oder Index-Überläufe zur Liste der identifizierten Schwachstellen. Weitere elf Schwachstellen wurden im Bereich Logging sowie in den Komponenten Container Network Interface (CNI) und CRI-O entdeckt.

### 4.3.4 Vergangene Schwachstellen: Container-Runtime

Schwachstellen der Docker-Architektur wurden in den Jahren 2015 - 2023 etwa 27 Schwachstellen gefunden, diese sind in Tabelle A.2 aufgeführt.

Die meisten hiervon entfallen auf docker und den Docker Daemon, die von ungewollt eingeschleusten Befehlen über Berechtigungsausweitungen durch Nutzung von CLI-Parametern bis hin zum Denial of Service führen. containerd und runc wurden jeweils 8 Schwachstellen zugeordnet. Die Schwachstellen der containerd reichen von der fehlerhaften Einrichtung von Gruppen innerhalb des Containers und der daraus folgenden Umgehung von Gruppenbeschränkungen über Dateizugriff auf dem Host bis hin zu Berechtigungsänderungen und unerlaubtes durchsuchen von Dateien und Verzeichnissen des Hosts. Runc ist für die Interaktion mit dem unterliegenden Betriebssystem, die Isolation der Container und des Dateisystems zuständig und übernimmt. Entsprechend fokussieren sich die Schwachstellen dieser Container-Runtime-Komponente auf das Dateisystem und dem Ausbruch auf das Host-Betriebssystem. Eine einzelne Schwachstelle wurde in containerd-shim gefunden.

## 4.4 Auswertung

Unter Einbezug der Angriffsvektoren und der aktuellen Schwachstellen, die in die Taktiken der Bedrohungsmatrix eingeordnet wurden, entsteht die in Abbildung 4.2 dargestellte Matrix. So entsteht ein gesamtheitliches Bild über die Bedrohungen eines Kubernetes Clusters. Am häufigsten können also Techniken des Lateral Movement genutzt werden, um im Cluster weitere Bereiche zu kompromittieren. Aber auch die Taktiken Execution und Defense Evasion sind mit ihren Techniken im Kubernetes Cluster am meisten nutzbar. Der

initiale Zugang zum Cluster konnte auch mithilfe der Angriffsvektoren und CVEs nicht mit weiteren Techniken ergänzt werden. Dort bleibt es bei den bekannten Zugriffsmöglichkeiten. Neu hinzugekommen ist die Exfiltration Taktik, die für den Daten-Diebstahl aus dem Netzwerk steht. Ebenso neu hinzugekommen sind Bedrohungstechniken die das Netzwerk und die Konfiguration des Clusters betreffen. Um ein noch größeres Verständnis dafür zu schaffen, welche Bedrohungstechnik sich auf welche Komponente bezieht, wurden diese zugeordnet. Die Zuordnung von Bedrohungstaktik, -technik und Kubernetes-Komponente ist in Tabelle 4.3 dargestellt. Die Zuordnung erfolgte aufgrund der Beschreibungen, aus denen sich die Komponenten ergeben, an denen diese Techniken Anwendung finden. Dabei konnten die Techniken nicht immer genau einer Komponente zugeordnet werden, weswegen sich diese in vielen Fällen doppeln. Die Übersicht der Zuordnung stellt heraus, wo die wirklichen Angriffsvektoren für das Cluster liegen. Am häufigsten und in sieben von 11 Taktiken finden sich Bedrohungstechniken des Containers wieder. Dabei sind die jeweiligen Techniken fast schon gleichmäßig über die Taktiken verteilt. Da Pods in engen Zusammenhang mit den Containern stehen, sind auch deren Bedrohungstechniken eng mit denen der Container verknüpft und identisch. Weiterhin sollte laut Tabelle auch auf der Management Ebene und dort vor allem auf den Controllern, dem API-Server und dem etcd ein erhöhter Sicherheitsfokus liegen. Ebenso wie das Netzwerk mit sieben Bedrohungstechniken sich in die Reihe einfügt. Auch die Cloud-Provider API, wie auch der Metadata-Service des Cloud-Providers sollten im Rahmen einer Sicherheitserhöhung nicht unbeachtet bleiben. Auch aus den vergangenen Schwachstellen können einige Erkenntnisse gewonnen werden. So war der API-Server am häufigsten anfällig für Schwachstellen, die meist im Bereich Privilege Escalation festgestellt wurden. Auch das etcd und die weiteren Bestandteile der Control-Ebene des Clusters, wie kubectl und Kubelet nehmen hierbei einen großen Bedrohungsvektor ein. Die Schwachstellen der Container-Runtime Docker verteilen sich nahezu gleichmäßig auf die Bestandteile dockerd (11) containerd (8) und runc (8). Im Kontrast zu Kubernetes API-Server mit 17 Schwachstellen, nimmt die Container-Runtime Docker im Vergleich jedoch mit 28 Schwachstellen Platz eins der am häufigsten betroffenen Komponenten ein.

Mit den in diesem Kapitel vorgestellten Herausforderungen und Angriffsvektoren und der erweiterten Bedrohungsmatrix wird eine Angriffsvektoren Heatmap erstellt. Diese Heatmap wird anhand von Punkten, die den einzelnen Komponenten zugeordnet werden, erstellt. Dabei werden die Punkte wie folgt vergeben:

- Hat die Komponente einen oder mehrere Ports (**1 Punkt**)
- Hat die Komponente eine nicht authentifizierte Schnittstelle (**3 Punkte**)
- Ist der Komponente ein oder mehrere Bedrohungstechniken zugeordnet (**je Technik 1 Punkt**)

Tabelle 4.3: Zuordnung der Bedrohungstaktiken und Techniken zu den jeweiligen Komponenten des Clusters. Die Bedrohungstaktiken und Techniken resultieren aus der Bedrohungsmatrix von Microsoft, die mit den Angriffsvektoren und Schwachstellen ergänzt wurde.

| Komponente | Taktik | Technik |
|---|---|---|
| Cloud Provider API | **Initial Access** | Using Cloud Credentials |
| | | Kubeconfig Datei |
| | **Privilege Escalation** | Access Cloud Ressources |
| | **Credential Access** | Access Managed Identity Credential |
| | **Lateral Movement** | Access Cloud Ressources |
| Container | **Initial Access** | Schwachstelle in einer Anwendung |
| | **Execution** | Exec in einen Container |
| | | bash/cmd innerhalb Containers |
| | | SSH-Server innerhalb des Containers |
| | | Anwendungs-Exploit (RCE) |
| | **Persistenz** | Backdoor Container |
| | | Writable hostPath Mount |
| | **Privilege Escalation** | Privileged Container |
| | | hostPath mount |
| | | Schwächen in der Konfiguration |
| | **Defense Evasion** | Pod und Container Name Similarity |
| | | Schwächen in der Konfiguration |
| | | Schadcode innerhalb des Images tarnen |
| | **Credential Access** | Mount Service Principal |
| | | Access Container Service Account |
| | | Application Credentials in Configuration Files |
| | **Lateral Movement** | Container Service Account |
| | | Application Credentials in Configuration Files |
| | | Writable Volume Mounts on the host |
| | | Using an vulnerability inside the container |
| (z.B. Controller) Control Plane | **Persistenz** | Kubernetes CronJob |
| | | Infizierter Admission Controller |
| | | manipuliere externalIP eines Service (CVE-2020-8554) |
| | **Privilege Escalation** | (CVE-2024-3177) |
| | **Credential Access** | (CVE-2024-3177) |
| | **Discovery** | Malicious Admission Controller |
| | **Collection** | (CVE-2020-8554) |
| | **Impact** | (CVE-2020-8554) |
| Node | **Execution** | Kubelet durch eine modifizierte Verison ersetzen. |
| | **Persistenz** | Kubelet durch eine modifizierte Verison ersetzen. |
| | **Defense Evasion** | Kubelet durch eine modifizierte Verison ersetzen. |
| | **Impact** | Physischer Zugriff auf den Node |
| API-Server | **Execution** | neuer Container |
| | **Discovery** | Access the Kubernetes API-Server |
| etcd | **Credential Access** | List Kubernetes Secrets |
| Kubelet API | **Discovery** | Access Kubelet API |
| Netzwerk | **Defense Evasion** | Schwächen in der Netzwerkisolation |
| | **Discovery** | Network Mapping |
| | **Exfiltration** | Unverschlüsselte Daten aus dem Netzwerk übertragen. |
| | **Lateral Movement** | Cluster internal Networking |
| | | ARP Poisining and IP Spoofing |
| | | CoreDNS Poisining |
| | | Netzwerkregeln manipulieren |
| Pods | **Execution** | Siudecar Injection |
| | **Privilege Escalation** | Schwächen in der Konfiguration |
| | **Defense Evasion** | Pod und Container Name Similarity |
| | | Schwächen in der Konfiguration |
| | **Credential Access** | Application Credentials in Configuration Files |
| | **Lateral Movement** | Application Credentials in Configuration Files |
| Interfaces (Kubernetes Dashboard, kubectl) | **Initial Access** | exponierte sensible Schwachstelle |
| | **Discovery** | Access Kubernetes Dashboard |
| | | Access the Kubernetes API-Server |

| Taktiken / Techniken | Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion | Credential Access |
|---|---|---|---|---|---|---|
| Bedrohung | Using Cloud Credentials | Exec in einen Container | Backdoor Container | Privilegierter Container | Container Logs löschen | Kubernetes Secrets auflisten |
| Bedrohung | Kompromitiertes Image in Registry | bash/cmd innerhalb Containers | Writable hostpath mount | Cluster-Admin Binding | Delete Kubernetes Events | Mount Service Principal |
| Bedrohung | Kubeconfig Datei | Neuer Container | Kubernetes Cronjob | hostPath mount | Pod/container name similarity | Zugriff auf Container Service Account |
| Bedrohung | Schwachstelle in einer Anwendung | Anwendungs-Exploit (RCE | Malicious Admission Controller | Access Cloud Ressourcen | Verbindung von Proxy Server | Anwendungs-Zugangsdaten in Konfigurationsdateien |
| Bedrohung | exponierte sensible Schnittstellen | SSH-Server innerhalb des Containers | manipuliere externalIP eines Service (CVE-2020-8554) | Umgehen einer Secret-Richtlinie des ServiceAccount Admission Plugin (CVE-2024-3177) | Kubelet durch eine modifizierte Version ersetzen | Zugriff auf managed Identität-Zugangsdaten |
| Bedrohung | | Sidecar Injection | Kubelet durch eine modifizierte Version ersetzen | Kubelet durch eine modifizierte Version ersetzen | Schwächen in der Konfiguration | Böswilliger AdmisionController |
| Bedrohung | | Kubelet durch eine modifizierte Version ersetzen | | Schwächen in der Konfiguration führen zur Manipulation von Konfigurationsdateien | Schwächen in der Netzwerk- isolation | Umgehen einer Secret-Richtlinie des ServiceAccount Admission Plugin (CVE-2024-3177) |
| Bedrohung | | Schwachstellen in Add-ons oder Plugins | | | Schadcode innerhalb des Images tarnen | |

| Taktiken / Techniken | Discovery | Exfiltration | Lateral Movement | Collection | Impact |
|---|---|---|---|---|---|
| Bedrohung | Zugriff auf Kubernetes API-Server | Unverschlüsselte Daten der Discovery-Phase aus dem Netzwerk übertragen. | Zugriff auf Cloud-Ressourcen | Images einer privaten Registry | Data Desctruction |
| Bedrohung | Zugriff auf die Kubelet API | | Container Service Account | manipuliere externalIP eines Service (CVE-2020-8554) | Resource Hijacking |
| Bedrohung | Netzwerk- Mapping | | Cluster-internes Networking | | Denial of Service |
| Bedrohung | Zugriff auf das Kubernetes Dashboard | | Anwendungs-Zugangdaten in Konfigurationsdateien | | manipuliere externalIP eines Service (CVE-2020-8554) |
| Bedrohung | Instance Metadata API | | Schreibbare Volumen mounts auf dem Host | | Physischer Zugriff auf den Node |
| Bedrohung | | | CoreDNS Poisining | | |
| Bedrohung | | | ARP poisoning und IP spoofing | | |
| Bedrohung | | | Netzwerkregeln manipulieren | | |
| Bedrohung | | | Nutzung einer Schwachstelle innerhalb des Containers | | |

Abbildung 4.2: Erweiterte Bedrohungsmatrix für Kubernetes Cluster. Ergänzt um die Techniken der Angriffsvektoren (blau markiert) und der aktuellen Schwachstellen CVE-2020-8554 und CVE-2024-3177 (orange markiert) für Kubernetes Cluster. Aufteilung der Bedrohungstechniken in die 11 Taktiken Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Exfiltration und Impact. Die Techniken der verschiedenen Grundlagen kann ein Angreifer nutzen, um das Cluster anzugreifen. Grundlage dieser Matrix ist die von Microsoft entwickelte Matrix aus dem Jahr 2021 unter Grundlage des MITRE ATT&CK Frameworks [Weia]

- Hat die Komponente eine aktuelle (zum Zeitpunkt dieser Arbeit) Schwachstelle (**je Schwachstelle 5 Punkte**)

- Hat die Komponente weniger als 5 Schwachstellen in vergangenen Versionen (**1 Punkt**)

- Wurden der Komponente mehr als 5, aber weniger als 10 Schwachstellen in der Vergangenheit zugeordnet (**3 Punkte**)

- Wurden der Komponente in der Vergangenheit mehr als 10 Schwachstellen zugeordnet (**5 Punkte**)

Jeder Punkt wurde so gewichtet, dass er die potenziellen Bedrohungen und Schwachstellen angemessen widerspiegelt.
Ein offener Port erhält 1 Punkt, da er zwar als Schnittstelle für den Datenverkehr dient, aber auch eine potenzielle Angriffsmöglichkeit darstellt. Ein offener Port allein ist noch kein großes Sicherheitsrisiko, solange die richtigen Sicherheitsmaßnahmen implementiert sind. Dennoch erhöht er das Risiko, wenn diese Maßnahmen nicht konsequent umgesetzt werden.
Eine nicht authentifizierte Schnittstelle wird mit 3 Punkten bewertet, da sie ein signifikant höheres Risiko birgt. Solche Schnittstellen ermöglichen es Angreifern, auf Funktionen oder Informationen zuzugreifen, ohne dass eine Authentifizierung erforderlich ist. Dies öffnet die Tür für unbefugte Dritte und kann zu einer direkten Ausnutzung der Komponente führen.
Die Zuordnung von Bedrohungstechniken zu einer Komponente zeigt, dass sie potenziell für spezifische Angriffe anfällig ist. Jede zugeordnete Technik erhöht das Risiko, dass die Komponente erfolgreich angegriffen wird. Daher wird für jede identifizierte Technik ein Punkt vergeben, um die erhöhte Bedrohungslage abzubilden.
Aktuelle Schwachstellen stellen die gravierendste Bedrohung dar, da sie Angreifern sofortige Möglichkeiten zur Ausnutzung bieten, bevor die Schwachstellen gepatcht oder behoben werden. Aus diesem Grund wird jeder aktuellen Schwachstelle 5 Punkte zugewiesen, was die Dringlichkeit und das Risiko einer sofortigen Behebung unterstreicht.
Vergangene Schwachstellen liefern wertvolle Informationen über die historische Sicherheit und das Patch-Management einer Komponente. Eine geringere Anzahl vergangener Schwachstellen (weniger als 5) deutet auf eine relativ stabile Sicherheit hin und wird daher nur mit 1 Punkt bewertet. Bei einer mittleren Anzahl von Schwachstellen (zwischen 5 und 10) wird das Risiko höher eingeschätzt und mit 3 Punkten bewertet. Hat eine Komponente jedoch mehr als 10 Schwachstellen in der Vergangenheit, deutet dies auf systematische Sicherheitsprobleme hin, die eine erhöhte Aufmerksamkeit erfordern, weshalb hierfür 5 Punkte vergeben werden.

Die Punktevergabe je Komponente, wie in Tabelle 4.4 dargestellt, ergibt sich aus den zuvor erläuterten Kriterien. Diese Bewertung hilft dabei, das relative Risiko jeder Komponente innerhalb des Kubernetes Clusters zu quantifizieren und zu priorisieren. Beispielsweise erhält die Cloud-Provider API sechs Punkte. Dies basiert auf der Tatsache, dass dieser Komponente die fünf Bedrohungstechniken zugeordnet werden und diese API ebenfalls einen Port bereitstellt.
Der Pod erhält in der Bewertung insgesamt sieben bis zehn Punkte. Sechs Punkte ergeben sich durch die Bedrohungstechniken, die von der Sidecar Injection bis hin zu Anwendungs-Zugangsdaten in Konfigurationsdateien reichen. Ein Punkt entstammt dem Hintergrund,

Tabelle 4.4: Bewertung der Angriffsvektoren, Schwachstellen und Bedrohungstaktiken einer Kubernetes-Architektur-Komponente. Einen Punkt gibt es für einen oder mehrere Ports. Drei Punkte für eine nicht authentifizierte Schnittstelle. Je Bedrohungstechnik der Komponente gibt es einen Punkt. Für eine aktuelle Schwachstelle gibt es zehn Punkte und für die Anzahl der zugeordneten Schwachstellen aus der Vergangenheit gibt es einen Punkt bei weniger als fünf Schwachstellen, drei Punkte bei mehr oder gleich fünf aber weniger als zehn Schwachstellen und fünf Punkte bei mehr oder gleich zehn Schwachstellen.

| Komponente | Punkte |
|---|---|
| Cloud-Provider API | 6 |
| Container Runtime | 26 |
| Pods | 7 - 10 |
| Control Plane (z. B. Controller) | 19 |
| Node | 4 |
| API-Server | 8 |
| etcd | 5 |
| Netzwerk | 10 |
| Kubelet | 8 |
| Interfaces (Kubernetes Dashboard, kubectl) | 6 |

dass ein Pod einen Port für seine Anwendung bereitstellt (sogenannte NodePorts, siehe Tabelle 4.2). Wenn diese nicht entsprechend authentifiziert werden, ergibt sich ein größerer Angriffsvektor, weswegen bis zu drei Punkte zusätzlich hierfür vergeben werden.
Am meisten Punkte in der Übersicht erhielt der Kube-API-Server mit 26 Punkten. Diese ergeben sich überwiegen aus der hohen Anzahl an Bedrohungstechniken und den 28 vergangenen Schwachstellen und der einen aktuellen Schwachstelle. Die Control Plane erhielt 19 Punkte basierend auf den beiden aktuellen Schwachstellen (CVE-2024-3177 und CVE-2020-8554) und den zusätzlichen vier weiteren Bedrohungstechniken, die dieser Komponente zugeordnet wurden.
Die Netzwerk-Komponente des Clusters kommt in der Auswertung auf genauso viele Punkte, wie die Pods. Mit sieben Bedrohungstechniken und weiteren drei Punkten für die sechs vergangenen Schwachstellen, kommt das Netzwerk des Clusters somit auf zehn Punkte.

Die daraus resultierende Heatmap ist in Abbildung 4.3 dargestellt. Bei der Einordnung der Farben wurde die Folgende Einordnung genutzt:

- **keine Färbung**: 0 Punkte,
- **grün**: 1 bis 5 Punkte,
- **gelb**: 6 bis 10 Punkte,
- **orange**: 11 bis 20 Punkte,

- **rot**: ab 21 Punkte

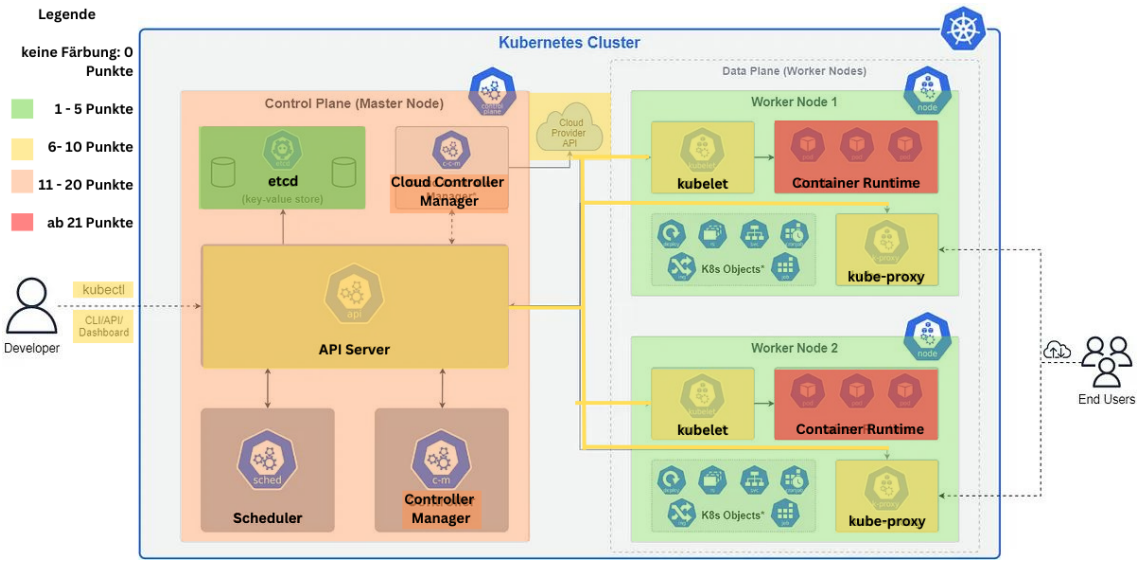



Abbildung 4.3: Schwachstellen-Heatmap basierend auf der Kubernetes Cluster-Architektur (Abbildung 2.3). Unterteilung der Bedrohungs-Gefahren für die einzelnen Komponenten nach den vergebenen Punkten (siehe Tabelle 4.4): 0 Punkte (keine Färbung), 1-5 Punkte (grün), 6-10 Punkte (gelb), 11-20 Punkte (orange), ab 21 Punkte (rot).

Aus der Heatmap geht somit eine ganzheitliche Bedrohungslage für das gesamte Cluster hervor. Am wenigsten betroffen sind die Worker Nodes. Die mit nur vier Punkten entsprechend grün eingefärbt wurden. Die Control Plane, die mit 19 Punkten orange und die Container Runtime mit 26 Punkten (rot)sind die gefährdetsten Bestandteile des Clusters. In gelb dazwischen finden sich alle Bestandteile, die zur API-Kommunikation des Clusters gehören, einschließlich der Cloud Provider API in öffentlichen Clouds.

# 5 Security Best-Practices & Zero-Trust-Architektur (ZTA)

In diesem Abschnitt der Arbeit werden die Security Best Practices, die Zero-Trust-Regeln und die Zero-Trust-Architektur für Kubernetes vorgestellt und implementiert, um zentrale Erkenntnisse für die Beantwortung der Forschungsfragen FF.1 und FF.5 zu liefern. Dazu wird zunächst sowohl das Setup und die Umgebung des Minikube Clusters und des EKS Clusters vorgestellt (Kapitel 5.1). Anschließend werden die Security Best Practices für Kubernetes vorgestellt und implementiert (Kapitel 5.2). Darauf aufbauend werden die Zero-Trust-Regeln der Cloud Native Computing Foundation vorgestellt (Kapitel 5.3), die anschließend konkret für Kubernetes Cluster vorgestellt und anschließend in Minikube und EKS implementiert werden (Kapitel 5.4). Die Implementierung beantwortet somit die Forschungsfrage FF.1. Zum Schluss wird die erstellte Architektur dargestellt und die einzelnen Komponenten anhand der Beispielapplikation hinsichtlich ihrer Komplexität und dem notwendigen Zeitaufwand eingeordnet (Kapitel 5.5). Anhand dieser Zusammenfassung wird Forschungsfrage FF.5 beantwortet.

## 5.1 Cluster-Setup: Minikube und Amazon EKS

In diesem Kapitel werden die Host-Umgebung und das Cluster-Setup der beiden Kubernetes-Cluster Minikube und Amazon EKS vorgestellt.

### 5.1.1 Minikube

Minikube ist ein ist eine leichtgewichtige K8s Distribution, die in der Regel zu Testzwecken auf Entwicklerrechner eingesetzt wird, die aus einem einzelnen Node besteht, der sowohl als Worker als auch als Master fungiert. Sie ist besonders für das Verständnis und Lernen von Kubernetes geeignet und bietet umfangreiche Unterstützung im Netz, um Security Best Practices und Zero-Trust-Architektur-Bestandteile zu integrieren.

Das Minikube-Cluster wird auf einem Linux-System mit der Distribution KDE neon 5.26 bereitgestellt, das auf Ubuntu-Version 22.04 (jammy) basiert. Die Hardware umfasst 16 GB

RAM und 16 CPU-Kerne eines Intel Core i7 der 11. Generation (11800H) mit 2.30GHz. Die Minikube-Server-Version ist Kubernetes v1.30.0 und die Client-Version v1.30.1.

Um das Minikube-Cluster zu starten, wird der Befehl `minikube start` verwendet. Durch den Parameter `--cni calico` wird das Container Networking Interface Calico installiert. Da Istio mindestens 8 GB RAM und 4 CPUs benötigt, werden mit den Parametern `--memory=8192mb --cpus=4` diese Anforderungen beim Setup von Minikube berücksichtigt.

### 5.1.2 Amazon EKS

Das Amazon EKS-Cluster wird auf AWS als Amazon Elastic Kubernetes Service bereitgestellt und besteht aus zwei Nodes. Beide Nodes laufen auf dem von Amazon bereitgestellten Betriebssystem Linux Amazon 2 und sind als Instanzen des Typs t3.large konfiguriert, welche jeweils 2 CPU-Kerne (2000ms) und 7,65 GB Arbeitsspeicher bereitstellen. Die Kubelet-Version ist v1.29.3-eks-ae9a62 und entspricht der Kubernetes-Version 1.29.

Das Cluster wurde von einem DevOps-Team eingerichtet, daher wird auf die spezifische Installation im Rahmen dieser Arbeit nicht näher eingegangen.

### 5.1.3 Installation: Beispiel-Anwendung

Für die Implementierung und den Test einer Zero-Trust-Architektur wird die Bookinfo-Anwendung von Istio genutzt. Diese Anwendung ist eine geeignete Wahl, da sie als Beispiel für die Funktionen von Istio gut dokumentiert ist und den Anforderungen der Cloud Native Computing Foundation für eine Zero-Trust-Architektur entspricht. [ista]

Die Bookinfo-Anwendung besteht aus mehreren Komponenten, die in Abbildung 5.1 dargestellt sind. Die Anwendung umfasst sechs Pods, die in verschiedenen Programmiersprachen entwickelt wurden: Python (Productpage), Java (Review-Container), NodeJS (Ratings) und Ruby (Details-Container). Jeder Pod verfügt über einen Service, der den Port der Anwendung (9080) definiert und jeweils einen Service Account, wobei die drei Review-Pods den Service-Account bookinfo-reviews und den Service gemeinsam nutzen.

Die Funktionsweise der Anwendung ist wie folgt: Anfragen werden an die Productpage weitergeleitet, die anschließend die Services Review und Details aufruft. Die Pods Reviews-v2 und Reviews-v3 rufen zusätzlich die Ratings-Anwendung auf, bevor die Productpage die Webseite mit allen Informationen bereitstellt.

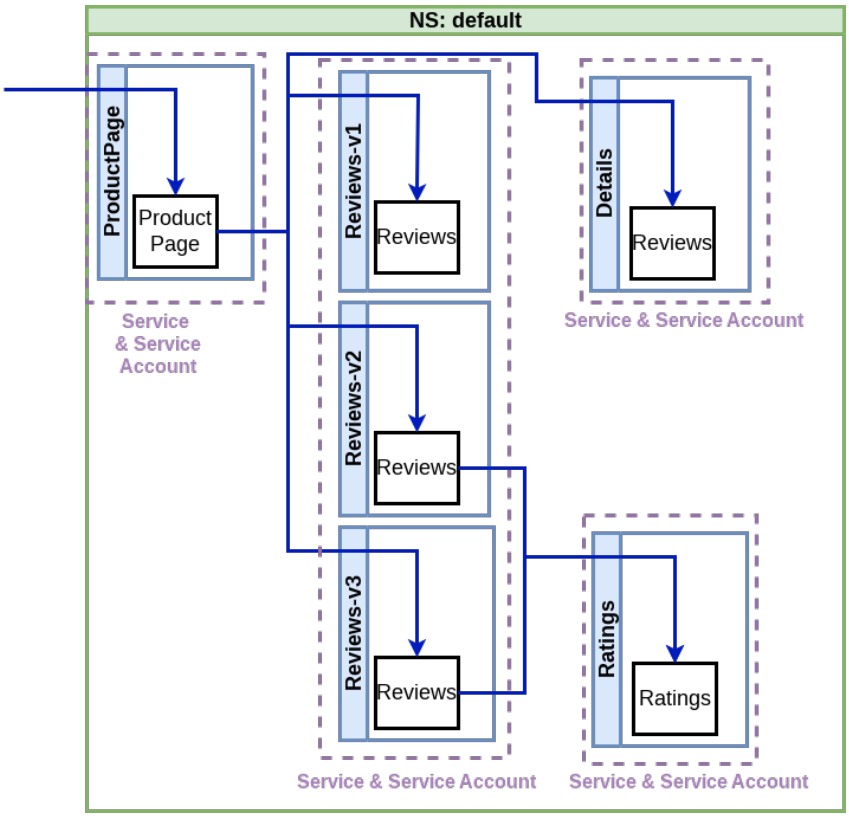



Abbildung 5.1: Bookinfo-Anwendung mit allen Anwendungen. Die Procutpage ist in Python geschrieben. Reviews sind in Java geschrieben. Ratings stellt eine NodeJS-Anwendung dar. Details wird mit Ruby ausgeführt. Die Darstellung zeigt den Netzwerkverkehr der Anwendung in dunkelblau. In Lila gestrichelt sind Service und Service Account dargestellt.

### 5.1.4 Installation von Istio

Istio ist eine Service Mesh Installation, die in Kapitel 5.4 bei der Integration der Zero-Trust-Architektur eine wesentliche Rolle spielt. Im Folgenden wird die Installation der Istio-Management-Ebene in Minikube und EKS durchgeführt.

**Minikube:** Um Istio im Minikube-Cluster bereitzustellen, können zwei Add-ons genutzt werden. Diese werden mit den Befehlen `minikube addons enable istio-provisioner` und `minikube addons enable istio` aktiviert. [k8s]

**EKS:** Im Amazon EKS-Cluster besteht ebenfalls die Möglichkeit, ein Service Mesh von Amazon zu nutzen, nämlich *AWS App Mesh* [amad]. Für einen fairen Vergleich zwischen Minikube und EKS wird jedoch auf diese Option verzichtet, da AWS App Mesh nicht nativ in Kubernetes integriert ist und auch für andere Instanzen und Dienste der Amazon Web Services (AWS) verwendet werden kann [amad]. Stattdessen wird Istio mithilfe von `istioctl` und dem Befehl `../istio-a.22.1/bin/istioctl install --set profile=demo` im EKS-Cluster installiert. [istb]

### 5.1.5 Installation des Open Policy Agenten

Um den Setup-Schritt zu vervollständigen, wird der Admission Controller des Open Policy Agent (OPA) in die jeweiligen Cluster integriert.

**Minikube und EKS:** Die Bereitstellung des Admission Controller des Open Policy Agenten erfolgt sowohl in Minikube als auch in EKS über die `quick_start.yaml`-Datei aus der Dokumentation des OPA. Diese Datei enthält eine Istio `AuthorizationPolicy`, die Istio anweist, sämtliche Autorisierungsprüfungen an den OPA-Envoy-Sidecar weiterzuleiten. Damit Istio den Endpunkt des OPA-Envoy-Sidecar-Containers findet, enthält die `quick_start.yaml` zudem einen `ServiceEntry`. Der Namespace `opa-istio` beinhaltet nun einen AdmissionController, der den OPA-Envoy-Sidecar automatisch in die Pods integriert, wenn der Namespace der Pods das Label `opa-istio-injection=enabled` trägt. [opeb] Anschließend muss die ConfigMap für Istio angepasst werden, um einen `extensionProvider` zu definieren, der auf den OPA-Envoy-Container verweist (siehe Listing A.1).

## 5.2 Security Best Practices

Im Folgenden werden die Security Best Practices für Kubernetes-Cluster vorgestellt. Sie werden unterteilt in Best Practices für Kubernetes-Komponenten, wie den API-Server und die Kubelet-API (siehe Kapitel 5.2.1) und Best Practices für Container (siehe Kapitel 5.2.2). Zu den Best Practices wird die jeweilige Implementierung in Minikube und EKS beschrieben.

### 5.2.1 Best Practices für Kubernetes-Komponenten

Die Analyse der Angriffsvektoren und Schwachstellen hat gezeigt, dass die Sicherheit des Kube-API-Servers und der Kubelet-API von Bedeutung ist. Die erste Verteidigungslinie ist daher die Beschränkung und Sicherung des Zugriffs auf API-Anfragen, da diese die Kontrolle über die Kubernetes-Plattform ermöglichen. In Kubernetes ist eine Authentifizierung erforderlich, bevor eine Anfrage autorisiert werden kann. Kubernetes erwartet hierfür Attribute, die für REST-API-Anfragen üblich sind. Das bedeutet, dass bestehende unternehmensweite oder Cloud-Provider-weite Zugriffskontrollsysteme, die andere APIs verarbeiten, auch mit der Kubernetes-Autorisierung funktionieren. API-Anfragen, die über den API-Server autorisiert werden, enthalten standardmäßig keine Berechtigungen. Alle Attribute der Anfrage werden anhand von Richtlinien überprüft, und die Anfrage wird entweder genehmigt oder abgelehnt. Jeder Teil der API-Anfrage muss von einer Richtlinie zugelassen werden, um fortfahren zu können. [owa]

### Externe Authentifizierung

Aufgrund der Schwächen der internen Authentifizierungsmechanismen von Kubernetes wird empfohlen, in größeren Clustern eine externe API-Authentifizierung zu verwenden. Hierfür bieten sich mehrere Optionen an. Durch die Nutzung von OpenID Connect (OIDC) kann die Authentifizierung ausgelagert werden. OIDC verwendet kurzlebige Tokens und ermöglicht die Einrichtung zentralisierter Gruppen für die Autorisierung. Public-Cloud-Provider bieten verwaltete Kubernetes-Distributionen wie Google Kubernetes Engine (GKE), Elastic Kubernetes Service (EKS) und Azure Kubernetes Service (AKS) an, die eine Integration mit den IAM-Authentifizierungssystemen des jeweiligen Providers unterstützen. Kubernetes-Impersonation kann sowohl in Public-Cloud- als auch in Private-Cloud-Clustern verwendet werden, um die Authentifizierung extern zu verlagern, ohne auf die Konfigurationsparameter des API-Servers zugreifen zu müssen. [owa]
Externe Authentifizierung ist in produktiven Clustern wichtig, wird jedoch in dieser Arbeit nicht weiter behandelt, da der Fokus auf einer Zero-Trust-Architektur und nicht auf externen Authentifizierungsmechanismen liegt.

