Ein Bild, das Schrift, Grafiken, Logo, Text enthält.

Automatisch generierte Beschreibung

Inwiefern erhöht die Einführung von Kubernetes die Sicherheit der Software-Bereitstellung?

**Bachelor-Thesis**

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.) im Studiengang Business Information Management vorgelegt dem Fachbereich Wirtschaftsinformatik der Provadis School of International Management and Technology von

Tim Dachs

D116

Betreuer:

Dr.-Ing. Florian Volk

Prof. Dr. Jörg Daubert

Johannes Fuchs

# Selbstständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit persönlich und selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder sinngemäß anderen Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Zeichnungen, Abbildungen und Tabellen in dieser Arbeit sind von mir selbst erstellt oder wurden mit einem entsprechenden Quellennachweis versehen. Diese Arbeit wurde weder in gleicher noch in ähnlicher Form von mir an anderen Hochschulen zur Erlangung eines akademischen Abschlusses eingereicht.

Liederbach, den 31.01.2025

Tim Dachs

# Inhaltsverzeichnis

[Selbstständigkeitserklärung II](#_Toc183536346)

[Sperrvermerk III](#_Toc183536347)

[Inhaltsverzeichnis IV](#_Toc183536348)

[Abbildungsverzeichnis VI](#_Toc183536349)

[Abkürzungsverzeichnis VII](#_Toc183536350)

[Listing-Verzeichnis VIII](#_Toc183536351)

[1 Einleitung 1](#_Toc183536352)

[1.1 Einführung in das betriebliche Umfeld 1](#_Toc183536353)

[1.2 Motivation 2](#_Toc183536354)

[1.3 Leitfrage 2](#_Toc183536355)

[1.4 Zielsetzung 2](#_Toc183536356)

[1.5 Aufbau 3](#_Toc183536357)

[2 Theorie 3](#_Toc183536358)

[2.1 Software-Bereitstellung 3](#_Toc183536359)

[2.2 IT-Sicherheit 6](#_Toc183536360)

[2.2.1 Vertraulichkeit 6](#_Toc183536361)

[2.2.2 Integrität 7](#_Toc183536362)

[2.2.3 Verfügbarkeit 7](#_Toc183536363)

[2.2.4 Erweiterte Schutzziele 8](#_Toc183536364)

[2.3 Kubernetes 8](#_Toc183536365)

[2.3.1 Kubernetes-Architektur 8](#_Toc183536366)

[2.3.2 Kubernetes-Konzept 11](#_Toc183536367)

[3 Methodik 12](#_Toc183536368)

[3.1 Forschungsvorgehen 12](#_Toc183536369)

[3.2 Literaturrecherche 12](#_Toc183536370)

[3.3 Fallstudien 12](#_Toc183536371)

[3.4 Experteninterviews 13](#_Toc183536372)

[3.5 Hypothesen 13](#_Toc183536373)

[4 Analyse des Kubernetes-Sicherheitskonzept 13](#_Toc183536374)

[4.1 Sicherheitsmerkmale von Kubernetes 14](#_Toc183536375)

[4.1.1 Zugriffskontrolle 14](#_Toc183536376)

[4.1.1.1 Rollenbasierte Zugriffskontrolle 14](#_Toc183536377)

[4.1.1.2 Attributbasierte Zugriffskontrolle 16](#_Toc183536378)

[4.1.2 Netzwerksicherheit 16](#_Toc183536379)

[4.1.3 Pod-Sicherheit 17](#_Toc183536380)

[4.1.4 Secret-Sicherheit 17](#_Toc183536381)

[4.1.5 Image-Sicherheit 17](#_Toc183536382)

[4.2 Umgang von typischen Angriffsvektoren mit Kubernetes 18](#_Toc183536383)

[4.3 Sicherheit beim Software-Deployment 19](#_Toc183536384)

[4.3.1 Rolling-Updates 19](#_Toc183536385)

[4.3.2 Rollback-Funktionalität 20](#_Toc183536386)

[4.4 Vergleich zur Bereitstellung ohne Orchestrierung 21](#_Toc183536387)

[5 Experteninterviews 23](#_Toc183536388)

[5.1 Vorstellung der Experten 23](#_Toc183536389)

[5.2 Auswertung der Interviewergebnisse 23](#_Toc183536390)

[5.3 Einordnung in den Kontext der Forschungsfrage 23](#_Toc183536391)

[6 Fallstudien 23](#_Toc183536392)

[6.1 Beschreibung des Anwendungsfall 23](#_Toc183536393)

[6.2 Analyse der Umsetzung 23](#_Toc183536394)

[6.3 Auswertung und Diskussion 23](#_Toc183536395)

[7 Diskussion 23](#_Toc183536396)

[7.1 Zusammenführen der wichtigsten Erkenntnisse 23](#_Toc183536397)

[7.2 Bewertung des Einflusses von Kubernetes 23](#_Toc183536398)

[7.3 Reflexion des Forschungsvorgehen 23](#_Toc183536399)

[8 Fazit und Ausblick 23](#_Toc183536400)

[8.1 Vorstellung der Ergebnisse 23](#_Toc183536401)

[8.2 Beantwortung der Hypothesen 23](#_Toc183536402)

[8.3 Beantwortung der Leitfrage 23](#_Toc183536403)

[8.4 Empfehlung 23](#_Toc183536404)

[8.5 Implikation für Forschung und Praxis 23](#_Toc183536405)

[9 KI-Verzeichnis 23](#_Toc183536406)

[10 Literaturverzeichnis 24](#_Toc183536407)

[11 Anhang 24](#_Toc183536408)

# Abbildungsverzeichnis

[Abbildung 1 - CI/CD-Pipeline 5](#_Toc183019740)

[Abbildung 2 - Kubernetes Architektur 9](#_Toc183019741)

[Abbildung 3 - Kubernetes Thread Matrix 18](#_Toc183019742)

# Abkürzungsverzeichnis

# Listing-Verzeichnis

[Listing 1 - Erstellung der Rolle 14](#_Toc182838890)

[Listing 2 - Zuweisung der Rolle 15](#_Toc182838891)

# Einleitung

Die Bereitstellung von Software, auch ‚Software Deployment‘ genannt, ist ein komplexer Prozess, der darauf abzielt, eine Software auf einem Zielsystem zu installieren und lauffähig zu machen. Ein Unternehmen, das Software für einen Kunden bereitstellen möchte, muss sicherstellen, dass es die dafür notwendigen Abläufe implementiert. Dabei spielt neben der tatsächlichen Bereitstellung auch die Sicherheit dieses Prozesses eine bedeutsame Rolle.

Bei der Implementierung von Sicherheitsmaßnahmen steht ein Unternehmen vor der Entscheidung, welche Software es dafür nutzen möchte. Kubernetes ist ein dafür speziell entwickeltes Open-Source-System von Google, das die Bereitstellung von Software durch Virtualiserung und Containerisierung unkompliziert und sicher machen soll. Bevor sich ein Unternehmen für die Nutzung von Kubernetes entscheidet, muss es sicherstellen, dass die eigenen Sicherheitsanforderungen durch Kubernetes abgedeckt werden. Daraus resultiert die Frage, was Kubernetes eigentlich genau macht, wie es die Software-Bereitstellung und dessen Sicherheit beeinflusst und ob Risiken bei der Nutzung entstehen.

Diese Forschungsarbeit soll den Prozess der Software-Bereitstellung untersuchen und den Einfluss von Kubernetes herausarbeiten. Dies soll durch die Beantwortung der Leitfrage: „Inwiefern erhöht die Einführung von Kubernetes die Sicherheit der Software-Bereitstellung?“, ermittelt werden. Der Sicherheitsbegriff im IT-Bereich soll erläutert werden und der Bereitstellungsprozess anhand der IT-Schutzziele analysiert werden: Verfügbarkeit, Vertraulichkeit und Integrität. Um eine Grundlage zur Bewertung von Kubernetes zu schaffen, soll mithilfe der Betrachtung eine Prozess mit und ohne Kubernetes ein Vergleich die positiven, wie negativen Auswirkungen auf die Bereitstellung herausgearbeitet werden. Hierfür werden Literaturrecherchen, Fallbeispiele sowie Experteninterview genutzt.

Hinweis:

Zur besseren Lesbarkeit wird in dieser Hausarbeit das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich – sofern nicht anders kenntlich gemacht – auf alle Geschlechter.

## Einführung in das betriebliche Umfeld

Die *S&N Invent GmbH* ist ein mittelständischer IT-Dienstleister mit Sitz in Eschborn. Der Großteil der Kunden stammt aus dem Finanzsektor, wie beispielsweise Banken oder Bausparkassen. Aber auch diverse Versicherungen greifen auf die Expertise der S&N Invent GmbH zurück. Derzeit sind 151 Mitarbeiter, davon 8 Duale-Studenten, bei der S&N Invent GmbH beschäftigt.

Die S&N Invent GmbH ist Teil der S&N Group AG, die mit über 500 Beschäftigten in mehr als 100 Kundenprojekten tätig ist. Die S&N Invent GmbH bietet ein breites Portfolio an. Die Leistungen umfassen nicht nur Softwareentwicklung und Applikation Management, sondern auch Beratungsdienstleistungen zur zielgerichteten Informationsversorgung. Als dualer Student bin ich Teil des Java-Development-Teams und somit selbst als Entwickler in einem Kundenprojekt bei einem Finanzdienstleister tätig. Sowohl in diesem Projekt als auch bei unternehmenseigenen Produkten wird Kubernetes zur Software-Bereitstellung genutzt.

## Motivation

Da die *S&N Invent GmbH* und der betreute Kunde Kubernetes zur Software-Bereitstellung einsetzen, ergibt sich daraus ein starkes berufliches, aber auch persönliches Interesse an einer tiefergehenden Untersuchung dieses System. Während meiner Tätigkeiten als Fullstack-Entwickler habe ich miterlebt wie Kubernetes implementiert und genutzt wurde. In einem internen Projekt habe ich bereits eine Deployment-Pipeline mit Docker und Jenkins aufgesetzt. Diese praktische Erfahrung hat mir Einblicke in den Umgang mit der Container-Orchestrierung gegeben.

Im Rahmen dieser Arbeit soll nun die Fähigkeiten von Kubernetes analysiert und eingeordnet werden, um ein fundierteres Verständnis für die Container-Orchestrierung zu gewinnen.

## Leitfrage

Inwiefern erhöht die Einführung von Kubernetes die Sicherheit der Software-Bereitstellung?

## Zielsetzung

Das Ziel der Forschungsarbeit ist es, den Einfluss von Kubernetes auf die Sicherheit der Deployment-Strategie zu untersuchen. Dabei sollen die zentralen Schutzziele der IT-Sicherheit durch die Aufstellung von drei Hypothesen einzeln betrachtet und bewertet werden:

1. *Deployment mit Kubernetes erhöht die Verfügbarkeit bei der Software-Bereitstellung.*
2. *Deployment mit Kubernetes erhöht die Integrität bei der Software-Bereitstellung.*
3. *Deployment mit Kubernetes erhöht die Vertraulichkeit bei der Software-Bereitstellung.*

Eine genauere Ausführung sowie die Darstellung möglicher Ergebnisse findet in Abschnitt 3.5 statt. Der Schwerpunkt liegt auf der Messung der Schutzziele durch geeignete Messverfahren sowie der Analyse der Ergebnisse. Mithilfe der Literaturrecherche soll eine stabile Grundlage geschaffen werden, um die für die Deployment-Strategien geeigneten Messverfahren zu ermitteln, anzuwenden und auswerten zu können. Des Weiteren soll die Literaturrecherche dazu dienen geeignete Fragen für Experteninterviews aufstellen zu können. Die Erkenntnisse aus Literatur, Fallstudien und Interviews sollen anschließend zusammengeführt und diskutiert werden. Durch die Überprüfung der Hypothesen und die Beantwortung der Leitfrage soll eine Empfehlung für den Umgang mit Kubernetes gegeben werden.