### Role-based Access Control

Rollenbasierte Zugriffskontrolle (RBAC) regelt den Zugang zu Computer- oder Netzressourcen basierend auf den Rollen einzelner Benutzer innerhalb einer Organisation. Kubernetes verfügt über eine integrierte RBAC-Komponente mit Standardrollen, die es ermöglichen, Benutzerverantwortlichkeiten nach ihren Aktionen zu definieren. Es wird empfohlen, RBAC zusammen mit dem Node Authorizer und dem NodeRestriction Admission Plugin zu verwenden. Die RBAC-Komponente ordnet Benutzern oder Gruppen eine Reihe von Berechtigungen zu, die mit Rollen verknüpft sind. Diese Rollen kombinieren Aktionen (Verben wie lesen, erstellen, löschen) mit Ressourcen (zum Beispiel Pods, Services, Nodes). Diese Berechtigungen können auf einen Namespace oder das gesamte Cluster angewendet werden. RBAC verwendet die API-Gruppe `rbac.authorization.k8s.io` für Autorisierungsentscheidungen, was eine dynamische Konfiguration von Richtlinien über die Kubernetes-API ermöglicht. Zur Umsetzung von Role-Based-Accounts innerhalb des Clusters werden Service Accounts mit `Role` und `ClusterRole` genutzt. Die `Role` bezieht sich auf einen Namespace, während eine `ClusterRole` Cluster-weit gilt. Beide definieren die Rechte der Aktionen, die ein Nutzer ausführen darf. Kubernetes ermöglicht dabei auch die Erstellung von benutzerdefinierten Rollen oder die Verwendung von Standardrollen wie Admin, Edit oder View. [Say17, owa]

**Minikube:** Bei der Installation von Minikube ist Role-based-Access Control auf dem API-Server standardmäßig aktiviert. Hierfür wird der parameter `AuthorizationMode=Node, RBAC` gesetzt.

**EKS:** Im EKS-Cluster ist RBAC ebenfalls standardmäßig aktiviert. Entsprechend müssen hierbei nur die entsprechenden Rollen und RoleBindings angelegt werden, um den Zugriff zu verwalten. Allerdings benutzt das in dieser Arbeit bereitgestellte EKS Cluster nicht die von AWS bereitgestellte Identity and Access Management (IAM) Funktionalität, sondern wird über die IAM-Prinzipale nur aus der aws-auth-ConfigMap konfiguriert. Anschließend können aber im Cluster ganz normal RBAC-Ressourcen bereitgestellt werden.

#### Kubelets

Die Kubelets stellen HTTPS-Endpunkte bereit, die standardmäßig einen nicht authentifizierten Zugriff auf die API ermöglichen. In Produktionsumgebungen sollten Authentifizierung und Autorisierung aktiviert werden. [owa]

**Minikube:** In Minikube kann die Authentifizierung und Autorisierung für die Kubelets über die Kubelet-Konfiguration bearbeitet werden. Dazu wird die kubelet Konfigurationsdatei geöffnet und die in Listing A.2 dargestellten Authentifizierungsparameter hinzugefügt. Dadurch wird die anonyme Authentifizierung deaktiviert und die webhook-Authentifizierung aktiviert. Weiterhin bestände die Möglichkeit den `authorization-mode` einfach auf Webhook festzulegen, dies kann über die gleiche Konfigurationsdatei oder über die kubelet-Startparameter erfolgen. Wird Webhook Authentifizierung jedoch genutzt sollte sichergestellt werden, dass ein Webhook-Server verfügbar ist, der anschließend ebenfalls in der webhook-config.yaml eingetragen werden muss. Anschließend wird der Kubelet-Dienst neugestartet. [kubg]

**EKS:** Da die Knoten von AWS verwaltet werden können, ist der Zugriff auf die Kubelet-Parameter nur begrenzt möglich. Zudem ist die primäre Methode zur Verwaltung des Zugriffs in EKS die Verwendung von IAM-Rollen und RBAC. Von einer manuellen Konfiguration sollte entsprechend abgesehen werden. Ähnlich wie in Minikube werden die Startparameter des Kubelets angepasst, sodass die Token-Webhook-Authentifizierung aktiviert wird.

#### Namespaces

Namespaces ermöglichen die Erstellung von logischen Partitionen innerhalb des Clusters. Sie erzwingen eine Trennung der Ressourcen und beschränken den Bereich für Nutzerberechtigungen. [owa]

**Minikube & EKS:** In Minikube und EKS können namespaces über `kubectl create namespace <Name>` angelegt werden, anschließend müssen Pods, Deployments und Services über deren Konfiguration diesem Namespace hinzugefügt werden. [owa]

### Limitierung des Ressourcenverbrauchs

Mit Kubernetes und Pod-Ressourcen-Limits lassen sich Begrenzungen für den CPU- und Speicherverbrauch festlegen. Dadurch wird das Risiko von Denial-of-Service-Angriffen und noisy neighbor-Szenarien reduziert. Durch die Festlegung einer initialen CPU- und Speichergröße kann der Anfangsressourcenverbrauch definiert werden. Benötigt der Pod während der Ausführung mehr Ressourcen, definieren Limits, wie viel insgesamt genutzt werden darf. [kubi]

**Minikube & EKS:** Bei der Implementierung von Ressourcen Limits in Minikube und EKS wurde deutlich, dass ein detailliertes Verständnis des CPU- und Speicherverbrauchs eines Pods notwendig ist, um geeignete Ressourcengrenzen festzulegen. Zwar können allgemeine Richtwerte genutzt werden, doch der Ressourcenbedarf variiert erheblich in Abhängigkeit von der Programmiersprache der jeweiligen Anwendung. So benötigt eine Java-Anwendung beispielsweise deutlich höhere Anfangswerte für den Start, während eine Ruby-Anwendung nur wenige Ressourcen beansprucht. Für den Container der Produktseite wurden die Speichergrenzen auf 200 bis 400 MiB und die CPU-Grenzen auf 500 bis 2000 ms festgelegt, wie in Listing A.3 dargestellt. Werden die Ressourcengrenzen beim Start des Containers zu niedrig angesetzt, kann Kubernetes den Container mit dem Status `OOMKilled` beenden.

### Pod-Unterbrechungsbudget

Das Pod-Unterbrechungsbudget eignet sich für hochverfügbare Anwendungen. Es unterscheidet zwischen freiwilligen und unfreiwilligen Unterbrechungen. Ein Pod verschwindet nicht einfach, es sei denn, er wird gelöscht oder es tritt ein unvermeidbarer Hardware- oder Systemfehler auf. Unfreiwillige Unterbrechungen umfassen alle unvermeidbaren Fälle, während freiwillige Unterbrechungen zum Beispiel das Löschen des Deployments oder einen Neustart des Pods umfassen. Bevor ein Pod-Unterbrechungsbudget eingeführt wird, sollte geprüft werden, ob unfreiwillige Unterbrechungen im Cluster aktiviert sind. Andernfalls sind Unterbrechungsbudgets irrelevant. Um freiwillige Unterbrechungen zu verhindern, können Pod-Unterbrechungsbudgets eingesetzt werden. Diese Budgets können für jede Anwendung erstellt werden und beschränken die Anzahl der Pods einer verteilten Anwendung, die gleichzeitig durch freiwillige Unterbrechungen nicht verfügbar sein dürfen. Ein `PodDisruptionBudget` bietet zwei Konfigurationsmöglichkeiten: `minAvailable`, das die mindestens verfügbare Anzahl von Pods des Deployments angibt, und `maxUnavailable`, das die maximale Anzahl nicht verfügbarer Pods definiert. Diese Budgets helfen dem Cluster sicherzustellen, dass die benötigten Ressourcen jederzeit verfügbar sind.
Aufgrund der simplen Beispielanwendung, die im Rahmen dieser Arbeit verwendet wird, ist eine Definition von Pod-Unterbrechungsbudgets für die Bookinfo-Anwendung nicht erforderlich. [kubj]

### Kubernetes NetworkPolicies

Um die Kommunikation zwischen Pods im Cluster einzuschränken, wird empfohlen, eine Netzwerksegmentierung zu implementieren. Dadurch kann sichergestellt werden, dass ein Container/Pod nur mit autorisierten Containern kommunizieren kann, was Angreifer daran hindert, sich seitlich über Container hinweg zu bewegen. [kubh]
Diese Best Practice spielt eine wesentliche Rolle in der Zero-Trust-Architektur und wird daher im Kapitel 5.4 in Minikube und EKS implementiert.

### Secrets

Um die Sicherheit von Secrets in Anwendungen zu gewährleisten, sollten diese in read-only Dateipfade des Containers gemountet werden, anstatt sie als Umgebungsvariablen zu verwenden. Dadurch wird verhindert, dass Secrets unbeabsichtigt durch den Container selbst oder dessen Umgebung offengelegt werden. Es ist wichtig, Secrets von Images, Pods oder jeglichen anderen Komponenten fernzuhalten, auf die jemand mit Zugriff auf das Image zugreifen kann, da dies sonst unweigerlich auch den Zugriff auf die Secrets ermöglicht. Kubernetes unterstützt die Verschlüsselung von Secrets, die im etcd-Datenbanksystem gespeichert werden, durch *encryption at REST*. Dies stellt sicher, dass selbst bei einem unbefugten Zugriff auf etcd der Inhalt der Secrets nicht ohne Weiteres ausgelesen werden kann. Zusätzlich sollte in Erwägung gezogen werden, regelmäßige Backups von etcd zu erstellen und diese mit einer vollständigen Festplattenverschlüsselung zu sichern. Dies ist wichtig, da etcd alle Informationen enthält, die über die Kubernetes-API zugänglich sind. Als Alternative zum internen etcd-Secret-Speicher kann ein externer Secret-Manager verwendet werden. Ein externer Secret-Manager bietet mehrere Vorteile, darunter die Möglichkeit, Secrets über mehrere Cluster oder Clouds hinweg zu nutzen, sowie eine zentrale Verwaltung und Rotation der Secrets. Im nächsten Kapitel wird dieser externe Secret-Manager als Key Management Service vorgestellt. [owa, kubf]

**Minikube:** Das etcd kann durch ein Cluster-internes base64-kodiertes Secret verschlüsselt werden. Hierfür ist zunächst eine Encryption-Configuration anzulegen. Diese benötigt, wie in Listing A.4 zu sehen, einen Provider, der Keys erhält. Das Secret, was in Zeile 10 angegeben ist, ist in base64 kodiert worden. Anschließend wird diese Datei als Volume-Mount und als volume.hostPath der kube-apiserver-Konfiguration hinzugefügt. Ebenso, wie noch für den Start des Containers des API-Servers folgendes Argument hinzugefügt wird: `- --encryption-provider-config=/etc/kubernetes/etcd/ec.yaml`. Listing A.5 zeigt die `kube-apiserver.yaml`, reduziert um die relevanten Ausschnitte. Somit können die Secrets mithilfe des neuen Verschlüsselungs-Providers verschlüsselt werden. Dies wird mit folgendem Befehl erreicht: `kubectl -n <namespace> get secrets -o json | kubectl replace -f -`. Durch diesen Befehl werden die Secrets des Namespaces neu erstellt und durch den neuen Provider auch direkt verschlüsselt. [owa, kubf]

Die Verschlüsselung des etcd ist somit im Minikube Cluster sichergestellt. Allerdings liegt der Schlüssel, der zur Verschlüsselung genutzt wird, im Cluster, was dazuführt, dass jeder, der Zugriff auf die EncryptionConfiguration hat, den Schlüssel in Klartext umwandeln kann, da base64 kein Verschlüsselungsverfahren darstellt, sondern ein Kodierungsverfahren in eine Zeichenfolge ist. Der in Listing A.4 genutzte Schlüssel ist somit die Repräsentation des Klartextes: `this-is-very-sec`. [owa, kubf]

**EKS:** In EKS werden die EBS-Verzeichnisse für etcd-Knoten verschlüsselt, mit EBS Verschlüsselung. Somit ist das Verzeichnis, in dem das Secret liegt, bereits verschlüsselt. Da jedoch nur das Verzeichnis verschlüsselt ist, ist es ratsam den Einsatz eines Key-Management System für Defense-in-Depth zu planen. [amac]

### Key Management Service

Ein Key Management Service (KMS) bietet zusätzliche Sicherheit durch Verschlüsselung von Daten, die über ein KMS-Plugin kommuniziert wird. In Cloud-nativen Umgebungen wird häufig HashiCorp Vault verwendet, um Passwörter, Schlüssel und Zertifikate zu verwalten. Die Integration eines KMS-Plugins ist entscheidend, um Secrets sicher zu speichern und zu verwalten. Die Kommunikation zwischen dem KMS-Plugin und dem Kubernetes API-Server ist in Abbildung 5.3 dargestellt. [hasb]

In Cloud-nativen Umgebungen kommt meist für Passwörter, Schlüssel und Zertifikate HashiCorp Vault zum Einsatz. Dieser Vault verwaltet die Secrets und schützt sensitive Daten. Anwendungsdaten werden auch über das etcd im Cluster gespeichert. Dieser Vorgang ist in Abbildung 5.2 dargestellt. Diese Daten sollten verschlüsselt im Cluster liegen, jedoch sollte der Schlüssel, der zur Ver- und Entschlüsselung genutzt wird, nicht im Cluster liegen. Die Secrets, die intern von Kubernetes benötigt werden, können anders als Secrets, die von Anwendungen genutzt werden, nicht in das externe Vault abgelegt werden. (s. Kapitel 5.2.1 - Externe Secret-Verwaltung für Anwendungen) [hasb]

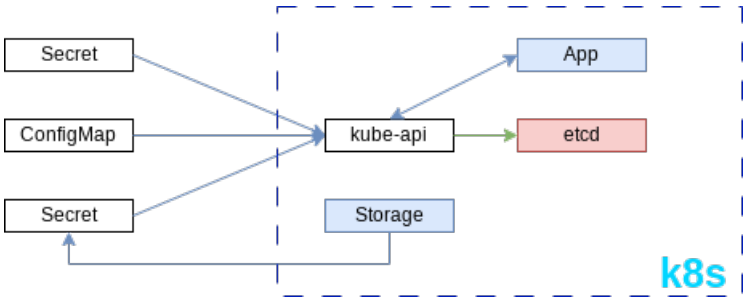



Abbildung 5.2: Darstellung der Kommunikation, wie im Kubernetes-Cluster Secrets verwaltet werden. Secrets, ConfigMaps werden über die kube-api aus dem etcd gelesen und im etcd gespeichert. Eine App bezieht die benötigten Secrets ebenfalls über den API-Server aus dem etcd.

Jedoch kommt HashiCorp Vault mit einem Service für die Ver- und Entschlüsselung von Secrets. Dieser Service heißt Vault Transit Engine und muss zunächst im Vault aktiviert werden. Da Vault in diesem Fall extern bereitsteht, wird auf die Konfiguration von Vault in diesem Kontext nicht eingegangen. Die Vault Transit Engine ist also ein Key Management Service, der von Kubernetes jedoch nicht einfach so aufgerufen werden kann. Denn hierfür wird ein weiteres Plugin benötigt, dass die Kommunikation mit der Vault Transit Engine und dem Kubernetes API-Server übernimmt. [hasb, tro]

**Minikube:** In Minikube wird zur Integration eines externen KMS-Systems das Plugin Trousseau.io verwendet, welches mit der Vault Transit Engine kommuniziert. [hasb, tro]

Die Installation und Konfiguration von HashiCorp Vault und der Vault Transit Engine sind nicht Bestandteil dieser Arbeit. Um das Plugin im Cluster bereitzustellen, wird zunächst ein Service Account mit `kubectl -n kube-system create serviceaccount trousseau-vault-auth` angelegt. Diesem Service Account wird anschließend ein Token zugeordnet, das in Listing A.6 dargestellt ist. Damit dieser Service Account als Delegierter die Secrets an die Transit Engine weiter leiten darf, wird ihm eine RBAC-Policy in Form eines ClusterRoleBinding zugeordnet. Der Service Account bekommt die Cluster-Rolle system:auth-delegator. Das ClusterRoleBinding ist in Listing A.7 dargestellt. Anschließend benötigt das Cluster noch eine Config map, die die Configuration für den Vault Agenten festlegt. Die ConfigMap, die in Listing A.8 dargestellt ist, definiert Authentifizierungs-Methoden und -Rollen und legt die Transit-Adressen, sowie den Provider fest. Um die Verschlüsselung über Trousseau zu ermöglichen, muss das KMS-Plugin zunächst in Kubernetes aktiviert werden. Hierfür wird eine EncryptionKonfiguration angelegt, dessen Provider auf das KMS Management des Trousseau Plugins verweist. Die YAML-Konfiguration hierzu ist in Listing A.9 dargestellt. Anschließend muss diese Konfiguration dem API-Server über den Parameter `--encryption-provider-config` mitgegeben werden. Die Konfiguration muss über einen Neustart des Clusters erfolgen. Die Konfiguration wird entsprechend in `/etc/ssl/certs/trousseau-vault-encryptionconfiguration.yml` angelegt. Minikube kann derzeit dem Apiserver keine Mounts mitgeben, weswegen der Dateipfad /etc/ssl/certs genutzt wird, da dieser bereits in den kubeapiserver gemountet wird. Über ein Daemonset, dass das KMS-Plugin im Cluster bereitstellt, wird die Verbindung zur Transit Engine ermöglicht. Der Neustart erfolgt dann mit dem Parameter: `--extra-config=apiserver.encryption-provider-config=/etc/ssl/certs/trousseau-vault-encryptionconf.yml`. Hierdurch wird die EncryptionConfiguration geladen und die Secret-Verschlüsselung über das KMS-Plugin durchgeführt. Der Neustart durch `minikube stop` und `minikube start ... --extra-config=apiserver.encryption-provider-config=/etc/ssl/certs/trousseau-kms-enc-conf.yaml` sollte eigentlich dazu dazuführen, dass das KMS-Provider aktiviert wird. Allerdings startet das Cluster dann nicht mehr erfolgreich. Eine weitere Möglichkeit, die Konfiguration des Apiservers im laufenden Betrieb zu editieren, führt ebenfalls zu einer nicht bereitstehenden Control Plane. Der Befehl `kubectl get pods --all-namespaces` resultiert in *The connec-*

*tion to the server 192.168.49.2:8443 was refused - did you specify the right host or port?.* [hasb, tro]

Minikube unterstützt somit zurzeit nicht die externe Secret-Verschlüsselung.

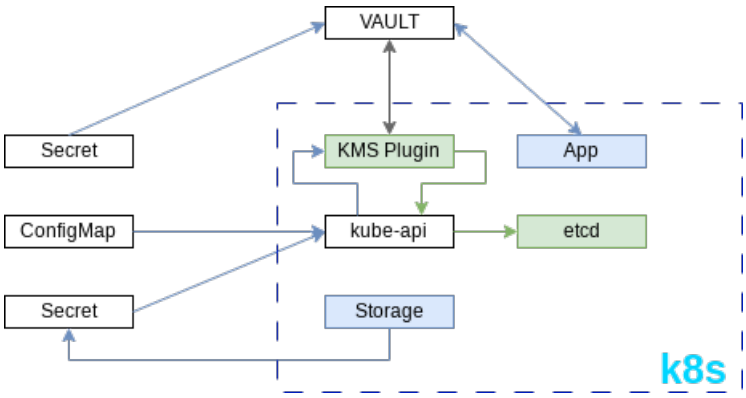



Abbildung 5.3: Darstellung der Kommunikation des Clusters mit KMS-Plugin zur Kommunikation mit dem Vault und der Encryption at REST. Das Secret kann direkt über den Vault der App zur Verfügung gestellt werden, neben dem besteht aber auch die Möglichkeit eine ConfigMap oder ein Secret verschlüsselt über Encryption at REST im etcd zu speichern, dazu wird das KMS-Plugin Trousseau (Minikube) oder ein externes KMS-Plugin des Cloud-Providers (EKS) genutzt. Dieses kommuniziert mit dem KMS (Vault für Minikube) und gibt die Antwort an den kube-api-Server zurück, der es dann im etcd speichert. [hasb, tro]

**EKS:** Innerhalb des Amazon EKS Cluster ist die Funktionsweise eines externen Key-Management-Systems ähnlich zu dem vorgestellten Ansatz im Minikube Cluster. Erstellt der Nutzer ein Secret, wird dafür im kube-apiserver ein DEK (Data Encryption Key) erstellt, der anschließend im externen Key Management System verschlüsselt, zurück an den API-Server geschickt wird und dann im etcd gespeichert wird.
Im eingesetzten EKS Cluster wurde bereits die Verschlüsselung der Secrets über ein KMS aktiviert. Dazu wird über die AWS Management Console die Verschlüsselung der Secrets aktiviert und ein KMS-Schlüssel hinzugefügt. Anschließend werden alle Secrets neu rotiert und verschlüsselt im etcd abgelegt. [amac]

#### Externe Secret-Verwaltung für Anwendungen mit Umgebungsvariable

In Kubernetes-Clustern müssen nicht alle Daten der Pods und Anwendungen im etcd gespeichert werden. Insbesondere für Anwendungen, die auf geteilte Secrets zugreifen sollen, kann die Nutzung einer externen Secret-Verwaltung sinnvoll sein. Externe Secret-Manager wie HashiCorp Vault bieten eine zentrale Lösung zur Speicherung und Verwaltung sensibler Informationen und bieten zusätzliche Funktionen wie Zugriffskontrollen und die automatische Rotation von Secrets. In Abbildung 5.3 wird dargestellt, wie eine Anwendung

auf ein externes Secret in einem Vault zugreifen kann, ohne die Kubernetes-API zu nutzen. Diese Methode erhöht die Sicherheit, da die Verwaltung der Secrets vollständig außerhalb des Clusters erfolgt. [hasb, tro]

**Minikube:** Um HashiCorp Vault mit Minikube zu verwenden, müssen einige vorbereitende Schritte unternommen werden, um eine reibungslose Integration zu gewährleisten. Hier wird davon ausgegangen, dass ein Vault-Server lokal oder auf einem Host bereitsteht, der für das Minikube-Cluster zugänglich ist, und dass ein Secret mit dem Pfad *secret/webapp/config* und den Werten *username=masterarbeit* und *password=ZTA* im Vault hinterlegt wurde. Um eine Verbindung zum Vault herzustellen, muss die IP-Adresse des Minikube-Clusters ermittelt werden. Dies kann durch den Befehl `minikube ip` erfolgen, der die IP-Adresse des Cluster-Nodes ausgibt. Beispiel: *192.168.49.2*, wie in Listing A.10 dargestellt. Anschließend wird die Adresse des Vault-Servers als Umgebungsvariable gesetzt, damit sie in den Pods verwendet werden kann. Diese kann in der Pod-Konfiguration verwendet werden, um sicherzustellen, dass die Anwendung die korrekte Adresse für den Vault-Server verwendet. Die Verwendung ist in Listing A.11 dargestellt. Die Kommunikation mit dem Vault kann anschließend über die Kommandozeile durch einen curl-Aufruf, aus dem Pod heraus, durchgeführt werden. Der Befehl hierfür lautet: `curl -vik -H "X-Vault -Token: $VAULT\_TOKEN" $VAULT_ADDR/v1/secret/data/devwebapp/config` Alternativ kann mit `kubectl exec devwebapp -- curl -s localhost:8080 ; echo` auch direkt auf den Pod devwebapp zugegriffen und auf die Anwendung (Port 8080) zugegriffen werden. Dieser Befehl greift auf den Pod devwebapp zu und ruft die auf Port 8080 laufende Anwendung auf. [hasb]

**EKS:** Für Amazon EKS (Elastic Kubernetes Service) erfolgt die Integration von HashiCorp Vault zur externen Secret-Verwaltung durch mehrere spezifische Schritte, die sich von einer lokalen Minikube-Installation unterscheiden. Zunächst sollte sichergestellt sein, dass der HashiCorp Vault korrekt konfiguriert ist und vom EKS-Cluster aus zugänglich ist. Die Vault-Instanz sollte sicher in einer Amazon Virtual Private Cloud (VPC) bereitgestellt sein und nur Verbindungen vom EKS-Cluster zulassen. Im Rahmen dieser Arbeit wird jedoch der Fall betrachtet, dass der Server extern außerhalb von Amazon EKS installiert wurde. Entsprechend ist als Umgebungsvariable einfach nur die Domain des Clusters zu setzen. Der Zugriff erfolgt dann wie zuvor gezeigt über curl oder die devwebapp. [hasb]

#### Externe Secret-Verwaltung für Anwendungen mit Service und Endpunkt

Ändert sich die Adresse des Vault regelmäßig oder wird der Vault für mehr als eine Applikation genutzt, kann ein Service und ein Endpunkt in das Cluster integriert werden. Dies ermöglicht die Aktualisierung des Endpunkts und nicht jeden Pods, der auf die Variable zugreift. [hasb]

**Minikube:** Innerhalb von Minikube werden Service und Endpunkt integriert. Diese sind in Listing A.12 dargestellt. Ein Service ist eine Methode, mit der eine Netzwerk-Anwendungen innerhalb des Clusters freigegeben werden kann. In diesem Fall wird der Endpunkt auf Port 8200 festgelegt werden. Der Endpunkt definiert sozusagen den Endpunkt der Cluster-internen Kommunikation und leitet den Request anschließend an die ip:port weiter. Anschließend kann die Pod-Konfiguration aus Listing A.11 angepasst werden, dass die Umgebungsvariable VAULT_ADDR auf die URL "http://external-vault:8200" verweist. Durch den im Cluster vorliegenden DNS-Server kann external-vault zum Service aufgelöst werden, der Anfragen auf Port 8200 an den Endpunkt weiterleitet, der dann den Aufruf am Vault durchführt. [hasb]

**EKS:** Innerhalb des Amazon EKS (Elastic Kubernetes Service) funktioniert die Integration ähnlich, aber es gibt einige zusätzliche Überlegungen und Optimierungen: In EKS wird ebenfalls der Kubernetes Service und ein Endpunkt verwendet, um den Vault-Server zu repräsentieren. Der Endpunkt kann auf eine externe IP-Adresse oder einen Load Balancer zeigen, je nachdem, wie der Vault-Server bereitgestellt wird. Dabei ist die Konfiguration der beiden Services ähnlich zu der in Listing A.12. Wenn der Vault-Server außerhalb des Clusters läuft, müssen die entsprechenden Sicherheitsgruppen so konfiguriert werden, dass der Zugriff von den EKS-Worker-Nodes auf den Vault-Port erlaubt ist. Innerhalb des Pods kann ähnlich wie bei Minikube die Pod-Konfiguration angepasst werden, um die Umgebungsvariable `VAULT_ADDR` auf die URL des Kubernetes Services zu setzen. [hasb]

#### Externe Secret-Verwaltung für Anwendungen mit Sidecar

Eine weitere Methode zur Secret-Verwaltung ist die Verwendung von HashiCorp Vault, welches durch Sidecar-Container in Anwendungen integriert werden kann. Diese Integration ermöglicht es, Secrets dynamisch zu verwalten und direkt in die Pods zu injizieren. Die Integration variiert je nach Kubernetes-Umgebung. Im Folgenden wird die Implementierung sowohl in einer lokalen Minikube-Umgebung als auch in einem verwalteten Amazon EKS-Cluster beschrieben. [hasb]

**Minikube:** Um dies zu ermöglichen, muss der Kubernetes Helm Chart `vault` von HashiCorp im Cluster installiert werden. Dieser Chart fügt den Pod `vault-agent-injector` und den ServiceAccount `vault` zum Cluster hinzu. Damit der Injektor weiß, dass es sich um eine externe Vault-Instanz handelt, wird ihm mithilfe des folgenden Parameters die Adresse dieser Instanz übergeben: `--set "global.externalVaultAddr=http://external-vault:8200"`. Der neue Service Account benötigt ein Token, das mithilfe der in Listing A.13 dargestellten Konfiguration gesetzt wird. Zusätzlich müssen einige Konfigurationen innerhalb des Vaults vorgenommen werden, die hier nicht weiter behandelt werden, aber essenziell für die Authentifizierung mit Kubernetes sind und durchgeführt werden müssen. Anschließend werden drei Annotationen auf dem Pod gesetzt, die in Listing A.14 dargestellt sind. Diese

Annotationen haben folgende Funktionen: *agent-inject* aktiviert die Bereitstellung eines Vault-Sidecar-Containers, *role: Webapp* ist die Vault Kubernetes Authentifizierungs-Rolle, und *agent-inject-secret-credentials.txt* definiert, aus welchem Pfad des Vaults die Daten bezogen und in welche Datei des Pods sie injiziert werden sollen. In diesem Fall liegen die Secrets aus *secret/data/webapp/config* in der Textdatei *credentials.txt* des Pods. Um die Secrets auszulesen, reicht im Container ein Befehl wie `cat /vault/secrets/credentials.txt` aus. [hasb]

Die Bereitstellung des Vault-Agenten im Pod funktioniert nicht, wenn ein Istio-Sidecar-Container ebenfalls in den Pod integriert ist. Laut [gitc] liegt dies an der Initialisierung des Istio-Sidecars, das die Firewall-Regeln (iptables) der Anwendung festlegt, bevor der Vault aufgerufen werden kann. Dadurch kann der Sidecar-Container des Vault-Agenten nicht auf Port 8200 zugreifen, was eine Verbindung zum Vault verhindert. Um dieses Problem zu beheben, muss dem Pod eine zusätzliche Annotation hinzugefügt werden, die in Listing A.15 dargestellt ist. [gitc]
Die Annotation *agent-init-first* sorgt dafür, dass der Sidecar-Container des Vaults *vault-agent-init* in der Container-Reihenfolge als erster gestartet wird. Wenn Vault innerhalb des Clusters läuft und nicht Teil des Service Mesh ist, könnte Istio angewiesen werden, den ausgehenden Verkehr auf Port 8200 zu ignorieren. [hasb]

**EKS:** In Amazon EKS (Elastic Kubernetes Service) können Secrets für Anwendungen extern über den AWS Secrets Manager verwaltet werden. Um diese Integration zu ermöglichen, wird der Kubernetes Operator `external-secrets` im Cluster installiert. Dieser Operator synchronisiert automatisch die Secrets aus dem AWS Secrets Manager in die Kubernetes Secrets, die dann von den Anwendungen genutzt werden können. Dazu müssen zunächst im AWS Secrets Manager die Secrets erstellt werden. Der Operator `external-secrets` kann im Cluster über helm installiert werden. Nachdem der Operator installiert ist, müssen die gewünschten Secrets im Cluster erstellt werden. Hierfür wird eine spezielle Ressource vom Typ *ExternalSecret* angelegt, die angibt, welche Secrets aus dem AWS Secrets Manager synchronisiert werden sollen. Listing A.16 zeigt eine Beispielkonfiguration. In dieser Ressourcen-Konfiguration wird festgelegt, dass auf den Secrets-Manager zugegriffen werden und die Schlüssel mit Namen username und password der Secret-gruppe mit Namen /secret/webapp/config in Kubernetes bereitgestellt werden sollen. Anschließend können die Secrets per Umgebungsvariablen in den Pods oder über als Dateien in Volumes eingebunden werden. [hasb]

#### Externe Zertifikat-Verwaltung

In Kubernetes-Clustern ist die Verwaltung von Zertifikaten von zentraler Bedeutung, um die Sicherheit und Integrität der Kommunikation zwischen verschiedenen Komponenten und Services zu gewährleisten. Zertifikate werden verwendet, um die Authentizität von Diensten

zu bestätigen und eine verschlüsselte Kommunikation zu ermöglichen, was die Vertraulichkeit der Daten sicherstellt. Eine externe Zertifikatsverwaltung bietet eine zentralisierte Lösung für die Erstellung, Erneuerung und Verwaltung von Zertifikaten. Diese Methode erhöht die Sicherheit und reduziert das Risiko von Fehlkonfigurationen, die zu Sicherheitslücken führen könnten. Durch die Verwendung eines externen Systems wie HashiCorp Vault können Unternehmen von zusätzlichen Funktionen wie der automatischen Rotation von Zertifikaten und der zentralen Kontrolle über die Zugriffsrechte auf Zertifikate profitieren. Dies ist besonders in dynamischen Umgebungen wie Kubernetes-Clustern wichtig, in denen sich die Konfiguration und die Anzahl der Dienste häufig ändern. [hasa]

**Minikube:** In diesem Abschnitt wird gezeigt, wie eine externe Zertifikatsverwaltung für ein Kubernetes-Cluster in Minikube mithilfe von HashiCorp Vault implementiert werden kann. Dabei wird davon ausgegangen, dass eine externe Verbindung zu Vault, wie im vorherigen Kapitel beschrieben, bereits besteht. Zusätzlich wird angenommen, dass die Public Key Infrastructure (PKI) bereits in Vault konfiguriert ist, eine maximale Time-to-live (TTL) für die Secrets festgelegt wurde und ein selbst signiertes Zertifikat mit dieser TTL sowie entsprechende Rollen und Richtlinien (Policies) im Vault erstellt wurden. [hasa] Für Kubernetes wird der cert-manager installiert, ein Add-on, das die Verwaltung und Erstellung von TLS-Zertifikaten automatisiert [cer]. Zunächst wird ein Namespace namens cert-manager erstellt. Anschließend wird das Repository in Helm hinzugefügt und der cert-manager installiert. [hasa] Um ein Zertifikat zu erstellen, wird zunächst ein Service Account mit dem Namen issuer angelegt, dem anschließend ein Secret vom Typ ServiceAccountToken zugewiesen wird. Diese Secret-Konfiguration ist in Listing A.17 dargestellt. Mithilfe des Secret-Namens kann eine Issuer-Konfiguration erstellt werden. Diese Konfiguration, die in Listing A.18 dargestellt ist, enthält unter anderem die Konfiguration für den Vault-Server, der unter der URL http://external-vault:8200 erreichbar ist, sowie die Definition, mit welchem Secret die Authentifizierung gegenüber Vault durchgeführt werden soll. Anschließend kann mithilfe dieses Tokens eine Konfiguration erstellt werden [hasa]. Der Server verwendet die EXTERNAL_VAULT_ADDR aus dem vorherigen Kapitel. Der Pfad definiert den internen Vault-Pfad, in dem die Zertifikate gespeichert werden sollen. Die auth-Konfiguration dient zur Authentifizierung mit der Rolle issuer und nutzt das zuvor erstellte Secret sowie den Vault-Authentifizierungsendpunkt mit mountPath [hasa]. Nun kann ein Zertifikat mit der Definition aus Listing A.19 erstellt werden. Das Zertifikat mit dem Namen *example-com* definiert, dass der zuvor erstellte Issuer `vault-issuer` verwendet wird. Der commonName und der DNS-Name müssen innerhalb der erlaubten Domains liegen, die für den Vault-Endpunkt konfiguriert wurden [hasa]. Um zu überprüfen, ob das Zertifikat korrekt ausgestellt wurde, kann der Befehl `kubectl describe certificat.cert-manager example-com` verwendet werden. Wenn die Ausstellung erfolgreich war, zeigt die letzte Nachricht im Eventlog den Hinweis *The certificate has been successfully issued* an. [hasa]

**EKS:** Ähnlich zum Minikube Cluster kann mithilfe des cert-manager die ACM Private Certificate Authority (CA) mit dem Amazon EKS Cluster verbunden werden. Somit wird eine Möglichkeit zur sicheren Zertifizierungstellenlösung für Kubernetes-Container geschaffen. [hasa]

### Audit Logging

Kubernetes unterstützt cluster-basiertes Logging, bei dem Container-Aktivitäten zentral erfasst werden können. Die Standard- und Fehlerausgaben jedes Containers werden mithilfe eines Fluentd-Agenten auf jedem Knoten protokolliert und können in Kibana angezeigt werden. Zusätzlich bietet Kubernetes einen Audit Logger, der Aktionen, die über die API ausgeführt werden, zur späteren Analyse aufzeichnet. Diese Audit-Protokollierung sollte aktiviert und die Audit-Dateien auf einem sicheren Server archiviert werden. [kubb]

Audit-Logging wird in Kubernetes durch Audit Policies geregelt, die festlegen, in welchem Detailgrad und zu welchem Zeitpunkt Anfragen protokolliert werden. Neben dem API-Server-Logging können auch einzelne Container über stdout und stderr geloggt werden, was allerdings weniger sicherheitsrelevant ist und daher nicht weiter behandelt wird. [kubb]

**Minikube:** Die Audit Policy kann dem Minikube Cluster durch den Parameter `--extra-config=apiserver.audit-policy-file=/etc/ssl/certs/audit-policy.yaml` und `--extra-config =apiserver.audit-log-path=-` mitgegeben beim Start des Clusters im API-Server mitgegeben werden. Als Policy-Speicherort wird in diesem Fall /etc/ssl/certs genutzt. Dieser ist bereits als Volume-Mount und der Pod-Volumes im kube-apiserver unter minikube integriert und muss somit nicht extra angelegt werden. Dies erspart das Editieren der Volumes nach Start des Containers. Dies ist auch nicht möglich, da Änderungen nur für bestimmte Felder der kube-apiserver Konfiguration erlaubt sind. Da es keinen dedizierten Ordner in ~/.minikube/ gibt, ist das genutzte Verzeichnis nur ein Workaround, bei dem die Datei vom Host in den Container durch den bereits vorhanden Volume-Mount kopiert wird. [kubb]

**EKS:** Auch innerhalb von Amazon EKS spielt die Sammlung und Analyse der Audit Logs eine große Rolle. In EKS werden die Audit-Logs direkt an den AWS-Service CloudWatch gesendet. Hierfür muss auf der Control Plane ein export zu CloudWatch erlaubt werden. Da der Service CloudWatch zusätzliche monetäre Ressourcen kostet, wird an dieser Stelle darauf verzichtet. [amab]

#### Monitoring Network Traffic

Containerisierte Anwendungen nutzen typischerweise meist exzessiv das Cluster-Networking, weshalb Monitoring des aktiven Netzwerkverkehrs eine gute Möglichkeit ist, um zu verstehen, wie Anwendungen miteinander interagieren und unerwartete Kommunikation zu erkennen. Somit lässt sich auch feststellen, ob die Kubernetes-Netzwerk-Richtlinien die notwendigen Anfragen abdecken. [owa]
Das Monitoring und im Besonderen das Monitoring des Netzwerkverkehrs wird in Kapitel 5.4 im Rahmen der Implementation der Zero-Trust-Architektur im Detail vorgestellt.

#### Rotation von Infrastruktur-Credentails

Je kürzer die Lebensdauer eines Geheimnisses oder einer Berechtigung sind, desto schwieriger ist es für einen Angreifer diese zu nutzen. Aus diesem Grund sollten kurze Laufzeiten für Zertifikate festgelegt und die Rotation automatisiert werden. Durch Verwendung eines Authentifizierungsanbieters können die Länge der ausgestellten Token kontrolliert werden. Token für Dienstkonten in externen Integrationen sollten entsprechend häufig gewechselt werden. [owa]

### 5.2.2 Containerbasierte Best Practices

Auch die Container sind, wie in Kapitel 4 beschrieben, von Angriffsvektoren und Schwachstellen betroffen und stellen somit ein erhebliches Sicherheitsrisiko für das Cluster dar. Aus diesem Grund gibt es beim Build eines Container-Images, sowie während des Deploys einige Securitys Best Practices, die im Folgenden kurz erläutert werden. Hierzu zählen auch grundlegende Sicherheitsmaßnahmen, wie die Prüfung der Aktualität der Container Images. Die Best Practices für den Build der Container-Images gliedern sich in die Punkte: 'autorisierte Images, Funktionen in Images reduzieren, Kontrolle und Identifizierung von Schwachstellen. Während sich die Security Best Practices für den Betrieb der Container in die folgenden Bereiche unterteilt: ImagePolicyWebhook, Container Schwachstellen-Scans, Least-Privileg Prinzip und Container Runtime Security.
Im Rahmen der Implementierung wird nur auf die Security Best Practices eingegangen, die sich auf den Betrieb der Container beziehen. Dennoch sind die Security Best Practices beim Build ebenso essenziell für die Sicherheit des Clusters, weswegen diese an dieser Stelle vorgestellt werden. [owa]

#### Funktionen in Images reduzieren

Es wird empfohlen, den Code innerhalb der Laufzeit-Container einzuschränken. Dieser Ansatz verbessert das Signal-Rausch-Verhältnis von Scannern, wie CVE und reduziert den Aufwand für die Ermittlung der Bestandteile auf das, was für die Aufgabe benötigt wird. Es sollte in Erwägung gezogen werden, minimale Container Images wie Distributionslose Images (zum Beispiel distroless) zu nutzen. Sollte dies nicht möglich sein, sollten keine Betriebssystem-Paketmanager oder Shell in das Image aufgenommen werden, da diese unbekannte Schwachstellen haben könnten. [owa]

**Minikube & EKS:** Wie eingangs erwähnt wird auf die Image-Erstellung (Build) nicht eingegangen. Da dies nichts direkt mit dem Cluster und dessen Zero-Trust-Architektur zu tun hat.

#### Kontrolle und Identifizierung von Schwachstellen

Kubernetes verwendet Container-Registries, die von externen Anbietern bereitgestellt werden, um Container-Images zu speichern und bereitzustellen. Je nach Bedarf können ein öffentliches oder ein privates Repository für die Container-Registry eingesetzt werden.
Best Practice ist es, eine private Registry zu nutzen, um nur genehmigte Images zu speichern. Dadurch wird die Anzahl der potenziellen Images, die in die Pipeline gelangen, automatisch reduziert. Aus den hunderttausenden öffentlichen Container Images wird so eine begrenzte Anzahl. Die Einrichtung einer CI-Pipeline, die eine Sicherheitsbewertung (Schwachstellen-Scans etc.) in den Build-Prozess integriert, wird ebenfalls empfohlen. Nur wenn keine Schwachstellen, und somit eine positive Sicherheitsbewertung vorliegt, sollte das Image in eine private Registry übertragen werden. [owa]

#### ImagePolicyWebhook

Während der Bereitstellung der Container im Cluster wird empfohlen, den Admission Controller ImagePolicyWebook zu nutzen. Dieser verhindert, dass nicht genehmigte Images genutzt werden und Pods, die solche Images nutzen, beendet werden. Ebenso werden Container Images verboten, die nicht vor kurzem gescannt wurden, die ein nicht explizit erlaubtes Base-Image oder ein Image, dass von einer unsicheren Registry bezogen wird, nutzen.
Im Rahmen dieser Arbeit wird lediglich die Istio-Beispielanwendung Bookinfo im Cluster deployt, weswegen eine ImagePolicyWebhook im Rahmen dieser Arbeit nicht betrachtet werden muss. Trotzdem wird empfohlen, die deploybaren Container-Images mithilfe der ImagePolicyWebhook zu begrenzen. [kuba]

### Container Schwachstellen-Scans

Es wird empfohlen, Container des Clusters regelmäßig auf Schwachstellen zu scannen. Dies schließt sowohl die First-Party Container, also die, die vor kurzem erstellt und gescannt wurden, als auch Third-Party Container ein. OWASP empfiehlt das Open-Source-Projekt ThreatMapper. ThreatMapper ist ein Open-Source-Tool, das von Deepfence entwickelt wurde und darauf abzielt, laufende Container und Microservices-Umgebungen auf Sicherheitsbedrohungen zu überwachen und zu schützen. Es bietet Funktionen wie Bedrohungsmodellierung, Erkennung von Schwachstellen in Echtzeit, Laufzeit-Schutz und Berichterstellung. [owa] ThreatMapper unterstützt die Erkennung von Bedrohungen und Sicherheitsverletzungen in Echtzeit und kann entsprechend darauf reagieren. Ebenso erkennt und verwaltet ThreatMapper Schwachstellen in Container-Images und laufenden Containern. ThreatMapper lässt sich automatisieren, um Sicherheitsrichtlinien und Reaktionen auf Bedrohungen automatisiert auszuführen und minimieren so den manuellen Aufwand und verbessern die Sicherheit insgesamt. [dee]

**Minikube & EKS:** Um die Container im Cluster regelmäßig auf Schwachstellen zu scannen, wird das Open Source-Projekt ThreatMapper im Cluster eingesetzt. ThreatMapper kann einfach über helm installiert werden. Zu den Voraussetzungen dazu gehört ein persistent Volume. Nach der Installation werden gut 13 Pods gestartet, die von Datenbanken (Postgres und Redis) bis hin zu Kafka und weiteren Schedulern, UIs und Workern reichen. Anschließend übernimmt ThreatMapp die Überwachung der Images und des Clusters und stellt über eine übersichtliche UI Schwachstellen, Secret-Exploits, Malware und weitere Angriffsvektoren dar. Runtime Erkennung ist jedoch mit ThreatMapper nicht mehr möglich, hierfür muss ein zusätzliches Add-on ThreatStryker erworben werden. Dennoch sollte über den Einsatz von ThreatMapper durchaus positiv befunden werden, auch wenn dieser im Cluster viele Ressourcen vereinnahmt. [dee]

### Least-Privilege

Es wird empfohlen, das Prinzip der minimalen Rechtevergabe (Least-Privilege) für Container anzuwenden, da die Risiken durch die Fähigkeiten, Rollenbindungen und Richtlinien des Containers beeinflusst werden. Jeder Container sollte nur die minimalen Privilegien und Fähigkeiten besitzen, die erforderlich sind, um die gewünschte Funktion bereitzustellen. [owa]
Pod Security Policies können diese sicherheitsrelevanten Attribute auf Pod-Ebene steuern. Die folgenden Bestimmungen sollten umgesetzt und als Sicherheitskontext für Pods und Container festgelegt werden:

- Anwendungsprozesse werden nicht als Root-Nutzer ausgeführt. [owa]

Tabelle 5.1: Auswertung der Pods, Nutzer-, Gruppen-Ids und Gruppen vor und nach dem Hinzufügen der Least Privilege Prinzipien

| Pod/App | Nutzer (vorher) | Nutzer (nachher) |
|---|---|---|
| Productpage | uid=1000, gid=0(root), groups=0(root) | uid=1000, gid=1000, groups=1000 |
| Ratings | uid=1000(node), gid=1000(node), groups=1000(node) | uid=1000(node), gid=1000(node), groups=1000(node) |
| Reviews (v1) | uid=1001(default), gid=0(root), groups=0(root) | ERROR |
| Reviews (v2) | uid=1001(default), gid=0(root), groups=0(root) | ERROR |
| Reviews (v3) | uid=1001(default), gid=0(root), groups=0(root) | ERROR |
| Details | uid=1000, gid=0(root), groups=0(root) | uid=1000, gid=1000, groups=1000 |

- Privilege Escalation ist nicht erlaubt. [owa]
- Das Root-Dateisystem ist read-only. [owa]
- Als Standard-Dateisystem wird /proc verwendet. [owa]
- Das Host-Netzwerk oder der Prozess-Bereich wird nicht genutzt. [owa]
- Nicht genutzte und nicht benötigte Linux-Fähigkeiten werden eliminiert. [owa]
- SELinux wird zur feingranularen Prozesskontrolle eingesetzt. [owa]
- Jede Anwendung erhält ihren eigenen Kubernetes Service Account. [owa]
- Container, die keinen Zugriff auf die Kubernetes API benötigen, sollten keinen Mount der Service Account Zugangsdaten erhalten. [owa]

**Minikube:** In Minikube wird das Least-Privilege-Prinzip angewendet, indem Container mit einem Minimum an Privilegien und Fähigkeiten ausgestattet werden. Es ist jedoch wichtig zu beachten, dass die Container der Bookinfo-Anwendung von Istio erstellt und verwaltet werden und im Rahmen dieser Arbeit nicht ausreichend analysiert werden konnten, um die vollständige Durchsetzung des Least-Privilege-Prinzips sicherzustellen. Für die bereitgestellten Pods der Komponenten *productpage*, *reviews*, *details* und *ratings* wurden die Nutzer-ID, Gruppen-ID und Dateisystem-ID ermittelt. Die Ergebnisse dieser Überprüfung sind in Tabelle 5.1 dargestellt. [kube]

Die deployten Pods verwenden alle, mit Ausnahme der *Reviews*, die Nutzer-ID 1000. Die *reviews*-Pods verwenden die Nutzer-ID 1001, die dem Standardwert entspricht. Dies bedeutet, dass die Berechtigungen der Pods bereits eingeschränkt sind, ohne dass eine spezielle Konfiguration für die Pods festgelegt wurde. Bei der Definition der Nutzer-ID gibt es jedoch eine Einschränkung durch die Nutzung des Service Meshes, da die UID 1337 für den Sidecar-Container, der IP-Tables bereitstellt, reserviert ist. Daher sollte kein *securityContext* eine UID von 1337 festlegen, um Konflikte mit dem Istio-Container zu vermeiden. Um die Sicherheit weiter zu erhöhen, sollten für die Pods der *productpage, reviews (v1-3)* und *details* zusätzliche Sicherheitskontexte hinzugefügt werden. Dies beinhaltet die Konfiguration von

*runAsUser: 1000*, *runAsGroup: 1000* und *fsGroup: 1000*. Weiterhin sollte der *securityContext* des Containers die in Listing A.20 aufgeführten weiteren Einschränkungen enthalten. Dies umfasst die Entfernung aller Fähigkeiten auf Container-Ebene, das Verbot von Privilege Escalation und das Setzen des Root-Dateisystems auf read-only. Nach der Aktualisierung der Konfiguration mithilfe von kubectl können die Ergebnisse ebenfalls in Tabelle 5.1 betrachtet werden. Abgesehen von den Pods und Containern der Review-Anwendung haben sich die Berechtigungen der Productpage- und Details-Anwendungen geändert. Die Review-Container können jedoch nicht starten, da diese offenbar zwingend Root-Zugriff auf das Dateisystem benötigen, was zum Status `CrashLoopBackOff` führt. Eine umfassendere Analyse des Containers wäre erforderlich, um dieses Problem vollständig zu lösen; dies geht jedoch über den Umfang dieser Arbeit hinaus. Daher wurden die entsprechenden Sicherheitskonfigurationen für die Review-Pods wieder rückgängig gemacht, sodass die Pods wieder verfügbar sind. *SELinux* kann verwendet werden, um den Dateizugriff von Containern einzuschränken. Auf dem vorliegenden Host wurde *SELinux* aktiviert, jedoch konnte keine Funktionsfähigkeit festgestellt werden, da *SELinux* weiterhin deaktiviert zu sein scheint. *AppArmor* wird derzeit von Minikube nicht unterstützt, sodass eine Implementierung in diesem Fall nicht möglich ist. [gitb]

**EKS:** Auch in Amazon Elastic Kubernetes Service (EKS) kann das Least-Privilege-Prinzip auf ähnliche Weise implementiert werden, um die Sicherheit der Container zu gewährleisten. In EKS wird empfohlen, Security Contexts zu verwenden, um die Berechtigungen der Pods und Container zu steuern. Für die Integration des Least-Privilege-Prinzips in EKS sollten die folgenden Schritte berücksichtigt werden: Somit sollten für jede Deployment- oder Pod-Konfiguratin ein securityContext festgelegt werden, um die Nutzer- und Gruppen-IDs (z.B. runAsUser und runAsGroup) sowie weitere sicherheitsrelevante Einstellungen (wie fsGroup und capabilities festzulegen. Seccomp, SELinux und Apparmor lassen sich im EKS-Cluster integrieren, allerdings ist die Festlegung der Profile aufgrund der verwalteten Eigenschaft eine Herausforderung.

Durch die Implementierung dieser Maßnahmen in EKS kann die Sicherheit der Container gemäß dem Least-Privilege-Prinzip gewährleistet werden, wodurch das Risiko von Sicherheitsverletzungen und unbefugtem Zugriff minimiert wird.

#### Container Runtime Security

Die Sicherheit von Containern zur Laufzeit ist ein wesentlicher Aspekt, um Angriffe und Anomalien schnell zu erkennen und darauf reagieren zu können, während die Container und Workloads weiterhin in einem funktionierenden Zustand bleiben. Typischerweise wird dies durch das Abfangen von Low-Level-Systemaufrufen (syscalls) und das Überwachen von Ereignissen ermöglicht, die auf eine Kompromittierung hinweisen. [fal, Noa] *Falco*, ein Open-Source-Tool von Sysdig, ist ein führendes Werkzeug zur Laufzeit-Sicherheitsüberwachung

von Containern. Es bietet eine erweiterte Sichtbarkeit in Container-Umgebungen, indem es ungewöhnliche Verhaltensmuster erkennt und darauf basierend Warnungen ausgibt. Falco unterstützt die Überwachung von verschiedenen Ereignisquellen und kann so flexibel auf unterschiedliche Sicherheitsanforderungen angepasst werden. [fal, Noa]

**Minikube:** In Minikube kann Falco, ein Open-Source-Tool zur Laufzeitüberwachung und Bedrohungserkennung, verwendet werden, um Container zur Laufzeit zu sichern. Falco ist in der Lage, Ereignisse von verschiedenen Quellen auszuwerten, darunter Syscalls und Kubernetes Audit Logs, was eine detaillierte Echtzeit-Überwachung ermöglicht. Falco kann einfach über Helm in das Minikube-Cluster installiert werden. Die Installation erfordert jedoch einige Voraussetzungen, darunter mindestens 4 CPUs, um effizient arbeiten zu können, insbesondere wenn mehrere Ereignisquellen überwacht werden sollen. Nach der Installation nutzt Falco standardmäßig Syscalls und Kubernetes-Audit-Logs als Quellen für die Echtzeitüberwachung. Die Überwachung in Falco ist sehr granular und kann so konfiguriert werden, dass sie auf unterschiedlichen Ebenen protokolliert:

- Änderungen an Pods werden auf der RequestResponse-Ebene protokolliert.
- Pod-Metadaten, wie Logs und Status, werden auf der Metadaten-Ebene erfasst.
- Bestimmte Anfragen, wie etwa an die ConfigMap Controller-Leader oder Watch-Requests vom System , sowie authentifizierte Anfragen an bestimmte Nicht-Ressourcen-URL-Pfade, werden nicht protokolliert.
- Änderungen an ConfigMaps und Secrets in anderen Namespaces werden auf unterschiedlichen Ebenen, entweder auf RequestResponse- oder Metadaten-Ebene, erfasst.
- Änderungen an Secrets in allen Namespaces werden auf Metadatenebene protokolliert.
- Alle anderen Ressourcen im Core und in Erweiterungen werden auf Anfrageebene protokolliert, während andere Ereignisse auf Metadatenebene protokolliert werden, außer RequestReceived-Events.

Für eine korrekte Integration muss auch eine Webhook-Konfiguration erstellt werden, damit der kube-apiserver die Audit-Logs an Falco senden kann. Nach der Konfiguration kann das Minikube-Cluster mit den angepassten Parametern, wie in Listing A.21 dargestellt, neu gestartet werden. Dies ermöglicht es Falco, die erforderlichen Informationen in Echtzeit zu sammeln und zu analysieren. Nach dem erfolgreichen Neustart sollte Falco deinstalliert und erneut installiert werden, um sicherzustellen, dass alle Konfigurationen korrekt angewendet wurden. Die Logs können dann über den Befehl `kubectl logs -l app.kubernetes.io/name =falco -f` eingesehen werden. [fal, Noa]

Falco überwacht die Ausführung ungewöhnlicher Prozesse oder Befehle innerhalb der Container. Zum Beispiel protokolliert Falco in einer Zero-Trust-Architektur eine kritische Ausführung einer binären Datei, die nicht Teil des Basis-Images ist. Diese Ereignisse werden durch das Falco-Syscall-Event geloggt und stammen beispielsweise aus dem Container Networking Interface (CNI) Calico. [fal, Noa]

Um zu testen, ob Falco die Audit-Logs gemäß der Webhook-Konfiguration protokolliert, kann ein einfaches Beispiel wie `kubectl create cm myconfigmap --from-literal=username =admin --from-literal=password=123456` verwendet werden. Dies sollte in den Protokollen eine entsprechende Fehlermeldung erzeugen, wie in Listing A.22 dargestellt. [fal, Noa]

**EKS:** Auch im Amazon Elastic Kubernetes Service (EKS) Cluster kann Falco zur Echtzeit-Erkennung von Angriffen und Anomalien eingesetzt werden. Die Installation von Falco erfolgt ebenfalls über Helm und umfasst die Bereitstellung von vier Pods, darunter zwei Falco-Sidekicks und zwei Falco-Pods. Diese Pods werden auf jeder Node des Clusters bereitgestellt, wobei pro Node ein Falco-Sidekick und zwei Falco-Container laufen. [fal, Noa]

Nach der Installation sammelt Falco Informationen, basierend auf der AuditPolicy von AWS. Um die Integration mit AWS-Diensten zu optimieren, kann ein CloudWatch-Plugin hinzugefügt werden. Dieses Plugin ermöglicht es, die gesammelten Logs und Ereignisse direkt an Amazon CloudWatch zu senden, um eine zentrale Überwachung und Alarmierung innerhalb der AWS-Umgebung zu ermöglichen. [fal, Noa]

Durch die Verwendung von Falco sowohl in Minikube als auch in EKS können Sicherheitsvorfälle und Anomalien in Echtzeit erkannt und verwaltet werden, was die Sicherheitslage der Containerumgebung erheblich verbessert. Die flexible Konfiguration und die umfangreichen Überwachungsmöglichkeiten machen Falco zu einem unverzichtbaren Werkzeug für die Container Runtime Security. [fal, Noa]

## 5.3 Zero-Trust-Regeln und Anforderungen

Die vorgestellten Security Best Practices bieten einen umfassenden Überblick über die Möglichkeiten zur Erhöhung der Sicherheit innerhalb des Clusters und dienen als Grundlage für eine Zero-Trust-Architektur. Bevor auf die technische Umsetzung einer Zero-Trust-Architektur in Kubernetes eingegangen wird, ist es wichtig, die relevanten Zero-Trust-Regeln und Anforderungen zu verstehen, die von der Cloud Native Computing Foundation (CNCF) und anderen Quellen definiert wurden.

Eine Zero-Trust-Architektur basiert auf folgenden sieben Regeln der CNCF, die für Kubernetes-Umgebungen empfohlen werden:

**Vermeide Komplexität** Es ist nicht erforderlich, neue Anwendungen zu entwickeln, um Zero-Trust in ein Cluster zu integrieren. Die notwendigen Tools sind als Open-Source-Kubernetes-Tools und allgemeine Lösungen für API-Management, Load Balancing und Anwendungssicherheit bereits vorhanden. Zero-Trust ist lediglich eine Weiterentwicklung bestehender Sicherheitsmaßnahmen, um aktuellen Herausforderungen wie verteilten Teams, API-Konnektivität und Cloud-basierten Anwendungstopologien zu begegnen. [Yac22]

**Minimaler Overhead für Entwickler und End-Nutzer** Zero-Trust sollte so integriert werden, dass radikale Änderungen der Workflows vermieden werden. Es sollte im Hintergrund laufen, ohne die tägliche Arbeit der Entwickler und Endnutzer zu beeinträchtigen. Zertifikat-Rotationen sollten transparent und Multifaktor-Prüfungen passiv durchgeführt werden, sodass Unterbrechungen minimiert und die Arbeitsabläufe nicht gestört werden. [Yac22]

**Umfassende Absicherung von Data und Control Plane** Zero-Trust sollte sowohl auf der Daten- als auch auf der Kontrollebene implementiert werden, um ausnutzbare Schnittstellen in der Vertrauenskette zu vermeiden. Die Control Plane schützt die Policies und verhindert unerlaubte Änderungen, während die Data Plane das Cluster vor Brute-Force-Angriffen und anderen Bedrohungen auf niedriger Ebene schützt. [Yac22]

**Ost-West- und Nord-Süd-Verkehr** Durch die Implementierung von Sicherheitsmaßnahmen sowohl für den internen (Ost-West) als auch den externen (Nord-Süd) Datenverkehr wird eine Perimetersicherheit geschaffen, die eine gründliche Überprüfung der Herkunft, Dauer und Art des Datenverkehrs ermöglicht. Ost-West-Verkehr bezieht sich auf den Service-zu-Service-Datenverkehr, der in modernen, API-gesteuerten und Microservices-basierten Anwendungen vorherrscht. [Yac22]

**Ingress Controller und Service Mesh** Diese Komponenten machen Kubernetes effizienter und sind wesentliche Bestandteile einer Zero-Trust-Architektur. Ingress-Controller verwalten den Nord-Süd-Verkehr und sind entscheidend für die Authentifizierung externer APIs, während Service Meshes die Interaktionen zwischen internen Microservices koordinieren und die automatische Zertifikatsrotation unterstützen. [Yac22]

**Integriere Ingress Controller und Service Mesh** Die Zusammenarbeit dieser Komponenten gewährleistet einen effektiven Zero-Trust-Ansatz und einen reibungslosen Betrieb der Anwendungen. Durch die Verwendung eines gemeinsamen Satzes von Verkehrsrichtlinien und die Einrichtung von If-Then-Ketten können Sicherheitsmaßnahmen flexibler gestaltet und die Kontrolle verbessert werden, insbesondere wenn Indicators of Compromise (IOCs) entdeckt werden. [Yac22]

**Zertifikat-Handhabung automatisieren** Die manuelle Zertifizierung von Services und Microservices ist nicht mehr praktikabel. Daher sollte dieser Prozess automatisiert werden, um eine nahezu kontinuierliche Überprüfung zu ermöglichen. Kubernetes bietet mehrere Möglichkeiten, die automatisierte Handhabung von Zertifikaten zu integrieren und so die Sicherheit und Effizienz zu erhöhen. [Yac22]

Zusammengefasst soll eine Zero-Trust-Architektur in Kubernetes minimalen zusätzlichen Aufwand für Entwickler und Teams verursachen. Sie sollte sowohl auf der Daten- als auch der Kontrollebene implementiert werden und sich auf den Netzwerkverkehr sowohl für Nord-Süd- als auch Ost-West-Kommunikation erstrecken. Die Integration von Ingress-Controllern und Service-Meshes ist unerlässlich, und die Zertifikatsverwaltung sollte automatisiert erfolgen, um die Sicherheitsmaßnahmen zu optimieren.

Zusätzlich gibt es mehrere Netzwerkanforderungen, die für die Unterstützung einer Zero-Trust-Architektur erfüllt werden müssen. Diese Anforderungen stellen sicher, dass alle Ressourcen und Kommunikationswege entsprechend geschützt sind. Zu diesen Anforderungen gehören:

1. **Unternehmensressourcen haben grundlegende Netzwerkkonnektivität.** Das local area network (LAN) bietet grundlegendes Routing und Infrastruktur. Die remote Unternehmensressource muss nicht alle Infrastruktur-Services nutzen. [Sta20]
   In Kubernetes-Clustern wird grundlegende Netzwerkkonnektivität durch das Kubernetes-Netzwerkmodell bereitgestellt, das Pod-to-Pod-, Pod-to-Service- und Service-to-External-Verbindungen ermöglicht. Kubernetes verwendet verschiedene Netzwerklösungen wie Calico, Flannel, Cilium oder Weave Net, um Netzwerkkommunikation und Routing innerhalb des Clusters zu unterstützen. Diese Lösungen stellen sicher, dass Pods unabhängig von ihrem Standort im Cluster miteinander kommunizieren können, während sie gleichzeitig Netzwerkregeln und -richtlinien erzwingen, um den Zugriff zu steuern. [kubd]

2. **Unterscheidung zwischen Geräten in Besitz oder verwaltet und aktuelle Sicherheitslage der Geräte** Dies geschieht durch vom Unternehmen ausgestellten Credentials und nicht durch Nutzung von Informationen, die nicht authentifiziert werden können. [Sta20]
   In Kubernetes kann diese Anforderung durch die Verwendung von Zertifikaten und Identitätsmanagementdiensten erfüllt werden. Kubernetes nutzt X.509-Zertifikate für die Authentifizierung von API-Server-Komponenten und Benutzern. Managed Kubernetes-Dienste wie Amazon EKS oder Azure AKS bieten zusätzlich Identitätsmanagement-Integrationen mit AWS IAM oder Azure Active Directory, die sicherstellen, dass nur autorisierte und authentifizierte Geräte und Benutzer auf den Cluster zugreifen können. [kubc]

3. **Beobachten des gesamten Netzwerkverkehrs.** Es werden Pakete auf der Datenebene aufgezeichnet, auch wenn es nicht möglich ist, alle Pakete auf der Anwendungsschicht zu inspizieren. Somit werden Metadaten über die Verbindungen herausgefiltert, um die Richtlinien dynamisch zu aktualisieren und die PE zu informieren, wenn es Zugriffsanfragen auswertet. [Sta20]
   Kubernetes bietet Tools wie Network Policy, um den Netzwerkverkehr zu kontrollieren und zu überwachen. Diese Richtlinien können dazu verwendet werden, den eingehenden und ausgehenden Verkehr zu beschränken und spezifische Verkehrsströme zu beobachten. Darüber hinaus können Netzwerk-Sniffing-Tools und Intrusion Detection Systeme (IDS) wie Falco oder Sysdig genutzt werden, um Netzwerkverkehr und -aktivitäten zu überwachen und aufzuzeichnen. [kubh]

4. **Ressourcen sind nicht ohne Zugriff auf ein PEP erreichbar.** Es werden keine willkürlichen eingehenden Verbindungen aus dem Internet akzeptiert. Benutzerdefinierte Verbindungen werden von Ressourcen nur akzeptiert, nachdem der Client authentifiziert und autorisiert wurde. Hierfür ist der PEP zuständig. Ohne Zugriffsanfrage auf einen PEP sind die Ressourcen nicht einmal auffindbar, was Angreifer daran hindert, Ziele durch Scannen und/oder starten von DoS-Angriffen zu identifizieren. Einige Netzinfrastrukturkomponenten, wie zum Beispiel DNS-Server müssen jedoch zugänglich bleiben. [Sta20]
   Kubernetes ermöglicht die Implementierung von Zero-Trust-Prinzipien durch die Verwendung von Network Policies und Service Meshes wie Istio, die den Datenverkehr zwischen Diensten steuern und absichern. Diese Werkzeuge erzwingen, dass alle Verbindungen authentifiziert und autorisiert werden müssen, bevor sie zugelassen werden, und verhindern damit unbefugten Zugriff. Die Ressource ist ohne die entsprechenden Berechtigungen weder erreichbar noch sichtbar. [kubh, cnc]

5. **Data und Control Plane sind logisch getrennt.** PEP, PE und PA kommunizieren in einem logisch getrennten Netzwerk, das nicht direkt zugänglich von Unternehmensressourcen ist. Die Datenebene wird für Anwendungs- und Dienst-Daten-Verkehr genutzt, während die Police Engine, der Policy Administrator und der Policy Enforcement Point über die Control Plane kommunizieren und darüber die Kommunikationspfade zwischen Ressourcen festlegen. Das PEP muss jedoch in der Lage sein, Nachrichten auf beiden Ebenen zu senden und zu empfangen. [Sta20]
   Kubernetes trennt die Datenebene (Pods und ihre Kommunikation) von der Kontrollebene (API-Server, Controller-Manager, etcd) logisch. Diese Trennung stellt sicher, dass die Control Plane isoliert und vor Datenverkehr geschützt ist, der für die Ausführung von Workloads im Cluster verwendet wird. Administrationszugriffe auf die Control Plane sind in der Regel auf bestimmte Netzwerksegmente oder VPNs beschränkt, um die Sicherheit zu erhöhen. [Say17]

6. **Ressourcen können die PEP Komponente erreichen.** Um Zugriff auf andere Ressourcen zu erlangen, muss eine Ressource Zugriff auf die PEP-Komponente haben. [Sta20]
   Innerhalb von Kubernetes können Ressourcen wie Pods über Kubernetes-native Mechanismen und Service Meshes wie Istio oder Linkerd auf Policy Enforcement Points (PEPs) zugreifen, um Zugriffsanfragen zu stellen und Richtlinien durchzusetzen. Die Verwendung von Sidecar-Proxy-Containern in einem Service Mesh sorgt dafür, dass alle Verbindungen über den PEP erfolgen und entsprechend den festgelegten Richtlinien verarbeitet werden. [cnc]

7. **PEP hat eine Verbindung zum PA** jedes PEP, dass auf dem Unternehmensnetzwerk arbeitet hat eine Verbindung zum Policy Administrator, um Kommunikationspfade zwischen Anwendern und Ressourcen zu ermöglichen. [Sta20]
   In Kubernetes können PEPs (Policy Enforcement Points) durch Werkzeuge wie Gatekeeper, Open Policy Agent (OPA) implementiert werden, die mit einer zentralen Policy Engine (PE) kommunizieren, um Zugriffskontrollen und Richtlinienentscheidungen zu treffen. Diese Komponenten arbeiten zusammen, um Richtlinienentscheidungen basierend auf den definierten Regeln zu erzwingen und sicherzustellen, dass alle Ressourcen nur gemäß den Richtlinien auf andere Ressourcen zugreifen können. [opea]

8. **Entfernte Ressourcen sollten auf Unternehmensressourcen zugreifen können, ohne erst die Netzwerkinfrastruktur des Unternehmens durchqueren zu müssen.** Ein entferntes Subjekt sollte nicht verpflichtet sein eine Rückverbindung zum Unternehmensnetz (VPN) zu nutzen, um auf Dienste zuzugreifen, die das Unternehmen nutzt und die von einem öffentlichen Cloud-Anbieter gehostet werden. [Sta20]
   In Kubernetes können entfernte Ressourcen direkt auf Cluster-Ressourcen zugreifen, indem sie Kubernetes-native Mechanismen wie Ingress-Controller oder VPN-Verbindungen verwenden. Managed Kubernetes-Dienste bieten oft zusätzliche Funktionen zur Integration mit externen Netzwerken, wodurch entfernte Ressourcen über das Internet oder spezielle Cloud-Verbindungen auf den Cluster zugreifen können, ohne durch eine interne Netzwerkstruktur gezwungen zu werden. [istg]

9. **Die Infrastruktur, die die ZTA Zugriffsentscheidungen unterstützt, sollte skalierbar sein, um Änderungen in der Prozesslast zu berücksichtigen.** Im Geschäftsprozess werden die Policy Engine, der Policy Administrator und der PEP zu den Schlüsselkomponenten. Verzögerungen oder Unfähigkeit einen PEP zu erreichen wirkt sich negativ auf die Fähigkeit zur Durchführung des Prozesses aus. Entsprechend müssen Komponenten für die erwartete Arbeitslast bereitstehen oder die Möglichkeit zum schnellen Skalieren der Infrastruktur bereitstehen, um eine erhöhte Nutzung zu bewältigen. [Sta20]
   Kubernetes ist von Natur aus eine skalierbare Plattform, die es ermöglicht, Workloads je nach Bedarf automatisch zu skalieren. Dies gilt auch für Sicherheitsinfrastrukturen

und Komponenten, die Zugriffskontrollen durchsetzen. Tools wie Horizontal Pod Autoscaler und Cluster Autoscaler können verwendet werden, um die Kapazität der Policy Engine, des Policy Administrators und der PEPs dynamisch zu skalieren, um eine erhöhte Nachfrage zu bewältigen. [Say17]

10. **Aufgrund von Richtlinien oder Entdeckungs-Faktoren sollen Unternehmensressourcen nicht in der Lage sein, einzelne PEPs zu erreichen.** Zum Beispiel könnte es eine Richtlinie geben, dass eine mobile Ressource nicht in der Lage ist, bestimmte Ressourcen erreichen zu können, wenn sich die anfragende Ressource außerhalb des Heimatlandes des Unternehmens befindet. Entsprechend vielfältig können diese Faktoren sein: zum Beispiel Standort oder Gerätetyp. [Sta20]
    Kubernetes kann durch den Einsatz von Network Policies und Service Meshes spezifische Zugriffsrichtlinien definieren, die den Zugriff auf bestimmte Ressourcen basierend auf Standort, Benutzeridentität oder Gerätetyp blockieren. Beispielsweise kann eine Network Policy konfiguriert werden, die den Zugriff von mobilen Geräten auf sensible Ressourcen verhindert, wenn diese sich außerhalb eines bestimmten geografischen Gebiets befinden. [kubh, kubd]

Diese Netzwerk-Anforderungen und Zero-Trust-Regeln bilden die Grundlage für eine robuste Zero-Trust-Architektur in Kubernetes, die darauf abzielt, die Sicherheitslücken zu minimieren und die Gesamtsicherheit des Systems zu erhöhen. Zusammenfassend bietet Kubernetes durch seine Netzwerkrichtlinien, Service Meshes, Sicherheitsintegrationen und skalierbare Infrastruktur eine robuste Grundlage zur Unterstützung einer Zero-Trust-Architektur. Dise Funktionen ermöglichen eine feingranulare Zugriffskontrolle und eine strenge Überwachung des Netzwerkverkehrs, die für die Umsetzung von Zero-Trust-Prinzipien erforderlich sind.