## Aufbau

Nachfolgend zu dieser Einleitung beginnt die Arbeit mit der Schaffung der theoretischen Grundlange durch die Darlegung des aktuellen wissenschaftlichen Stands in den Bereichen Software-Bereitstellung, IT-Sicherheit und Kubernetes. Anschließend werden das Forschungsvorgehen und die entsprechenden Forschungsmethoden vorgestellt.

Im vierten Kapitel wird das Sicherheitskonzept von Kubernetes analysiert, um die Sicherheitsmerkale der Software vorzustellen. Im anschließenden Kapitel erfolgen die Beschreibung und Analyse der Fallstudie. Auf Grundlage dessen werden anschließend Experteninterviews durchgeführt. Dieses Kapitel beinhaltet die Vorstellung der Experten, die Auswertung der Interviews sowie die Einordnung in den Kontext der Leitfrage.

Abschließend soll das nächste Kapitel die gewonnen Kenntnisse diskutieren und zusammenführen, bevor im letzten Abschnitt die Ergebnisse vorgestellt werden, die Hypothesen beantwortet sowie eine Empfehlung und Implikation für dem Umgang von Kubernetes in der Praxis und Forschung geben wird.

# Theorie

Diese Arbeit soll dem Leser alle notwendigen Informationen liefern, sodass keine eigene Recherche notwendig ist. Daher wird nachfolgend der aktuelle Forschungsstand zu den wichtigsten Themen dargestellt.

## Software-Bereitstellung

Die Software-Bereitstellung (engl. Software-Deployment) beschreibt einen Prozess, der sicherstellt, dass eine Anwendung bzw. Software aus der Entwicklungsumgebung auf eine Produktionsumgebung überführt wird, sodass der Endnutzer diese nutzen kann. Die Aufgabe eines Deployment-Prozesses beschränkt sich nicht nur auf die Installation auf dem Zielsystem, sondern umfasst eine Vielzahl von weiterführenden Aufgaben. Dazu gehören die Umgebungskonfiguration, automatisierte Testabläufe oder vordefinierte deployment-spezifische Sicherheitsmaßnahmen, wie Rollbacks oder Back-Ups (vergleiche (vgl). [1, pp. 8-11]).

In modernen Unternehmen ist eine effiziente und automatisierte Software-Bereitstellung von hoher Bedeutung. Die Implementierung der kontinuierlichen Lieferung und Aktualisierung von Software, auch CI/CD genannt, ermöglicht dem Unternehmen eine schnelle Reaktionszeit auf Kundenanforderungen oder Marktveränderungen. Durch den Einsatz dieser Werkzeuge kann ein Unternehmen seine Bereitstellungsprozess zuverlässig und schnell konzipieren (vgl. [1, p. 11]). Die Software-Bereitstellung entwickelt sich stetig weiter. Von der Übertragung mittels Lochkarten und Magnetbändern in den 1960er über die Verarbeitung von Disketten und CDs in den 1980er-und 1990er-Jahren bis hin zur digitalen Verarbeitung von Installationsdateien Anfang der 2000er. Seit 2010 haben Cloud-Computing, die Einführung von DevOps-Prozessen sowie vor allem die Container-Orchestrierung das Software-Deployment durch die Einführung von CI/CD revolutioniert.

CI/CD ist eine Sammlung von Werkzeugen und Techniken, die sowohl die Entwicklung als auch die Auslieferung von einer Software verbessern sollen. Continuous Integration (CI), also die kontinuierliche Integration, ist ein automatisierter Prozess, der Entwicklungsteams dabei unterstützt Codeänderungen kontinuierlich zusammenzuführen. Durch automatisierte Tests und Konfliktprüfungen können mehrere Entwickler zeitgleich an derselben Codebasis arbeiten, ohne dass es zu Problemen bei der Zusammenführung gibt. Dies ist vor allem in der heutigen Zeit wichtig, da in der modernen Anwendungsentwicklung oft größere Teams in einem Projekt arbeiten (vgl. [1, p. 15]). CD steht sowohl für Continuous Delivery als auch Continuous Deployment. Continuous Delivery setzt direkt nach CI an. Es bereitet den zuvor getesteten Code final für ein Produktions-Deployment vor. Durch das Bereitstellen in einer Abnahmeumgebung für die Qualitätssicherung und den Kunden, können Akzeptanz-, System und End-to-End-Tests durchgeführt werden. Des Weiteren sind Integrationstests sowie Code-Scans Teil dieses Prozess, um zu überprüfen, ob der Code den eigenen Sicherheitsstandard entspricht. Hierfür für der Code nach nicht verschlüsselten Passwörter oder Secrets und potenziellen Sicherheitslücken geprüft. Continuous Deployment ist der letzte Schritt von CI/CD. Er erweitert das Continuous Delivery um das Deployment in eine Produktionsumgebung. Oft ist dieser Schritt an einen Freigabeprozess gekoppelt, sodass ein Produktions-Deployment nur mit der entsprechenden Autorisierung stattfinden kann. Eine CI/CD-Pipeline ist ein Workflow, der all diese Schritte zusammenführt und weiter automatisiert. So werden Entwicklungsteams entlastet und müssen keine manuellen Aufgaben bei der Code-Bereitstellung übernehmen. Abbildung 1 zeigt die Schritte einer CI/CD übersichtlich (vgl. [1, pp. 19-23]).

Ein Bild, das Text, Screenshot, Schrift, Rechteck enthält.

Automatisch generierte Beschreibung

Abbildung 1 - CI/CD-Pipeline

Quelle: Red Hat, 2020, https://www.redhat.com/de/topics/devops/what-is-continuous-delivery

Für ein Deployment wird ein Zielsystem benötigt. In einer CI/CD-Pipeline werden für die Test- und Produktionsumgebungen jeweils mindestens ein Server benötigt, auf die die Anwendung bei einem Deployment installiert werden muss. Dafür ist die Container-Orchestrierung zuständig.

Die Containerisierung ermöglicht es eine Software sowie ihre Konfigurationsdateien und weitere Abhängigkeiten in isolierte und portable Container zu verpacken. Container können mit virtuellen Maschinen verglichen werden. Sie virtualisieren das Betriebssystem eines Server sind jedoch voneinander und der Host-Umgebung isoliert (vgl. [2, pp. 10-13]). Eine Anwendung kann auf mehrere Container skaliert werden. Dies sorgt dafür, dass die Anwendung auf mehreren unabhängigen Instanzen läuft. Kommt es bei einer dieser Instanzen zu Problemen, läuft die Anwendung auf den anderen Container problemlos weiter. Da sich die Container-Technologie in der modernen Anwendungsentwicklung durchgesetzt hat, stieg der Bedarf an einem effizienten Management der Container-Instanzen. Orchestrierungstools wie Kubernetes oder Docker Swarm haben sich durchgesetzt. Orchestrierung beschreibt einen Prozess zur Koordination und Steuerung verschiedener Systeme, um ein komplexes Ziel zu erreichen. Dabei werden einzelne Prozessschritte aufeinander abgestimmt gesteuert und ihre Aufgaben effizient auszuführen. Im Bezeug auf die Containerisierung sorgt die Orchestrierung für das Starten von Container, die Skalierung der Anzahl von Container je nach Last, die Verwaltung der Netzwerkeinstellungen, die Kommunikation zwischen verschiedenen Anwendungen und die Sicherheit der einzelnen Container (vgl. [3]).

In CI/CD-Pipelines und in Container-Orchestrierungstools wie Kubernetes sind diverse Sicherheitsmechanismus verankert. Dies ist für Unternehmen und den Kunden von großer Bedeutung. Der gesamte Prozess hat sich von vielen manuellen Schritten zu einem komplex aufgebauten automatisierten Prozess entwickelt. Die Implementierung verschiedenster Workflows bringt Vorteile mit, birgt aber auch Risiken, da jedes System angreifbar oder fehlerhaft sein kann. Wenn automatisierte Tests oder Code-Scans falsch oder fehlerhaft konfiguriert werden, können Entwickler, die sich auf dessen Korrektheit verlassen, unbewusst Sicherheitslücken verursachen. Daher ist es wichtig den gesamten Prozess so sicher wie möglich zu gestalten.

## IT-Sicherheit

IT-Sicherheit (engl. IT-Security) umfasst sämtliche Maßnahmen, die Informationen und Systeme vor unerlaubten Zugriff sowie Datenmanipulation zu schützen. Moderne Unternehmen profitieren von ihrer Datenintegrität und -verfügbarkeit und sind deshalb auf fortschrittliche Sicherheitskonzepte angewiesen. In modernen Zeiten nehmen die Bedrohungen für IT-Systeme in Komplexität und Quantität kontinuierlich zu. Um für neue Angriffsvektoren eine Lösung bieten zu können, entwickelt sich die IT-Sicherheit auf technischer und organisatorischer Ebene weiter (vgl. [5, Kapitel 1.1 – 1.3]). Das zentrale Konzept, dass zur Konzeption von Sicherheitsmaßnahmen dient, sind die Schutzziele der IT-Sicherheit. Sie bilden den Rahmen, um Gefahren zu identifizieren und Lösungen zu entwickeln. Die Hauptziele sind: Vertraulichkeit, Integrität und Verfügbarkeit (vgl. [4, p. 439]). Diese und weitere Ziele der IT-Sicherheit werden im Folgenden beschrieben.

### Vertraulichkeit

Das Ziel der Vertraulichkeit besteht darin, Daten vor unbefugtem Zugriff zu schützen (vgl. [5, p. 7]). Für Unternehmen ist der Schutz vertraulicher Daten von großer Bedeutung. In erster Linie soll dadurch das Unternehmen und ein möglicher Vorteil gegenüber Mitbewerber geschützt werden. Allerdings ist das Umsetzen von entsprechend Maßnahmen durch die Einführung des Geschäftsgeheimnisgesetzes von 2019 auch gesetzlich geregelt. Das Gesetzt verpflichtet dazu Geheimhaltungsmaßnahmen zu ergreifen, damit der Schutz von vertrauliche Geschäftsinformationen rechtlich geregelt ist. Als vertrauliche Informationen bezeichnet man Daten, die vor unbefugter Offenlegung bewahrt werden und deren Zugang zur Verarbeitung restriktiv geregelt sein muss[[1]](#footnote-1).

Zur Gewährleistung der Vertraulichkeit können Zugangskontrollen und Verschlüsselung beitragen (vgl. [4, pp. 441-442]).