## 5.4 Zero-Trust-Architektur-Komponenten

Basierend auf den Security Best Practices für Kubernetes Cluster und anhand der Zero-Trust-Regeln für Kubernetes, können die folgenden Komponenten für eine Zero-Trust-Architektur in einem Kubernetes Cluster genutzt werden. Eine Zero-Trust-Architektur in Kubernetes muss sich, wie jede Zero-Trust-Architektur, an den Prinzipien von Zero-Trust orientieren und beschreibt die Funktionen und Wechselbeziehungen, Workflows der Kubernetes Komponenten genauso gut, wie deren Zugriffs-Richtlinien und Durchsetzungsinfrastruktur.

### 5.4.1 Netzwerk-Richtlinie

Auf der Netzwerkebene sollen gemäß den Zero-Trust-Regeln für Kubernetes der Cloud Native Computing Foundation sowohl der Ost-West- als auch der Nord-Süd-Verkehr gesichert

werden. Eine Möglichkeit, dies zu erreichen, ist die Implementierung einer Netzwerk-Richtlinie (NetworkPolicy) im Cluster. Dabei gibt es jedoch einige Einschränkungen für die Nutzung von NetworkPolicies innerhalb des Clusters: [kubh]

- Es wird ein Kubernetes-Cluster mit NetworkPolicy-Unterstützung benötigt.
- Ein Network-Policy Admission Controller muss aktiviert sein, um Netzwerk-Richtlinien durchzusetzen.

Das bedeutet, dass im Cluster ein Container Networking Interface (CNI) aktiviert sein muss, welches als Network-Policy Admission Controller fungiert und die Netzwerk-Richtlinien durchsetzt. Die meisten Cluster-Installationen werden ohne NetworkPolicy-Unterstützung bereitgestellt, weshalb diese Funktion oft nachträglich aktiviert werden muss. WeaveNet und Calico sind gängige Network-Policy-Provider, die für diese Aufgabe geeignet sind. [kubh]

NetworkPolicies ermöglichen die Kontrolle der Netzwerkkommunikation auf Schicht 3 (Netzwerk) und Schicht 4 (Transport) des OSI-Schichtenmodells. Ohne NetworkPolicies ist jeglicher Verkehr im Cluster erlaubt, was bedeutet, dass Anwendungen uneingeschränkt miteinander kommunizieren können. Dies bietet Angreifern die Möglichkeit, von einem Pod aus auf andere Komponenten im Cluster zuzugreifen oder diese zu kompromittieren. Durch den Einsatz von NetworkPolicies kann sowohl der Ingress- und Egress-Verkehr (Nord-Süd) als auch die interne Kommunikation im Cluster (Ost-West) auf den Schichten 3 und 4 (Netzwerk- und Transportschicht des OSI-Schichtenmodells) für die Daten- und Kontrollebene eingeschränkt werden. [kubh]

Da das Minikube-Cluster bereits mit dem Container Networking Interface (CNI) Calico gestartet wurde, unterstützt das Cluster nun NetworkPolicies und aktiviert einen Network-Policy Admission Controller. [kubh]

Für die Integration der Netzwerkrichtlinien ist es zunächst wichtig, festzulegen, wie die Kommunikation innerhalb des Clusters zwischen den Pods stattfindet. Innerhalb der Bookinfo-Anwendung findet die in Abbildung 5.6 dargestellte Kommunikation statt. Diese kann wie folgt beschrieben werden: [kubh]

- Das Ingress-Gateway kommuniziert ausschließlich mit dem Productpage-Service.
- Der Productpage-Service kommuniziert mit dem Details-Service.
- Der Productpage-Service kommuniziert mit den Reviews-Services.
- Die Reviews-Services (v2, v3) kommunizieren mit dem Ratings-Service.

Die NetworkPolicy soll zunächst jeglichen Ingress-Verkehr verhindern. Dies wird durch die in Listing A.23 in den Zeilen 1-10 definierte Policy erreicht. Anschließend wird der Zugriff auf das Ingress-Gateway für die Ports 80 und 443 geöffnet.

Daraufhin werden die spezifischen Kommunikationsregeln für die Pods festgelegt. Ein Beispiel für die erlaubte Ingress-Kommunikation für den Productpage-Service ist in den Zeilen 32-49 dargestellt. Hierbei wird über den `podSelector` festgelegt, für welche Pods diese Richtlinie gelten soll. Die `ingress`-Konfiguration definiert, dass der Ingress-Verkehr über Port 9080 erfolgen muss und nur von einem Pod mit dem Label `istio-ingressgateway` kommen darf. Die Productpage-Anwendung erlaubt somit die Kommunikation vom Istio-Ingress-Gateway auf Port 9080. Ähnliche Konfigurationen würden für die Services Details, Reviews und Ratings erstellt, wobei jeweils der entsprechende zugreifende Service in der `from`-Klausel definiert wird. Die korrekte Funktionsweise der definierten Netzwerk-Richtlinien kann wie folgt überprüft werden:

- Die ProductPage sollte auf die Reviews- und Details-Services zugreifen können. Dies wird durch den Zugriff auf die URL `<GATEWAY_URL>/productpage` überprüft. Der Zugriff funktioniert wie zuvor, und auch der Zugriff von Reviews auf Ratings funktioniert nach einmaligem Aktualisieren der Seite, was durch das Erscheinen der schwarzen Sterne angezeigt wird.

- Der Reviews-Service sollte keinen Zugriff auf die ProductPage haben. Dies wird getestet, indem eine Shell im Pod `reviews-v1` gestartet wird und der Befehl `curl productpage:9080/productpage` ausgeführt wird. Der Versuch sollte aufgrund eines Verbindungstimeouts fehlschlagen.

- Der Ratings-Service sollte keinen Zugriff auf die Productpage haben. Auch hier wird der Test über den Curl-Befehl vom Pod der Ratings auf `productpage:9080/productpage` durchgeführt. Auch hier sollte die Kommunikation aufgrund eines Verbindungstimeouts fehlschlagen.

**EKS:** Im Amazon EKS-Cluster ist für die Unterstützung von Netzwerk-Richtlinien ein Add-on notwendig. Das Add-on *Amazon VPC CNI* aktiviert Pod-Netzwerke innerhalb des Clusters und stellt Calico als Container Networking Interface zur Verfügung. Zusätzlich zum Add-on muss der Konfigurationswert `ënableNetworkPolicy": "true"` in der AWS Management-Konsole oder über die AWS CLI gesetzt werden.

Nach Anwendung der Netzwerkrichtlinien aus Listing A.23 können die Richtlinien erneut überprüft werden. Hierbei schlägt der Aufruf mit dem Befehl `kubectl exec (kubectl get pod -l app=ratings -o jsonpath='{.items[0].metadata.name}')-- curl -sS productpage :9080/productpage` aufgrund eines *upstream connect error* fehl. Der Grund hierfür ist der LoadBalancer, den das Istio-Ingress-Gateway erstellt. Dieser LoadBalancer wird in Amazon

außerhalb des Clusters als EC2 LoadBalancer bereitgestellt, wie in Abbildung 5.4 dargestellt. [amaf]

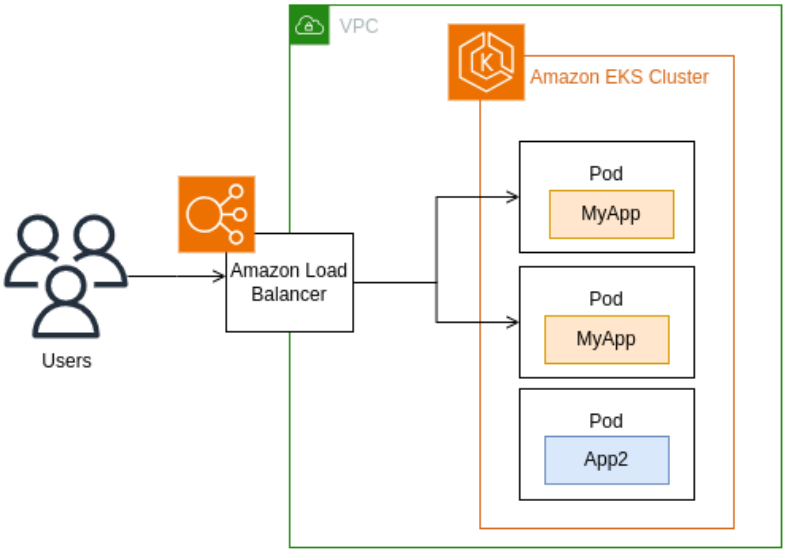



Abbildung 5.4: Darstellung der architekturellen Lage des Amazon Load Balancers in Amazon EKS. Der Load Balancer liegt in der Virtual Private Cloud (VPC) und nicht im Cluster. Dies sorgt dafür, dass Network Policies nicht korrekt ausgewertet werden. [amaf]

Aufgrund der Konfiguration, die jeglichen Ingress-Verkehr, außer dem explizit erlaubten, blockiert, schlägt die Kommunikation mit dem Ingress-Gateway fehl. Daher müssen die Netzwerk-Richtlinien aus Listing A.23 um die folgenden drei Richtlinien reduziert werden: `default-deny-all`, `istio-ingress-lockdown` und `product-page-ingress`. Somit greifen die Netzwerk-Richtlinien nur für die Anwendungen, Reviews, Details und Ratings. Gemäß den sieben Zero-Trust-Regeln für Kubernetes ist somit der Ost-West-Verkehr geregelt. Für den Nord-Süd-Verkehr bietet Istio bereits ein Ingress-Gateway, das den Zugang zum Cluster regelt. [amaf]

### 5.4.2 Service Mesh

Ein Service Mesh ist eine weitere Empfehlung der Cloud Native Computing Foundation. Ein Service Mesh unterstützt hierbei gleich weitere Funktionen, die für die Zero-Trust-Architektur genutzt werden können. Die Anzahl an ServiceMesh-Implementationen für Kubernetes wächst stetig. Die Cloud Native Computing Foundation (CNCF) fördert die Projekte Istio und Linkerd für Service Meshs. [istb] Beide bringen den fast identischen Funktionsumfang mit und unterscheiden sich hauptsächlich in der Performance.

Aufgrund dessen, dass Istio, weitaus öfter im Zusammenhang mit Kubernetes genutzt wird und auf Github eine höhere Referenz (Sterne) hat, betrachtet diese Arbeit den Einsatz von Istio in Kubernetes-Clustern. [gita]

Das Service Mesh unterstützt eine sichere Service-zu-Service-Kommunikation, indem es mutual TLS (mTLS) und Identiäten-basierte Authentifizierung und Authorization einsetzt. Weiterhin unterstützt ein Service Mesh Load Balancing, die Kontrolle des Netzwerkverkehrs, Richtlinienverwaltung, protokolliert Metriken und zeichnet Verkehrspfade innerhalb des Clusters auf. [istg]

Ein Service Mesh kontrolliert somit mit Policies den Netzwerkverkehr auf Layer 7 (Anwendungs-Schicht), stellt Ingress und Egress-Gateways zur Verfügung und prüft durch die Envoy Proxies auch die Authentifizierung und Autorisierung der Services und ermöglicht Enduser-Authentication. [istg]

Somit findet sich ein Service Mesh auf der Datenebene wieder, kontrolliert auf dieser aber auch Ost-West und Nord-Süd-Verkehr und integriert eine eigene Control Plane, die die Kommunikation zwischen den Proxys überwacht. [istg]

#### Istio Sidecar injektion

**Minikube & EKS:** Da Istio bereits im Cluster installiert ist, ist in diesem Schritt nur noch die Sidecar-Injektion und somit der Aktivierung des Service Meshs in den Clustern Minikube und Amazon EKS notwendig. [ista]
Hierzu muss ein Label auf dem `default` Namespace festgelegt werden. Das Label `istio-injection=enabled` sorgt dafür, dass Istio automatisch einen Sidecar-Container zu den Pods hinzufügt.[ista]
Anschließend werden die Pods der Bookinfo-Anwendung entfernt und neu mit Sidecar bereitgestellt. Im Amazon EKS Cluster kann es vorkommen, dass die Sidecar Container und somit die Pods nicht erfolgreich gestartet werden können. AWS gibt in der *SecurityGroup* die Ports 443 und 15017 nicht frei. Beide Ports müssen somit über die Security Group des Nodes freigegeben werden. Nach einer erneuten Bereitstellung der Pods, werden die Pods und die zugehörigen Container erfolgreich gestartet. [ista]

#### Access Control auf Workload-Ebene

Durch die Netzwerk-Richtlinien wurde die Kommunikation zwischen Pods zugelassen. Auf Ebene der Workloads und Anwendungen können mithilfe von AuthorizationPolicies die Netzwerkkommunikation eingeschränkt werden. [ista] Somit ergeben sich die folgenden Richtlinien:

1. Sämtlicher Netzwerkverkehr im Namespace sollte deaktiviert sein.

2. Die Productpage sollte nur über die GET http-Methode zugreifbar sein.

3. Die Details sollten nur von der Productpage über GET abrufbar sein.

4. Die Reviews sollten nur von der Productpage über GET abrufbar sein.

5. Die Ratings sollten nur von den Reviews über GET abrufbar sein.

**Minikube & EKS:** Die Richtlinien werden somit wie in Listing A.24 zu sehen umgesetzt. So wird zunächst in den Zeilen 1–7 die Richtlinie Nummer 1 umgesetzt. durch das leere spec-Objekt wird sämtlicher Request verboten. In den Zeilen 10–23 wird der productpage-Zugriff über GET definiert. Durch die fehlende 'from'-Definition wird festgelegt, dass der Zugriff von allen Bestandteilen des Clusters erfolgen darf. Die Zeilen 26-42 stehen repräsentativ für die Richtlinien 3,4 und 5. Hierbei wird die from Regel auf den Service Account der Productpage festgelegt. Somit darf die Productpage auf die Details-Anwendung zugreifen. Ähnlich würde diese Konfiguration für die Review-Anwendung aussehen, nur für die Ratings muss die 'from'-Konfiguration auf den Service-Account der Review Page festgelegt werden Die nächste AuthorizationPolicy aus den Zeilen 10 - 23 steht repräsentativ für die Richtlinie Nummern 2, 3,4 und 5. Die to-Konfiguration legt fest, dass in diesem Fall auf die Productpage (selector.matchLabels.app=productpage) nur per GET-Methode zugegriffen werden darf. Die fehlende from Route-Legt fest, dass der Zugriff von allen Bestandteilen des Clusters aus erfolgen darf. Für die Details und die Reviews werden ebenfalls zwei AuthorizationPolicies festgelegt, die in den Zeilen 26 bis 61 in Listing A.24 niedergeschrieben sind. Da in den Richtlinien 3 und 4 explizit verlangt wird, dass diese nur von der Productpage aufgerufen werden dürfen, findet sich in diesen nun eine from-Regel, die den Prinzipal auf den Service-Account (sa) der bookinfo-productpage festlegt. Der Zugriff darf nach to-Regel nur über die Methode GET, wie in der Richtlinie definiert, erfolgen. Ähnlich ist es sich dann für die letzte Richtlinie Nummer 5, die in den Zeilen 64 bis 80 niedergeschrieben ist. Hier ändert sich jedoch der Prinzipal für die Ausführung, denn es soll nur ein Zugriff über die Reviews möglich sein, wie unter from.sources.principals festgelegt. Somit wurden die Access-Control Richtlinien für die Workloads der Bookinfo-Anwendung festgelegt. [ista]

#### mTLS Service-zu-Service Verschlüsselung

mTLS ist eine Verbesserung von TLS, während TLS die Verbesserung von SSL darstellte. Das mTLS-Protokoll findet sich im OSI-Schichten-Modell zwischen der Transport Layer (Layer 4) und der Application Layer (Layer 5) wieder. [Pan]

mTLS kommt gerade für den neuen Ansatz der Microservice Architekturen infrage, da sich nun beide Kommunikations-Beteiligte, der web Client und der Webserver, authentifizieren müssen, bevor es zum Handshake kommt. [Pan]

Das Istio Service Mesh erstellt zusammen mit jeder Anwendung auch gleichzeitig ein Layer 4 und Layer 7 Sidecar-Proxy, den "Envoy proxy", der sämtliche Netzwerk-Kommunikation überwacht. Somit übernimmt dieser die Aufgaben eines Policy Enforcement Points. [Pan]

Das Konzept sidecar container "Envoy-Proxy" erstellt somit eine Zero-Trust-Architektur für jede Anwendung des Clusters. [Pan]

Abbildung 5.5 zeigt die Architektur des Istio Envoy Proxy und den Pfad der Kommunikation. [Pan]

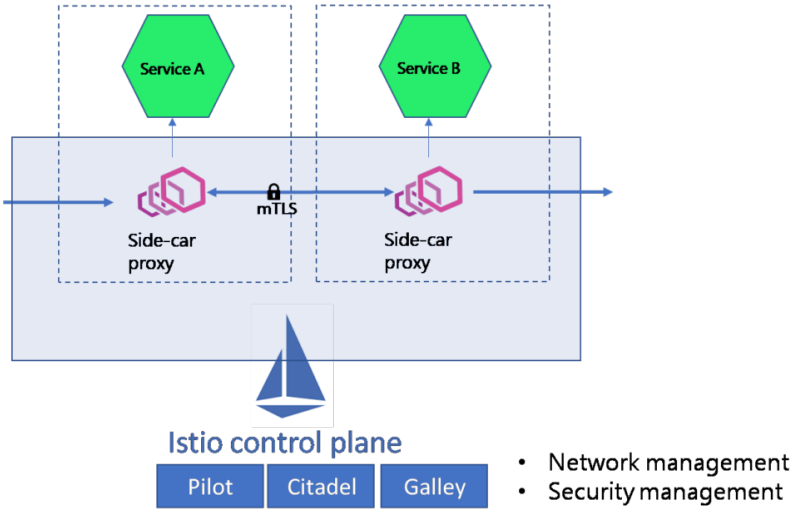



Abbildung 5.5: Darstellung der Istio-Envoy mTLS Kommunikation mit zwei Pods und deren Services A und B. Darstellung der Istio Control Plane im unteren Teil, die Aufgaben des Netzwerkmanagements und Sicherheitsmanagement übernimmt. [Pan]

**Minikube & EKS** Mithilfe der in Listing A.25 dargestellten mTLS PeerPolicy wird mTLS zwischen allen Teilen im Netz verpflichtend aktiviert. Standardmäßig führt Istio die Kommunikation über mTLS aus, leitet jedoch auch Anfragen ohne mTLS unverschlüsselt weiter. Aus diesem Grund muss mTLS verpflichtend aktiviert werden. Würde nun eine Applikation ohne Sidecar Container und mTLS installiert werden, würde dies dazu führen, dass dieser nicht auf andere Teile des Clusters, die mit mTLS arbeiten, zugreifen darf. Um die Funktionsweise der mtls-Verschlüsselung zu testen, wird ein Namespace namens `legacy` genutzt, darin wird ein Pod namens sleep aus den Beispielen von Istio deployt. Ohne Berücksichtigung der network Policy testen wir den Aufruf mit folgendem Curl Befehl: `kubectl exec sleep-5577c64d7c-b9gs2 -n legacy --stdin --tty -- curl productpage:9080/productpage -s`. Während selbiger Aufruf vom pod der Ratings erfolgreich durchgeführt werden kann, schlägt der Aufruf des sleep Pods des legacy Namespaces fehlt. Entsprechend ist die mtls-Kommunikation im Namespace `default` sichergestellt

und es kann zu keiner ungeschützten Kommunikation kommen. Standardmäßig würde Istio die Kommunikation sonst ohne mTLS durchführen. [istd]

#### Endnutzer-Authentifizierung mit JWT

Istio als Service Mesh unterstützt die Enduser-Authentifizierung für den Zugriff auf das Ingress-Gateway. Dazu lässt sich eine JWT-Enduser-Authentifizierung erstellen. [istc]

**Minikube & EKS:** Dazu wird eine RequestAuthentication Konfiguration angelegt, die jwtRules festlegen. Zur Validierung des JWT-Tokens wird hier ein JSON Web Key (JWK)-Service aus den Beispielen von Istio definiert. Zusammen mit einer E-Mail-Adresse des Issuers ist diese Konfiguration valide für die JWT-Authentifizierung. [istc]
Istio wird nun Anfragen ohne und mit validem Token erfolgreich (Status 200) akzeptieren. Anfragen mit invalidem Token werden mit Status 401 beantwortet. [istc]
Eine verpflichtende JWT-Validierung muss anschließend über eine AuthorizationPolicy für das Ingress-Gateway festgelegt werden. Diese ist in Listing A.26 dargestellt. [istc]
Die Aktion wird auf 'DENY' festgelegt und als Regel wird festgelegt, dass die Anfrage abgelehnt werden soll, wenn diese keinen RequestPrincipal enthält. RequestPrincipals sind nur verfügbar, wenn ein valides JWT-Token in der Anfrage übergeben wird. [istc]

### 5.4.3 Ingress Gateway

Ein Ingress (Gateway) ist ein API-Objekt, das externen Zugriff auf Services innerhalb des Clusters verwaltet. Es stellt Load Balancing, SSL-Termination und namensbasiertes virtuelles Routing bereit. [ista]
Der Vorteil gegenüber normalen Services ist, dass die Gateway-API einen erweiterbaren, rollenbasierten und protokollbasierten Konfigurationsmechanismus darstellt. Zusätzlich ist es ein Add-on, dass durch verschiedene API-Arten eine dynamische Infrastrukturbereitstellung und ein erweitertes Traffic-Routing ermöglicht. Während das Gateway selbst dafür zuständig ist, einen Endpunkt für Netzwerk-Anfragen festzulegen, übernehmen `HTTPRouten` das eigentliche Routing der Requests. So lassen sich verschiedene HTTP-Routen zum Beispiel anhand eines Hostnamens und der Zuordnung des passenden `PathPrefix` festlegen. [ista]

#### Ingress-Gateway integrieren

Um das Service-Mesh komplett zu machen, wird anschließend noch ein Ingress-Gateway in das Cluster eingefügt. Das Ingress-Gateway legt Port 80 (HTTP) als Zugangspunkt

zum Cluster fest. Ebenso wird mit dem Ingress-Gateway zusammen ein VirtualService in das Clsuter integriert, dass ein Route-Matching der Routen /productpage, /static, /login /logout, api/v1/products festlegt und als Routenziel die Productpage auf Port 9080 definiert. Zusammen mit Sidecar-Containern (Envoy) und Ingress-Gateway (Ingress Envoy) besteht die Bookinfo-Anwendung nun aus den in Abbildung 5.6 dargestellten Elementen. Die Netzwerkkommunikation wird nun über das istio-ingressgateway und den dortigen istio-proxy durchgeführt. Innerhalb der Anwendungs-Pods kamen die Istio-Sidecar-Container (rot) hinzu. [ista]

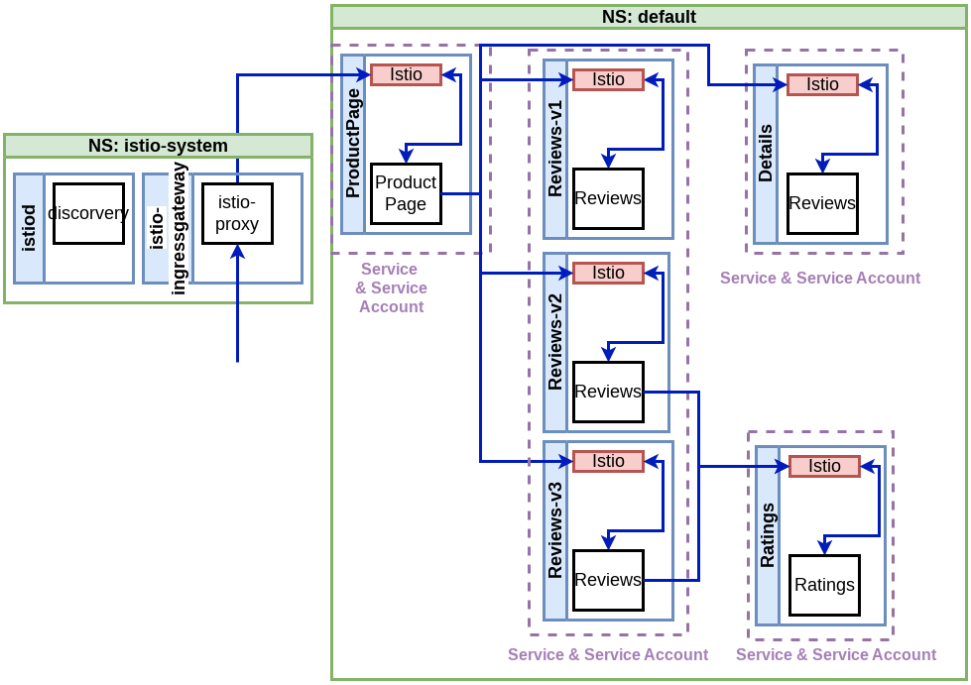



Abbildung 5.6: Bookinfo mit Istio-Sidecar und Ingress-Gateway dargestellt, mit den Namespaces (grün). Darstellung der Istio-Proxy (rot). Ebenso werden Service und ServiceAccounts (lila) integriert. Die Netzwerkkommunikation mit Service Mesh ist in blau dargestellt.

**Minikube & EKS:** Die Integration des Ingress-Gateways erfolgt in Minikube und EKS über `kubectl`. Dieser stellt eher einen digitalen Service (zu vergleichen mit einem Load Balancer; In EKS wird dieser als Load Balancer bereitgestellt) und einen ´Routing Option bereit, damit Anfragen an die ProductPage-Komponente weitergeleitet werden, wie es in Abbildung 5.6 dargestellt ist. [ista]

### 5.4.4 Zertifikatshandhabung automatisieren

Gemäß der Regeln für Zero-Trust in Kubernetes, soll die Zertifikatshandhabung und Aktualisierung automatisiert werden.

**Minikube & EKS:** Die Erneuerung von Zertifikaten erfolgt in Minikube und Amazon EKS durch kubeadm während eines Control Plane Upgrades. Dieses Verfahren ist in Umgebungen,

die regelmäßige Versionsaktualisierungen durchführen, oft ausreichend, da die Zertifikate dabei automatisch erneuert werden. Für Umgebungen mit speziellen Anforderungen oder Sicherheitsvorgaben kann die manuelle Erneuerung eine Alternative bieten.

Für die Zero-Trust-Architektur in Kubernetes-Clustern wird jedoch empfohlen, den cert-manager zu verwenden, um die Zertifikatshandhabung für Arbeitslasten zu automatisieren. Der cert-manager kann mit verschiedenen Certificate Authorities (CA) wie Let's Encrypt, HashiCorp Vault oder internen CAs integriert werden. Er generiert automatisch den Private Key und das Zertifikat, speichert diese sicher in einem Kubernetes-Secret und sorgt für deren automatische Erneuerung. Diese Vorgehensweise stellt sicher, dass die Zertifikate stets aktuell sind und reduziert das Risiko von Sicherheitslücken durch abgelaufene Zertifikate.

Die Integration von Add-ons wie csi-driver, csi-driver-spiffe oder istio-csr bietet zusätzliche Sicherheit, indem sie sicherstellen, dass private Schlüssel nur bei Bedarf erstellt werden und niemals die Node verlassen. Diese Add-ons ermöglichen eine engere Integration mit Service Meshes wie Istio, indem sie die Istio CA mit der CA des cert-managers verbinden und so eine zentrale Verwaltung und Erneuerung von Zertifikaten innerhalb des Clusters ermöglichen. Dies entspricht den Zero-Trust-Prinzipien, indem es sicherstellt, dass alle Verbindungen authentifiziert und autorisiert werden.

Im Rahmen dieser Arbeit wurde im Rahmen der Security Best Practices der cert-manager nur für die Zertifikatsverwaltung von Anwendungen betrachtet (siehe Kapitel 5.2.1), wobei eine externe Zertifikatsverwaltung zum Einsatz kommt. Diese Methode bietet eine zusätzliche Sicherheitsebene, indem sie den Schlüsselverwaltungsprozess aus dem Cluster heraus verlagert und so potenzielle Angriffspunkte minimiert.

### 5.4.5 Open Policy Agent (OPA)

Der Open Policy Agent (OPA) ist ein leistungsfähiges Werkzeug zur Laufzeitauswertung von Richtlinien. Mit OPA können Richtlinien zentral verwaltet und auf verschiedene Anwendungen und Umgebungen angewendet werden. Die Richtlinien werden in einer zentralen Repository gespeichert und in der maschinenlesbaren Rego-Sprache verfasst. Es gibt zwei Hauptmethoden, wie OPA innerhalb eines Kubernetes-Clusters eingesetzt werden kann: als Teil eines Gatekeepers oder in Kombination mit einem Service Mesh. Im Folgenden werden diese beiden Ansätze genauer betrachtet. [opea]

**OPA mit Gatekeeper:** Der OPA Gatekeeper fungiert als validierender Kubernetes-Zugangscontroller. Er sorgt dafür, dass Richtlinien auf Kubernetes-Objekte angewendet werden, wenn diese erstellt oder geändert werden. Sobald ein Objekt durch einen Nutzer oder einen Prozess verändert wird, fängt der OPA Gatekeeper die Anfrage ab und prüft sie

gegen die definierten Richtlinien. Bei einem Verstoß wird die Anfrage abgelehnt. Ein typisches Anwendungsbeispiel wäre die zentrale Kontrolle der Herkunft von Container-Images innerhalb eines Clusters. Zudem kann festgelegt werden, dass jeder Pod ein app-Label tragen muss. Obwohl diese Anwendungen nützlich sind, gehen sie über die Anforderungen dieser Arbeit hinaus und werden daher nicht weiter thematisiert.

**OPA mit Service Mesh:** Ein alternativer Einsatz von OPA erfolgt in Verbindung mit einem Service Mesh. In diesem Szenario stellt der Open Policy Agent sicher, dass der Netzwerkverkehr sicher ist und den definierten Richtlinien entspricht. So kann beispielsweise sichergestellt werden, dass Anfragen an einen Service blockiert werden, wenn sie nicht die richtige Autorisierung oder Verschlüsselung aufweisen. [opea]

Da im Rahmen dieser Arbeit eine Zero-Trust-Architektur mit einem Service Mesh implementiert wird, liegt der Fokus im Folgenden auf der Integration von OPA in ein solches Service Mesh. [opea]

#### Service Mesh vs. OPA

Beim Vergleich zwischen Service Meshes und Open Policy Agents fällt der Fokus auf fünf zentrale Richtlinienprüfungen [Das]:

1. Verschlüsselung während der Übertragung
2. Service-Identität und Authentifizierung
3. Service-zu-Service-Autorisierung
4. Endbenutzer-Identität und Authentifizierung
5. Endbenutzer-zu-Ressourcen-Autorisierung

Während ein Service Mesh eine dedizierte Infrastruktur bietet, die die ersten vier Richtlinien vollständig und die fünfte teilweise abdeckt, geht der Open Policy Agent in der Endbenutzer-zu-Ressourcen-Autorisierung noch einen Schritt weiter. Mit OPA können anwendungsspezifische Richtlinien für einzelne Anfragen definiert werden. Dadurch erweitert OPA die Möglichkeiten über die grundlegenden Zero-Trust-Prinzipien hinaus. Tabelle 5.2 illustriert die Unterschiede zwischen Istio und Open Policy Agent hinsichtlich der Autorisierung. [Das]

Tabelle 5.2: Vergleich der Funktionalitäten zwischen Istio Authorization (Authz) und Open Policy Agent (OPA). Die Tabelle zeigt die Unterschiede in der Unterstützung von Datengruppen und Feldern, die in Richtlinien verwendet werden können. Während beide Systeme HTTP-Anfragen, Kubernetes-spezifische Informationen und JWT-Token verarbeiten können, bietet OPA zusätzliche Flexibilität durch die Unterstützung von kontextbezogenen Daten und dem Auswerten des HTTP-Request-Bodys, was in Istio Authz nicht möglich ist. [Das]

| Data Group | Field | Istio Authz | OPA |
|---|---|---|---|
| HTTP | Request header, connection SNI | ✓ | ✓ |
| Kubernetes | IP address, IP block, namespaces, request principals | ✓ | ✓ |
| JWT | Token validation, fields | ✓ | ✓ |
| Contextual Data | Other static data or data not in request | x | ✓ |
| HTTP Request Body | | x | ✓ |

#### Zero-Trust-Architektur mit OPA und Service Mesh

Der Open Policy Agent wird im Folgenden in Kombination mit dem Istio-Service Mesh betrachtet. Im Kapitel 5.1 wurde bereits die Installation des Open Policy Agent Admission Controller und der Authentifizierungskonfiguration durchgeführt. Die Konfigurationsdatei ist in Listing A.27 dargestellt. Die Benutzer Alice und Bob werden in diesem Beispiel definiert, wobei Alice die Rolle guest und Bob die Rolle admin erhält. Für die Rollen wird festgelegt, dass guest nur auf /productpage zugreifen darf, während admin Zugriff auf /productpage und /api/v1/products erhält. [opeb]

**Minikube & EKS:** Innerhalb des Minikube und EKS-Clustern muss nun noch der `default` Namespace mit dem Label `opa-istio-injection="enabled"` versehen werden. Anschließend werden die Pods erneut bereitgestellt, wodurch der OPA-Sidecar-Container in die Pods integriert wird. Nun können die zuvor definierten Regeln getestet werden. Gemäß der Autorisierungsrichtlinie hat Alice keinen Zugriff auf /api/v1/products, was durch den Befehl `curl --user alice -i http://$GATEWAY_URL/api/v1/products` mit einem HTTP-Status 403 bestätigt wird. Der Zugriff auf /productpage hingegen liefert den HTTP-Status 200 zurück. Bob hat wie festgelegt vollen Zugriff auf beide Ressourcen. [opeb]

### 5.4.6 Monitoring

In modernen Kubernetes-Clustern ist die Überwachung von Netzwerkverkehr, Performance und Ressourcenverbrauch von entscheidender Bedeutung, um die Stabilität und Effizienz der Anwendungen sicherzustellen. Netzwerk-Monitoring ermöglicht es, ungewöhnliche Aktivitäten zu erkennen, Engpässe zu identifizieren und die Kommunikation zwischen den Diensten zu optimieren. Gleichzeitig sorgt das Monitoring von Performance und Ressourcenverbrauch dafür, dass Anwendungen effizient laufen und schnell auf Änderungen in der Last oder bei Fehlern reagiert werden kann. Ohne eine gründliche Überwachung könnten Probleme

unentdeckt bleiben, was zu einer verminderten Systemleistung oder sogar zu Ausfällen führen kann.

### Monitoring des Netzwerkverkehrs

Der Netzwerkverkehr in Kubernetes-Clustern sollte stets überwacht werden, um die Performance und Sicherheit der Anwendungen zu gewährleisten.

**Minikube:** In einem Minikube-Cluster kann dies effektiv mit Kiali erfolgen, einem Tool, das als Add-on für Istio geliefert wird. Kiali bietet eine umfassende Visualisierung der Service-Mesh-Topologie und ermöglicht es, den Datenfluss zwischen den Services zu beobachten. Es zeigt detaillierte Metriken wie Request-Raten, Latenzen und Fehlerraten, die für die Diagnose von Netzwerkproblemen essenziell sind. Kiali wird mithilfe einer YAML-Konfigurationsdatei in das Cluster integriert und kann über den Befehl `istioctl dashboard kiali` lokal auf localhost:2001 gestartet werden. Dies bietet eine benutzerfreundliche Oberfläche, auf der Administratoren und Entwickler die Netzwerkaktivitäten in Echtzeit überwachen können. [opeb]

**EKS:** Für das Monitoring des Netzwerkverkehrs in einem Amazon EKS-Cluster stehen verschiedene Tools zur Verfügung. AWS CloudWatch ist eine besonders geeignete Lösung, da sie nicht nur detaillierte Metriken und Log-Daten sammelt, sondern auch für das Audit-Logging verwendet werden kann. Für eine tiefere Analyse und visuelle Darstellung von Netzwerkpfaden und Abhängigkeiten können zusätzlich AWS Distro für OpenTelemetry (ADOT) und AWS X-Ray eingesetzt werden. Diese Tools bieten erweiterte Funktionen zur Überwachung und Diagnose von verteilten Anwendungen. Amazon DevOps Guru unterstützt zudem bei der automatischen Erkennung von Anomalien und potenziellen Problemen, was proaktive Maßnahmen ermöglicht. Prometheus kann ebenfalls für das Monitoring verwendet werden, benötigt jedoch eine tiefere Integration im EKS-Cluster. Im Rahmen dieser Arbeit wird die Integration von Kiali in Kubernetes Clustern betrachtet. Die Integration erfolgt in gleicher Weise, wie im Minikube Cluster. Das Dashboad wird ebenfalls lokal unter localhot:20001 dargestellt. [opeb, amaa]

### Performance- und Ressourcen-Monitoring

Die Überwachung der Performance und des Ressourcenverbrauchs ist unerlässlich, um die Effizienz und Verfügbarkeit von Anwendungen in Kubernetes-Umgebungen zu gewährleisten.

**Minikube:** In einem Minikube-Cluster kann dies durch den Einsatz des metrics-server-Addons sowie durch Prometheus erfolgen. Der metrics-server liefert grundlegende Metriken

zur Ressourcennutzung, die für automatische Skalierungsentscheidungen von Kubernetes verwendet werden können. Prometheus hingegen bietet eine tiefgehende Überwachung, indem es detaillierte Metriken über den Zustand des Clusters und der laufenden Anwendungen sammelt. Diese Metriken können in Grafana visualisiert werden, das vorkonfiguriert ist, um sowohl den Ressourcenverbrauch als auch die Istio Control Plane zu überwachen. Grafana ermöglicht auch das Setzen von Alarmschwellen, um Administratoren bei der Überschreitung bestimmter Metrikwerte zu warnen. Das Dashboard von Grafana kann lokal auf localhost:3000 geöffnet werden, indem der Befehl istioctl dashboard grafana ausgeführt wird, was eine intuitive Überwachung der Clusterressourcen ermöglicht.

**EKS:** In Amazon EKS ist AWS CloudWatch die zentrale Lösung für das Performance- und Ressourcen-Monitoring. CloudWatch bietet eine umfassende Übersicht über die Clusterressourcen und deren Auslastung. Durch die Integration mit anderen AWS-Diensten wie AWS Lambda und AWS X-Ray ermöglicht es eine lückenlose Überwachung und Fehlerbehebung. Für Benutzer, die eine tiefere Integration benötigen, kann Prometheus verwendet werden, welches zusammen mit einem agentenlosen Scraper von AWS betrieben wird. Dies ermöglicht das Sammeln von Metriken ohne zusätzliche Installationen im Cluster, was den Verwaltungsaufwand reduziert und die Leistung verbessert. Diese Kombination von Tools stellt sicher, dass die Cluster sowohl effizient als auch stabil betrieben werden können, indem sie Echtzeitinformationen und historische Daten für fundierte Entscheidungsprozesse bereitstellen. Im Rahmen dieser Arbeit wird jedoch die Integration von Grafana und Prometheus betrachtet. Dabei erfolgt die Integration im gleicher Weise wie im Minikube Cluster. Das Dashboard zeigt unter Localhost dann die Monitoring-Daten des EKS-Clusters. [amaa]

## 5.5 Architekturübersicht und Zusammenfassung

In diesem Kapitel wurden die Security Best Practices vorgestellt und implementiert. Weiterhin wurde eine Zero-Trust-Architektur innerhalb von Kubernetes-Umgebungen am Beispiel von Minikube und Amazon Elastic Kubernetes Service (EKS) entwickelt und implementiert. Die Abbildungen 5.7 und 5.8 geben einen detaillierten Überblick über die Architektur dieser Zero-Trust-Implementierung in den jeweiligen Kubernetes-Clustern.

Die Architektur innerhalb des Minikube-Clusters, wie in Abbildung 5.7 dargestellt, umfasst die Bookinfo-Anwendung im Namespace `default` sowie essenzielle Systemkomponenten für Istio, Open Policy Agent (OPA), und das Monitoring von Netzwerkverkehr und Ressourcen. Diese Komponenten ermöglichen eine detaillierte Überwachung und Steuerung des Netzwerkverkehrs sowie die Durchsetzung von Richtlinien, was zentral für die Zero-Trust-Sicherheitsarchitektur ist. Der `kube-system`-Namespace wird nur teilweise abgebildet, um die Übersichtlichkeit zu bewahren, da eine vollständige Darstellung aufgrund der Vielzahl an Pods unpraktikabel wäre.

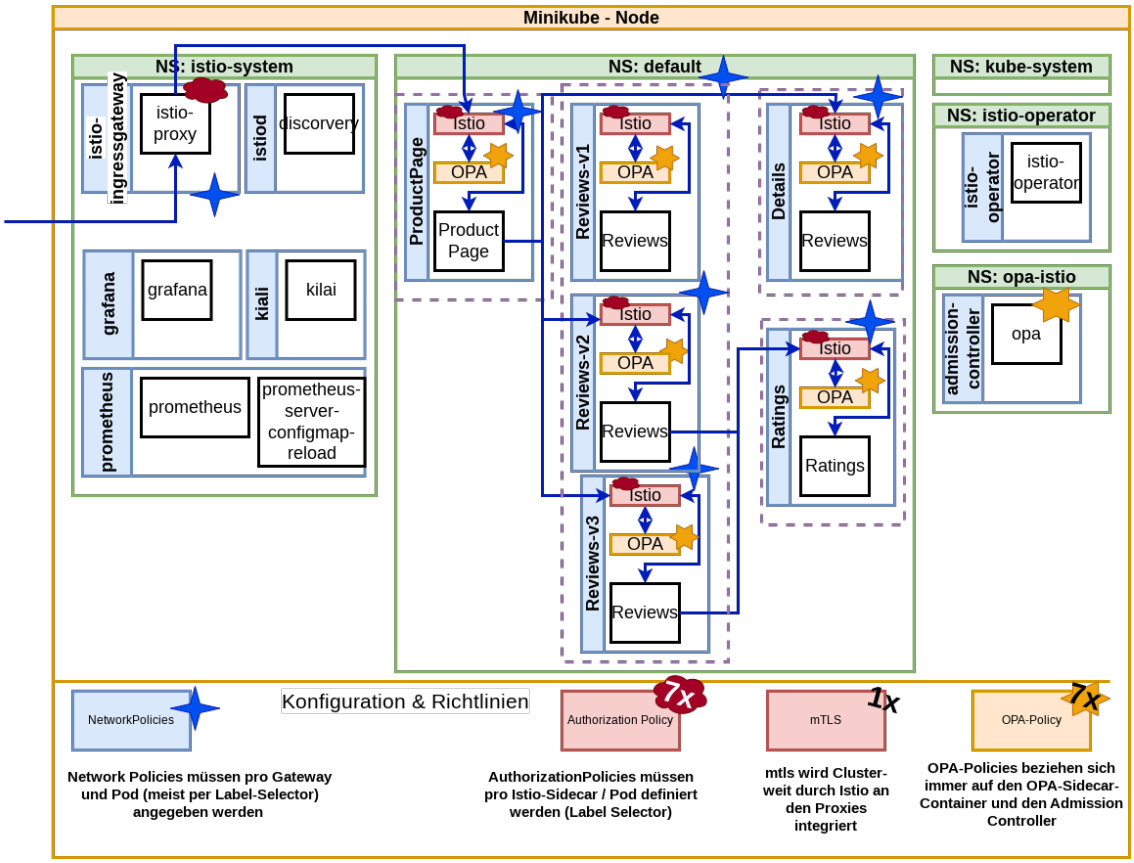



Abbildung 5.7: Überblick über die Zero-Trust-Architektur innerhalb des Minikube-Clusters. Dargestellt sind die Nodes (gelb), die Namespaces (grün), die Services und Service Accounts (lila) und die Pods (blau) mit ihren Containern. Ebenfalls dargestellt sind die notwendigen Richtlinien wie Network Policies, Authorization Policies, mTLS und die OPA-Policy. Die farbigen Symbole zeigen die Zuordnung der NetworkPolicies und Authorization Policy zu den Pods und Sidecar-Proxies.

Die Architektur des EKS-Clusters, wie in Abbildung 5.8 gezeigt, weist ähnliche Strukturen auf, jedoch mit spezifischen Anpassungen und Erweiterungen auf zwei Nodes, die für den Betrieb in einer skalierbaren und verwalteten Cloud-Umgebung notwendig sind.

Die Implementierung der Zero-Trust-Prinzipien in diesen Kubernetes-Clustern erfordert eine umfangreiche Konfiguration und Anpassung auf verschiedenen Ebenen:

Die Kommunikation zwischen den Pods wird durch eine Vielzahl von NetworkPolicies und AuthorizationPolicies reguliert, die sowohl auf Pod- als auch auf Namespace-Ebene definiert werden. Jede dieser Richtlinien muss sorgfältig erstellt und konfiguriert werden, um sicherzustellen, dass die Zero-Trust-Prinzipien wie „Least Privilege“ und „Deny by Default“ konsequent durchgesetzt werden. Dies führt zu einer erheblichen Komplexität, insbesondere wenn die Anzahl der Pods und deren Interaktionen steigt. Die Implementierung von Network Policies und Authorization Policies auf den Schichten des OSI-Modells, nämlich

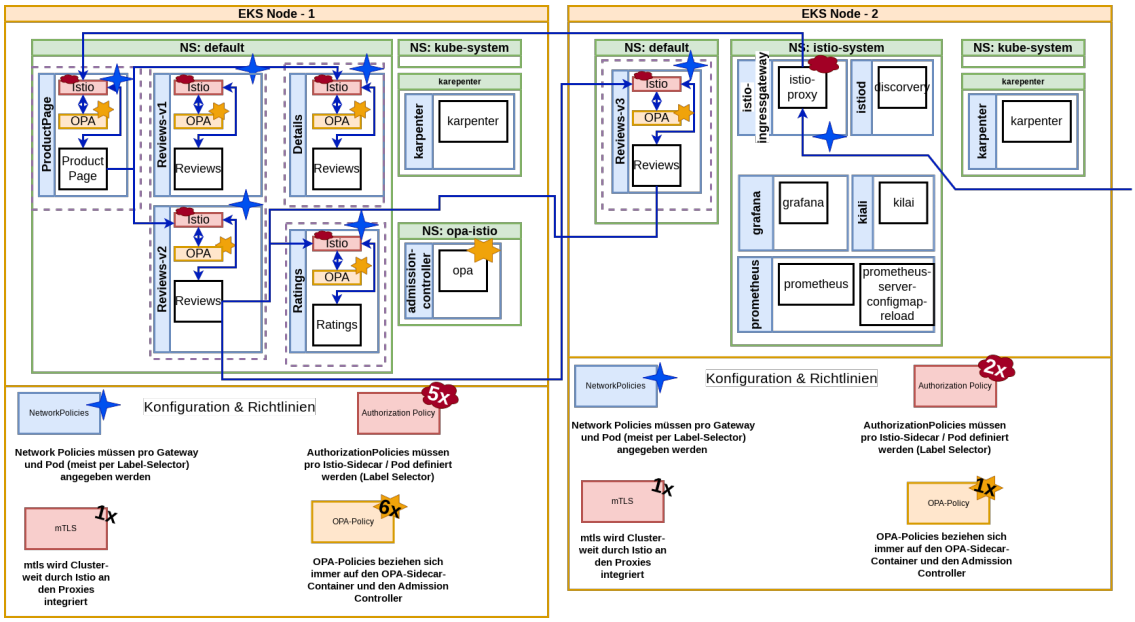



Abbildung 5.8: Überblick über die Zero-Trust-Architektur innerhalb des EKS-Clusters. Dargestellt sind die Nodes (gelb), die Namespaces (grün), die Services und Service Accounts (lila) und die Pods (blau) mit ihren Containern. Ebenfalls dargestellt sind die notwendigen Richtlinien wie NetworkPolicies, AuthorizationPolicies, mTLS und die OPA-Policy. Die farbigen Symbole zeigen die Zuordnung der NetworkPolicies und Authorization Policy zu den Pods und Sidecar-Proxies.

der Netzwerkschicht (Layer 3), Transportschicht (Layer 4) und Anwendungsschicht (Layer 7), verdeutlicht den Defense-in-Depth-Ansatz der Zero-Trust-Architektur.

Zur Erhöhung der Sicherheit und zur Umsetzung der Zero-Trust-Architektur werden Sidecar-Container in Form von Istio-Proxies eingesetzt. Diese ermöglichen die Durchsetzung von mTLS (mutual TLS), was sicherstellt, dass die Kommunikation zwischen den Services verschlüsselt und authentifiziert ist. Die Konfiguration dieser Proxies und die Pflege der zugehörigen Sicherheitszertifikate stellen weitere administrative Aufgaben dar.

Neben den Netzwerk- und Autorisierungsrichtlinien sind zusätzliche Sicherheitspraktiken wie die Begrenzung von Pod-Ressourcen, die Einschränkung von Containerberechtigungen und das Setzen von Read-Only-Root-Dateisystemen unerlässlich. Diese Maßnahmen erfordern ein tiefes Verständnis der Anwendungen, da die Sicherheitsrichtlinien individuell auf die Bedürfnisse jeder Anwendung zugeschnitten sein müssen.

Die Integration von Tools wie Kiali, Prometheus und Grafana ermöglicht eine kontinuierliche Überwachung des Netzwerkverkehrs und der Ressourcennutzung. Allerdings muss für jede Anwendung spezifisch konfiguriert werden, welche Metriken erfasst und wie diese visualisiert werden. Die damit verbundene Komplexität wächst mit der Anzahl der Anwendungen und ihrer spezifischen Anforderungen an die Überwachung.

Der Überblick über die Security Best Practices und die Komponenten der Zero-Trust-Architektur zeigt, dass die Abgrenzung der beiden Sicherheitsmaßnahmen nicht strikt durchgeführt werden kann. So finden sich Network Policies und Monitoring-Vorgaben auch in den Security Best Practices wieder, während Least-Privilege Container-Richtlinien auch ein wesentlicher Bestandteil von Zero-Trust-Architekturen ist. Somit ist eine Zero-Trust-Architektur eine erweiterte Sicherheitslösung für Kubernetes Cluster, die auf den Security Best Practices aufbaut und diese ergänzt.

### Zusammenfassung

Die Implementierung einer Zero-Trust-Architektur in Kubernetes-Clustern wie Minikube und EKS erfordert eine detaillierte Planung und umfangreiche Konfigurationsarbeiten. Die Notwendigkeit, individuelle Sicherheitsrichtlinien für jede Anwendung zu erstellen und zu pflegen, und die kontinuierliche Überwachung und Anpassung an neue Anforderungen machen die Verwaltung solcher Architekturen komplex und zeitaufwändig. Besonders in bestehenden Clustern, die ohne Unterbrechungen migriert werden müssen, stellt dies eine besondere Herausforderung dar. Dennoch bietet die Zero-Trust-Architektur erhebliche Sicherheitsvorteile, da sie eine granularere Kontrolle über den Netzwerkverkehr und die Zugriffsrechte innerhalb des Clusters ermöglicht. Die Fähigkeit, auf potenzielle Bedrohungen schnell zu reagieren und Sicherheitsrichtlinien dynamisch anzupassen, macht diese Architektur besonders wertvoll in Umgebungen, die eine hohe Sicherheit erfordern.

# 6 Evaluation

In diesem Kapitel wird die Effizienz der Zero-Trust-Architektur in Kubernetes-Clustern bewertet. Dabei sollen die Forschungsfragen FF.2, FF.3 und FF.4 beantwortet werden. Zunächst wird die Einhaltung der Zero-Trust-Grundprinzipien erläutert (Kapitel 6.1). Anschließend folgt eine Sicherheitsbewertung anhand einer erweiterten Bedrohungsmatrix und Heatmap (Kapitel 6.2). Zudem wird der zusätzliche Ressourcenverbrauch der Zero-Trust-Architektur in Minikube- und Amazon EKS-Clustern analysiert und deren wirtschaftliche Auswirkungen auf Amazon EKS und Public Clouds betrachtet (Kapitel 6.3). Abschließend erfolgt eine Bewertung der Sicherheitsrisiken, die durch die Einführung der Zero-Trust-Architektur entstehen können (Kapitel 6.4).

## 6.1 Einhaltung der Zero-Trust-Grundsätze

Das National Institute of Standards and Technology (NIST) hat sieben Grundprinzipien für Zero-Trust festgelegt, die in den Grundlagen zu Zero-Trust in Kapitel 2.2 vorgestellt wurden. Die erstellte Zero-Trust-Architektur wird nun daraufhin untersucht, wie gut diese Prinzipien eingehalten werden.

Eines der ersten Prinzipien des NIST für Zero-Trust ist die *Klassifizierung aller Geräte*, um unternehmenseigene von persönlichen Geräten zu unterscheiden. Innerhalb eines Kubernetes-Clusters kann diese Klassifizierung durch die Verwendung von Security Best Practices, wie zum Beispiel Namespaces und Labels, umgesetzt werden. Durch die eindeutige Zuordnung von Labels zu den jeweiligen Geräten oder Pods können spezifische Sicherheitsrichtlinien angewendet werden, die sicherstellen, dass nur autorisierte Geräte Zugriff auf bestimmte Ressourcen haben.
Ein weiteres wichtiges Prinzip ist, dass *jede Kommunikation abgesichert* sein muss, unabhängig von der Netzwerkadresse. In einem Kubernetes-Cluster wird dieser Grundsatz durch den Einsatz eines Service Mesh, wie Istio, umgesetzt. Das Service Mesh ermöglicht die sichere Kommunikation zwischen den Services durch Authentifizierung und Autorisierung jeder Anfrage, unabhängig von der physischen oder logischen Netzwerkposition der Dienste. Dies verhindert, dass unbefugte Zugriffe allein durch die Kenntnis der Netzwerkadresse erfolgen können.
Das NIST fordert, dass der Zugang zu Ressourcen nach dem *Prinzip der minimalen Pri-*

*vilegien* gewährt wird. In der Zero-Trust-Architektur eines Kubernetes-Clusters wird dies durch mehrere Sicherheitsebenen realisiert. Auf der Netzwerkschicht sorgt das Service Mesh für Authentifizierung und Autorisierung zwischen den Diensten. Darüber hinaus wird Role-based Access Control (RBAC) eingesetzt, um feingranulare Zugriffsrechte zu definieren. Auf Container- und Pod-Ebene werden die Best Practices für Least Privilege Security aus Kapitel 5.2.2 angewendet, um die Berechtigungen der Container auf das absolute Minimum zu beschränken.
Ein zentraler Bestandteil der Zero-Trust-Prinzipien ist die *kontinuierliche Überwachung und Bewertung* der Sicherheitslage. Dies wird in der Kubernetes-Architektur durch Tools wie Falco erreicht, das in Kapitel 5.2.2 beschrieben wurde. Falco ermöglicht eine Echtzeit-Diagnose von ungewöhnlichen Aktivitäten innerhalb des Clusters, bietet jedoch keine automatische Schadensbegrenzung. Die kontinuierliche Überwachung ermöglicht eine schnelle Reaktion auf potenzielle Bedrohungen und gewährleistet, dass verdächtige Aktivitäten schnell erkannt und untersucht werden können.
Um die Sicherheit weiter zu erhöhen, empfiehlt das NIST den *Einsatz von Identitäts-, Berechtigungs- und Zugriffsmanagement*, einschließlich Multifaktor-Authentifizierung (MFA). In Cloud-Umgebungen, wie beispielsweise Amazon AWS, Google Cloud Platform oder Microsoft Azure, wird dies durch integrierte Dienste wie AWS Identity and Access Management (IAM) realisiert. Diese Tools bieten eine einfache Möglichkeit, Benutzer und Dienste zu authentifizieren und zu autorisieren sowie Zugriffskontrollen durchzusetzen.
*Monitoring*-Tools wie Prometheus, Kiali und Grafana sind essenziell, um eine kontinuierliche Überwachung und Neubewertung der Zugriffsrechte zu ermöglichen. Diese Tools sammeln umfangreiche Daten über den Sicherheitsstatus des Clusters, den Netzwerkverkehr und die Zugriffsanfragen, wodurch eine umfassende Sicht auf die Aktivitäten im Netzwerk entsteht und Schwachstellen frühzeitig identifiziert werden können.
Schließlich legt das NIST großen Wert auf eine *strenge Identitätsüberprüfung* bei jedem Zugriff auf Ressourcen. Dies wird in der Kubernetes-Umgebung durch die Kombination von RBAC, Policies und dem Einsatz von MFA bei der Authentifizierung von Benutzern und Services erreicht. Durch die konsequente Durchsetzung dieser Maßnahmen wird sichergestellt, dass nur autorisierte und authentifizierte Benutzer und Dienste Zugang zu den Ressourcen erhalten.

Zusammenfassend lässt sich sagen, dass die sieben zentralen Grundsätze für Zero-Trust in der implementierten Zero-Trust-Architektur unter Einbezug der Security Best Practices erfüllt werden. Die Verwendung von Sicherheitsmechanismen wie Service Mesh, RBAC, MFA und kontinuierlichem Monitoring stellt sicher, dass die Sicherheitsanforderungen auf allen Ebenen des Kubernetes-Clusters berücksichtigt werden und die Prinzipien des Zero-Trust-Modells umfassend eingehalten werden.

## 6.2 Verhinderung von Bedrohungen

Dieses Kapitel analysiert, wie eine Zero-Trust-Architektur zur Verringerung von Bedrohungen in einem Kubernetes-Cluster beiträgt. Anhand der erweiterten Bedrohungsmatrix und Heatmap aus Kapitel 4.4 wird aufgezeigt, welche Bedrohungen durch die Zero-Trust-Architektur verhindert oder reduziert werden können.

Die Abbildungen 6.1 und 6.2 zeigt die erweiterte Bedrohungsmatrix für Kubernetes-Cluster und die möglichen Gegenmaßnahmen für die verschiedenen Bedrohungstechniken. Bedrohungen, die durch Security Best Practices oder Komponenten der Zero-Trust-Architektur vollständig verhindert werden können, sind in Grün dargestellt. Maßnahmen, die zur Reduktion der Bedrohung beitragen, jedoch nicht vollständig verhindern, sind Gelb eingefärbt. Bedrohungstechniken, für die es keine Gegenmaßnahmen gibt, sind Rot eingefärbt und mit dem Hinweis *keine* versehen. Die Angriffsvektoren und erarbeiteten Schwachstellen sind zur Vergleichbarkeit mit der Matrix aus Kapitel 4.4 in Blau und Orange dargestellt.

Ein genauer Blick auf die Matrix zeigt, dass das Konzept der Role-based Access Control (RBAC) eine zentrale Rolle spielt. RBAC war bereits in den Security Best Practices entscheidend und trägt wesentlich zur Verhinderung von Bedrohungen bei.

Konkret für die einzelnen Bedrohungstaktiken kommen folgende Gegenmaßnahmen infrage: **Initial Access** umfasst Techniken, die Angreifer nutzen, um erstmals Zugriff auf ein Kubernetes-Cluster zu erlangen. Es ist der erste Schritt in einem Angriff, bei dem der Angreifer versucht, eine Präsenz im Cluster zu etablieren. Die folgenden Gegenmaßnahmen können die Gefahr dieser Angriffe deutlich reduzieren:

**a)** Using Cloud Credentials: Role-based Access Control (RBAC) ist eine effektive Methode, um den Zugriff auf das Cluster zu steuern. Durch RBAC kann sichergestellt werden, dass nur autorisierte Benutzer und Dienste mit den erforderlichen Berechtigungen auf das Cluster zugreifen können. Jeder Benutzer und jede Rolle sollte so konfiguriert werden, dass nur die minimal notwendigen Rechte gewährt werden.

**b)** Kompromittierte Images in Registry: Die Verwendung von privaten Registries und Image-Scanning-Tools wie Snyk kann helfen, kompromittierte Images frühzeitig zu erkennen und aus der Produktionsumgebung fernzuhalten. Regelmäßige Scans und die Kontrolle der Images vor der Bereitstellung können das Risiko von Schadsoftware oder unsicheren Konfigurationen reduzieren.

**c)** Kubeconfig Datei: Da der Zugriff auf die Kubeconfig-Datei oft über den Public Cloud Provider erfolgt, ist es wichtig, den Zugang zu dieser Datei auf autorisierte Benutzer zu beschränken und sichere Speicherorte und Verfahren für diese Datei zu etablieren.

| Taktiken / Techniken | Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion | Credential Access |
|---|---|---|---|---|---|---|
| Bedrohung | Using Cloud Credentials | Exec in einen Container | Backdoor Container | Privilegierter Container | Container Logs löschen | Kubernetes Secrets auflisten |
| Gegen-maßnahme | RBAC | RBAC | RBAC | SecurityContext. Seccomp, AppArmor, SELinux | Falco | Vault, KMS |
| Bedrohung | Kompromitiertes Image in Registry | bash/cmd innerhalb Containers | Writable hostpath mount | Cluster-Admin Binding | Delete Kubernetes Events | Mount Service Principal |
| Gegen-maßnahme | private Registry & z.B. Tool, wie Snyk | SecurityContext, AppArmor, Seccomp, SELinux | RBAC | RBAC, Least Privilege | Falco | RBAC |
| Bedrohung | Kubeconfig Datei | Neuer Container | Kubernetes Cronjob | hostPath mount | Pod/container name similarity | Zugriff auf Container Service Account |
| Gegen-maßnahme | Keine | RBAC | RBAC | RBAC, Least Privilege | vergabe eindeutiger Namen für Pods und Container | eigener Service Account pro Pod/Service |
| Bedrohung | Schwachstelle in einer Anwendung | Anwendungs-Exploit (RCE) | Malicious Admission Controller | Access Cloud Ressourcen | Verbindung von Proxy Server | Anwendungs-Zugangsdaten in Konfigurationsdateien |
| Gegen-maßnahme | SeucrityContext, AppArmor, Seccomp, SELinux | regelmäßige Patches und Falco | Falco | RBAC, IAM-Richtlinien | IP-Adress-Einschränkung, Network Policies | Secret-Management-Dienste (KMS), |
| Bedrohung | exponierte sensible Schnittstellen | SSH-Server innerhalb des Containers | Manipuliere externalIP eines Service (CVE-2020-8554) | Umgehen einer Secret-Richtlinie des ServiceAccount Admission-Plugin (CVE-2024-3177) | Kubelet durch eine modifizierte Version ersetzen | Zugriff auf managed Identität Zugangsdaten |
| Gegen-maßnahme | Authentifizierung, TLS | Vermeiden und kubectl nutzen | NetzwerkPolicy, ServiceMesh, OPA | strikte Konfiguration und Audits, OPA | Audits | IAM-Richtlinien (Cloud Provider) |
| Bedrohung | | Sidecar Injection | Kubelet durch eine modifizierte Version ersetzen | Kubelet durch eine modifizierte Version ersetzen | Schwächen in der Konfiguration | Böswilliger AdmisionControl ler |
| Gegen-maßnahme | | Kontrolle und Audits mit Falco | Audits | Audits | Anwendung der Security Best Practices, Audits | vertrauenswürdige Quellen |
| Bedrohung | | Kubelet durch eine modifizierte Version ersetzen | | Schwächen in der Konfiguration führen zur Manipulation von Konfigurationsdateien | Schwächen in der Netzwerk-isolation | Umgehen einer Secret-Richtlinie des ServiceAccount Admission Plugin (CVE-2024-3177) |
| Gegen-maßnahme | | TLS-Verschlüsselung, RBAC | | Anwendung der Security Best Practices, Audits | Network Policies und Segmentierung | strikte Konfiguration und Audits, OPA |
| Bedrohung | | Schwachstellen in Add-ons oder Plugins | | | Schadcode innerhalb des Images tarnen | |
| Gegen-maßnahme | | regelmäßige Patches und Nutzung vermindern | | | Schwachstellen Scanning | |

Abbildung 6.1: Bedrohungsmatrix mit Gegenmaßnahmen der Security Best Practices und der Zero-Trust-Architektur. Grün gibt eine gänzliche Verhinderung an, gelb sind Möglichkeiten zur Verhinderung, ohne gänzliche Verhinderung. Rot stellen keine Gegenmaßnahmen dar. Blaue Bedrohungen zeigen einen Angriffsvektor, Orange Bedrohungen zeigen die aktuellen Schwachstellen, die in Kapitel 4 erarbeitet wurden. Fortsetzung in Abbildung 6.2

**d)** Schwachstelle einer Anwendung: Die Anwendung des SecurityContext und des Least Privilege-Prinzips kann verhindern, dass Schwachstellen in Anwendungen ausgenutzt werden, um Zugriff auf das Cluster zu erlangen. Diese Maßnahmen sorgen dafür, dass Container nur mit den minimal erforderlichen Berechtigungen ausgeführt werden und potenziell gefährliche Aktionen blockiert werden.

**e)** Ungeschützte sensitive Interfaces: Öffentliche Schnittstellen wie das Kubernetes Dashboard sollten mit TLS und starker Authentifizierung gesichert werden. Zudem sollte der Zugriff auf diese Interfaces regelmäßig überwacht und auf das Nötigste beschränkt werden.

**Execution** umfasst Techniken, die Angreifer nutzen, um schädlichen Code im Cluster auszuführen. Diese Techniken zielen darauf ab, Kontrolle über die Cluster-Ressourcen zu erlangen oder weitere Privilegien zu erweitern. Effektive Gegenmaßnahmen umfassen:

**a)** Exec in einen Container: Die Nutzung von kubectl exec sollte durch strikte RBAC-Richtlinien eingeschränkt werden, um zu verhindern, dass unautorisierte Benutzer oder Prozesse Shell-Zugriff auf Container erhalten.

**b)** bash und cmd innerhalb eines Containers: Container sollten so konfiguriert werden, dass unnötige Shells oder Befehlskonsole deaktiviert sind. Durch die Verwendung von SecurityContext kann sichergestellt werden, dass diese Umgebungen nur die minimal erforderlichen Befehle unterstützen.

**c)** Neuer Container: Die Bereitstellung neuer Container sollte durch die Verwendung von ImagePolicyWebhooks reguliert werden, die sicherstellen, dass nur sichere und genehmigte Konfigurationen verwendet werden.