*Zugangskontrollen* sollen sicherstellen, dass nur autorisierte Anwender und Systeme Zugriff auf entsprechende Daten haben. Physische Zugangskontrollen in Form von Zugangskarten, biometrischen Scannern oder logische Zugangskontrollen durch Vergabe von Berechtigungen und Benutzerrollen in IT-Systemen können dies sicherstellen.

*Authentifizierungsmethoden* dienen zu Überprüfung der Identität beim Zugriff auf Daten. Man kann dies auch als Erweiterung der Zugangskontrollen verstehen. Durch eine zusätzliche Überprüfung in Form von Passwortschutz, biometrische Verfahren oder durch eine Multi-Faktor-Authentifizierung (MFA) soll das Risiko auf unbefugtem Zugriff reduziert werden.

Mit der *Verschlüsselung* von Daten werden Informationen in eine nicht lesbare Form umwandelt. Nur durch die Authentifizierung mit einem speziellen Passwort oder Schlüssel werden diese Informationen für den Nutzer in Klartext freigegeben. Durch die Verschlüsselung werden ruhende Daten (z.B auf Festplatten) und Daten im Netzwerkverkehr geschützt.

### Integrität

Unter der Integrität von Daten versteht man in erster Linie die Unversehrtheit, also die Korrektheit von Daten. Das heißt, alle durchgeführten Änderungen an Daten sollen erkannt und nachvollziehbar sein, ebenso sollen nur gewollte Änderungen möglich sein. Ungewollte Änderungen von Daten müssen verhindert werden, um einen hohen Grad an Datenintegrität vorweisen zu können. Die Wahrung der Integrität wird durch den Einsatz von Prüfmechanismen wie Hashing und digitale Signaturen gewährleistet (vgl. [6, p. 16]).

*Hashing* ist ein Mechanismus, bei dem beliebig große Daten in eine feste Zeichenfolge, auch „Hash“ genannt, umwandelt werden. Jede Änderung sorgt für die Generierung eines neuen anderen Hashwertes (vgl. [5, Kapitel 4.3]).

Eine *digitale Signatur* ist ein kryptografisches Verfahren. Durch asymmetrischer Verschlüsselung soll es die Datenintegrität wahren. Ein Nutzer erstellt einen privaten Schlüssel (engl. Private Key), der geheim bleibt, damit nur dieser die Daten mit seiner Signatur verschlüssel kann. Des Weiteren erstellt der Nutzer einen öffentlichen Schlüssel (engl. Public Key). Diesen kann der Nutzer öffentlich zur Verfügung stellen, damit Empfänger seine Daten entschlüsseln können. Schlägt das Entschlüsseln fehl, so weiß der Empfänger, dass die Daten manipuliert wurden (vgl. [5, Kapitel 5.2]).

### Verfügbarkeit

Durch das Schutzziel Verfügbarkeit soll sichergestellt werden, dass Daten und Systeme zu den gewünschten Zeiten von Personen genutzt werden können (vgl. [5, Kapitel 8.1.1). Kritische Systeme müssen unter Umständen dauerhaft erreichbar sein. Ein Ausfall oder Störung kann für ein Unternehmen einen erheblichen Schaden verursachen. Durch das Durchführen einer Risikoanalyse können unternehmensspezifische Maßnahmen abgeleitet werden. Dabei kommt es stark darauf an, wie man die unternehmenseigenen Systeme betreibt. Setzt ein Unternehmen auf eine selbst betriebene On-Premises-Lösung sind redundante Hardware und Netzwerke mit einem Failover-System, bei der eine produktions-gleichen Infrastruktur aufgebaut wird, zur der automatisch im Ausfall gewechselt wird, sowie Lokale Backups und ein Disaster-Recovery-Plan gängige Methoden zum Wahren der Verfügbarkeit (vgl. [5, Kapitel 8.1.3]).

Betriebt ein Unternehmen seine Infrastruktur über einen Dienstleister in der Cloud, so kann es diese entweder selbst verwalten (Infrastructur as a Service) oder die Verwaltung ebenfalls an den Dienstleister abgeben (Platform as a Service oder Software as a Service).

***IaaS (Infrastructure as a Service)*** Stellt IT-Ressourcen wie Server und Speicher bereit. Unternehmen hat Kontrolle über Betriebssysteme und Anwendungen (vgl. [7, p. 15]).

***PaaS (Platform as a Service)*** Bietet eine Plattform für die Entwicklung sowie Ausführung von Anwendungen ohne Infrastrukturverwaltung (vgl. [7, p. 15]).

***SaaS (Software as a Service)*** Bereitstellung vollständiger Anwendungen über das Internet, ohne Verwaltungsaufwand für den Nutzer (vgl. [7, p. 16]).

Die Maßnahmen zum Wahren des Schutzziel Verfügbarkeit sind jedoch gleich. Multi-Region- bzw. Multi-Zone-Bereitstellung sind ein Konzepte, bei der Anwendungen auf verschiedene Regionen oder Zone verteilt werden, damit eine Verfügbarkeit dauerhaft gewährleistet werden kann. Durch das Aufsetzen von Back-Up-Prozessen kann ein Datenverlust minimiert werden. Des Weiteren können durch automatisiertes Monitoring im Fehlerfall oder bei Störung bereits vordefinierte Prozesse zur Lastverteilung oder Wiederherstellung angestoßen werden.

### Erweiterte Schutzziele –rausnehmen?

Für Unternehmen kann es in bestimmten Kontexten sinnvoll sein noch zusätzliche Schutzziele zu definieren, um den Schutz zu optimieren. Je nach Literatur werden unterschiedliche zusätzliche Ziele angegeben, wie:

***Authentizität*** Sicherstellung der Identität des Kommunikationspartners, Person oder System

***Nichtabstreitbarkeit*** Versand und Empfang einer Information soll nachträglich nicht abstreitbar sein

***Zuverlässigkeit*** Sicherstellung der technischen Funktionsfähigkeit von IT-System, Erweiterung des Schutzziel Verfügbarkeit

***Intervenierbarkeit*** Auskunft über verarbeitete Daten und Möglichkeit zur Korrektur

***Transparenz*** Verarbeitung von personenbezogen Daten muss nachvollziehbar sein

Eine vollständige Liste aller Schutzziele kann daher nicht angegeben werden.

## Kubernetes

Kubernetes (k8s) ist ein Open-Source-Container-Orchestrierungssystem. Es soll die Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen automatisieren. Es ist eine Weiterentwicklung des Google-internen Programm Borg (vgl. [2, p. 19]). 2014 als Open-Source-Projekt von Google vorgestellt, wurde 2015 eine erste Version veröffentlich und an die neu gegründete Cloud Native Computing Foundation (CNCF), ein Projekt zur Förderung von Cloud-Technologien, gespendet. In der CNCF sind die größten Tech-Unternehmen der Welt zusammengeschlossen (vgl. [8, p. 8]).

### Kubernetes-Architektur

Ein Kubernetes Cluster besteht aus einem Control Panel und Nodes. Ein Cluster ist eine Gruppe von physischen und virtuellen Maschinen, die zusammenarbeiten, um Anwendungen und Ressourcen zu steuern. Das Control Panel ist die höchste und zentrale Verwaltungsebene. Hier werden Entscheidungen getroffen und der Status des gesamten Clusters überwacht und bei Störungen entsprechende Maßnahmen eingeleitet. Sie besteht aus mehreren Komponenten (vgl. [2, p. 22]).

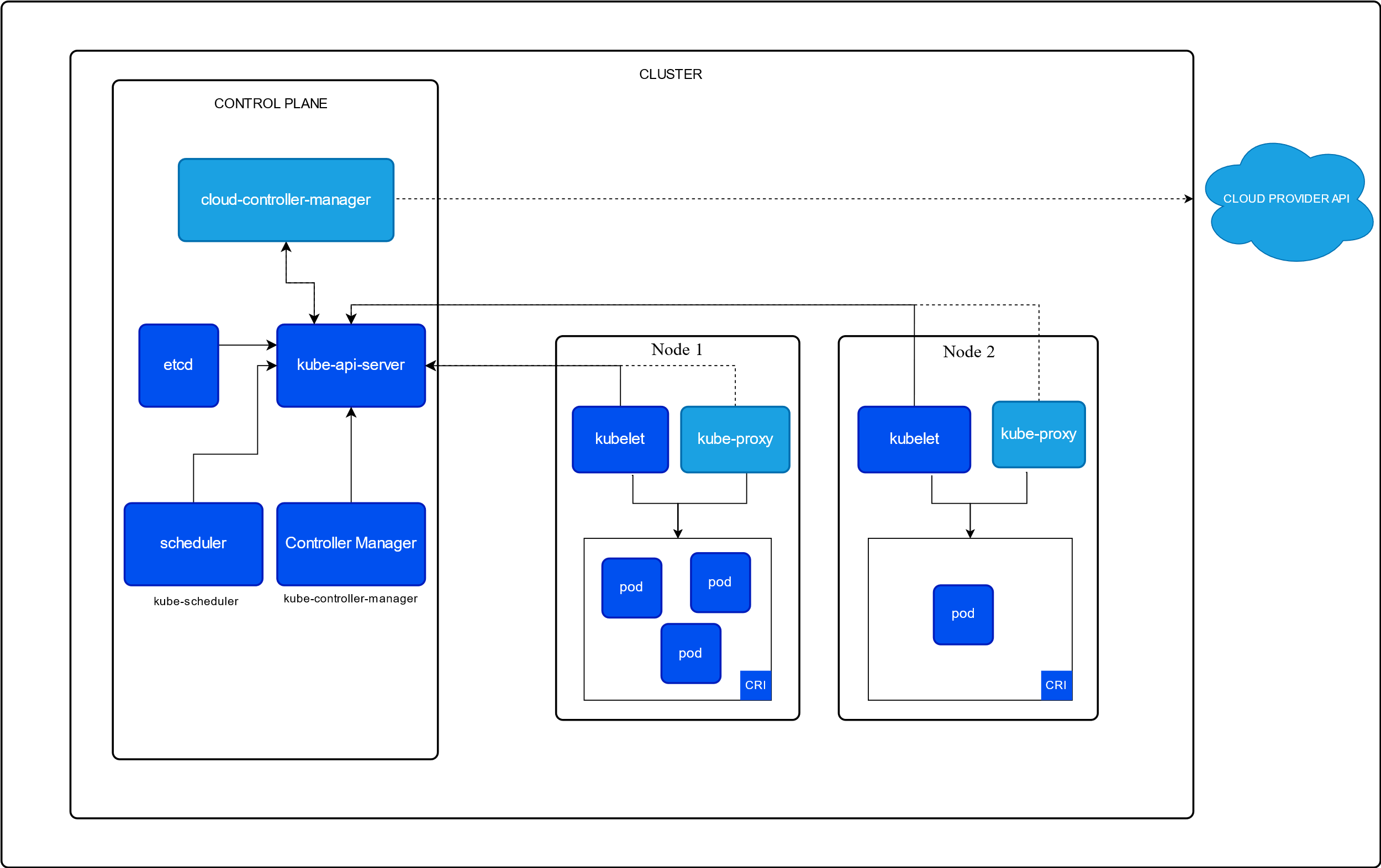


Abbildung 2 - Kubernetes Architektur

Quelle: Kubernetes, 2024, https://kubernetes.io/docs/concepts/architecture/

Um das Control Panel als Nutzer ansprechen zu können, wurde ein Application Programming Interface (API) bereitgestellt. Eine API ist eine Schnittstelle, die es ermöglicht mit der Anwendung zu kommunizieren und Daten auszutauschen. Die Kubernetes API fungiert durch einen API-Server, den kube-apiserver, der Anfragen und Daten validiert (vgl. [8, p. 13]).