**d)** Anwendungs-Exploit (RCE): Um Remote Code Execution zu verhindern, sollten Anwendungen innerhalb von Containern regelmäßig gepatcht und aktualisiert werden. Zusätzlich kann ein Intrusion Detection System (IDS) wie Falco verwendet werden, um verdächtige Aktivitäten zu erkennen.

**e)** SSH-Server innerhalb des Containers: Die Bereitstellung von SSH-Servern in Containern sollte vermieden werden. Stattdessen sollten sichere Kubernetes-Mechanismen wie kubectl exec verwendet werden, um die Angriffsfläche zu minimieren.

**f)** Sidecar Injektion: Die Verwendung von Sidecars sollte streng kontrolliert und auditiert werden. Sichere Richtlinien und strikte Kontrollen können helfen, das Risiko bösartiger Injektionen zu minimieren.

**g)** Kubelet durch eine modifizierte Version ersetzen: Der Zugriff auf Kubelet sollte sicher konfiguriert und regelmäßig überprüft werden. Verwenden Sie TLS-Verschlüsselung und RBAC, um den Zugriff zu kontrollieren und sicherzustellen, dass nur autorisierte Änderungen vorgenommen werden können.

**h)** Schwachstellen in Add-ons oder Plugins: Add-ons und Plugins sollten regelmäßig aktualisiert und überprüft werden. Die Nutzung solcher Erweiterungen sollte minimiert werden, um die Angriffsfläche zu reduzieren.

**Persistence** beschreibt Techniken, die es Angreifern ermöglichen, eine dauerhafte Präsenz im Cluster zu etablieren. Solche Techniken umfassen die Platzierung von Backdoor-Containern oder die Manipulation von Konfigurationen, um den langfristigen Zugriff zu sichern. Die folgenden Gegenmaßnahmen können die Gefahr dieser Angriffe minimieren:

**a)** Backdoor Container: Regelmäßige Überwachung und Scanning von Containern mit Tools wie Falco können helfen, Backdoor-Container zu identifizieren. Strikte Zugriffskontrollen und Überwachungsmechanismen sind ebenfalls notwendig. (RBAC)

**b)** Writable hostPath mount: Die Nutzung von hostPath-Mounts sollte streng kontrolliert und auf das notwendige Minimum beschränkt werden. Wenn hostPath verwendet wird, sollte es immer als read-only konfiguriert sein, um zu verhindern, dass Container das Host-Dateisystem manipulieren.

**c)** Kubernetes Cronjob: Cronjobs sollten durch RBAC kontrolliert und auditiert werden, um sicherzustellen, dass nur autorisierte Benutzer diese erstellen oder ändern können. Regelmäßige Überprüfungen und Scans können helfen, unerwünschte Änderungen zu erkennen.

**d)** Malicious Admission Controller: Admission Controller sollten sorgfältig konfiguriert und regelmäßig auf Integrität überprüft werden. Nur vertrauenswürdige Quellen sollten verwendet werden, und Audits sollten sicherstellen, dass keine bösartigen Controller im Cluster aktiv sind.

**e)** Manipuliere externalIP eines Service (CVE-2020-8554): Um diese Schwachstelle zu verhindern, sollten Cluster-Administratoren den Zugriff auf die API einschränken und sicherstellen, dass nur autorisierte Benutzer Änderungen an den Service-Spezifikationen vornehmen können. Network Policies und Service Mesh können ebenfalls helfen, den Datenverkehr zu sichern.

**f)** Kubelet durch modifizierte Version ersetzen: Das Ersetzen von Kubelet durch eine modifizierte Version kann durch regelmäßige Audits, Härtung des Betriebssystems und Überwachung der Integrität verhindert werden.

**Privilege Escalation** beschreibt Techniken, die Angreifer verwenden, um ihre Berechtigungen im Cluster zu erweitern. Diese Techniken nutzen oft Fehlkonfigurationen oder Schwachstellen aus, um erweiterte Zugriffsrechte zu erlangen. Gegenmaßnahmen umfassen:

**a)** Privilegierter Container: Die Bereitstellung privilegierter Container sollte strikt eingeschränkt und durch Pod Security Standard (PSS) verhindert werden. Nur spezifische Anwendungen, die unbedingt Root-Zugriff benötigen, sollten unter sorgfältiger Überwachung ausgeführt werden. Durch SecurityContext und Diensten wie Seccomp, AppArmor und SELinux können weitere Folgen verhindert werden.

**b)** Cluster-Admin Binding: Die Vergabe von Cluster-Admin-Rechten sollte minimiert und regelmäßig überprüft werden. RBAC-Richtlinien sollten sicherstellen, dass nur autorisierte Benutzer und Dienste solche Privilegien erhalten.

**c)** hostPath mount: Der Einsatz von hostPath-Mounts sollte streng kontrolliert und möglichst vermieden werden. Wenn hostPath verwendet wird, sollten Mounts als read-only und auf spezifische Verzeichnisse beschränkt sein.

**d)** Access Cloud Ressourcen: Zugriff auf Cloud-Ressourcen sollte durch strikte IAM- und RBAC-Richtlinien beschränkt werden, um sicherzustellen, dass nur autorisierte Benutzer und Anwendungen Zugriff auf sensible Ressourcen haben.

**e)** Umgehen einer Secret-Richtlinie des Serviceaccounts Admission Plugin (CVE-2024-3177): Diese Schwachstelle kann durch strikte Konfigurationen des Admission-Plugins und regelmäßige Überprüfungen von Richtlinien verhindert werden. Der Einsatz von OPA und externem Secret Management kann ebenfalls helfen.

**f)** Kubelet durch eine modifizierte Version ersetzen: Regelmäßige Audits und die Verwendung von Tools zur Integritätsüberwachung können unautorisierte Änderungen am Kubelet verhindern.

**g)** Schwächen in der Konfiguration führen zur Manipulation von Konfigurationsdateien: Regelmäßige Audits und die Verwendung von Konfigurationsmanagement-Tools können sicherstellen, dass alle Konfigurationen sicher und wie vorgesehen sind.

**Defense Evasion** umfasst Techniken, die darauf abzielen, Sicherheitsmaßnahmen zu umgehen oder zu deaktivieren, um unbemerkt zu bleiben. Diese Taktik nutzt oft Fehlkonfigurationen und Schwachstellen aus, um den Erkennungsmechanismen zu entgehen. Gegenmaßnahmen umfassen:

**a)** Container Logs löschen: Logs sollten extern gespeichert und auf sichere Weise verwaltet werden. Der Einsatz von Tools wie Falco zur Überwachung von Log-Löschereignissen kann helfen, Manipulationen zu erkennen.

**b)** Delete Kubernetes Events: Kubernetes-Events sollten sicher gespeichert und auf ungewöhnliche Löschvorgänge überwacht werden. Auditing-Tools, wie Falco, können dabei helfen, diese Aktivitäten zu protokollieren und zu erkennen.

**c)** Pod/Container name similarity: Die Vergabe eindeutiger und konsistenter Namen für Pods und Container kann helfen, Verwechslungen zu vermeiden und Angreifer daran zu hindern, legitime Ressourcen zu tarnen.

**d)** Verbindung von Proxyserver: Die Nutzung von Proxyservern sollte durch Network Policies und Überwachung auf unautorisierte Verbindungen eingeschränkt werden.

**e)** Kubelet durch eine modifizierte Version ersetzen: Regelmäßige Audits und die Verwendung von Tools zur Integritätsüberwachung können helfen, unautorisierte Änderungen am Kubelet zu verhindern.

**f)** Schwächen in der Konfiguration: Regelmäßige Audits und die Anwendung von Best Practices für Konfigurationsmanagement können helfen, Schwachstellen zu identifizieren und zu beheben.

**g)** Schwächen in der Netzwerkisolation: Netzwerkisolation sollte durch den Einsatz von Netzwerk Policies und die Segmentierung von Ressourcen gewährleistet werden, um die seitliche Bewegung innerhalb des Clusters zu verhindern.

**h)** Schadcode innerhalb des Images tarnen: Regelmäßiges Scannen von Container-Images mit Tools wie Snyk kann helfen, bösartigen Code zu identifizieren, bevor Images bereitgestellt werden.

**Credential Access** beschreibt Techniken, die Angreifer nutzen, um Zugangsdaten zu stehlen oder zu verwenden, um Zugriff auf Cluster-Ressourcen zu erhalten. Gegenmaßnahmen umfassen:

**a)** Kubernetes Secrets auflisten: Der Zugriff auf Secrets sollte durch strikte RBAC-Richtlinien und Verschlüsselung eingeschränkt werden, um sicherzustellen, dass nur autorisierte Benutzer diese auflisten können.

**b)** Mount Service Prinzipal: Service Prinzipals sollten sicher gespeichert und nur für autorisierte Anwendungen verfügbar gemacht werden. IAM- und RBAC-Richtlinien sollten die Nutzung von Service Prinzipals kontrollieren.

**c)** Zugriff auf Container Service Account: Der Zugriff auf Service Accounts sollte minimiert und nur für autorisierte Anwendungen zugelassen werden. Jeder Pod sollte einen eigenen Service-Account haben. Secrets sollten immer sicher gespeichert und nicht in Containern eingebettet werden.

**d)** Anwendungs-Zugangsdaten in Konfigurationsdateien: Anwendungs-Zugangsdaten sollten nie im Klartext in Konfigurationsdateien gespeichert werden. Secret-Management-Dienste oder verschlüsselte Variablen sollten verwendet werden.

**e)** Zugriff auf managed Identität Zugangsdaten: Managed Identitäten sollten sicher verwaltet und der Zugriff auf sie durch strikte IAM-Richtlinien reguliert werden, um den Missbrauch zu verhindern.

**f)** Böswilliger AdmissionController: Admission Controller sollten nur von vertrauenswürdigen Quellen geladen und regelmäßig auf Integrität überprüft werden.

**g)** Umgehen einer Secret-Richtlinie des Serviceaccounts Admission Plugin (CVE-2024-3177): Diese Schwachstelle kann durch strikte Konfigurationen des Admission Plugins und regelmäßige Überprüfungen von Richtlinien verhindert werden.

**Discovery** beschreibt Techniken, die Angreifer verwenden, um Informationen über das Cluster und seine Konfiguration zu sammeln. Gegenmaßnahmen umfassen:

**a)** Zugriff auf Kubernetes API-Server: Der Zugriff auf den API-Server sollte durch Network Policies, RBAC-Richtlinien und starke Authentifizierung geschützt werden, um sicherzustellen, dass nur autorisierte Benutzer Zugriff haben.

**b)** Zugriff auf Kubelet API: Der Zugriff auf Kubelet sollte auf autorisierte Benutzer und Anwendungen beschränkt werden. Eine strikte Konfiguration und regelmäßige Audits können dazu beitragen, den Missbrauch der Kubelet-API zu verhindern.

**c)** Netzwerk Mapping: Netzwerk-Mapping-Techniken können durch die Verwendung von Netzwerk Policies und segmentierten Netzwerken eingeschränkt werden.

**d)** Zugriff auf das Kubernetes Dashboard: Das Dashboard sollte durch starke Authentifizierung und TLS gesichert werden.

**e)** Instance Metadata API: Der Zugriff auf die Instance Metadata API sollte durch Netzwerkpolicies eingeschränkt werden.

**Exfiltration** umfasst Techniken, die Angreifer nutzen, um Daten aus dem Cluster herauszuschmuggeln. Gegenmaßnahmen umfassen:

**a)** Unverschlüsselte Daten der Discovery-Phase aus dem Netzwerk übertragen: Alle Datenübertragungen sollten durch TLS oder andere sichere Protokolle verschlüsselt werden.

**Lateral Movement** beschreibt Techniken, die Angreifer verwenden, um sich seitlich innerhalb des Clusters zu bewegen. Gegenmaßnahmen umfassen:

**a)** Zugriff auf Cloud-Ressourcen: Der Zugriff auf Cloud-Ressourcen sollte durch strikte IAM- und RBAC-Richtlinien eingeschränkt werden.

**b)** Container Service Account: Der Zugriff auf Service Accounts sollte minimiert und regelmäßig überprüft werden.

**c)** Cluster-internes Networking: Netzwerk Policies sollten so konfiguriert werden, dass der interne Datenverkehr auf das Nötigste beschränkt wird.

**d)** Anwendungs-Zugangsdaten in Konfigurationsdateien: Zugangsdaten sollten sicher gespeichert und durch Verschlüsselung geschützt werden.

**e)** Schreibbare Volume-mounts auf dem Host: Schreibbare Volume-mounts sollten vermieden oder auf das Nötigste beschränkt werden.

**f)** CoreDNS Poisining: Die CoreDNS-Konfiguration sollte regelmäßig überprüft und gehärtet werden.

**g)** ARP poisining und IP spoofing: Netzwerk-Sicherheitsmechanismen wie ARP-Spoofing-Schutz und IP-Spoofing-Erkennung sollten implementiert werden.

**h)** Netzwerkregeln manipulieren: Netzwerkregeln sollten regelmäßig überprüft und überwacht werden.

**i)** Nutzung einer Schwachstelle innerhalb des Containers: Container sollten regelmäßig gepatcht und auf Schwachstellen überprüft werden.

**Collection** bezieht sich auf Techniken, die Angreifer nutzen, um Informationen oder Ressourcen innerhalb des Clusters zu sammeln, die ihnen helfen, ihre Angriffe weiter auszubauen. Gegenmaßnahmen umfassen:

**a)** Images einer privaten Registry: Der Zugriff auf private Registries sollte durch strikte IAM- und RBAC-Richtlinien eingeschränkt werden.

**b)** manipuliere externalIP eines Service (CVE-2020-8554): Um diese Schwachstelle zu verhindern, sollten Cluster-Administratoren den Zugriff auf die API einschränken und sicherstellen, dass nur autorisierte Benutzer Service-Spezifikationen ändern können.

**Impact** beschreibt Techniken, die Angreifer nutzen, um direkten Schaden im Cluster zu verursachen, sei es durch Datenmanipulation, Denial of Service oder andere destruktive Aktionen. Gegenmaßnahmen umfassen:

**a)** Data Destruction: Um die Zerstörung von Daten zu verhindern, sollten regelmäßige Backups erstellt und sicher aufbewahrt werden. Zudem sollten Zugriffe auf Daten durch strikte RBAC-Richtlinien kontrolliert werden.

**b)** Ressource Hijacking: Ressourcenmissbrauch kann durch Überwachung des Ressourcenverbrauchs und Implementierung von Quoten verhindert werden. Tools wie Prometheus und Grafana können helfen, ungewöhnliche Aktivitäten zu erkennen.

**c)** Denial of Service: Um DoS-Angriffe zu verhindern, sollten Rate-Limiting und Network Policies implementiert werden, die den Zugriff auf Cluster-Ressourcen regulieren.

**d)** manipuliere externalIP eines Service (CVE-2020-8554): Um das Ausnutzen dieser Schwachstelle zu verhindern, sollten Cluster-Administratoren den Zugriff auf die API einschränken.

**e)** Physischer Zugriff auf den Node: Physischer Zugang zu Nodes sollte durch physische Sicherheitsmaßnahmen wie Schlösser und Zugangskontrollen beschränkt werden, um den unautorisierten Zugriff auf Cluster-Ressourcen zu verhindern. Entsprechend lässt sich dies nicht durch Security Best practices und die Zero-Trust-Architektur verhindern.

Die beschriebenen Gegenmaßnahmen bieten einen umfassenden Schutz gegen verschiedene Bedrohungstaktiken, die Angreifer nutzen könnten, um ein Kubernetes-Cluster zu kompromittieren. Es ist entscheidend, diese Maßnahmen regelmäßig zu überprüfen und zu aktualisieren, um sicherzustellen, dass sie den aktuellen Bedrohungen und Best Practices entsprechen. Die Verhinderung von Bedrohungen in Kubernetes-Clustern erfordert eine Kombination aus mehreren Sicherheitsmaßnahmen und Best Practices. Zero-Trust-Ansätze, die Network Policies, Service Mesh, Open Policy Agent (OPA), und Authorization Policies umfassen, spielen eine unterstützende Rolle, sind jedoch nicht die Hauptverteidigung gegen die meisten Bedrohungen. Obwohl Zero-Trust-Mechanismen wie Network Policies und Service Mesh wichtige Funktionen zur Netzwerksegmentierung und sicheren Kommunikation bieten, verhindern sie nicht direkt Bedrohungen wie kompromittierte Images, unverschlüsselte Datenübertragungen oder das Ausnutzen von Schwachstellen in Anwendungen. Entscheidend sind Maßnahmen wie Role-based Access Control (RBAC), strikte Zugriffskontrollen, sichere

Konfigurationen und regelmäßige Audits, um den anfänglichen Zugang, die Privilegien-Eskalation und die Persistenz von Angreifern zu verhindern. Darüber hinaus spielen Tools zur Sicherheitsüberwachung und Laufzeitüberprüfung wie Falco sowie Image-Scanning-Tools wie Snyk und Clair eine wesentliche Rolle bei der Erkennung und Vermeidung von Bedrohungen. Ein umfassender Sicherheitsansatz, der über Zero-Trust hinausgeht und mehrere Schichten der Abwehr integriert, ist notwendig, um Kubernetes-Umgebungen effektiv zu schützen.

Diese Einschätzungen lassen sich auch in der Heatmap in Abbildung 6.3 darstellen. Die Best Practices wurden auf die in Blau dargestellten Bestandteile des Clusters angewendet, während die orangefarbenen Bereiche die Zero-Trust-Mechanismen repräsentieren. Die Heatmap zeigt, dass Zero-Trust-Mechanismen am besten in der Netzwerksegmentierung und sicheren Kommunikation eingesetzt werden. Es braucht aber für die Sicherung des Clusters einen Sicherheitsansatz, der über Zero-Trust hinausgeht und mehrere Schichten der Abwehr integriert, um Kubernetes Umgebungen effektiv zu schützen.

## 6.3 Ressourcenverbrauch und wirtschaftliche Auswirkungen

Die Einführung eines Service Mesh, Open Policy Agent (OPA) und Monitoring-Tools hat einen Einfluss auf den Ressourcenverbrauch in Kubernetes-Clustern [iste]. In diesem Kapitel wird untersucht, wie sich diese Komponenten auf den Speicher- und CPU-Verbrauch in Minikube- und Amazon EKS-Clustern auswirken. Zudem werden die wirtschaftlichen Auswirkungen auf das Amazon EKS-Cluster analysiert.

### 6.3.1 Ressourcenverbrauch

Um den Ressourcenverbrauch der Zero-Trust-Architektur zu bestimmen, wurden mehrere Schritte durchgeführt:

1. Initiale Installation (nur Management-Ebene): Das Cluster wurde ohne zusätzliche Anwendungen oder Service Mesh-Komponenten aufgesetzt, um die Basiswerte für Speicher und CPU-Verbrauch zu ermitteln.

2. Installation der Bookinfo-Anwendung ohne Service Mesh: Die Bookinfo-Anwendung wird bereitgestellt, um den Ressourcenverbrauch ohne zusätzliche Sicherheitsmaßnahmen zu messen.

3. Installation des Istio Service Mesh: Die Management-Ebene des Service Mesh wurde installiert, um die Auswirkungen auf die Management-Ebene zu analysieren.

4. Hinzufügen der Istio-Sidecar-Container: Die Istio-Sidecars wurden den Anwendungs-Pods hinzugefügt, um den Einfluss auf den Ressourcenverbrauch zu messen.

5. Hinzufügen notwendiger Policies: Sicherheitsrichtlinien, wie Netzwork Policies, Authoriziation Policies und mTLS-Konfigurationen wurden angewendet, um die Auswirkungen auf den Ressourcenverbrauch zu bewerten.

6. Installation des Open Policy Agent: Der OPA wurde hinzugefügt, um die Sicherheitsfunktionen zu erweitern.

7. Hinzufügen der OPA-Sidecar-Container: Diese Sidecars wurden zur Durchsetzung von Sicherheitsrichtlinien implementiert.

8. Integration der Monitoring-Tools (Grafana, Prometheus, Kiali): Diese Tools wurden integriert, um den Ressourcenverbrauch zu überwachen und zu visualisieren.

#### Ermittlung der Werte

Die Messungen des Speicher- und CPU-Verbrauchs werden mithilfe des Kubernetes metrics-server durchgeführt. Dieser Server sammelt Metriken von den Knoten, Pods und Containern des Clusters und stellt sie für die Auswertung zur Verfügung. Der metrics-server ermöglicht eine Echtzeitüberwachung des Ressourcenverbrauchs, was für die genaue Analyse der Auswirkungen jeder neuen Komponente und Konfiguration entscheidend war.

#### Rollen der einzelnen Namespaces:

`default` Namespace: Enthält die Hauptanwendung (Bookinfo-Anwendung). Alle Pods und Services, die nicht explizit einem anderen Namespace zugewiesen wurden, werden standardmäßig hier platziert.

`kube-system` Namespace: Beinhaltet die Management-Ebene des Kubernetes-Clusters, einschließlich der Kernkomponenten wie der API-Server, Controller-Manager und Scheduler.

`istio-system` Namespace: Enthält die Komponenten des Istio Service Mesh, einschließlich der Istio Control Plane, die für die Verwaltung des Service Meshes verantwortlich ist. Zusätzlich enthält dieser Namespace auch die Monitoring Tools Grafana, Kiali und Prometheus.

`istio-operator` Namespace: Dieser Namespace enthält den Istio Operator, der die Installation und Verwaltung des Istio Service Meshes automatisiert.

`opa-istio` Namespace: Beinhaltet den Admission Controller des Open Policy Agent (OPA), der für die Durchsetzung der Sicherheitsrichtlinien innerhalb des Service Meshes zuständig ist.

`karpenter` Namespace: Enthält das von AWS bereitgestellte Tool Karpenter, das für die automatische Skalierung und Optimierung der Ressourcennutzung im EKS-Cluster verwendet wird.

#### Analyse des Speicherverbrauchs:

**Minikube (Abbildung 6.4):** Der Speicherverbrauch steigt mit jedem Schritt kontinuierlich an. Vor der Installation des Service Meshes liegt der Verbrauch bei etwa 500 MiB. Nach der Installation der Bookinfo-Anwendung erhöht sich der Verbrauch auf etwa 750 MiB. Das Hinzufügen des Istio Service Meshes führt zu einem weiteren Anstieg um etwa 100 MiB für die Istio Control Plane und 200 MiB für die Management-Ebene. Die Integration der Istio-Sidecars steigert den Verbrauch um weitere 170 MiB, da die Sidecar-Container pro Pod zusätzlichen Speicherplatz beanspruchen. Die Anwendung der Sicherheitsrichtlinien und die Installation des Open Policy Agenten zeigen einen geringfügigen Einfluss auf den Speicherverbrauch, der minimal um etwa 30 MiB anstieg. Nach dem Hinzufügen der OPA-Sidecars und der Monitoring-Tools steigt der Gesamtverbrauch des Clusters auf mehr als das Doppelte des Ausgangswerts. Die vollständigen Zahlen zum Speicherverbrauch pro Namespace und Schritt sind in Tabelle A.3 dargestellt.

**Amazon EKS (Abbildung 6.5):** Der Speicherverbrauch im Amazon EKS-Cluster zeigt ein ähnliches Muster wie bei Minikube, jedoch mit einigen Unterschieden. Zu Beginn liegt der Speicherverbrauch bei etwas über 250 MiB. Nach der Installation der Bookinfo-Anwendung steigt der Verbrauch auf über 600 MiB. Die Installation des Istio Service Meshes und der Sidecar-Proxies hatte nur geringe Auswirkungen auf den Speicherverbrauch der Management- und Anwendungsebenen. Erst die Integration der Monitoring-Tools führt zu einem deutlichen Anstieg des Speicherverbrauchs auf mehr als das Fünffache des Ausgangswertes. Die detaillierten Daten zum Speicherverbrauch in EKS sind ebenfalls in Tabelle A.5 zusammengefasst.

#### Analyse des CPU-Verbrauchs:

**Minikube (Abbildung 6.6):** Der CPU-Verbrauch bleibt bis zur Integration der Monitoring-Tools relativ konstant, zwischen 90 und 140 Millicores. Nach der Installation der Monitoring-Tools steigt der Verbrauch jedoch auf etwa 118 Millicores. Der höhere Verbrauch ist auf den

zusätzlichen Aufwand des API-Servers zurückzuführen, der die Kommunikation verwaltet. Die vollständigen CPU-Verbrauchsdaten für jeden Schritt sind in Tabelle A.4 aufgeführt.

**Amazon EKS (Abbildung 6.7):** Der CPU-Verbrauch beginnt bei etwa 33 Millicores und steigt nach der Installation der Monitoring-Tools auf 504 Millicores. Dies zeigt, dass das EKS-Cluster nach der Integration der Zero-Trust-Architektur und der Monitoring-Tools etwa dreimal so viele CPU-Ressourcen verbraucht wie zu Beginn. Die Unterschiede im Verbrauch zwischen Minikube und EKS sind hauptsächlich auf die größere Skalierung und die zusätzlichen Verwaltungsaufgaben in EKS zurückzuführen. Die detaillierten Zahlen zum CPU-Verbrauch sind in Tabelle A.6 dargestellt.

**Vergleich und Auswertung der Node-Verbräuche:**

**Minikube-Node (Abbildung 6.8):** Der Speicherverbrauch des Minikube-Nodes lag nur leicht über dem kumulierten Verbrauch der Namespaces. Der Node verbrauchte etwa 500 MiB mehr als die Summe der einzelnen Namespaces. Der CPU-Verbrauch des Nodes lag ebenfalls geringfügig höher als der kumulierte Verbrauch der Namespaces, was auf die zusätzliche Verwaltungsaufgaben des Nodes zurückzuführen ist. Nach der Integration der Monitoring-Tools stieg der CPU-Verbrauch des Nodes auf etwa 300 Millicores, was den erhöhten Verwaltungsaufwand widerspiegelt. Die detaillierten Verbräuche der Nodes sind in den Tabellen A.3 und A.5 aufgeführt.

**EKS-Nodes (Abbildung 6.9):** Im EKS-Cluster zeigte sich ein ähnliches Muster. Der Speicherverbrauch der Nodes lag konstant etwas über dem Verbrauch der Namespaces und stieg nach der Integration der Zero-Trust-Architektur. Der CPU-Verbrauch der beiden Nodes glättete die Schwankungen, die bei den einzelnen Namespace-Verbräuchen zu beobachten waren, und blieb nach der Integration der Monitoring-Tools leicht erhöht. Der EKS-Cluster zeigte insgesamt eine höhere Konsistenz im Speicher- und CPU-Verbrauch im Vergleich zu Minikube, was auf die zusätzliche Verwaltungsebene und Skalierbarkeit von EKS zurückzuführen ist. Die genauen Werte für die Nodes sind ebenfalls in den Tabelle A.3 und A.5 zu finden.

Zusammenfassend zeigt die Analyse, dass die Einführung von Zero-Trust-Komponenten wie Istio und OPA sowie die Integration von Monitoring-Tools zu einem signifikanten Anstieg des Speicher- und CPU-Verbrauchs in Kubernetes-Clustern führen. Diese Erhöhungen sind besonders in produktionsähnlichen Umgebungen wie Amazon EKS bemerkbar.

### 6.3.2 Istio Benchmarking

Istio selbst erstellt Benchmarks zu Geschwindigkeit und Skalierbarkeit. Für die in dieser Arbeit verwendete Version, 1.21. werden die folgenden Daten zur Verfügung gestellt. Da das Istio Load Test Service Mesh aus 1000 Services und 2000 Sidecars besteht und mit 70.000 Anfragen pro Sekunde getestet wird, erscheinen die Daten durchaus nachvollziehbar und lassen sich im Rahmen dieser Arbeit als solche auswerten. [iste]

Gerade hinsichtlich des CPU- und Arbeitsspeicherverbrauchs lassen sich die ersten Gemeinsamkeiten finden, denn die Werte, die der Istio Sidecar Proxy für 1000 Requests angeben liegen bei 0.5 vCPU pro 1000 Requests. Der durchschnittliche Verbrauch von Arbeitsspeicher des Proxies liegt bei etwa 50 MB. Werte, die sich auch aus der vorherigen Auswertung entnehmen lassen. Requests haben aufgrund des nicht vorhandenen Puffers auf dem Sidecar keine Auswirkungen auf den Arbeitsspeicher, auch dies konnte lokal nachgestellt werden. [iste]

Hinsichtlich der Latenz, die die Istio-Sidecar-Container mitbringen, gibt Istio etwa eine Verzögerung um 0,182 Milicores und 0,248 Milicores im 90 und 99. Perzentil der Latenz für 2 bis 64 Verbindungen an. Gesendet wurden 1kB Payload bei 1000 Anfragen pro Sekunde und mTLS war aktiviert. [iste]

### 6.3.3 Wirtschaftliche Auswirkungen

Aus den gewonnenen Erkenntnissen der Bestimmung des Ressourcenverbrauchs und des Istio Benchmarkings lässt sich einiges für die Wirtschaftlichkeit des Clusters aussagen. Um einen Einblick in die Auswirkungen zu geben, wird in diesem Fall das Amazon EKS-Cluster und ein Kubernetes Cluster auf Hetzner verglichen. Ziel ist es herauszustellen, was das Cluster vor der Integration der Zero-Trust-Architektur und was es anschließend kostet. Dabei bezieht sich die Auswertung auf den Verbrauch der Nodes des EKS Clusters. Die Umrechnung der CPU-Werte ist dabei wie folgt: 1000 m (milicores) sind 1 Kern oder 1 vCPU. Zum Betrieb der Bookinfo-Anwendung verbrauchte das EKS Cluster etwa 680 und 930 MiBytes je Knoten. Bei der CPU verbrauchten die Nodes 45 und 57 Milicores. Die Nodes, die für die Ausführung der Anwendung ausreichend gewesen wären, sind micro Instanzen, deren Preis bei 0,0096 USD (t4g.micro) und 0,116 USD (t2.micro) pro Stunde liegt. t2.micro stellt 1 vCPU und 1 GiB RAM zur Verfügungh, t3.micro 2 vCPU und 1 GiB RAM, ebenso wie t3a.micro und t4g.micro.

Nach Integration der Zero-Trust-Architektur verbrauchen die Nodes 1500 und 1290 MiBytes Arbeitsspeicher. Der CPU-Verbrauch bleibt mit 115 und 121 Milicores dabei jedoch gering. Das Benchmarking stellt jedoch heraus, dass nochmal 0.5 vCPU also 500 Milicores pro 1000 anfragen benötigt werden, sodass auch in diesem Fall insgesamt 1 vCPU durchaus

| Instanzname | vCPU | Arbeitsspeicher (GiB) | Stundensatz (USD) | Monatskosten (730 Std.) (USD) |
|---|---|---|---|---|
| *vor Integration ZTA* | | | | |
| t4g.micro | 2 | 1 | 0,0096 | 7,01 |
| t3.micro | 2 | 1 | 0,012 | 8,76 |
| t3a.micro | 2 | 1 | 0,0108 | 7,884 |
| t2.micro | 1 | 1 | 0,0134 | 9.782 |
| *nach Integration ZTA* | | | | |
| t4g.small | 2 | 2 | 0,0192 | 14,016 |
| t3.small | 2 | 2 | 0,024 | 17,52 |
| t3a.small | 2 | 2 | 0,0216 | 15,768 |
| t2.small | 1 | 2 | 0,0268 | 19,564 |

Tabelle 6.1: Kosten der AWS Instanzen für die Bereitstellung der Anwendung ohne Zero-Trust-Architektur (oben) und mit Zero-Trust-Architektur (unten)

ausreichen dürfte. Lediglich der Arbeitsspeicher muss aufgrund des Ressourcenbedarfs auf 2 GiB erhöht werden. Hierfür kommen dann die Instanzen t4g.small, t3.small, t3a.small und t2.small infrage. Die Kosten für die EC2-Instanzen sind übersichltich in der Tabelle 6.1 dargestellt. In der oberen Hälfte finden sich die Kosten für die EC2-Instanzen vor der Integration der Zero-Trust-Architektur. In der unteren Hälfte sind die Kosten für die EC2-Instanzen nach der Integration der Zero-Trust-Architektur dargestellt.

Tabelle 6.1 zeigt ganz übersichtlich, dass sich die Kosten für das EKS Cluster verdoppelt haben. Entsprechend sollte ein Unternehmen, dass Zero-Trust im Unternehmen mit Monitoring wie Prometheus, Kiali und Grafana in Amazon EKS einsetzen möchte, mit den doppelten Kosten rechnen. Zur Vergleichbarkeit wird nun noch der Public Cloud Provider Hetzner in den Vergleich aufgenommen. Hetzner bietet keinen verwalteten Kubernetes Service an. Kubernetes lässt sich bei Hetzner jedoch über mehrere Cloud-Server-Instanzen einsetzen, was eine Vergleichbarkeit zum EKS-Cluster ermöglicht. Hetzner bietet shared vcpus, die in der kleinsten möglichen Konfiguration bereits 2 vCPUs (2000 Milicores und 4 GB RAM) bereitstellen. Mit Blick auf die zuvor gesammelten Verbräuche ist bereits diese Instanz ausreichend, um das Cluster zu unterstützen. Die Kosten hierfür betragen 0,006 € pro Stunde und 3,92 € im Monat. Bei Hetzern würde somit durch die Größe der kleinstmöglichen Server-Instanz keine große wirtschaftliche Auswirklung entstehen und man würde selbst mit 3 Knoten (2 Worker, 1 Arbeiter Knoten) hinter dem Preis von AWS liegen. Die Gesamtkosten für EKS-Cluster + 2 EC2 Instanzen und 3 Hetzner CX22-Server sind in Tabelle 6.2 dargestellt. Damit wäre Hetzner auch mit Zero-Trust-Architektur fast 1/10 günstiger als das Amazon EKS Cluster.

Zusammenfassend lässt sich sagen, dass durch die Integration der Zero-Trust-Architektur und der damit verbundenen gestiegenen Verbräuche größere Instanztypen notwendig werden, die je nach Cloud Provider unterschiedlich wirtschaftlich zu bewerten sind. Der Vergleich mit AWS EKS und Hetzner hat gezeigt, dass bei AWS fast mit den doppelten Kosten zu rechnen ist. Bei Hetzner hingegen würden drei Server des Typs CX22 in diesem Vergleich ausreichen.

| AWS | | | Hetzner | |
|---|---|---|---|---|
| Kosten EKS (USD) | Kosten EC2 Instanz (USD) | **Kosten AWS Gesamt (USD)** | Kosten Hetzner CX22 (EUR) | **Gesamtkosten Hetzner 3x CX22 (EUR)** |
| *vor Integration ZTA* | | | | |
| 73 | 7,01 | **87,02** | 3,92 | **11,76** |
| *nach Integration ZTA* | | | | |
| 73 | 14,016 | **101,032** | 3,92 | **11,76** |

Tabelle 6.2: Vergleich der Preise für Amazon EKS Cluster mit 2 EC2 Instanzen (t4g.micro und t4g.small) und selbst gehostetem Kubernetes Cluster bei Hetzner mit drei CX22 Instanzen (zwei Worker, ein Master), um ein vergleichbares Bild zu schaffen. Vergleich der Kosten vor und nach der Integration der Zero-Trust-Architektur.

Natürlich sollten Instanzgrößen sowohl im EKS-Cluster als auch bei der Integration von Kubernetes auf Servern von z.B. Hetzner ausreichend groß gewählt werden. Somit kann es sein, dass die Kosten bei der Betrachtung des Clusters gar nicht doppelt so hoch ausfallen, da die Instanzgröße bereits zuvor ausreichend groß dimensioniert wurde.

## 6.4 Herausforderungen und Risiken der Zero-Trust-Architektur

Durch die Integration der Zero-Trust-Architektur entstehen neue Sicherheitsrisiken und Probleme, die im Folgenden betrachtet werden.

### Angriffsvektoren

Aufseiten der Angriffsvektoren entstehen durch die Nutzung des Service Meshs Istio eine Reihe offener Ports, für die einschränkende Maßnahmen ergriffen werden sollten. So öffnet istio auf der Control Plane die Ports 8080 und den Port 15010.
Port 8080 stellt ein Debug-Interface zur Verfügung. Dieses Interface bietet Lese-Zugriff auf eine Fülle an Informationen und Details über den Cluster-Status. Viele istioctl-Befehle hängen jedoch von diesem Interface ab und werden nach Deaktivierung nicht mehr funktionieren. Aus diesem Grund sollte der Port nach abgeschlossener Konfiguration und Erstellung des Clusters diese Schnittstelle geschlossen werden. Die Umgebungsvariable `ENABLE_DEBUG_ON_HTTP=false` muss hierfür auf dem istiod gesetzt werden.
Port 15010 liefert einen XDS-Service aus. Der ebenfalls über ein Kennzeichen –grpcAddr= auf dem istiod geschlossen werden kann. [istf]

Auf der Istio Data Plane stellen vor allem die Proxies Ports zur Verfügung, die über localhost freigegeben werden. Hierzu zählen: 15099 (Telemetrie), 15021 (Health Check und die Ports 15020 und 15000, die als Debugging Endpunkte dienen. Da diese nur über localhost

ausgeliefert werden, sind diese nur im gleichen Pod zugänglich, in denen auch der Proxy verortet ist. Es gibt jedoch keine Vertrauensgrenze zwischen Proxy und Anwendung.

Grafana, Prometheus und Kiali stellen, sofern deren Dashboards genutzt werden, ebenfalls einige Ports zur Verfügung, die dann ebenfalls als Angriffsvektor ins Cluster genutzt werden können. Grafana wird über Port 3000 bereitgestellt. Prometheus wird hingegen über Port 9090 und Kiali über 20001 bereitgestellt. Da Prometheus von den Anwendungen verlangt einen metrics Pfad und Port freizugeben, sind dies weitere interne Angriffsvektoren, die gegenüber der Anwendungen entstehen, wenn diese Metrik-Schnittstellen nicht ausreichend abgekapselt werden.

Die Schnittstellen für Grafana, Prometheus und Kiali lassen sich mit Authentifizierung und TLS-Authentifizierung sichern.

#### Schwachstellen der Zero-Trust-Architektur

Neue Bestandteile und Add-ons bieten natürlich auch neue Schwachstellen. Ein Blick auf Schwachstellen der Vergangenheit gibt einen Überblick über die Bedrohungslage der einzelnen Komponenten. Gerade im Istio Service Mesh wurden in der Vergangenheit 19 Schwachstellen gefunden, die von der Identitätsnachahmung über Umgehungen von Berechtigungen bis hin zum Zugriff auf Zugangsdaten reichen. [cvee]

Der Open Policy Agent war in der Vergangenheit nur 3 Schwachstellen ausgeliefert. Sowohl CVE-2022-23628, CVE-2022-33082 als auch CVE-2022-36085 beziehen sich jeweils auf die Übersetzung der Rego-Sprache innerhalb des Gatekeepers. [cveg]

#### Nachteile und Probleme

Die Integration eines Service Meshs hat neben Vorteilen für die Sicherheit auch Nachteile, die im Folgenden beschrieben werden.

Gerade die Implementierung von Istio ist keine leichte Aufgabe. Die Einrichtung und Verwaltung von Istio kann erhebliche interne Fähigkeiten erfordern und ist daher eher ein Einsatzfeld von größeren Unternehmen. [Iye]
Der Einsatz von mTLS sichert wie beschrieben den Ost-West-Verkehr des Clusters, das eine transparente, bidirektionale Verschlüsselung von Dienst zu Dienst mit Public-key-Zertifikaten bietet. Diese Zertifikate fungieren als Maschinenidentitäten und bieten eine gegenseitige Authentifiziertung. Es wird also sichergestellt, dass die beiden Parteien, die sich verbinden, die sind, für die sie sich ausgeben, damit sie sicher miteinander kommunizieren können. [Iye]

Wie in den Beispielen mit Minikube und EKS zu sehen, wird der Entwickler in den Prozess der Zertifikat-Erstellung nicht eingebunden und muss sich somit nicht um die Verwaltung der Maschinenidentitäten kümmern. Allerdings liegt hier eins der großen Probleme, denn Maschinenidentitäten als Authentifizierungssystem erfordern ein gewisses Maß an Kontrolle, um effektiv zu sein. Um sofort einsatzbereit zu sein unterstützt Istio jedoch nur selbstsignierte Maschinenidentitäten. Das spart zwar Zeit und Geld, setzt Untenehmen aber auch einem weiteren Sicherheitsrisiko aus. [Iye]
Die Kontrolle die das Unternehmen zuvor mit Einrichtung von Istio im Cluster erlangen wollte, wird nun durch Istio aufgegeben. Die Sichtbarkeit wird ebenso verrringert, da selbstsignierte Maschinenidentitäten außerhalb der bestehenden Infrastruktur für die Verwaltung von Maschinenidentitäten liegen. Jedoch sind dies beides Punkte, die von Sicherheitsteams gefordert werden. [Iye]
Die Zertifikate, die Istio nutzt sind also nicht von einer öffentlichen Certificate Authority (CA) signiert worden, noch können sie widerrufen werden oder ablaufen. Was sich wie ein Vorteil anhört ist ein Problem, wenn der private Schlüssel kompromittiert ist. Denn dann führt die Unfähigkeit ihn zu widerufen dazu, dass Angreifer weiteren Zugriff auf das Cluster erlangen. [Iye]
Unklar bleibt also auch, welche Identitäten ausgestellt wurden, ob sie von einer vertrauenswürdigen Zertifizierungsstelle ausgestellt wurden, und in welchem Kontext sie erstellt wurde und verwendet wird.
Zur Behebung der Risiken, setzen viele Istio-Kunden ihre bewährte Zertifizierungsstellen und Infrastrukturen ein, um ihre eigenen Zertifikate auszustellen. Allerdings ist die Verwaltung dieser Identitäten kaum manuell zu verwalten und die kurzzeitige Gültigkeit der Zertifikate verbessert die Situation nicht. Also wurden vermehrt länger Lebensdauern und Wildcard-zertifikate eingesetzt.
Längere Lebensdauer führt jedoch dazu, dass das Zertifikat als weniger sicher eingestuft wird und Wildcard-Zertifikate reduzieren zunächst einmal die Kosten und erhöhen die Flexibilität. Doch wenn genau dieses Zertifikat kompromittiert wurde, droht der single Point of Failure für das gesamte Cluster.
Diese Praktiken führen nun also dazu, das Entwickler nicht in der Lage sind, dynamisch neue Arbeitslasten und Dienste zu Anwendungen hinzufügen können. Im schlechtesten Fall können schlecht verwaltete Identitäten zu Ausfällen führen, da ein Ablaufdatum scheinbar aus heiterem Himmel eintritt und ganze Anwendungen und Dienste vom Netz genommen werden.
Abhilfe kann durch eine externe Zertifikatsverwaltung Vault, wie sie im Rahmen dieser Arbeit vorgestellt wurde, geschaffen werden um X.509 (TLS)-Zertifikate bereitzustellen. Mithilfe von cert-manager, der Konfiguration des Issuers und des CSR können die Zertifikate anschließend über Vault verwaltet werden. Der Prozess bleibt hierbei derselbe, wie er in Kapitel 5.2.1 dargestellt wurde.

Die Einführung von Zero-Trust bringt auch einige Nachteilte mit sich. Ein offensichtlicher Nachteil von Zero-Trust ist der erhöhte Aufwand für das Management von Anwendungen,

Geräten und Benutzern. Jedes Benutzerkonto und jede Anwendung muss individuell registriert, überwacht und verwaltet werden. Dies kann insbesondere bei einer großen Anzahl an Nutzern, Kunden, Drittanbietern und Anwendungen einen erheblichen Arbeitsaufwand darstellen. [Hew]
Durch das Service Mesh und den Open Policy Agenten und der weiteren Security Best Practices, die als Grundlage für Zero-Trust in Kubernetes Clustern umgesetzt wurden, steigt das Datenaufkommen und der Ressourcenverbrauch. Unter anderem auch deswegen, weil mehr Verbindungen entstehen. Die Sicherung dieser Verbindungen ist eine der größten Herausforderungen. [Hew]

| Taktiken / Techniken | Discovery | Exfiltration | Lateral Movement | Collection | Impact |
|---|---|---|---|---|---|
| Bedrohung | Zugriff auf Kubernetes API-Server | Unverschlüsselte Daten der Discovery-Phase aus dem Netzwerk übertragen | Zugriff auf Cloud-Ressourcen | Images einer privaten Registry | Data Desctruction |
| Gegen-maßnahme | Network Policies, RBAC, Authentifizierung | TLS, mTLS | RBAC, IAM | Keine | RBAC |
| Bedrohung | Zugriff auf die Kubelet API | | Container Service Account | manipuliere externalIP eines Service (CVE-2020-8554) | Resource Hijacking |
| Gegen-maßnahme | RBAC, Audits | | Zugriff minimieren, eigener Service Account pro Pod/Service | strikte Konfiguration und Audits, OPA | Ressourcen-verbrauch einschränken, Quotas, Monitoring |
| Bedrohung | Netzwerk-Mapping | | Cluster-internes Networking | | Denial of Service |
| Gegen-maßnahme | NetworkPolicies, Service Mesh | | NetworkPolicies | | DisruptionBudgets, RessourceBudgets |
| Bedrohung | Zugriff auf das Kubernetes Dashboard | | Anwendungs-Zugangdaten in Konfigurationsdateien | | manipuliere externalIP eines Service (CVE-2020-8554) |
| Gegen-maßnahme | Authentifizierung, TLS | | KMS | | strikte Konfiguration und Audits, OPA |
| Bedrohung | Instance Metadata API | | Schreibbare Volumen mounts auf dem Host | | Physischer Zugriff auf den Node |
| Gegen-maßnahme | Network Policies | | auf das nötigste reduzieren | | Keine |
| Bedrohung | | | CoreDNS Poisining | | |
| Gegen-maßnahme | | | Audits | | |
| Bedrohung | | | ARP poisoning und IP spoofing | | |
| Gegen-maßnahme | | | Audits, Netzwerk-Sicherheitsmechanismen | | |
| Bedrohung | | | Netzwerkregeln manipulieren | | |
| Gegen-maßnahme | | | Audits | | |
| Bedrohung | | | Nutzung einer Schwachstelle innerhalb des Containers | | |
| Gegen-maßnahme | | | Least Privilege, Network Policies, OPA, Service Mesh | | |

Abbildung 6.2: Fortsetzung der Bedrohungsmatrix von Abbildung 6.1 mit Gegenmaßnahmen der Security Best Practices und der Zero-Trust-Architektur. Grün gibt eine gänzliche Verhinderung an, gelb sind Möglichkeiten zur Verhinderung, ohne gänzliche Verhinderung. Rot stellen keine Gegenmaßnahmen dar. Blaue Bedrohungen zeigen einen Angriffsvektor, Orange Bedrohungen zeigen die aktuellen Schwachstellen, die in Kapitel 4 erarbeitet wurden.

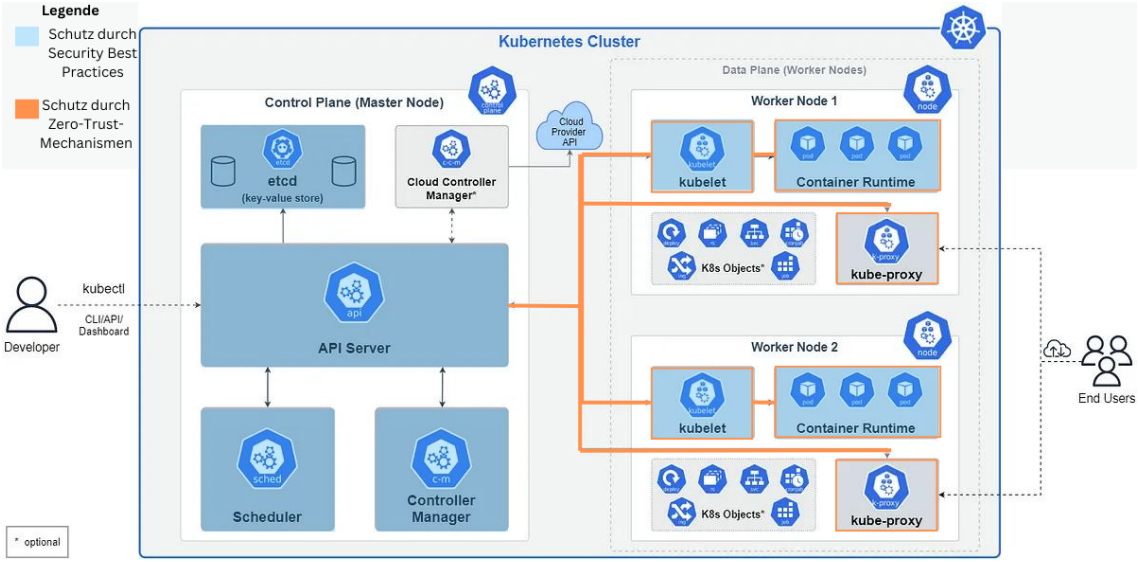



Abbildung 6.3: Bedrohungs-Heatmap, die die Möglichkeiten der Verhinderung von Bedrohungstechniken auf dem Cluster zeigt. Dabei wurde einheitlich alles, was von Security Best Practices abgedeckt wird, in Blau eingefärbt. Tragen Zero-Trust-Mechanismen zur Verbesserung der Sicherheit der Komponenten bei, sind die Komponenten orange eingefärbt. Somit entsteht die Einschätzung, dass Zero-Trust-Ansätze eine unterstützende Rolle, jedoch nicht die Hauptverteidigung des Clusters übernehmen und somit ein umfassender Sicherheitsansatz, der über Zero-Trust hinausgeht, notwendig ist.

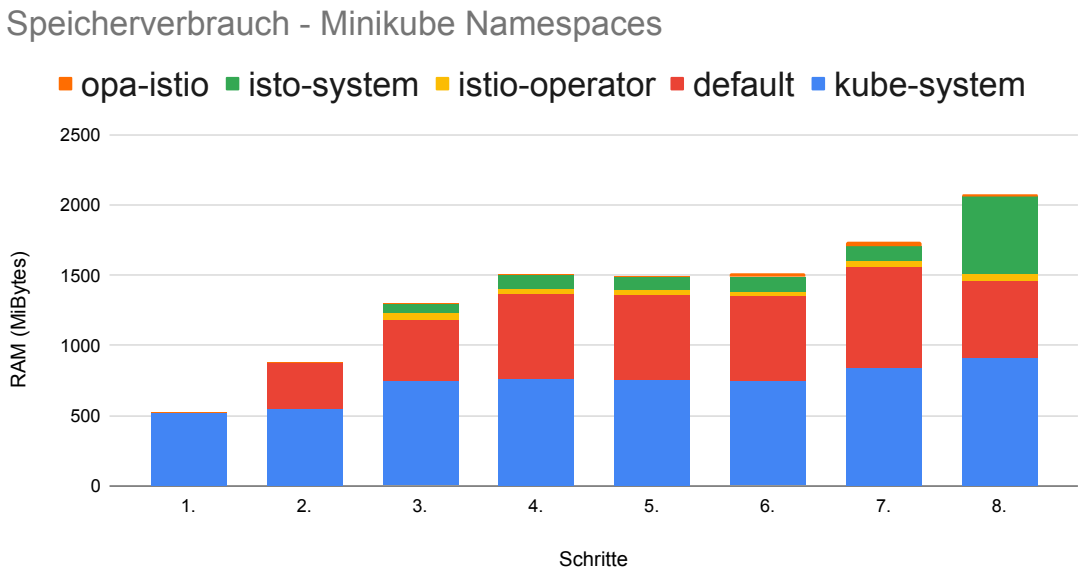



Abbildung 6.4: Speicherverbrauch des Minikube-Clusters pro Namespace: `default`: Enthält die Bookinfo-Anwendung; `kube-system`: Beinhaltet die Management-Ebene des Kubernetes Clusters; `istio-system`: Enthält die Komponenten (Istio Control Plane) des Istio-Service Meshs und die Monitoring-Tools; `istio-operator`: Dieser Namespace enthält den Istio-Operator; `opa-istio`: Beinhaltet den Admission Controller des Open Policy Agent (OPA). Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)

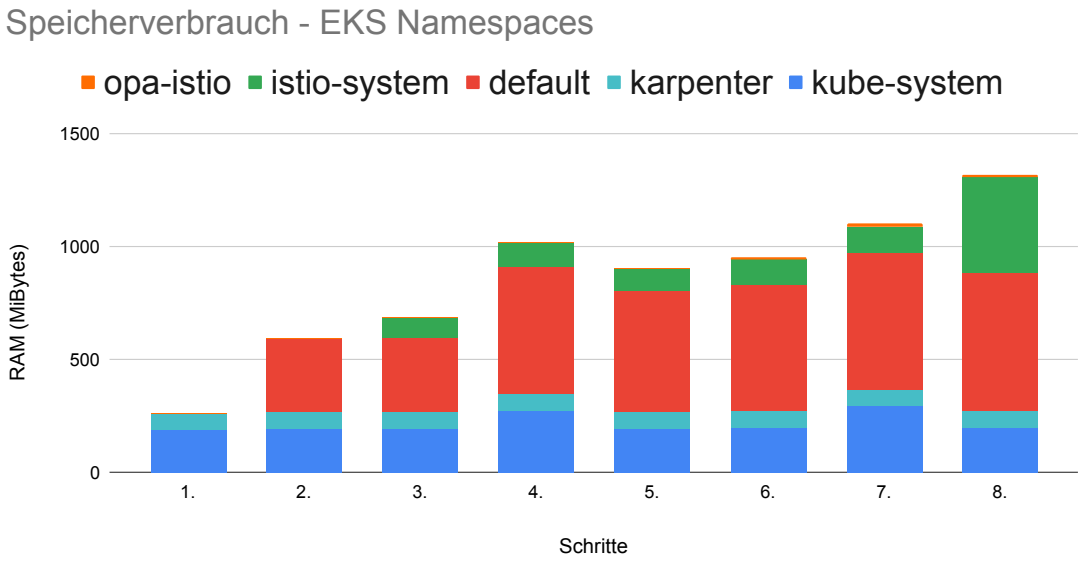



Abbildung 6.5: Speicherverbrauch des EKS-Clusters pro Namespace: `default:` Enthält die Bookinfo-Anwendung; `kube-system:` Beinhaltet die Management-Ebene des Kubernetes Clusters; `karpenter:` Enthält das AWS-Tool Karpenter. `istio-system:` Enthält die Komponenten (Istio Control Plane) des Istio-Service Meshs und die Monitoring-Tools; `opa-istio:` Beinhaltet den Admission Controller des Open Policy Agent (OPA).Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)

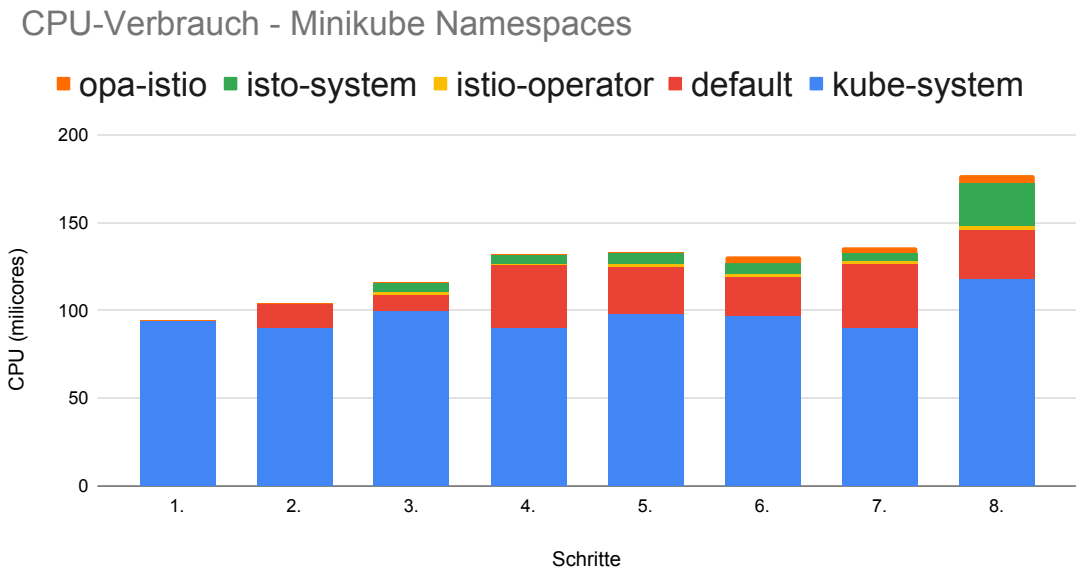



Abbildung 6.6: CPU-Verbrauch des Minikube-Clusters pro Namespace: `default`: Enthält die Bookinfo-Anwendung; `kube-system`: Beinhaltet die Management-Ebene des Kubernetes Clusters; `istio-system`: Enthält die Komponenten (Istio Control Plane) des Istio-Service Meshs und die Monitoring-Tools; `istio-operator`: Dieser Namespace enthält den Istio-Operator; `opa-istio`: Beinhaltet den Admission Controller des Open Policy Agent (OPA). Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)

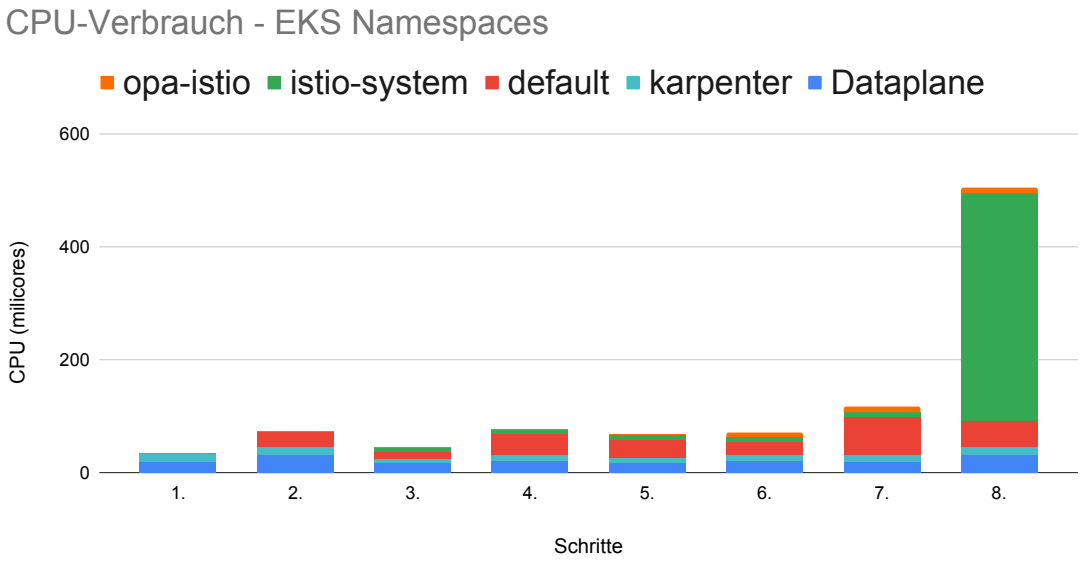
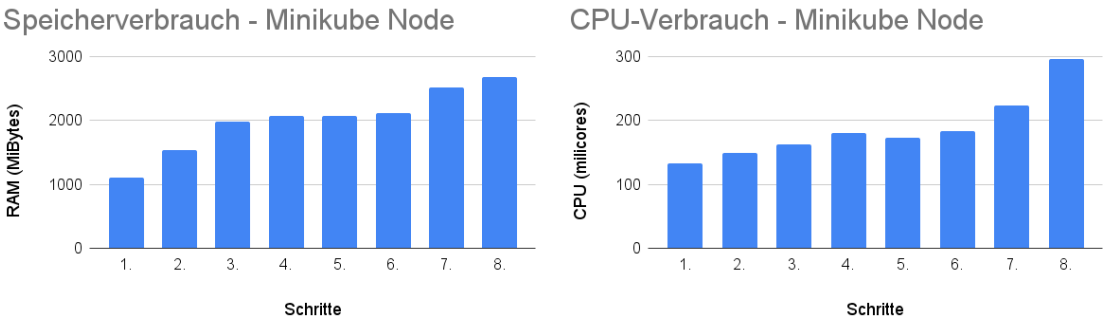
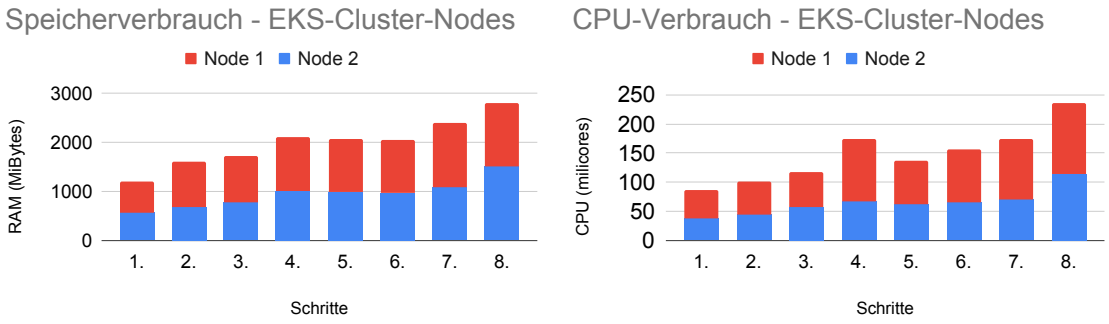



Abbildung 6.7: CPU-Verbrauch des EKS-Clusters pro Namespace: `default`: Enthält die Bookinfo-Anwendung; `kube-system`: Beinhaltet die Management-Ebene des Kubernetes Clusters; `karpenter`: Enthält das AWS-Tool Karpenter. `istio-system`: Enthält die Komponenten (Istio Control Plane) des Istio-Service Meshs und die Monitoring-Tools; `opa-istio`: Beinhaltet den Admission Controller des Open Policy Agent (OPA). Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)




(a) Speicherverbrauch des Minikube Nodes (b) CPU-Verbrach des Minikube Nodes

Abbildung 6.8: Speicher- (a) und CPU-Verbrauch (b) des Minikube-Cluster Node, unterteilt in Node 1. Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)




(a) Speicherverbrauch der EKS Nodes

(b) CPU-Verbrach der EKS Nodes

Abbildung 6.9: (a) Speicher- und (b) CPU-Verbrauch der EKS-Cluster Nodes, unterteilt in Node 1 und Node 2. Die x-Achse zeigt die Schritte 1 - 8, die zur Bestimmung des Ressourcenverbrauchs gewählt wurden: 1. Initial, 2. Installation der Bookinfo-Anwendung, 3. Installation des Istio Service Meshs, 4. Hinzufügen der Istio-Sidecar-Container, 5. Hinzufügen der notwendigen Policies, 6. Hinzufügen des Open Policy Agenten, 7. Hinzufügen der OPA-Sidecar-Container, 8. Hinzufügen der Monitoring-Tools (Grafana, Prometheus, Kiali)

# 7 Schluss

Im Rahmen dieser Arbeit wurden die fünf Forschungsfragen zur Beantwortung der Wirksamkeit einer Zero-Trust-Architektur für Kubernetes-Cluster beantwortet. Eine Zero-Trust-Architektur umfasst mehrere Netzwerkrichtlinien, ein Service Mesh sowie Monitoring-Tools wie Prometheus, Grafana und Kiali. Zur Erweiterung der Authentifizierungsmöglichkeiten sollte auch ein Open Policy Agent in die Zero-Trust-Architektur integriert werden. Entscheidend für eine erfolgreiche Zero-Trust-Implementierung ist jedoch die konsequente Umsetzung von Security Best Practices (FF.1).

Die wirtschaftlichen Auswirkungen einer Zero-Trust-Architektur, insbesondere durch den erhöhten CPU- und Speicherverbrauch, wurden ausführlich in Kapitel 6 diskutiert. Insbesondere in der Public Cloud (Amazon EKS) führen die zusätzlichen Verbräuche von Istio und Open Policy Agent zu einer Verdopplung der Kosten im Vergleich zu einer Umgebung ohne Zero-Trust-Architektur (FF.2).

Auch die Sicherheitsbewertung der Zero-Trust-Architektur wurde in dieser Arbeit eingehend untersucht. Die Analyse zeigt, dass die Zero-Trust-Architektur Bedrohungen aus der Bedrohungsmatrix nur begrenzt verhindern kann. Ein wesentlicher Vorteil liegt jedoch in der Reduzierung des Lateral Movement durch Netzwerkrichtlinien, AuthorizationPolicies sowie die Authentifizierung und Autorisierung von Services und Anwendungen. Dadurch trägt Zero Trust erheblich zur Verringerung von Insider-Bedrohungen bei, da jedes Gerät, jede Anwendung und jeder Nutzer kontinuierlich überwacht und überprüft wird, bevor Zugriff gewährt wird. Die Heatmap verdeutlicht, dass diese Maßnahmen vor allem die Netzwerksicherheit durch die Verbesserung der Netzwerkkommunikation stärken (FF.3).

Das Service Mesh, der Open Policy Agent und die Monitoring-Tools bringen jedoch auch neue Herausforderungen für die Sicherheit des Clusters mit sich. Aspekte wie offene Ports, Schwachstellen und das Management der Zertifikate für mTLS müssen bei der Beurteilung der Bedrohungslage berücksichtigt werden und erfordern zusätzliche Maßnahmen und Security Best Practices. Das erhöhte Datenvolumen und das damit verbundene Sicherheitsmanagement, ebenso wie der gesteigerte Verwaltungs- und Überwachungsaufwand, sind die offensichtlichsten Nachteile dieser Architektur (FF.4).

Die Integration der Zero-Trust-Architektur hat zudem gezeigt, dass ein tiefes Verständnis für die Kommunikation der Anwendungen erforderlich ist, um NetworkPolicies, Authorization-

Policies und Open Policy Agent-Richtlinien effektiv umzusetzen. Die Komplexität und der Zeitaufwand steigen mit der Größe des Clusters und der darin enthaltenen Anwendungen (FF.5).

## 7.1 Fazit

Diese Arbeit untersucht die Effektivität der Zero-Trust-Architektur in Kubernetes-Clustern im Vergleich zu traditionellen Sicherheitsansätzen.

Die Ergebnisse zeigen, dass Security Best Practices allein bereits einen umfassenden Schutz gegen die identifizierten Bedrohungen bieten können. Während die Zero-Trust-Architektur insbesondere bei der Reduzierung interner Bedrohungen innerhalb des Clusters unterstützt, ist deren Implementierung komplex und herausfordernd.

Die Integration der Zero-Trust-Komponenten, wie Network Policies, Authorization Policies und Open Policy Agenten, erfordert ein tiefes Verständnis der Anwendungen im Cluster. Diese Integration gestaltet sich besonders zeitaufwändig und erfordert kontinuierliche Überwachung und Anpassung der Richtlinien, um auf Veränderungen im Cluster angemessen zu reagieren. Der Aufwand skaliert mit der Anzahl und Größe der Anwendungen.

Zusätzlich hat die Implementierung der Zero-Trust-Architektur signifikante Auswirkungen auf den Ressourcenverbrauch. Besonders in der Public Cloud (Amazon EKS) wurden nahezu doppelt so hohe Kosten festgestellt, was auf einen erheblichen Anstieg des Ressourcenverbrauchs hinweist. Dies ist besonders relevant für Organisationen, die kosteneffiziente Lösungen suchen und den zusätzlichen Verwaltungsaufwand der Zero-Trust-Praktiken berücksichtigen müssen.

Weitere Herausforderungen ergeben sich durch offene Ports, Schwachstellen in neu hinzugefügten Komponenten und das Management von Zertifikaten für mTLS. Diese Aspekte erfordern zusätzliche Maßnahmen und verstärken den Aufwand für Sicherheitsmanagement und Überwachung. Die Balance zwischen Wirtschaftlichkeit und Sicherheitsgewinn wird daher zu einem zentralen Entscheidungsfaktor.

Zusammenfassend lässt sich sagen, dass die Zero-Trust-Architektur einen Beitrag zur Verbesserung der Sicherheit gegen interne Bedrohungen leisten kann. Sie bringt jedoch erhebliche Herausforderungen in Bezug auf Komplexität, Verwaltungsaufwand und wirtschaftliche Belastungen mit sich. Eine sorgfältige Kombination von Best Practices und gezielten Zero-Trust-Maßnahmen könnte für die meisten Organisationen eine ausgewogene Lösung darstellen, die sowohl Sicherheit als auch Effizienz berücksichtigt.

## 7.2 Ausblick

Diese Arbeit gab einen groben Einblick in die Möglichkeiten zur Erreichung von Security Best Practices und setzte dabei viele verschiedene Tools und Möglichkeiten ein. Aufgrund der Fülle an Tools für das Monitoring und die Sicherheitsanalyse des Clusters konnten diese nicht im Rahmen dieser Arbeit im Detail betrachtet werden. Entsprechend sollte auf Grundlage dieser Arbeit im Kontext ein Vergleich zwischen den Tools durchgeführt werden und ein oder mehrere Tools erwählt werden, die die Security Best-Practices bestens ergänzen und somit die Grundlage der Zero-Trust-Architektur weiter verstärken.

Eine weitere Arbeit zur Sicherheit in Kubernetes könnte basierend auf dieser Arbeit die Gesamteffektivität der Security Best-Practices und der Zero-Trust-Architektur bewerten und auswerten, ob das Verhältnis von Ressourcenverbrauch, Komplexität und Aufwand für die Erreichung der vollständigen Sicherheit innerhalb des Clusters akzeptabel ist.