Zum Speichern der Konfigurationsdaten nutzt Kubernetes eine etcd-Datenspeicher, dies ist eine Open-Source-Schlüsselwert-Datenbank, auch Key-Value-Store genannt. Etcd ist ein konsistenter Datenspeicher, der für die Konfiguration von verteilten Systemen entwickelt wurde (vgl. [9, p. 10]). Speziell im Kontext der Zusammenarbeit in einem Kubernetes Cluster sichert und repliziert er die Daten und Informationen über alle Instanzen hinweg und dient als „Single Point of Truth“ – also als einziger und allgemeingültiger Datenbestand. Ohne diesen Speicher kann das Cluster nicht existieren. Das Verteilen des etcd-Speichers auf mehreren Instanzen ist wird daher empfohlen (vgl. [8, p. 13]). Das Bereistellen des Speichers auf nur eine Instanz ist möglich, birgt aber Verfügbarkeitsrisiken für das gesamte Cluster, wenn diese Instanz ausfällt.

Ein Pod im Kubernetes-Kontext ist eine Sammlung von Container auf denen Anwendungen laufen können. Ein Pod ist die kleinste deploybare Einheit in einem Kubernetes Cluster. Pods werden in einem Node (Arbeitsknoten) bereitgestellt (vgl. [10]). Ein Node ist die kleinste Recheneinheit in einem Cluster, es enthält verschiedene Komponenten (vgl. [11]). Der kube-scheuduler bestimmt auf welchem Node ein neuer Pod ausgeführt wird. Unter Berücksichtigung diverser Faktoren wie Ressourcenanforderungen, Hard- oder Software-Einschränkungen, Spezifikationen oder Datenstandort entscheidet der Scheduler auf welchem Node der Pod bereitgestellt wird (vgl. [2, pp. 350-351]).

Der kube-controller-manager ist eine Sammlung von Controllern, die das Cluster am Laufen halten. Die Controller warten auf Anfragen an die API, um dann die gewünschten Änderungen umzusetzen. Es gibt einen Knoten-Controller, der den Status der verschiedenen Nodes überwacht und auf Ausfälle durch das Starten weitere Node reagiert. Der Replication-Controller sorgt dafür das die konfigurierte Anzahl an Pod-Repliken in einem Node laufen und wird von einem Replication-Manager gesteuert. Zum Verknüpfen der Anwendungen mit Pods untereinander im Cluster gibt es den Endpoint-Controller. Zur Verwaltung der Namespace und Volumes gibt es jeweils einen eigenen Kontroller. Des Weiteren gibt es einen Deployment-Controller, der die Rollouts neuer Anwendungsversionen koordiniert. Zur Überwachung von Pods gibt es zusätzlich noch den Status-Controller. Zum Starten von Kubernetes-Jobs, einmalige und kurzfristige Aufgaben, gibt es den Job-Controller. Der Dienst-Controller kann Dienste wie das LoadBalacing, also die Lastverteilung, steuern (vgl. [2, pp. 352-257]).

Die letzte Komponente des Control Panel ist der cloud-controller-manager. Dies ist ein separater Controller der speziell für die Integration von Kubernetes mit Cloud-Provider entwickelt wurde. Durch ihn kann das gesamte Cluster an die API eines Cloud-Providers angebunden werden. Er ist horizontal skalierbar, um mehrere Instanzen gleichzeitig laufen zu lassen. So wird die Leistung verbessert und Fehler können toleriert werden. Der Controller ist nur verfügbar, wenn Kubernetes in der Cloud läuft. Wenn Kubernetes on-premises läuft, ist dieser Controller nicht vorhanden. Er kann verschiedene Aufgaben übernehmen, die über einzelne Controller verwaltet werden. Der Node-Controller überwacht durch die API des Cloud-Provider den Status der Nodes. Die Kommunikation zwischen Nodes und der entsprechenden Cloud-Umgebung wird der Route-Controller verwendet. Zur Lastverteilung innerhalb der Cloud wird der Service-Controller verwendet (vgl. [12]).

Ein Node besteht nicht nur aus einem oder mehreren Pods. Er besitzt weitere Komponenten. Dazu gehören kubelet, CRI (Container Runtime Interface) und optional kube-proxy (vgl. [2, p. 340]).

Kubelet ist ein Agent, der in jedem Node läuft. Er stellt in erster Linie sicher, dass die Pods laufen. Dafür betrachtet er die Pod-Spezifikationen (PodSpec) (vgl. [2, p. 357]). Diese werden in Form einer YAML oder JSON von Entwicklern an den kube-apiserver bereitgestellt. YAML (YAML Ain’t Markup Language) und JSON (JavaScript Object Notation) sind formatierete Textformate, die zur Übertragung von strukturierten Daten verwendet werden. Beide Formate werden von der Kubernetes-API akzeptiert (vgl. [2, p. 69]).

Das Container Runtime Interface ermöglicht es unterschiedliche Container-Software in das Kubernetes-Cluster zu integrieren. Die Container und deren Umgebung werden nicht direkt durch Kubernetes bereitgestellt, daher braucht man diese Schnittstelle, um eine zusätzliche Software dafür anzubinden (vgl. [9, p. 12]). Die bekanntesten Runtimes sind die Docker Engine, containerd oder die Kubernetes-eigenentwicklung CRI-O.

Kube-Proxy übernimmt die Netzwerkkommunikation innerhalb des Clusters und mit externen Netzwerken. Er läuft auf jedem Node. Ein Proxy ist ein Vermittler, der Anfragen zwischen Client und Server weiterleitet. Der Kube-Proxy ist eine optionale Komponente im Node (vgl. [2, pp. 358-359]). Eine andere Möglichkeit einen Proxy im Netzwerk einzubinden ist ein Plugin in Kubernetes einzubinden. Ein Plugin ist eine Software-Erweiterung, die der Hauptanwendung zusätzliche Funktionen zur Verfügung stellt (vgl. [12]).

### Kubernetes-Konzept

Zur Verwaltung von Containern werden neben Pods noch weitere grundlegende Konzepte innerhalb eines Clusters eingehalten, um die Verwaltung performant zu halten.

Durch Namespaces ist es möglich innerhalb eines Clusters, verschiedene Umgebungen (z.B Test- und Produktionsumgebung) logisch zu trennen und sie so im selben Cluster verwalten zu können. Ein Namespace kann entweder über die Kommandozeile kubectl oder durch das Übergeben einer YAML- oder JSON-Datei an die API erstellt werden. Nach Erstellung eines Namespace kann dieser Parameter in die Deployment-yaml eines Services als Metadaten eingetragen werden (vgl. [2, pp. 87-89]).

Das Service-Konzept ermöglicht es einer Anwendung innerhalb des Clusters eine feste und langlebige IP-Adresse zuzuordnen. Die Anwendung läuft unter Umständen auf mehreren Pods gleichzeitig. Diese können bzw. werden automatisch erstellt und gelöscht, um die Lastverteilung und Verfügbarkeit sicherzustellten. Durch das Definieren eines Services und einer eigenständigen IP-Adresse, kann dauerhaft auf die Anwendung zugegriffen werden (vgl. [8, pp. 22-23]).

Um Daten dauerhaft und unabhängig vom Lebenszyklus eines Containers oder Pods zu speichern, nutzt Kubernetes Volumes. Dies ist eine externe Speicherressource, die Pods auch bei Neustart oder Verschiebung zur Verfügung stehen. Ein Volume wird an einen Pod gebunden. Die sich darin befinden Container können dann über einen festgelegten Pfad auf ihre Daten zugreifen. Wird der Pod neugestartet ordnet kubelet das Volume wieder zu. (vgl. [2, pp. 178-181]).

Secrets oder ConfigMaps werden in Kubernetes in der etcd-Datenbank gespeichert. Passwörter, Tokens oder Zertifikate werden in Kubernetes als verschlüsselte Secrets gespeichert. ConfigMaps sind allgemeine Konfigurationsdateien und werden standardmäßig nicht verschlüsselt abgelegt. Die Daten können als Umgebungsvariable oder über ein Volume an den Pod übergeben werden (vgl. [9, pp. 17-18]).

# Methodik

In diesem Abschnitt werden das Forschungsvorgehen und die Forschungsmethoden vorgestellt.

## Forschungsvorgehen

Nachfolgend zum theoretischen Grundwerk soll durch eine weiterführende Literaturrecherche das Sicherheitskonzept von Kubernetes analysiert werden. Dies ist notwendig, um für die Fallstudien und vor allem Experteninterviews eine geeignete Wissensgrundlage zu erarbeiten. Die Notwendigkeit zeichnet sich dadurch aus, dass durch die Analyse des Sicherheitskonzepts nicht nur potenzielle Lücken und Risiken aufgedeckt, sondern auch Stärken und gut umgesetzte Sicherheitsmaßnahmen identifiziert, werden können. Diese können dann gezielt in den Experteninterviews besprochen und in den Fallstudien untersucht werden.

## Literaturrecherche

Die Literaturrecherche soll zur Erarbeitung der theoretischen Grundlage sowie der analytischen Betrachtung des Kubernetes-Sicherheitskonzept dienen. Durch diese sollen der aktuelle Forschungsstand im Bereich der IT-Sicherheit und Software-Bereitstellung dargestellt werden. Des Weiteren soll die Literaturrecherche auch eine Rolle bei der Erstellung der Experteninterviews dienen und dabei helfen den Deployment-Prozess ohne Kubernetes aufzusetzen. Die Messverfahren zur Bewertung der Fallstudien sollen ebenfalls durch die Literaturrecherche ermittelt werden. Vor allem die systematische Literaturrecherche, bei der Online-Bibliotheken und Datenbanken nach vorher definierten Schlüsselwörtern durchsucht, um möglichst viel Literatur zu dem gewählten Themen zu finden, soll für diese Forschungsarbeit genutzt werden.

## Fallstudien

Es sollen Messungen für einen

1. *Deployment-Prozess mit Kubernetes*
2. *Deployment-Prozess ohne Kubernetes*

durchgeführt werden. Faktoren für diese Messung sollen die zentrale IT-Schutzziele (Verfügbarkeit, Integrität und Vertraulichkeit) sein. Die genauen Parameter zur Untersuchung werden im Rahmen der hierfür aufgestellten Hypothesen vorgestellt. Teil dieses Abschnittes soll ebenso die Ermittlung und Vorstellung der Messverfahren sein.

## Experteninterviews

Die Ergebnisse der Literaturrecherche und der Fallstudien sollen mit den Experten diskutiert und durch diese eingeschätzt werden. Die Experteninterviews sollen mit der qualitativen Inhaltsanalyse nach Mayring ausgewertet werden. Diese Methode folgt einem festen Schema zur Auswertung von textbasierten Daten und eignet sich dementsprechend für die Auswertung von Experteninterviews. Nach Festlegung des zu untersuchenden Materials, wird ermittelt, wie und warum das Material ausgewertet wird. Anschließend wird der Charakter des textes sowie die Analyserichtung festgelegt. Mayring bietet drei verschiedene Auswertungstechniken an: Die Zusammenfassung, die Explikation und die Strukturierung. Je nach vorliegendem Material muss eine passende Technik ausgewählt werden. In der Regel werden alle drei Methoden in Kombination verwendet.

## Hypothesen

Durch das Forschungsvorgehen sollen folgende Hypothesen untersucht werden:

1. *Deployment mit Kubernetes erhöht die Verfügbarkeit bei der Software-Bereitstellung.*
2. *Deployment mit Kubernetes erhöht die Integrität bei der Software-Bereitstellung.*
3. *Deployment mit Kubernetes erhöht die Vertraulichkeit bei der Software-Bereitstellung.*

Mögliche Ergebnisse sind jeweils: wird besser, bleibt gleich, wird schlechter.

Die Hypothesen zur Verfügbarkeit und Integrität sollen durch die Messung technischer Parameter bei Deployments durchgeführt werden.

* Verfügbarkeit: Verfügbarkeitskennzahlen (Uptime, Downtime), Anzahl und Dauer von Ausfällen
* Integrität: Vergleich Fehlerquote, Häufigkeit bzw. Gründe für Rollbacks

Die Vertraulichkeit soll theoretisch untersucht werden und nicht gemessen werden, da sich dies bei einem Deployment nicht verändert.

Für alle drei Hypothesen soll sowohl die Auswertung der Experteninterviews sowie die Literaturrecherche zusätzlich betrachtet werden.

# Analyse des Kubernetes-Sicherheitskonzept

Die Analyse des Sicherheitskonzept umfasst sowohl allgemeine Sicherheitsmerkmale wie Authentifizierungs-, Passwort- und Netzwerkmanagement, aber auch die spezifischen Funktionen, um die Sicherheit von Deployments zu gewährleisten. Diese Forschungsarbeit untersucht zwar explizit die Sicherheitsfunktionen von Kubernetes bei der Software-Bereitstellung, allerdings soll eine Analyse des allgemeinen Sicherheitskonzept sicherstellen, dass die generelle Nutzung von Kubernetes kein Sicherheitsrisiko darstellt. Das Schutzziel Vertraulichkeit kann vor allem durch die allgemeineren Sicherheitsmerkmale theoretisch untersucht werden, wohingegen die Verfügbarkeit und Integrität primär im Verlauf eines Deployments von großer Bedeutung sind. Die einzelnen Mechanismen werden vorgestellt und ihr jeweiliger Einfluss auf die IT-Schutzziele analysiert.

## Sicherheitsmerkmale von Kubernetes

Nachfolgenden werden die Zugriffskontrollen, die Netzwerkkonfiguration sowie die Sicherheitsmechanismen von Pods, Secrets und Images untersucht.

### Zugriffskontrolle

In Kubernetes ist die Authentifizierung von der Autorisierung getrennt. Zur Authentifizierung unterstützt Kubernetes verschiedene Mechanismen. Vor allem die Integration externer Dienste wie *OpenID Connect* oder *OAuth2* hat sich durchgesetzt. Diese ermöglichen eine Authentifizierung über Google, Entra ID oder GitHub. Die Autorisierung wird in der Regel durch die rollenbasierte oder attributbasierte Zugriffskontrolle verwaltet.

#### Rollenbasierte Zugriffskontrolle

In Kubernetes wird die rollenbasierte Zugriffskontrolle (Role Based Access Control, RBAC) oft als zentrales Sicherheitskonzept zur Verteilung von Benutzerrechten verwendet, da es nativ in Kubernetes integriert ist. Es gibt auch andere Methoden, jedoch hat sich RBAC als „Best-Pratice“ etabliert (vgl. [9, pp. 19-20]). Die Idee von RBAC ist es Berechtigungen an Rollen und nicht an Benutzer zu binden, nur die notwendigsten Berechtigungen zu verteilen und diese zentral zu verwalten. RBAC ist in Kubernetes über die API *rbac.authorization.k8s.io* verwaltbar und legt insgesamt vier Objekte zur Rollenverteilung fest:

Eine *Role* oder *ClusterRole* beinhalten eine Reihe von Rechten. Eine Role legt nur Berechtigungen für einen Namespace fest, eine ClusterRole für das gesamte Kubernetes-Cluster. Ein *RoleBinding* oder *ClusterBinding* verknüpft eine Role mit einen Benutzer, eine Gruppe oder einem System. Sie geben an wer die in der Role definierten Berechtigungen wirklich nutzen darf. Analog zur Role und ClusterRole, gilt ein RoleBinding nur für den Namespace und das ClusterBinding für den gesamten Cluster (vgl. [2, p. 386 ff.]). Um einem Benutzer eine Rolle zuzuweisen, muss diese zunächst erstellt werden.

Mit *kubectl apply -f role.yaml* kann eine erstellte Role an die Kubernetes-API übergeben werden. Der Befehl setzt sich aus -f für file und dem tatsächlichen Dateinamen zusammen.

Eine Role kann so definiert werden:

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

  namespace: my-namespace # Namespace der Rolle

  name: reader             # Name der Rolle

rules:

- apiGroups: [""]

  resources: ["pods"]      # Zielressourcen: Pods

  verbs: ["get", "list"]   # Erlaubte Aktionen: Lesen (get) und auflisten (list)

Listing 1 - Erstellung der Rolle

Nun muss diese Role an einen Benutzer zu gewiesen werden. Das RoleBinding wird ebenfalls mit *kubectl apply -f rolebinding.yaml* an die API übergeben:

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

  name: reader-binding      # Name des RoleBindings

  namespace: my-namespace   # Namespace der Verknüpfung

subjects:                   # Subjekte, die Zugriff erhalten

- kind: User

  name: testuser            # Name des Benutzers

  apiGroup: rbac.authorization.k8s.io

roleRef:                    # Verweis auf die Rolle

  kind: Role

  name: reader              # Name der verknüpften Rolle

  apiGroup: rbac.authorization.k8s.io

Listing 2 - Zuweisung der Rolle

Die Zuweisung und Erstellung von ClusterRoles folgt dem gleichen Prinzip, nur die Angaben in der yaml-Datei müssen entsprechend auf Cluster-Ebene angepasst werden (vgl. [13]).

Kubernetes setzt bei RBAC auf einige bewährte Praktiken. Eines davon ist das „Least Privilege-Principle“ also das Prinzip der minimalen Privilegien. Nutzer und Systeme sollten nur die notwendigsten Rechte auf der kleinsten Ebene zugewiesen bekommen. Damit einher geht, dass Berechtigungen, soweit es geht auf der Namespace-Ebene zugewiesen werden sollen. Die Vergabe von Wildcard-Berechtigungen soll, wenn möglich vermieden werden. Wildcard-Rechte werden durch die Verwendung eines ‘\*‘ bei den Parameter *resources* und *verbs* vergebe. So werden die Rechte nicht nur für bestehende, sondern auch für zukünfigt festgelegte Objekte im Cluster festgelegt. Benutzer sollten die Rolle des *cluster-admins* nur zugewiesen bekommen, wenn dies explizit gefordert ist. Die Gruppe *system:masters*, welche alle Konfigurationen des RBAC ignoriert und dem Benutzer uneingeschränkten Zugang gewährt, sollte ebenfalls vermieden werden (vgl. [13]).

Pods bekommen ihre Rechte auch durch RBAC. Dafür werden Pods an Service Accounts gebunden, denen über RBAC Berechtigungen zu gewiesen werden. Die Anzahl von Nodes auf denen Pods mit weitreichenden Rechte soll minimal gehalten werden, damit ein potenzieller Schadensradius im Angriffsfall niedrig gehalten wird. Alle Änderungen an Pods oder Konfigurationsdateien können über Audit Logs (Überwachungsdateien) von Cluster-Adminstratoren nachvollzogen werden

*Vertraulichkeit*:

Die Implemtierung von Authentifizerungs- und Autorisierungmechanismen schützt das Cluster und dessen Daten vor unbefugtem Zugriff Dritter. Des Weiteren wird durch das Verteilen von Benutzer- und Anwendungsrollen sichergestellt, dass diese nur begrenzten Zugriff auf Ressourcen haben. Die Gefahr der Offenlegung von vertraulichen Daten wird so weiterführend eingeschränkt.

*Integrität*:

Durch die Rollenbaiserte Zugriffskontrollen wird die Integrität von Daten und Ressourcen gewahrt, da diese vor unbeatsichtiger Manipulation geschützt werden. Werden Daten geändert, kann dies über Audit Logs nachvollzogen werden. So kann identifiziert werden, welcher Nutzer die Änderungen durchgeführt hat.

*Verfügbarkeit*:

Indirekt tragen die Zugriffskontrollen auch zur Verfügbarkeit bei. Es wird verhindert, dass unbefugte Nutzer Aktionen ausführen, die direkten Einfluss auf den Betrieb oder ein Deployment haben.

#### Attributbasierte Zugriffskontrolle

Alternativ zu RBAC bietet Kubernetes auch andere Möglichkeiten, um API-Anfragen zu authentifizieren. Vor allem die attributbasierte Zugriffskontrolle (Attribute-based access control, ABAC) ist eine oft gewählte Alternative (vgl. [2, p. 386]). Anders als bei RBAC werden Berechtigungen nicht an Rollen, sondern an Attribute gekoppelt. So können Zugriffsentscheidungen dynamisch geregelt werden. Bei ABAC lassen sich Benutzerattribute (Mitgliedschaften, Identität), Ressourcenattribute (Namespace, Pod, Anwendungen) und Kontextbezogene Attribute (Zeitpunkt, Netzwerk) definieren.

{

   "apiVersion":"abac.authorization.kubernetes.io/v1beta1",

   "kind":"Policy",

   "spec":{

      "user":"testuser",

      "namespace":"\*",

      "resource":"\*",

      "apiGroup":"\*"

   }

}

Listing 3 – ABAC

### Netzwerksicherheit

Die Netzwerk-Architektur eines Clusters kann je nach Anforderungen unterschiedlich sein. Um ein Netzwerk zu konfigurieren, bietet Kubernetes eine Vielzahl von Möglichkeiten. Die Netzwerkkonfiguration der Pods bzw. Container hängt im hohem Maße von der Wahl des *Container Network Interface*-Plugin ab. Je nach Wahl lassen sich bestimmte Netzwerk-Richtlinien einstellen oder nicht. Mit ihnen lässt sich der eingehende (Ingress) und ausgehende (Egress) Datenverkehr genaustens regeln (vgl. [2, p. 434]).

Die Kommunikation zum etcd-Datenspeicher kann durch die Verwendung von *mutual TLS* (mTLS) eingeschränkt werden. mTLS ist eine Erweiterung von TLS. Hier müssen sich Client und Server jeweils gegenseitig authentifizieren, damit eine sichere Verbindung hergestellt werden kann. Der Kubernetes-API-Server und die kubelet-api sind standardmäßig öffentlich erreichbar. Durch die Implementierung von Authentifizierung- und Autorisierungtechniken (vgl. Abschnitt 4.1.1) kann das Cluster von außen und innen gesichert werden, sodass nur autorisierte Personen und Anwendungen Änderungen am Cluster vornehmen können. Zur Authentifizierung von Pods gegenüber der API, werden diese an ServiceAccounts gebunden (vgl. [2, p. 381]). Die Rechte der ServiceAccounts können ebenfalls für RBAC oder ähnliche Plugins vergeben werden (vgl. [13]). Die Verwendung von Namespace sorgt dafür, dass Pods nur auf ServiceAccount innerhalb ihreres Namespace zugreifen können. Ein Pod kann nur auf einen ServiceAccount zugreifen, allerdings kann ein ServiceAccount von mehreren Pod benutzt werden (vgl. [2, pp. 381-382]).

*Vertraulichkeit*:

Durch den Einsatz von CNI-Plugns und ServiceAccounts wird sichergestellt, dass Pods und Container isoliert werden und nur Rechte innerhalb ihreres zugewiesen Bereich erhalten. Pods können nur auf sensible Daten zugreifen, wenn ihn diese Rechte explizit zugewiesen werden. Wird ein Pod kompromittiert, wird durch die Isolierung sichergestellt, dass der Angreifer keinen Zugriff auf das gesamte Cluster hat und Daten einsehen kann. Der Schutz der API verhindert den Zugriff auf vertrauliche Daten von Unbefugten. Der Einsatz von mTLS zur Kommunikation mit dem etcd-Speicher verhindert die Offenlegung von hochsensiblen Cluster-Daten wie Secrets und Konfigurationsdateien.

*Integrität*:

Die Pod-Isolierung und Verwendung con CNI sorgt dafür, dass Daten nicht von Unbefugten manipuliert werden können. Der Schutz des API-Servers und etcd-Speichers verhindert, dass Dritte hochsensible Daten ändern können.

*Verfügbarkeit*:

Durch die Implementierung der vorgestellten Netzwerksicherheits-Mechanismen wird die Verfügbarkeit des Clusters verstärkt, da die möglichen Angriffsflächen minimiert werden. Die Verfügbarkeit profiert davon, da der laufenden Betrieb oder Deployments so nicht bzw. unwahrscheinlicher gestört werden.

### Pod-Sicherheit

Die Sicherheit eines Pods profitiert von den Zugangskontrollen und den Netzwerkeinstellungen. Dennoch gibt es weitere Möglichkeiten einen Pod zu schützen. Kubernetes bietet dafür drei vordefinierte Sicherheitsprofile an: *Privileged*, *Baseline* und *Restricted*. Die Profile legen fest, wie eine Pod-Spezifikation konfiguriert werden darf. Pro Namespace kann ein Sicherheitsprofil gesetzt werden.

Das Privileged-Profil gewährt maximale Berechtigungen und lässt Privilegien-Eskalationen zu. Durch die Zuweisung dieses Profil kann ein Container oder Pod die Pod-Isolierung möglicherweise umgehen (vgl. [14]). Unter einer Privilegien-Eskalationen versteht man im Kubernetes-Kontext die unbefugte Erlangung höher Berechtigungen durch einen Pod. Das Baseline-Profil verhindert bekannte Eskalationen und schränkt die Konfiguration eines Pods minimal ein. Mit dem Restricted-Profil werden die Einstellung eines Pods stark eingeschränkt. Die Profile lassen sich manuell bearbeiten und erweitern.

Die Auswirkung auf IT-Schutzziele hängt von der Auswahl und Konfiguration der Profile ab.

*Vertraulichkeit*:

Die Sicherheitsprofile können gewährleisten, dass Pods nur die minimalen Berechtigungen haben und schränken so den Zugriff auf Daten ein.

*Integrität*:

Das Einschränken der Berechtigungen und die Verstärkung der Pod-Isolierung wird eine Manipulation von Daten eingeschränkt.

*Verfügbarkeit*:

Die Verfügbarkeit profitiert indirekt von diesen Sicherheitsmechanismen. Die Verhinderung von Privilegien-Eskalationen und die erweiterbaren Einstellung bezüglich der Pod-Isolierung verhindern die Störung des Clusterbetriebs.

### Secret-Sicherheit

Ein Secret kann im etcd-Datenspeicher verschlüsselt hinterlegt werden. Werden Secrets im etcd-Speichert abgelegt, kann eine Verschlüsselungskonfiguration in Form einer yaml-Datei an den API-Server übergeben werden. Dort kann festgelegt werden, welche Daten wie verschlüsselt werden sollen. Durch RBAC oder ABAC können die Zugriffe auf Secrets generell verwaltet werden.

Zusätzlich ist die Integration eines Drittanbieter für das Speichern von Secrets möglich. Dafür gibt es das Plugin *Secrets Store CSI Driver*. Dieses ermöglicht es Secrets aus externen Speichern im Cluster bereitzustellen (vgl. [15], Kapitel 1). Cloud-Dienste wie Azure Key Vault, AWS Secrets Manager oder Google Secret Manager bieten eine sichere Möglichkeit zu Ablegen von Secrets. Allerdings ist dafür neben einer Konfiguration im Cluster auch die Einbindungen der Abhängigkeiten im Code-Projekt nötig.

*Vertraulichkeit*:

Die Verschlüssung von sensiblen Daten wie Secrets oder Konfigurationsdateien im etcd-Speicher und in Datenbanken von Drittanbietern, sorgt für eine erhöhe Sicherheit dieser.

*Integrität*:

Des Weiteren sorgt die Verschlüssung und die Zugriffskontrollen auf diese Daten, dass nur befugte Nutzer Änderungen vornehmen können. Durch die Audit Logs wird sichergestellt, dass alle Änderungen an diesen Daten nachvollzogen werden können.

*Verfügbarkeit*:

Wird die Sceret-Verwaltung im etcd-Datenspeicher verwaltet, so kann das Cluster auch bei Netzwerkunterbrechungen zu Dirttanbietern weiterhin auf diese zugreifen. Die Verfügbarkeit der Cluster wird so gewahrt. Bei der Anbindung von Drittanbieter-Software profiertiert man von deren Sicherheits- und Verfügbarkeitsmechanismen. Allerdings besteht die Gefahr, dass das Cluster bei Netzwerkstörung nicht auf die eigenen Secrets zugreifen kann.

### Image-Sicherheit

Ein Image bzw. Container-Image ist eine Datei, die eine Anwendung ausführt. Sie enthält den Code sowie alle Abhängigkeiten, Konfigurationsdateien sowie zusätzliche Bibliotheken, damit die Anwendung auf dem Container ausgeführt werden kann (vgl. [2, p. 15]). Ein Image und dessen Bestandteile können eine Schwachstelle in einem Cluster sein, da sie gegeben, falls eine Hintertür oder Sicherheitslücken für Angriffe bieten. Daher hat Kubernetes mehrere Mechanismen implementiert, um das Image und dessen Verwendung zu überprüfen.

Als bewährte Praktik hat sich zunächst durchgesetzt, dass ein Image nur die notwendigsten Abhängigkeiten enthalten sollte. Braucht man zusätzliche Werkzeuge wie ein Debug-Tool (Werkzeug zum Identifizieren von Fehler) bietet Kubernetes die Möglichkeit diese in einem temporären Container bereitzustellen. Generell wird davon aber in Produktionsumgebungen abgeraten. Im Docker-Umfeld gibt es die Möglichkeit Container immer mit dem aktuellen bzw. letzten Image auszustatten. Diese Funktion ist durch den tag „*latest*“ gegeben. Dadurch kann eine potenzielle Sicherheitslücke entstehen, falls es Angreifern gelingt, ein Image in den Docker-Kontext einzuschleusen. Kubernetes bietet daher die Möglichkeit, durch den Admission Controller oder eine Richtlinie, festzulegen, dass immer eine eindeutige sha256-Nummer angegeben werden muss. Sha256 ein Hash-Algorithmus. Er erzeugt für jeden Images eine Zeichenkette, um Container-Images zu identifizieren. Der Admission Controller ist Teil des API-Servers und nimmt Anfragen entgegen, um diese zu validieren, ggf. anzupassen oder abzulehnen. Er kann des Weiteren dazu verwenden werden, um die digitale Signatur eines Images zu verifizieren. So kann sichergestellt werden, dass ein Image nur von einer vertrauenswürdigen Quelle bereitgestellt werden können (vgl. [15]).

*Vertraulichkeit*:

Die Verwendung von sha-256-Hashwerten und die Sicherstellung der Vertrauenswürdigkeit über den Admission Controller stellt sicher, dass nur vertraute Images im Cluster bereitgestellt werden.

*Integrität*:

Die Manipulation von Images bzw. dessen Tag wird durch den Admission Controller und die Prüfung auf eine vertrauenswürdige Signatur verhindert.

*Verfügbarkeit*:

Der Admission Controller verhindert, dass unsichere Images im Cluster bereitgestellt werden, die zu potenziellen Störungen führen könnten.

## Umgang von typischen Angriffsvektoren mit Kubernetes

Das Framework MITRE ATT&CK® und Microsoft haben jeweils eine Thread Matrix für Kubernetes erstellt, die eine Übersicht über potenzielle Schwachstellen und Angriffsvektoren in einem Cluster aufzeigen. In der Matrix von Microsoft werden sowohl die eigenen Erkenntnisse dargestellt, aber auch die des Frameworks MITRE ATT&CK®.

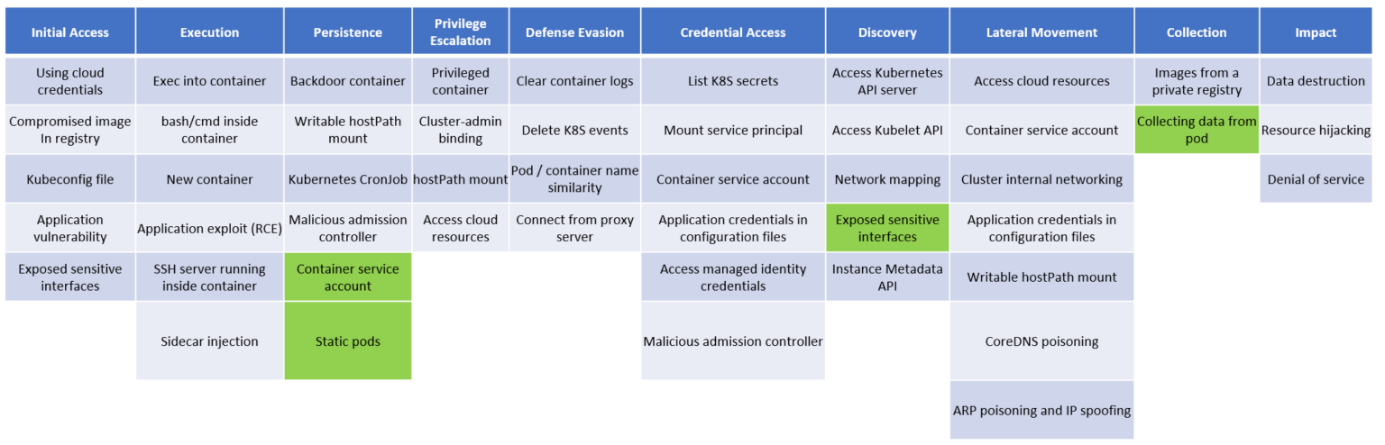


Abbildung 3 - Kubernetes Thread Matrix

Quelle: Yossi Weizmann, Dotan Patrich, Microsoft, 2022, https://www.microsoft.com/en-us/security/blog/2022/12/07/mitigate-threats-with-the-new-threat-matrix-for-kubernetes/

Die Matrix zeigt in der obersten Zeile jeweils die Taktiken, die die übergeordneten Ziele eines Angriffs darstellen. In den jeweiligen Spalten werden die Techniken zur Erreichung der Ziele aufgezeigt. Die grünmarkierten Felder zeigen neue Techniken auf.

Eine Analyse der Matrix wird in dieser Arbeit nicht durchgeführt, da dies zu umfangreich wäre. Die Bereistellung der Matrix zeigt einem Anwender jedoch alle bekannten Sicherheitslücken oder Angriffsmöglichkeiten auf und bietet zudem mögliche Lösungen an. Davon profitieren die IT-Schutzziele und der Anwender, allerdings auch potenzielle Angreifer.

## Sicherheit beim Software-Deployment

Im vorherigen Abschnitt wurden generelle Sicherheitsmechanismen und eine Thread Matrix für Kubernetes dargestellt. Der Kern der Forschungsarbeit liegt auf der Analyse der Sicherheit während eines Software-Deployment. Dies soll in diesem Abschnitt untersucht werden, um eine theoretische Grundlage für die Experteninterviews und die Fallstudien zu schaffen.

Um eine Anwendung in einem Cluster bereitzustellen oder eine Bestehende zu aktualisieren, nutzt Kubernetes die *Rolling-Update-Strategie*. Ein Rolling-Update soll eine Anwendung ohne Ausfallzeiten in einem Kubernetes-Cluster bereitstellen. Dieses und weitere Mechanismen werden nachfolgend vorgestellt und theoretisch bewertet.

### Rolling-Updates

Ein Rolling-Update ermöglicht während der Entwicklung, aber vor allem während des Live-Betriebs einer Anwendung, dass nahtlose Aktualisieren dieser. Heutzutage erwarten Unternehmen, dass ihre Anwendungen immer verfügbar sind. Es soll in der Regel zu keinen Ausfallzeiten kommen, da dies zu betrieblichen bzw. finanziellen Schäden führen kann. Durch das inkrementelle Starten von Pods, während eines Rolling-Updates, wird dies in Kubernetes ermöglicht (vgl. [16, pp. 123-126]). Der Ablauf eines Rolling-Deployments, zu sehen in Abbildung 4, sieht wie folgt aus. Im Kubernetes-Cluster läuft eine Anwendung (v1) in diesem Beispiel auf zwei Pod-Instanzen. Die Anzahl der Instanzen ist konfigurierbar. Wenn nun eine neue Version (v2) der Anwendung bereitgestellt werden soll, startet zunächst eine neue Pod-Instanz mit der aktualisierten Version. Erst wenn diese ohne Probleme starten konnte, werden die bestehenden Instanzen inkrementell abgebaut und eine weitere Instanz der neuen Version gestartet. Es wird also erst eine neue Version erfolgreich bereitgestellt, bevor die vorherige Version heruntergefahren wird.

Ein Bild, das Kreis, Diagramm, Design enthält.

Automatisch generierte Beschreibung

Abbildung 4 - Kubernetes Rolling Update

So kann verhindert werden, dass aufgrund von falscher Konfiguration oder infrastruktureller Probleme zu einer Ausfallzeit kommt. Im Fehlerfall würde das Deployment gestoppt werden und der Pod mit der neuen Version würde sich in einem CrashLoopBackOff befinden. D.h der Pod bzw. Container ist nach dem Start abgestürzt und versucht mit einem BackOff, also einer Wartezeit, neu zu starten. Sowohl die Wartezeit als auch die maximalen Versuche können konfiguriert werden. Zur Identifikation des Fehlers können über *kubectl logs <podname>* oder *kubectl describe <podname>* die Log-Nachrichten inklusive der Fehlermeldung abgerufen werden (vgl. [2, pp. 451-452]). Entwickler können so den Fehler entdecken und beheben. Während sich die neue Version im CrashLoopBackOff befindet, läuft die vorherige Version auf den Pod-Instanzen weiter. Kubernetes-Deployments erfolgen demnach atomar. Es wird entweder die gesamte neue Version bereitgestellt oder die alte Version bleibt vollständig aktiv (vgl. [8, pp. 21-22]).

*Vertraulichkeit*:

Entsteht durch ein Deployment eine Sicherheitslücke, kann diese entweder durch ein Rollback (siehe Abschnitt 4.3.2) behoben werden, oder durch durch eine neue nachgebesserte Version.

*Integrität*:

Durch Rolling-Updates wird die Integrität in einem Cluster verstärkt, da nur korrekte bzw. fehlerfreie Pods hochgefahren werden.

*Verfügbarkeit*:

Die Verfügbarkeit einer Anwendung profitert von Rolling-Updates. Eine Ausfallzeit wird verhindert, da es eine Zero-Downtime-Bereistellung ist. Durch die eingebauten Skalierungs- und Load-Balacingmethoden wird die Verfügbarkeit während und nach einem Deployment sichergestellt.

### Rollback-Funktionalität

Nach einem abgeschlossen Deployment kann es trotzdem Gründe für das Zurückkehren zu einer Vorgängerversion geben. Nachträglich entdeckte Sicherheitslücken oder Fehlverhalten, schlechtere Leistung oder Inkompatibilität mit anderen Anwendungen können Gründe für einen Rollback sein. In Kubernetes ist ein Rollback mit *kubectl rollout undo deployment/<deployment-name>* möglich. So wird ein Deployment mit der vorherigen Version gestartet. Mit *kubectl rollout undo deployment/<deployment-name> --to-revision=2* kann zu einer spezifischen Version zurückgekehrt werden (vgl. [17]).

*Vertraulichkeit*:

Die Rollback-Funktion in Kubernetes hat keinen direkten Einfluss auf die Vertraulichkeit. Allerdings kann diese dabei helfen, wenn nach einem Deployment neue Sicherheitslücken entdeckt werden. Durch einen schnell und einfach durchzuführenden Rollback können diese vorerst behoben werden.

*Integrität*:

Wird eine neue Version einer Anwendung mit einer erst nachträglich entdecken Sicherheitslücke, Hintertür für mögliche Angreifer oder fehlerhaften Code, bereitgestellt, wirkt sich dies negativ auf die Integrität der Anwendung und des Cluster aus, da Geschäftsprozesse falsch ablaufen oder manipuliert werden können. Ein Rollback auf die Vorgängerversion und die Wiederherstellung der korrekten Anwendung hat einen positiven Einfluss auf die Integrität

*Verfügbarkeit*:

Das Ausführen eines Rollbacks folgt dem Prinzip des Rolling-Updates. Es kommt also zu keinen Ausfallzeiten.

## Vergleich zur Bereitstellung ohne Orchestrierung

Eine Software-Bereitstellung ohne Kubernetes oder einen ähnlichen Orchestrator erfordert hohen manuellen Aufwand. Nicht nur die Konzeption eines grundlegenden Deployment-Prozess, sondern auch die Wartung im Live-Betrieb kann mitunter komplex und zeitaufwendig sein.

Kubernetes bietet einem Anwender viele Mechanismen mit Standardkonfigurationen und die Möglichkeit diese manuell anzupassen. Der Anwender findet demnach eine grundlegend funktionierende Infrastruktur vor, die er nach seinen Vorstellungen anpassen kann. Der Aufbau einer eigenen ständigen Server- und Netzwerkinfrastruktur wird dem Nutzer zum großen Teil abgenommen. Lediglich den Aufbau bzw. die Verknüpfung zu einer Container Runtime wie Docker muss der Anwender selbstständig durchführen. Deployments erfordern die Erstellung von Skripten, die mitunter fehleranfällig sein könnten, während Kubernetes Rolling Updates oder Rollbacks in einer Kommandozeile abfertigt. Ein fehlerhaftes Deployment kann zu einer Ausfallzeit führen. Kubernetes verhindert dies und ermöglicht eine einfache Fehlersuche. Die Skalierung der Server-Instanzen erfordert manuelles Starten weiterer Server oder die Integration von weiterer Software. In Kubernetes werden die Skalierung-Mechanismen durch eine Konfigurationsdatei festgelegt und die Kontrollebene von Kubernetes führt diese automatisch aus. Die Ressourcenverwaltung, die Zuweisung von CPU und RAM, kann ständige Anpassungen erfordern, wenn dies nicht wie in einem Kubernetes-Cluster automatisiert werden kann. Für Berechtigungen, Zugriffskontrollen und Passwortmanagement müssen auf Serverebene Zugriffsrechte auf Datenbanken und Berechtigungen erstellt und zugewiesen werden. Durch rollen- oder attributbasierte Zugriffskontrollen wird der Nutzer bei Kubernetes hierbei entlastet. Ebenso die Implementierung und Konzeption eines korrekten Netzwerkschutz durch Firewall und VPN verlangt von einem Anwender hohen Aufwand und ist ohne das nötige Wissen mitunter sehr komplex. Durch die vordefinierten Mechanismen in Kubernetes kann der Nutzer hier ebenfalls entlastet werden.

Eine Software-Bereitstellung ohne einen Orchestrator ist möglich, ist jedoch mit erheblicher Arbeit verbunden. Vor allem im Hinblick auf die Automatisierung des Deployment-Prozesses ist eine Orchestrierungsplattform von hohem Wert. Bei eigenem eigenständigen Prozess hängt der Grad der Automatisierung von verschiedenen Faktoren ab. Die Wahl der Infrastruktur sowie eventuell zusätzlich angebundener Software kann hier entscheidend sein. Für Unternehmen einen derartig wichtig und mitunter hoch frequentierten Prozess nicht im höchsten Maße zu automatisieren, kann fatal sein. Während für eine Entwicklungsumgebung ein manuelles Deployment durchaus sinnvoll seien kann, damit der Entwickler seine Anwendung vollumfänglich testen kann, sollten Unternehmen für eine Produktivumgebung ein Deployment mithilfe eines Orchestrators weitestgehend automatisieren.

## Fazit

Unter Betrachtung der Sicherheitskonzepte von Kubernetes kann ein vorläufiges Fazit gezogen werden. Allein der Einsatz von Kubernetes und dessen vorkonfiguierten Einstellungen hat bereits einen positiven Einfluss auf die Deployment-Strategie eines Unternehmens, wenn vorher kein ähnlicher Orchestrator eingesetzt wurde. Die Sicherheit der Strategie und des Clusters hängt jedoch stark von Konfiguration einzelnen Mechanismen ab. Damit die Anwendungen im Cluster sicher verwaltet und bereitgestellt werden können, müssen die Administratoren eine Konfigurationen vornehmen. Der weitverbreite Einsatz und die Unterstützung durch eine Open-Source-Community sorgt jedoch dafür, dass der Anwender durch eine ausführlichen Dokumentation und diversen Abhandlungen hierbei unterstützt wird. Bei der Konfiguration einer eigenen Deployment-Strategie kann dies unter Umständen nicht der Fall sein.

# Fallstudien

Nachfolgend werden die durchgeführten Fallstudien sowie deren Auswertung vorgestellt.

## Beschreibung der Anwendungsfälle

Zu Beginn wurde eine Spring-Boot-Applikation aufgesetzt. Spring-Boot ist ein Open-Source-Java-Framework, das die Entwicklung von Java-basierter Software vereinfacht. Die Grundstruktur der Anwendung für diese Fallstudien wurde mit dem Spring Initializr angelegt. Dieser ermöglicht es, durch die Eingabe und Auswahl von Metadaten die Grundstruktur einer Anwendung zu generieren. Für diese Fallstudie wurde eine Applikation mit folgenden Metadaten generiert:

* Project: Maven
* Language: Java
* Spring Boot: 3.4.1
* Project Metadata:
* Packaging: Jar
* Java: 17

Diese Programmiersprache, das Framework sowie vorgestellten Einstellungen wurden so gewählt, da für diese Technologien bereits Vorkenntnisse aus Firmen-Projekten bestehen.

Die Messungen beider Deployment-Prozesse werden mit den Open-Source-Tools Prometheus und Grafana durchgeführt. Prometheus ist eine Überwachungssoftware, die Metriken wie die Up- und Downtime eines Servers sammelt. Diese Daten lassen sich durch Grafana visuell darstellen. Eine Kombination beider Systeme ist in der Praxis üblich, sodass diese für die Durchführung der Messungen im Rahmen dieser Arbeit geeinet sind.

Damit Prometheus die gewünschten Metriken sammeln kann, wurden im Code-Projekt die Abhängigkeiten spring-boot-starter-actuator und micrometer-registry-prometheus eingebunden und die benötigten Endpunkte in der application.properties eingeschaltet. Die Images der Applikation werden über ein Dockerfile mit dem Befehl *docker build -t name:TAG .* gebaut und können für beide Deployment-Prozesse genutzt werden. Im Laufe der Vorbereitungen wurden weitere Files dem Projekt hinzugefügt. Diese werden in den Abschnitten 5.1.1 und 5.1.2 vorgestellt.

Beide Deployment-Prozesse werden lokal aufgesetzt, damit diese mit den gleichen Systemvorrausetzungen laufen und eine bessere Vergleichbarkeit gegeben ist.

### Deployment-Prozess mit Kubernetes

Der Deployment-Prozess mit Kubernetes wird mit dem Open-Source-Tool Minikube erstellt. Dieser ermöglicht es ein Kubernetes Cluster lokal auf einem PC bzw. in einem Docker Container aufzusetzen und dort zu betreiben.

Das Aufsetzen des Minikube und dem Command-Line-Tool Kubectl erfolgte mit den bereigstellen Dokumentationen von Kubernetes (vgl. [18]) und Minikube (vgl. [19]) selbst. In diesem Cluster wurde dann die Spring-Boot-App bereitgestellt. Die Bereistellung erfolgt über eine Deployment.yaml. In dieser werden die Deployment-Strategie, Container-Einstellungen und Metadaten des Deployments konfiguriert. Für diese Messungen wurde die Anzahl der Container, auf denen die Software bereitgestellt wurde auf 1 belassen, damit eine bessere Vergleichbarkeit zum zweiten Testfall gegeben ist. Die Deployment.yaml ermöglicht es ein Deployment so zu konfigurieren, dass mit der Rolling-Update-Strategie zu keiner Downtime kommt. Durch das Hinzufügen der readinessProbe-Konfiguration wird bei einem Deployment daraufgewartet, dass der neue Pod nicht nur korrekt hochgefahren wird, sondern auch Anfragen verarbeiten kann, bevor das bestehende Deployment inklusive Pod heruntergefahren wird.

Die Einbindung von Grafana und Prometheus in das Cluster erfolgte anhand einer Dokumentation von Grafana zur Einbindung beider Tools über Helm-Files (vgl. [20]). Anschließend wurde ein ServiceMonitor aufgesetzt. Damit kann Prometheus seinen eigenen in Code-Projekt implementierten Endpunkte (/actuator/prometheus) abfragen. Der Interval der Abfrage wurde für die Messungen auf 1 Sekunde gesetzt. Der ServiceMonitor kann dann in Grafana über die Metrik *up{job="spring-sample-app-k8s"}* abgerufen werden und in verschiedenen Formen wie Tabellen oder Zeitserien visualisiert werden.

Es wurden insgesamt 50 Deployments durchgeführt. Dafür wurden zwei Images vom selben Code-Stand erzeugt und abwechelnd durch die Deployment.yaml mit dem Befehl *kubectl apply -f deployment.yaml* deployt. Damit dies über die Command-Line ausgeführt werden kann, muss zuvor in das Code-Verzeichnis gewechselt werden. Die Daten wurden von Prometheus gesammelt und von Grafana visuell dargestellt. Die Messungen wurde über einen Zeitraum vom 1 Stunde durchgeführt. Gemessen wurde die Verfügbarkeit des Service in Kubernetes. Dafür prüft Prometheus jede Sekunde die Verfügbarkeit des aktuellen Container. Während eines Deployment prüft Prometheus sowohl die Verfügbarkeit des aktuellen Containers als auch die des neuen Deployments. Abbildung 5 zeigt einen Zeitraum während den Messungen, der dies deutlich macht.

Ein Bild, das Diagramm, Reihe, Screenshot, Text enthält.

Automatisch generierte Beschreibung

Abbildung 5 - Grafana Dashboard: Rolling Update (Eigendarstellung)

Ist der Container bzw. der Endpunkt der Anwendung, welche über den Container bereitgestellt wird, verfügbar gibt die Metrik den Wert 1 für UP zurück. Konnte der Endpunkt nicht erreicht werden, wird 0 für DOWN zurückgegeben. Die Rohdaten der Messungen können direkt aus Grafana in eine CSV-Datei exportiert werden. In dieser sind alle gestarteten Container so wie deren Up- und Downtime in einer Zeitlinie dokumentiert. Für jeden Zeitraum, in dem kein Container verfügbar war, ist es dementsprechend zu einer Downtime gekommen.

Zur Auswertung der Daten wurde eine Kopie der Rohdaten-Datei erstellt. In dieser wurden die Daten in einer Tabellenform dargestellt und eine zusätzliche Spalte „Verfügbarkeit“ hinzugefügt. In dieser wird soll zu jedem Zeitdatenpunkt in der Tabelle ermittelt werden, ob die Anwendung verfügbar war. Zu einem Deployment gehören in diesem Fallbeispiel zwei Container: Der Container vor und der Container nach dem Deployment. Um die Up- und Downtime während der gesamten Messungen zu ermitteln, muss dementsprechend geprüft werden, welche Werte Prometheus, während eines Deployments von den beteilgiten Container erhalten hat. War die Anwendung dauerhaft verfügbar, muss sich zu jedem Zeitpunkt mindestens ein Container finden lassen, der erreichbar war. Wenn zu einem Zeitpunkt kein Container eine Verfügbarkeit zurückgemeldet hat, ist es zu einer Downtime gekommen. Da Prometheus die Uptime durch eine 1 und die Downtime durch eine 0 gemessen hat, kann für jede Spalte in der Tabelle folgende Excel-Funktion die Verfügbarkeit der Anwendung ermitteln:

=WENN(ZÄHLENWENN(C2:BD2;1)<1;"Downtime";"Uptime")

Diese Funktion prüft, ob es in einer Spalte weniger als eine 1 gibt und schreibt die Uptime bzw. Downtime in die hinzugefügte Spalte Verfügbarkeit.

*Verfügbarkeit:*

Die Messungen zeigen, dass es während der Deployments zu keiner Downtime gekommen ist und die Anwendung dauerhaft verfügbar war. Dementsprechend konnten keine Ausfälle und deren Dauer dokumentiert werden.

**Uptime: 100%**

**Downtime: 0%**

**Anzahl von Ausfällen: 0**

**Dauer von Ausfällen: 0**

*Integrität:*

Die Messungen der Uptime durch Prometheus liefert auch Daten, um die Integrität der Deployment-Strategie zu bewerten. Integrität beschreibt die Korrektheit von Daten. Im Bezug auf die hier vorgestellten Fallbeispiele bezieht sich die Integrität also auf die Korrektheit eines Deployments. Über den gesammten Messzeitraum sind alle Deployments im ersten Versuch erfolgreich durchgeführt worden. Dementsprechend ist es zu keinen Fehlern gekommen und kein Rollback war nötig.

**Fehlerqoute: 0**

**Anzahl Rollbacks:0**

### Deployment-Prozess ohne Kubernetes

## Auswertung und Diskussion

# Experteninterviews

## Vorstellung der Experten

## Auswertung der Interviewergebnisse

## Einordnung in den Kontext der Forschungsfrage

# Diskussion

## Zusammenführen der wichtigsten Erkenntnisse

## Bewertung des Einflusses von Kubernetes

## Reflexion des Forschungsvorgehen

# Fazit und Ausblick

## Vorstellung der Ergebnisse

## Beantwortung der Hypothesen

## Beantwortung der Leitfrage

## Empfehlung

## Implikation für Forschung und Praxis

# KI-Verzeichnis

Es wurde keine KI verwendet.

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | R. Leszko, Continous Delivery with Docker and Jenkins, Packt Publishing Ltd., 2017. |
| [2] | M. Luksa, Kubernetes in Action: Anwendungen in Kubernetes-Clustern bereitstellen und verwalten, Hanser, 2018. |
| [3] | RedHat, „Was ist Container-Orchestrierung?,“ 2022. |
| [4] | T. Probst, „Generische Schutzmaßnahmen für Datenschutz-Schutzziele,“ *Datenschutz und Datensicherheit - DuD,* Nr. 6, pp. 439-444, 2012. |
| [5] | R. Hellmann, IT Sicherheit: Methoden und Schutzmaßnahmen für Sichere Cybersysteme, Ausgabe 2, De Gruyter Oldenbourg, 2022. |
| [6] | T. E. Florian Deusch, Beauftragte für IT-Sicherheit und Informationssicherheit, Fachmedien Recht und Wirtschaft, Deutscher Fachverlag GmbH, 2024. |
| [7] | N. Kratzke, Cloud-native Computing: Software Engineering von Diensten und Applikationen für die Cloud, Hanser, 2022. |
| [8] | N. Poulton, The Kubernetes Book, 2021. |
| [9] | G. Sayfan, Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes, Packt Publishing Ltd, 2019. |
| [10] | Kubernetes, „Pods“. |
| [11] | Kubernetes, „Nodes“. |
| [12] | Kubernetes, „Kubernetes Architecture“. |
| [13] | Kubernetes, „Using RBAC Authorization“. |
| [14] | Kubernetes, „Pod Security Standards“. |
| [15] | Kubernetes, „Security Checklist“. |
| [16] | J. B. K. H. Brendan Burns, „Kubernetes Up & Running, Dive into the Future of Infrastructure (2.Ausgabe),“ O'Reilly, 2019. |
| [17] | Kubernetes, „Deployments“. |
| [18] | Kubernetes, „Kubernetes lokal über Minikube betreiben“. |
| [19] | Minikube, „minikube start“. |
| [20] | Grafana, „Migrate a Kube-Prometheus Helm stack to Grafana Cloud“. |
| [21] | Docker, „Use containers to Build, Share and Run your applications“. |
| [22] | „Kubernetes CSI Developer Documentation“. |

# Anhang

1. Vgl. Gesetz zum Schutz von Geschäftsgeheimnissen vom 18. April 2019 (BGBl. I S. 466) [↑](#footnote-ref-1)