Gerade was die Komplexität und den administrativen Aufwand der Zero-Trust-Architektur angeht, könnte sich eine zukünftige Arbeit auf die Optimierung der Verwaltung und die Reduktion des administrativen Aufwands konzentrieren, um die Vorteile der Zero-Trust-Architektur noch effizienter auszuschöpfen.

# Literaturverzeichnis

[amaa] *Amazon EKS and Kubernetes Container Insights metrics - Amazon CloudWatch — docs.aws.amazon.com.* https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Container-Insights-metrics-EKS.html, . – [Accessed 02-09-2024]

[amab] *Send control plane logs to CloudWatch Logs - Amazon EKS — docs.aws.amazon.com.* https://docs.aws.amazon.com/eks/latest/userguide/control-plane-logs.html#enabling-control-plane-log-export, . – [Accessed 02-09-2024]

[amac] *Using EKS encryption provider support for defense-in-depth | Amazon Web Services — aws.amazon.com.* https://aws.amazon.com/de/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/, . – [Accessed 31-07-2024]

[amad] *Was ist AWS App Mesh? - AWS App Mesh — docs.aws.amazon.com.* https://docs.aws.amazon.com/de_de/app-mesh/latest/userguide/what-is-app-mesh.html, . – [Accessed 03-09-2024]

[amae] *Was ist Containerisierung? – Containerisierung erklaert – AWS — aws.amazon.com.* https://aws.amazon.com/de/what-is/containerization/, . – [Accessed 01-07-2024]

[amaf] *What is the AWS Load Balancer Controller? - Amazon EKS — docs.aws.amazon.com.* https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html, . – [Accessed 17-06-2024]

[BBHD20] Burns, B. ; Beda, J. ; Hightower, K. ; Demmig, T.: *Kubernetes: Eine kompakte Einführung.* dpunkt.verlag, 2020 https://books.google.de/books?id=FCIPEAAAQBAJ. – ISBN 9783969100486

[BBM+21] Budigiri, Gerald ; Baumann, Christoph ; Muhlberg, Jan ; Truyen, Eddy ; Joosen, Wouter: Network Policies in Kubernetes: Performance Evaluation and Security Analysis, 2021, S. 407–412

[BOS+21] Buck, Christoph ; Olenberger, Christian ; Schweizer, André ; Völter, Fabiane ; Eymann, Torsten: Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. In: *Computers & Security* 110 (2021), S. 102436

[cer] *cert-manager — cert-manager.io*. https://cert-manager.io/docs/, . – [Accessed 26-08-2024]

[cnc] *Service mesh: A critical component of the cloud native stack — cncf.io*. https://www.cncf.io/blog/2017/04/26/service-mesh-critical-component-cloud-native-stack/, . – [Accessed 16-06-2024]

[cvea] *CVE-2020-8554 : Kubernetes API server in all versions allow an attacker who is able to create a ClusterIP service and set the spec.exter — cvedetails.com*. https://www.cvedetails.com/cve/CVE-2020-8554/, . – [Accessed 07-05-2024]

[cveb] *CVE-2024-21626 : runc is a CLI tool for spawning and running containers on Linux according to the OCI specification. In runc 1.1.11 and e — cvedetails.com*. https://www.cvedetails.com/cve/CVE-2024-21626/, . – [Accessed 04-08-2024]

[cvec] *CVE-2024-3177 : A security issue was discovered in Kubernetes where users may be able to launch containers that bypass the mountable sec — cvedetails.com*. https://www.cvedetails.com/cve/CVE-2024-3177/, . – [Accessed 17-08-2024]

[cved] *Versions of Docker Docker : Versions and number of related security vulnerabilities — cvedetails.com*. https://www.cvedetails.com/version-list/0/28125/1/?q=Docker, . – [Accessed 03-09-2024]

[cvee] *Versions of Istio Istio : Versions and number of related security vulnerabilities — cvedetails.com*. https://www.cvedetails.com/version-list/0/55469/1/?q=Istio, . – [Accessed 03-09-2024]

[cvef] *Versions of Kubernetes Kubernetes : Versions and number of related security vulnerabilities — cvedetails.com*. https://www.cvedetails.com/version-list/0/34016/1/?q=Kubernetes, . – [Accessed 03-09-2024]

[cveg] *Versions of Openpolicyagent Open Policy Agent : Versions and number of related security vulnerabilities — cvedetails.com.* https://www.cvedetails.com/version-list/0/109683/1/?q=Open+Policy+Agent, . – [Accessed 03-09-2024]

[DA21] D'Silva, Daniel ; Ambawade, Dayanand D.: Building A Zero Trust Architecture Using Kubernetes. In: *2021 6th International Conference for Convergence in Technology (I2CT)*, 2021, S. 1–8

[Das] Das, Pulak: *5 Steps to Integrate Istio with OPA — imesh.ai.* https://imesh.ai/blog/istio-opa/, . – [Accessed 28-08-2024]

[dee] *ThreatMapper - Open Source CNAPP by Deepfence — deepfence.io.* https://www.deepfence.io/threatmapper, . – [Accessed 27-08-2024]

[DHA22] Darwesh, Ghadeer ; Hammoud, Jaafar ; Alexandrovna, Alisa: Security in kubernetes: best practices and security analysis. (2022), Nr. 2 (44), S. 63–69

[doc] *Docker overview — docs.docker.com.* https://docs.docker.com/guides/docker-overview/#docker-architecture, . – [Accessed 27-06-2024]

[enda] *Docker Engine — endoflife.date.* https://endoflife.date/docker-engine, . – [Accessed 17-08-2024]

[endb] *Kubernetes — endoflife.date.* https://endoflife.date/kubernetes, . – [Accessed 03-09-2024]

[fal] *Learning Environment — falco.org.* https://falco.org/docs/install-operate/third-party/learning/, . – [Accessed 13-07-2024]

[Fer] Fernando, Kavishka: *Exploring the Kubernetes Architecture: A Foundation for Modern Application Deployment — kavishkafernando.* https://medium.com/@kavishkafernando/exploring-the-kubernetes-architecture-a-foundation-for-modern-application-deployment-f2c0f15d661e, . – [Accessed 18-06-2024]

[gita] *GitHub - istio/istio: Connect, secure, control, and observe services. — github.com.* https://github.com/istio/istio, . – [Accessed 02-09-2024]

[gitb] *Issues · kubernetes/minikube — github.com.* https://github.com/kubernetes/minikube/issues/8299, . – [Accessed 02-09-2024]

[gitc] *vault-k8s and istio service mesh don't work together · Issue 41 · hashicorp/vault-k8s — github.com.* https://github.com/hashicorp/vault-k8s/issues/41, . – [Accessed 11-07-2024]

[hasa] *Configure Vault as a certificate manager in Kubernetes with Helm | Vault | HashiCorp Developer — developer.hashicorp.com.* https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-cert-manager, . – [Accessed 06-07-2024]

[hasb] *Integrate a Kubernetes cluster with an external Vault | Vault | HashiCorp Developer — developer.hashicorp.com.* https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-external-vault#determine-the-vault-address, . – [Accessed 05-07-2024]

[Hew] Hewitt, Nik: *The Pros and Cons of Zero Trust Security • TrueFort — truefort.com.* https://truefort.com/pros-and-cons-of-zero-trust-security/, . – [Accessed 03-09-2024]

[ista] *Bookinfo Application — istio.io.* https://istio.io/latest/docs/examples/bookinfo/, . – [Accessed 31-07-2024]

[istb] *Getting Started — istio.io.* https://istio.io/latest/docs/setup/getting-started/, . – [Accessed 31-07-2024]

[istc] *JWT Token — istio.io.* https://istio.io/latest/docs/tasks/security/authorization/authz-jwt/, . – [Accessed 02-09-2024]

[istd] *Mutual TLS Migration — istio.io.* https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/, . – [Accessed 02-09-2024]

[iste] *Performance and Scalability — istio.io.* https://istio.io/latest/docs/ops/deployment/performance-and-scalability/, . – [Accessed 03-09-2024]

[istf] *Security Best Practices — istio.io.* https://istio.io/latest/docs/ops/best-practices/security/#control-plane, . – [Accessed 03-09-2024]

[istg] *What is Istio? — istio.io.* https://istio.io/latest/docs/overview/what-is-istio/, . – [Accessed 02-09-2024]

[Iye] Iyer, Sitaram: *Warum die mTLS-Authentifizierung von Istio Probleme birgt — dev-insider.de.* https://www.dev-insider.de/

warum-die-mtls-authentifizierung-von-istio-probleme-birgt-a-6cb6a0b22139bb10f158a0423646dbaf/, . – [Accessed 10-07-2024]

[k8s] *Using the Istio Addon — minikube.sigs.k8s.io.* https://minikube.sigs.k8s.io/docs/handbook/addons/istio/, . – [Accessed 31-07-2024]

[kuba] *Admission Controllers Reference — kubernetes.io.* https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#imagepolicywebhook, . – [Accessed 02-09-2024]

[kubb] *Auditing — kubernetes.io.* https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/, . – [Accessed 31-07-2024]

[kubc] *Certificate Management with kubeadm — kubernetes.io.* https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-certs/, . – [Accessed 03-09-2024]

[kubd] *Cluster Networking — kubernetes.io.* https://kubernetes.io/docs/concepts/cluster-administration/networking/, . – [Accessed 03-09-2024]

[kube] *Configure a Security Context for a Pod or Container — kubernetes.io.* https://kubernetes.io/docs/tasks/configure-pod-container/security-context/, . – [Accessed 02-09-2024]

[kubf] *Encrypting Confidential Data at Rest — kubernetes.io.* https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/, . – [Accessed 31-07-2024]

[kubg] *Kubelet authentication/authorization — kubernetes.io.* https://kubernetes.io/docs/reference/access-authn-authz/kubelet-authn-authz/, . – [Accessed 31-07-2024]

[kubh] *Network Policies — kubernetes.io.* https://kubernetes.io/docs/concepts/services-networking/network-policies/, . – [Accessed 31-07-2024]

[kubi] *Resource Management for Pods and Containers — kubernetes.io.* https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/, . – [Accessed 31-07-2024]

[kubj] *Specifying a Disruption Budget for your Application — kubernetes.io.* https://kubernetes.io/docs/tasks/run-application/configure-pdb/, . – [Accessed 31-07-2024]

[Kub19] Kubernetes, T: Kubernetes. In: *Kubernetes. Retrieved May* 24 (2019), S. 2019

[Kui] Kuiry, Mukesh: *How Docker Evolved | History of Containerization — dev.to.* https://dev.to/mukeshkuiry/evolution-of-docker-kubernetes-virtualization-1a9f, . – [Accessed 22-08-2024]

[LLG+19] Li, Wubin ; Lemieux, Yves ; Gao, Jing ; Zhao, Zhuofeng ; Han, Yanbo: Service Mesh: Challenges, State of the Art, and Future Research Opportunities. In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, S. 122–1225

[MB20] Mehraj, Saima ; Banday, M T.: Establishing a zero trust strategy in cloud computing environment. In: *2020 International Conference on Computer Communication and Informatics (ICCCI)* IEEE, 2020, S. 1–6

[MBR+21] Minna, Francesco ; Blaise, Agathe ; Rebecchi, Filippo ; Chandrasekaran, Balakrishnan ; Massacci, Fabio: Understanding the security implications of kubernetes networking. In: *IEEE Security & Privacy* 19 (2021), Nr. 5, S. 46–56

[Myt20] Mytilinakis, Panagiotis: *Attack methods and defenses on Kubernetes*, Πανεπιστήμιο Πειραιώς, Diplomarbeit, 2020

[Noa] Noah: *Kubernetes Security Tools: Falco — noah_h.* https://medium.com/@noah_h/kubernetes-security-tools-falco-e873831f3d3d, . – [Accessed 12-07-2024]

[opea] *Introduction — openpolicyagent.org.* https://www.openpolicyagent.org/docs/latest/, . – [Accessed 02-09-2024]

[opeb] *Tutorial: Istio — openpolicyagent.org.* https://www.openpolicyagent.org/docs/latest/envoy-tutorial-istio/, . – [Accessed 31-07-2024]

[owa] *Kubernetes Security - OWASP Cheat Sheet Series — cheatsheetseries.owasp.org.* https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html, . – [Accessed 02-08-2024]

[Pac21] Pace, Matteo: *Zero Trust networks with Istio*, Politecnico di Torino, Diss., 2021

[Pan] Panda, Debasree: *What is mTLS and How to implement it with Istio — imesh.ai.* https://imesh.ai/blog/what-is-mtls-and-how-to-implement-

it-with-istio/#ssl, . – [Accessed 28-06-2024]

[ROM+21] Rodigari, Simone ; O'Shea, Donna ; McCarthy, Pat ; McCarry, Martin ; McSweeney, Sean: Performance analysis of zero-trust multi-cloud. In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)* IEEE, 2021, S. 730–732

[Say17] Sayfan, Gigi: *Mastering kubernetes*. Packt Publishing Ltd, 2017

[SBR20] Shamim, Md Shazibul I. ; Bhuiyan, Farzana A. ; Rahman, Akond: Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. In: *2020 IEEE Secure Development (SecDev)* (2020), S. 58–64

[SCS+22] Sarkar, Sirshak ; Choudhary, Gaurav ; Shandilya, Shishir K. ; Hussain, Azath ; Kim, Hwankuk: Security of zero trust networks in cloud computing: A comparative review. In: *Sustainability* 14 (2022), Nr. 18, S. 11213

[SI20] Surantha, Nico ; Ivan, Felix: Secure kubernetes networking design based on zero trust model: A case study of financial service enterprise in indonesia. In: *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 13th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2019)* Springer, 2020, S. 348–361

[Sta20] Stafford, VA: Zero trust architecture. In: *NIST special publication* 800 (2020), S. 207

[Tan23] Tan, Kee H.: Mitigating Insider Threats in AWS: A Zero Trust Perspective. (2023), 8. http://dx.doi.org/10.1184/R1/23931384.v1. – DOI 10.1184/R1/23931384.v1

[Tan24] Tan, Kee H.: Mitigating Insider Threats in Amazon Elastic Kubernetes Service (EKS): A Zero Trust Perspective. (2024), 2. http://dx.doi.org/10.1184/R1/24923931.v1. – DOI 10.1184/R1/24923931.v1

[tro] *Deploy Trousseau - Documentation — docs.trousseau.io*. https://docs.trousseau.io/trousseau/v1/deployment/, . – [Accessed 06-07-2024]

[WA20] Weever, Catherine de ; Andreou, Marios: Zero trust network security model in containerized environments. In: *University of Amsterdam: Amsterdam, The Netherlands* (2020)

[Weia] Weizman, Yossi: *Secure containerized environments with updated threat matrix for Kubernetes*. https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-kubernetes/. – Zugriff am: 14.12.2023

[Weib] Weizman, Yossi: *Threat matrix for Kubernetes*. https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-kubernetes/. – Zugriff am: 14.12.2023

[WRK+19] Watada, Junzo ; Roy, Arunava ; Kadikar, Ruturaj ; Pham, Hoang ; Xu, Bing: Emerging Trends, Techniques and Open Issues of Containerization: A Review. In: *IEEE Access* 7 (2019), S. 152443–152472. http://dx.doi.org/10.1109/ACCESS.2019.2945930. – DOI 10.1109/ACCESS.2019.2945930

[Yac22] Yacobucci, Matthew: *Seven zero trust rules for Kubernetes — cncf.io*. https://www.cncf.io/blog/2022/11/04/seven-zero-trust-rules-for-kubernetes/, 2022. – [Accessed 15-05-2024]

[ZKL+23] Zeng, Qingyang ; Kavousi, Mohammad ; Luo, Yinhong ; Jin, Ling ; Chen, Yan: Full-stack vulnerability analysis of the cloud-native platform. In: *Computers & Security* 129 (2023), S. 103173

# A Anhang / Appendix

## A.1 Tabellen

Tabelle A.1: Anzahl und CVE-Nummern der Schwachstellen der Komponenten in Abhängigkeit des Angriffs für Kubernetes Versionen bis 28.0 [cvef]

| Komponente/ Angriff | Privilege Escalation | Denial of Service | Information Leakage | # |
|---|---|---|---|---|
| API-Server | CVE-2021-25735, CVE-2020-8561, CVE-2021-25737, CVE-2019-11247, CVE-2018-1002102, CVE-2018-1002105, CVE-2016-1905, CVE-2020-8559, CVE-2020-8554, CVE-2022-3294 | CVE-2019-11254, CVE-2019-1002100, CVE-2020-8552, CVE-2019-1125 | CVE-2020-8555, CVE-2022-3172 | 16 |
| Kubectl | CVE-2021-25743, CVE-2019-11251, CVE-2019-11246, CVE-2019-1002101, CVE-2019-11249, CVE-2019-11246 | CVE-2019-11244 | - | 7 |
| CRI-O | CVE-2018-100040, CVE-2022-0811 | CVE-2023-6476 | - | 3 |
| Kubelet | CVE-2020-8558, CVE-2020-8557, CVE-2021-25741, CVE-2023-2431 | CVE-2020-8551, CVE-2019-11248 | - | 6 |
| etcd | CVE-2023-32082, CVE-2020-15113, CVE-2020-15115, CVE-2020-15136, CVE-2021-28235, CVE-2023-0296 | CVE-2020-15106, CVE-2020-15112, CVE-2020-15114 | - | 9 |
| CNI | CVE-2020-8562, CVE-2021-25740, CVE-2020-10749, CVE-2019-9946, CVE-2020-11091, CVE-2020-26728 | - | - | 6 |
| Logging | - | - | CVE-2020-8565, CVE-2019-11252 | 2 |
| Gesamt | 34 | | 4 | 49 |

Tabelle A.2: Anzahl und CVE-Nummern der Schwachstellen der Docker-Architektur in Abhängigkeit des Angriffs [cved]

| Komponente/ Angriff | Privilege Escalation | Denial of Service | Information Leakage | Anzahl |
|---|---|---|---|---|
| Docker/dockerd | CVE-2021-21284, CVE-2018-10892, CVE-2018-9862, CVE-2019-13139, CVE-2018-15664 | CVE-2020-13401, CVE-2021-21285, CVE-2018-20699, CVE-2017-14992 | CVE-2019–13509, | 11 |
| containerd | CVE-2020-15257, CVE-2021-41103, CVE-2021-32760, CVE-2021-21334 | CVE-2023-25153, CVE-2022-31030, CVE-2022-23471, | CVE-2022-23648 | 8 |
| containerd-shim | CVE-2020-15257 | - | - | 1 |
| runc | CVE-2023-27561, CVE-2023-25809, CVE-2022-29162, CVE-2021-30465, CVE-2019-5736, CVE-2019-16884, CVE-2016-9962 | CVE-2021-43784, | - | 8 |
| Gesamt | 22 | 6 | | 27 |

Tabelle A.3: Speicherverbrauch (MiBytes der Namespaces und des Nodes im Minikube-Cluster während der Installation der Zero-Trust-Architektur in den Schritten: 1. Initial; 2. Installation der Bookinfo-Anwendung; 3. Installation des Istio Service Mesh; 4. Hinzufügen der istio-Sidecar-container; 5. Hinzufügen der notwendigen Policies; 6. Installation des Open Policy Agent (OPA); Hinzufügen der OPA-Sidecar-Container; 8. Integration der Monitoring Tools

| Schritte | kube-system | default | istio-operator | isto-system | opa-istio | Gesamt | Gesamt Node |
|---|---|---|---|---|---|---|---|
| 1. | 527 | | | | | 527 | 1108 |
| 2. | 550 | 327 | | | | 877 | 1529 |
| 3. | 749 | 434 | 46 | 69 | | 1298 | 1980 |
| 4. | 759 | 609 | 37 | 101 | | 1506 | 2072 |
| 5. | 754 | 604 | 37 | 97 | | 1492 | 2066 |
| 6. | 747 | 603 | 38 | 99 | 27 | 1514 | 2116 |
| 7. | 842 | 723 | 38 | 108 | 28 | 1739 | 2510 |
| 8. | 911 | 549 | 47 | 553 | 18 | 2078 | 2679 |

Tabelle A.4: CPU-Verbrauch (milicores) der Namespaces und des Nodes im Minikube-Cluster während der Installation der Zero-Trust-Architektur in den Schritten: 1. Initial; 2. Installation der Bookinfo-Anwendung; 3. Installation des Istio Service Mesh; 4. Hinzufügen der istio-Sidecar-container; 5. Hinzufügen der notwendigen Policies; 6. Installation des Open Policy Agent (OPA); Hinzufügen der OPA-Sidecar-Container; 8. Integration der Monitoring Tools

| Schritte | kube-system | default | istio-operator | isto-system | opa-istio | Gesamt | Gesamt Node |
|---|---|---|---|---|---|---|---|
| 1. | 94 | | | | | 94 | 133 |
| 2. | 90 | 14 | | | | 104 | 149 |
| 3. | 100 | 9 | 2 | 5 | | 162 | |
| 4. | 90 | 36 | 1 | 5 | | 132 | 181 |
| 5. | 98 | 27 | 2 | 6 | | 133 | 173 |
| 6. | 97 | 22 | 2 | 6 | 4 | 131 | 183 |
| 7. | 90 | 37 | 1 | 5 | 3 | 136 | 223 |
| 8. | 118 | 28 | 2 | 25 | 4 | 177 | 296 |

Tabelle A.5: Speicherverbrauch (MiBytes) der Namespaces und der Nodes im EKS-Cluster während der Installation der Zero-Trust-Architektur in den Schritten: 1. Initial; 2. Installation der Bookinfo-Anwendung; 3. Installation des Istio Service Mesh; 4. Hinzufügen der istio-Sidecar-container; 5. Hinzufügen der notwendigen Policies; 6. Installation des Open Policy Agent (OPA); Hinzufügen der OPA-Sidecar-Container; 8. Integration der Monitoring Tools

| Schritte | kube-system | karpenter | default | isto-system | opa-istio | Gesamt | Node 1 | Node 2 | Gesamt Nodes |
|---|---|---|---|---|---|---|---|---|---|
| 1. | 190 | 74 | | | | 264 | 564 | 638 | 1202 |
| 2. | 192 | 74 | 327 | | | 593 | 679 | 930 | 1609 |
| 3. | 193 | 74 | 329 | 89 | | 685 | 783 | 932 | 1715 |
| 4. | 274 | 74 | 557 | 111 | | 1016 | 1001 | 1092 | 2093 |
| 5. | 194 | 76 | 532 | 100 | | 902 | 985 | 1077 | 2062 |
| 6. | 197 | 76 | 554 | 116 | 9 | 952 | 989 | 1081 | 2050 |
| 7. | 293 | 76 | 601 | 117 | 13 | 1100 | 1076 | 1311 | 2387 |
| 8. | 197 | 76 | 608 | 425 | 10 | 1316 | 1500 | 1290 | 2790 |

Tabelle A.6: CPU-Verbrauch (milicores) der Namespaces und der Nodes im EKS-Cluster während der Installation der Zero-Trust-Architektur in den Schritten: 1. Initial; 2. Installation der Bookinfo-Anwendung; 3. Installation des Istio Service Mesh; 4. Hinzufügen der istio-Sidecar-container; 5. Hinzufügen der notwendigen Policies; 6. Installation des Open Policy Agent (OPA); Hinzufügen der OPA-Sidecar-Container; 8. Integration der Monitoring Tools

| Schritte | kube-system | default | istio-operator | isto-system | opa-istio | Gesamt | Node 1 | Node 2 | Gesamt Nodes |
|---|---|---|---|---|---|---|---|---|---|
| 1. | 19 | 14 | | | | 33 | 38 | 48 | 86 |
| 2. | 32 | 13 | 28 | | | 73 | 45 | 57 | 102 |
| 3. | 18 | 8 | 11 | 8 | | 45 | 58 | 60 | 118 |
| 4. | 22 | 10 | 38 | 8 | | 78 | 67 | 107 | 174 |
| 5. | 18 | 10 | 31 | 8 | 9 | 67 | 63 | 74 | 137 |
| 6. | 22 | 10 | 23 | 8 | 9 | 72 | 65 | 91 | 156 |
| 7. | 20 | 12 | 68 | 8 | 10 | 118 | 70 | 105 | 175 |
| 8. | 32 | 13 | 46 | 404 | 9 | 504 | 115 | 121 | 236 |

## A.2 Listings

Listing A.1: Konfiguration von Istio, sodass der Open Policy Agent als externer Autorisierer genutzt wird

```
1 data:
2   mesh: |-
3     # Add the following lines to define the ServiceEntry
          previously created as an external authorizer:
4     extensionProviders:
5     - name: opa-ext-authz-grpc
6       envoyExtAuthzGrpc:
7         service: opa-ext-authz-grpc.local
8         port: "9191"
```

Listing A.2: Kubelet-Konfiguration, um Authentifizerung zu aktivieren

```
1 authentication:
2   anonymous:
3     enabled: false  # Deaktiviert die anonyme Authentifizierung
4   webhook:
5     enabled: true   # Aktiviert die Webhook-Authentifizierung
6   x509:
7     clientCAFile: /var/lib/minikube/certs/ca.crt
```

Listing A.3: Ressourcen-Limits beispielhaft für die Productpage mit einer Request Size von 200 MiBytes und 500 milicores und Limits von 400Mi und 2000 milicores

```
1     containers:
2     - name: productpage
3       image: docker.io/istio/examples-bookinfo-productpage-v1:1.19.1
4       resources:
5         requests:
6           memory: "200Mi"
7           cpu: "500m"
8         limits:
9           memory: "400Mi"
10          cpu: "2000m"
```

Listing A.4: etcd EncryptionConfiguration, um das etcd mit dem secret zu verschlüsseln

```
1 apiVersion: apiserver.config.k8s.io/v1
2 kind: EncryptionConfiguration
3 resources:
4   - resources:
5     - secrets
6     providers:
7     - aesgcm:
8         keys:
9         - name: key1
```

Listing A.5: Darstellung der API-Server-Konfiguration, um ihn anzuweisen, die Verschlüsselung des etcd mit dem zuvor angelegten Schlüssel (EncryptionConfiguration) durchzuführen

```
spec:
  containers:
  - command:
    - kube-apiserver
...
    - --encryption-provider-config=/etc/kubernetes/etcd/ec.yaml
...
    volumeMounts:
    - mountPath: /etc/kubernetes/etcd
      name: etcd
      readOnly: true
...
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/kubernetes/etcd
      type: DirectoryOrCreate
    name: etcd
...
```

Listing A.6: ServiceAccount-Token für Trousseau mit Namen trousseau-vault-auth [tro]

```
---
apiVersion: v1
kind: Secret
metadata:
  namespace: kube-system
  name: trousseau-vault-auth
  annotations:
    kubernetes.io/service-account.name: "trousseau-vault-auth"
type: kubernetes.io/service-account-token
```

```
          secret: dGhpcy1pcy12ZXJ5LXNlYw==
    - identity: {}
```

Listing A.7: RBAC ClusterRoleBinding system:auth-delegator trousseau SA [tro]

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
   name: role-tokenreview-binding
   namespace: kube-system
roleRef:
   apiGroup: rbac.authorization.k8s.io
   kind: ClusterRole
   name: system:auth-delegator
subjects:
- kind: ServiceAccount
  name: trousseau-vault-auth
  namespace: kube-system
```

Listing A.8: Trousseau Vault ConfigMap [tro]

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: trousseau-vault-agent-config
  namespace: kube-system
data:
  vault-agent-config.hcl: |
    exit_after_auth = true
    pid_file = "/home/vault/pidfile"
    auto_auth {
        method "kubernetes" {
            mount_path = "auth/kubernetes"
            config = {
                role = "trousseau"
            }
        }
        sink "file" {
            config = {
                path = "/home/vault/.vault-token"
            }
        }
    }

    template {
    destination = "/etc/secrets/config.yaml"
    contents = <<EOT
    {{- with secret "secret/data/trousseau/config" }}
    ---
    provider: vault
    vault:
      keynames:
      - {{ .Data.data.transitkeyname }}
      address: {{ .Data.data.vaultaddress }}
      token: {{ .Data.data.vaulttoken }}
    {{ end }}
    EOT
    }
```

Listing A.9: EncryptionConfiguration für trousseau.io. Endpunkt ist der unix-daemon des trousseau services[hasb]

```
kind: EncryptionConfiguration
apiVersion: apiserver.config.k8s.io/v1
resources:
  - resources:
      - secrets
    providers:
      - kms:
          name: trousseau-vault-plugin
          endpoint: unix:///opt/trousseau-kms/vaultkms.socket
          cachesize: 1000
      - identity: {}
```

Listing A.10: IP Adresse des Hosts bestimmen

```
$ minikube ip
192.168.49.2
$ ifconfig
...
br-e005d919e3a8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
mtu 1500
  inet 192.168.49.1  netmask 255.255.255.0
broadcast 192.168.49.255
  inet6 fe80::42:59ff:fe4c:3434  prefixlen 64  scopeid 0x20<link>
  ...
```

Listing A.11: Pod Konfiguration mit VAULT_ADDRES [hasb]

```
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp
spec:
  serviceAccountName: internal-app
  containers:
    - name: app
      image: webapp
      env:
      - name: VAULT_ADDR
        value: "http://$EXTERNAL_VAULT_ADDR:8200"
      - name: VAULT_TOKEN
        value: root
```

Listing A.12: Service und Endpoint für Vault [hasb]

```
---
apiVersion: v1
kind: Service
metadata:
  name: external-vault
  namespace: default
spec:
  ports:
  - protocol: TCP
    port: 8200
---
apiVersion: v1
kind: Endpoints
metadata:
  name: external-vault
subsets:
  - addresses:
      - ip: $EXTERNAL_VAULT_ADDR
    ports:
      - port: 8200
```

Listing A.13: Secret Service-Account-Token [hasb]

```
apiVersion: v1
kind: Secret
metadata:
  name: vault-token-g955r
  annotations:
    kubernetes.io/service-account.name: vault
type: kubernetes.io/service-account-token
```

Listing A.14: Pod Konfiguration mit Annotationen [hasb]

```
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp
  annotations:
    vault.hashicorp.com/agent-inject: 'true'
    vault.hashicorp.com/role: 'web-app'
    vault.hashicorp.com/agent-inject-secret-credentials.txt: '
       secret/data/webapp/config'
spec:
  serviceAccountName: internal-app
  containers:
    - name: app
      image: webapp
```

Listing A.15: Istio Vault-Agent zusätzliche Annotation [gitc]

```
  annotations:
    vault.hashicorp.com/agent-init-first: 'true'
    ...
```

Listing A.16: Secrets with Secrets Manager. [hasb]

```
---
apiVersion: kubernetes-client.io/v1
kind: ExternalSecret
metadata:
  name: my-secret
  namespace: default
spec:
  backendType: secretsManager
  data:
    - key: username
      name: secret/webapp/config
    - key: password
      name: secret/webapp
```

Listing A.17: Secret Service-Account-Token [hasb]

```
apiVersion: v1
kind: Secret
metadata:
  name: issuer-token-lmzpj
  annotations:
    kubernetes.io/service-account.name: issuer
type: kubernetes.io/service-account-token
```

Listing A.18: Issuer Konfiguration cert-manager [hasb]

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    server: http://external-vault:8200
    path: pki/sign/example-dot-com
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: issuer-token-lmzpj
          key: token
```

Listing A.19: Certificate Definition cert-manager [hasa]

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  commonName: www.example.com
  dnsNames:
  - www.example.com
```

Listing A.20: SecurityContext für die Container

```
securityContext:
  capabilities:
    drop:
    - all
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
```

Listing A.21: Minikube start mit audit-policy und webook-Konfiguration

```
minikube start \
    --cni calico \
    --memory=8192mb \
    --cpus=4 \
    --extra-config=apiserver.audit-policy-file=/etc/ssl/certs/
       audit-policy.yaml \
    --extra-config=apiserver.audit-log-path=- \
    --extra-config=apiserver.audit-webhook-config-file=/etc/ssl/
       certs/webhook-config.yaml \
    --extra-config=apiserver.audit-webhook-batch-max-size=10 \
    --extra-config=apiserver.audit-webhook-batch-max-wait=5s
```

Listing A.22: Warnung innerhalb der Logs anhand der Falco-Konfiguration

```
Warning K8s configmap with private credential (user=minikube-user
   verb=create resource=configmaps configmap=myconfigmap config
   ={"password":"123456","username":"admin"})
```

Listing A.23: Auschnitt der Netwerk-Richtlinien

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress

---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: istio-ingress-lockdown
  namespace: default
spec:
  podSelector:
    matchLabels:
      istio : ingressgateway
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443

---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: product-page-ingress
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: productpage
  ingress:
  - ports:
    - protocol: TCP
      port: 9080
    from:
    - podSelector:
        matchLabels:
          istio: ingressgateway
```

Listing A.24: Ausschnitt Istio AuthorizationPolicy

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: allow-nothing
  namespace: default
spec:
  {}

---
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: "productpage-viewer"
  namespace: default
spec:
  selector:
    matchLabels:
      app: productpage
  action: ALLOW
  rules:
  - to:
    - operation:
        methods: ["GET"]

---
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: "details-viewer"
  namespace: default
spec:
  selector:
    matchLabels:
      app: details
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/default/sa/bookinfo-productpage"]
    to:
    - operation:
        methods: ["GET"]
```

Listing A.25: Istio mTLS-peer-policy

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: "mTLS-peer-policy"
  mtls:
    mode: STRICT
```

Listing A.26: Autorisierungs Richtlinie zur verpflichtenden Autorisierung mittels JWT für das Ingress Gateway [istc]

```
1
2 apiVersion: security.istio.io/v1
3 kind: AuthorizationPolicy
4 metadata:
5   name: "frontend-ingress"
6   namespace: istio-system
7 spec:
8   selector:
9     matchLabels:
10      istio: ingressgateway
11  action: DENY
12  rules:
13  - from:
14    - source:
15        notRequestPrincipals: ["*"]
```

Listing A.27: Open Policy Agent Policy für die Authorization und die Image-Quelle

```
package istio.authz

import rego.v1

import input.attributes.request.http as http_request
import input.parsed_path

default allow := false

allow if {
        parsed_path[0] == "health"
        http_request.method == "GET"
}

allow if {
        some r in roles_for_user
        r in required_roles
}

roles_for_user contains r if {
        some r in user_roles[user_name]
}

required_roles contains r if {
        some perm in role_perms[r]
        perm.method == http_request.method
        perm.path == http_request.path
}

user_name := parsed if {
        [_, encoded] := split(http_request.headers.authorization,
            " ")
        [parsed, _] := split(base64url.decode(encoded), ":")
}

user_roles := {
        "alice": ["guest"],
        "bob": ["admin"],
}

role_perms := {
        "guest": [{"method": "GET", "path": "/productpage"}],
        "admin": [
                {"method": "GET", "path": "/productpage"},
                {"method": "GET", "path": "/api/v1/products"},
        ],
}
```

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings