HOCHSCHULE ESSLINGEN

IT DESIGNERS GRUPPE

Masterarbeit

Entwicklung und Implementierung einer GitOps-gesteuerten, Multi-Tenant-fähigen DevSecOps Kubernetes-Plattform für die Cloud-basierte Softwareentwicklung in einer Hybrid-Cloud-Umgebung

Development and implementation of a GitOps-driven, multi-tenant DevSecOps Kubernetes platform for cloud-based software development in a hybrid cloud environment

im Master-Studiengang
Angewandte Informatik
Fakultät Informationstechnik

vorgelegt von

Alexander Efremidis
Matrikelnummer: 771514

vorgelegt am
31. August 2024

Erstprüfer/in:	Prof. Dr. rer. nat. Mirko Sonntag
Zweitprüfer/in:	Dipl.-Phys. Stefan Kraft

Verfasser der Masterarbeit

Alexander Efremidis

Bearbeitungszeitraum

Der Bearbeitungszeitraum erstreckte sich vom 01.03.2024 bis zum 31.08.2024.

Betreuung

Betreuung an der Hochschule
Prof. Dr. rer. nat. Mirko Sonntag
Flandernstraße 101
73732 Esslingen
mirko.sonntag@hs-esslingen.de

Betreuung im Unternehmen
Dipl.-Phys. Stefan Kraft
Alleenstraße 26
73730 Esslingen am Neckar
stefan.kraft@stz-softwaretechnik.de

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird in dieser Masterarbeit auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Formulierungen gelten gleichermaßen für alle Geschlechter.

Hinweis zur Nutzung von KI-Tools

Für die Erstellung dieser Masterarbeit wurden DeepL und KI-basierte Tools ausschließlich zur Überprüfung von Grammatik und Rechtschreibung verwendet. Der Inhalt und der wissenschaftliche Anspruch der Arbeit sowie die Argumentationen, Informationen, Grafiken und Schlussfolgerungen stammen vollständig vom Autor selbst bzw. sind mit den entsprechenden Quellen gekennzeichnet.

Abstract

This thesis deals with the development and implementation of a multi-tenant Kubernetes DevSecOps platform that is specially optimised for remote development. Against the backdrop of the growing importance of cloud computing and the increasing shift of software development processes to the cloud, the focus is on increasing the efficiency and productivity of development teams. Centralised GitOps approaches are used, which enable a clear separation between infrastructure and end users. The paper first looks at the technological foundations of Kubernetes, multi-tenancy and DevOps. This is supplemented by an online survey to ascertain the current level of knowledge and the DevOps culture in the supporting company. Based on this, the requirements for the platform are formulated and prioritised in order to ensure targeted implementation. The implementation is carried out on a Proxmox test system using modern open source tools such as Kamaji and Capsule to ensure strict separation and isolation of the teams and developers. In addition, a new CI/CD pipeline based on GitLab will be developed and supplemented by a CLI tool to support developers with cluster deployment and onboarding. Finally, the platform is evaluated and the objectives of the work are successfully confirmed. In the future, extensions such as the integration of a graphical user interface, the use of external storage solutions and the further automation of infrastructure management are conceivable.

Kurzfassung

Diese Arbeit beschäftigt sich mit der Entwicklung und Implementierung einer mandantenfähigen Kubernetes DevSecOps Plattform, die speziell für Remote Development optimiert ist. Vor dem Hintergrund der wachsenden Bedeutung von Cloud Computing und der zunehmenden Verlagerung von Softwareentwicklungsprozessen in die Cloud steht die Steigerung der Effizienz und Produktivität von Entwicklungsteams im Mittelpunkt. Dabei werden zentrale GitOps-Ansätze genutzt, die eine klare Trennung zwischen Infrastruktur und Endanwendern ermöglichen. Die Arbeit beleuchtet zunächst die technologischen Grundlagen von Kubernetes, Mandantenfähigkeit und DevOps. Ergänzt wird dies durch eine Online-Umfrage, die den aktuellen Wissensstand und die DevOps-Kultur im betreuenden Unternehmen erhebt. Darauf aufbauend werden die Anforderungen an die Plattform formuliert und priorisiert, um eine zielgerichtete Umsetzung zu gewährleisten. Die Implementierung erfolgt auf einem Proxmox Testsystem unter Verwendung moderner Open Source Tools wie Kamaji und Capsule, um eine strikte Trennung und Isolation der Teams und Entwickler zu gewährleisten. Zusätzlich wird eine neue CI/CD Pipeline auf Basis von GitLab entwickelt und durch ein CLI Tool ergänzt, das die Entwickler beim Cluster Deployment und Onboarding unterstützt. Abschließend wird die Plattform evaluiert und die Ziele der Arbeit erfolgreich bestätigt. Perspektivisch sind Erweiterungen wie die Integration einer grafischen Benutzeroberfläche, die Nutzung externer Speicherlösungen und die weitere Automatisierung des Infrastrukturmanagements denkbar.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CRD Custom Ressource Definition

VM Virtuelle Maschine

ML Maschinelles Lernen

VPN Virtual Private Network

SDLC Software Development Life Cycle

NIST National Institute of Standards and Technology

OWASP Open Worldwide Application Security Project

DAG Directed Acyclic Graph

CI Continuous Integration

CLI Command Line Interface

CD Continuous Delivery

HA High Availability

IPC Interprocess Communication

LVM Logical Volume Manager

RAID Redundant Array of Independent Disks

RBAC Role-Based-Access-Control

IaC Infrastructure as Code

DRBD Distributed Replicated Block Device

CNCF Cloud Native Computing Foundation

1 Einleitung

1.1 Der Wandel zur Cloud-Native Softwareentwicklung

Die Digitalisierung hat die Methoden und Anforderungen im Bereich der Softwareentwicklung in den letzten Jahrzehnten stark verändert. Während klassische Modelle wie das Wasserfallmodell heute in den meisten Unternehmen von agilen Modellen abgelöst wurden, haben sich auch im Bereich der Infrastruktur große Veränderungen ergeben. In den letzten 20 Jahren wurde mit dem Aufkommen zahlreicher Cloud-Anbieter wie Amazon AWS oder Microsoft Azure eine neue Ära in der Softwareentwicklung eingeläutet (vgl. Reinheimer 2018, S. 6-7). Das Thema Cloud Computing spielt in Unternehmen eine immer wichtigere Rolle, da sich durch die stetig steigende Rechenleistung zahlreiche neue strategische, operative und finanzielle Vorteile durch Themen wie Big Data und Machine Learning ergeben haben, von denen Unternehmen in vielen Bereichen profitieren (vgl. Reinheimer 2018, S. 15). Damit gehen aber auch einige technische und finanzielle Hürden einher, die unter anderem durch eine engere Zusammenarbeit zwischen Entwicklungs- und Betriebsteams überwunden werden sollen. Dieser Kulturwandel hat dazu geführt, dass im Zeitalter des Cloud Computing die schnelle Entwicklung und Auslieferung von Software mit agilen Methoden zum De-facto-Standard geworden ist. Die daraus entstandene Cloud-native Softwareentwicklung konnte sich zusammen mit DevOps in den letzten Jahren zunehmend am Markt etablieren (vgl. GitLab 2022 und DeBellis und Peters 2022).

Bei näherer Betrachtung des Konzepts der Cloud-nativen Softwareentwicklung zeigt sich jedoch, dass häufig nicht der gesamte Softwareentwicklungsprozess (SDLC) in der Cloud stattfindet. Ein Großteil der eigentlichen Entwicklung findet lokal in den Entwicklungsteams statt und wird lediglich in Cloud-Umgebungen getestet und ausgeliefert. Dieses klassische Vorgehen führt je nach Größe und Komplexität der Anwendung sowie der Größe des Teams zu einem erheblichen Mehraufwand an Konfiguration, Wartung und Zeit. Gerade bei Teams mit hoher Fluktuation führt dies zu langen Einarbeitungsphasen, verringert die Effizienz und erfordert viel Dokumentation und technisches Know-how, um neue Teammitglieder in bestehende Prozesse einbinden zu können. Gerade bei Themen rund um Künstliche Intelligenz (KI) stoßen Entwickler immer schneller an die Kapazitätsgrenzen der verfügbaren Hardware, was sie häufig dazu zwingt, auf cloudbasierte Lösungen auszuweichen (vgl. DeBellis und Peters 2022). Unter anderem dadurch hat sich ein neues Feld der Cloud-nativen Softwareentwicklung eröffnet, das sogenannte Remote Development, bei dem der gesamte Entwicklungsprozess zentral in der Cloud stattfindet. Dies ermöglicht es Entwicklern, von nahezu jedem internetfähigen Endgerät aus zu arbeiten, und bietet neue flexible Lösungen im Bereich DevOps, um Kosten und Verwaltungsaufwand zu reduzieren. Gleichzeitig entstehen jedoch auch Herausforderungen im Ressourcenmanagement und in der Sicherheit, für die es noch keine allgemeingültige Lösung gibt.

1.2 Zielsetzung und Leitfragen

Ziel dieser Arbeit ist der Entwurf und die Implementierung eines mandantenfähigen DevSecOps Frameworks, das speziell für Remote Development optimiert ist und bestehende Entwicklungsteams bei der Anwendungsentwicklung effektiv unterstützt. Der Fokus liegt dabei auf der Erstellung eines optimalen Konzepts, das versucht, die zahlreichen am Markt existierenden Lösungen zu vergleichen, um anschließend ein anfängerfreundliches Self-Service-Modell zu entwickeln, das es vor allem neuen Entwicklern ermöglicht, ohne umfangreiche Unterstützung durch Betriebsteams und ohne Kenntnisse im Bereich der Cloud-nativen Entwicklung auf die für ihre Arbeit benötigten Ressourcen zuzugreifen und diese zu verwalten. Dabei geht es vor allem um die generische Erstellung neuer Entwicklungsumgebungen in einer Private Cloud sowie um die Optimierung bestehender Release-Strukturen für die finalen Releases der verschiedenen Anwendungen. Das System soll vollständig in einer zentralen Cloud-Umgebung, z. B. einem On-Premise Kubernetes Cluster, betrieben werden. Die Self Service Plattform soll vollständig auf Open Source Technologien basieren, um die Kosten so gering wie möglich zu halten.

Um dieses Ziel zu erreichen werden folgende vier Leitfragen formuliert:

- **LF1: Inwieweit erfüllen bestehende multimandantenfähige Lösungen für Kubernetes die Anforderungen an eine „harte“ Isolation von Entwicklungs- und Produktionsumgebungen?**
- **LF2: Welchen Wissensstand haben Entwickler in den Bereichen sichere Entwicklung, Cloud-native Entwicklung, Continuous Delivery (CD) und Remote Development?**
- **LF3: Können unterstützende Werkzeuge die Bewältigung von operationellen Herausforderungen (Ops Challenges) in multimandantenfähigen Cluster-Umgebungen erleichtern, insbesondere für diejenigen Entwickler, die keine umfangreiche Erfahrung in der Verwaltung solcher Systeme haben?**
- **LF4: Wie kann die Entwicklung einer kostengünstigen Self-Service-Plattform gestaltet werden, um die täglichen Entwicklungsprozesse innerhalb eines bestehenden Teams mit hoher Fluktuation zu unterstützen, und wie wirkt sich dies auf die Effizienz und Effektivität aus?**

In der Gesamtbetrachtung sollen die positiven Aspekte der DevOps-Philosophie und die damit verbundene Verschmelzung von Entwicklung, Sicherheit und Endbetrieb weiter optimiert werden, um eine sichere und moderne multimandantenfähige Plattform zu schaffen und die Vorteile von Remote Development in der Praxis zu evaluieren.

1.3 Methodische Herangehensweise

Der methodische Ansatz dieser Arbeit umfasst mehrere Kapitel, die inkrementell aufeinander aufbauen. Im zweiten Kapitel werden zunächst die theoretischen Grundlagen sowie domänenspezifische Themen und Begriffe zum besseren Verständnis des Gesamtkontextes erläutert. Im Anschluss daran erfolgt eine ausführliche Literaturrecherche, um vertiefte Kenntnisse zu den Kernthemen wie DevSecOps, Mehrmandantenfähigkeit, Cloud-native Entwicklung zu erlangen und anschließend geeignete Frameworks und Tools zu finden, die zur Beantwortung der Leitfragen notwendig sind.

Um eine geeignete Grundlage für eine spätere Anforderungsanalyse zu haben, wird eine quantitative und qualitative Online-Befragung im betreuenden Unternehmen durchgeführt, die sich zum größten Teil aus festangestellten Mitarbeitern und zu ca. 20% aus Studierenden zusammensetzt. Ziel ist es, einen Überblick über den aktuellen Wissensstand in den Bereichen Kubernetes, Secure Development, Cloud-native Development, Continuous Delivery und Remote Development zu erhalten. Diese werden im Detail analysiert, um die genauen Anforderungen zu definieren, die die gesamte Self-Service Plattform später erfüllen soll.

Anschließend erfolgt die Anforderungsanalyse auf Basis der bestehenden Infrastruktur im Unternehmen und den Ergebnissen der Befragung. Dabei wird das bestehende System im Unternehmen als Vergleichsbasis herangezogen. Die Anforderungen werden im Wesentlichen so definiert, dass das zu erreichende Forschungsziel bestmöglich erfüllt wird. Für die Implementierung und Evaluierung wird ein isolierter Kubernetes Testcluster auf Basis virtueller Maschinen nachgebildet, der die technischen Spezifikationen und Vorgaben des Unternehmens, in dem dieses System später eingesetzt werden soll, bestmöglich erfüllt. Darüber hinaus werden Konzepte entwickelt, um den GitOps-Ansatz zusammen mit der Mandantenfähigkeit bestmöglich zu integrieren.

Daraufhin erfolgt die Konzeptions- und Implementierungsphase des Unterstützungswerkzeugs in Form einer CLI für Entwicklungsumgebungen. Hierfür wird eine Auswahl gängiger Frameworks getroffen, mit denen dieses umgesetzt wird.

1.4 Aufbau der Arbeit

In diesem Abschnitt ist die Gliederung mit kurzen Erläuterungen aufgeschlüsselt.

- **Kapitel 2 - Grundlagen:** In diesem Kapitel werden die theoretischen Grundlagen und die für das Verständnis der Arbeit relevanten Begriffe vorgestellt. Es dient als Basis für die nachfolgenden Vergleiche und Auswertungen.
- **Kapitel 3 - Nutzerbefragung und Auswertung:** Dieses Kapitel umfasst eine quantitative und qualitative Online-Umfrage innerhalb des Unternehmens. Ziel ist es, einen Überblick über den aktuellen Wissensstand in den Bereichen Kubernetes, Secure Development, Cloud-native Development, Continuous Delivery und Remote Development zu erhalten.
- **Kapitel 4 - Analyse der Anforerungen an das Gesamtsystem:** Dieses Kapitel befasst sich mit der Definition der Anforderungen und der Analyse der aktuellen Kubernetes-Infrastruktur im Unternehmen.
- **Kapitel 5 - Realisierung der Kubernetes-as-a-Service Plattform:** Dieses Kapitel befasst sich mit der Konzeption und dem Vergleich von Technologien bis hin zur Implementierung der Kubernetes-Plattform.
- **Kapitel 6 - Konzeption des CLI-Unterstützungswerkzeugs:** In diesem Kapitel wird die Planung des zu entwickelnden Unterstützungsinstruments detailliert beschrieben. Sowohl die technischen Aspekte als auch die methodischen Überlegungen werden erläutert.
- **Kapitel 7 - Implementierung des CLI-Unterstützungswerkzeugs:** Dieses Kapitel dokumentiert den Prozess der Implementierung des Unterstützungswerkzeugs. Es beschreibt die technischen Herausforderungen, die Lösungsstrategien und die endgültige Implementierung der CLI.
- **Kapitel 8 - Evaluation:** In diesem Kapitel wird die Evaluierung der Kubernetes-Plattform durchgeführt. Die angewandten Methoden, die Ergebnisse der Umsetzung usw. werden diskutiert.
- **Kapitel 9 - Zusammenfassung und Ausblick:** Das Schlusskapitel fasst die wichtigsten Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Forschungsrichtungen und Entwicklungen.

2 Grundlagen

Zunächst soll anhand von Literatur ein Verständnis für die verschiedenen Begriffe, Ansätze und Architekturen erarbeitet werden. Zu Beginn wird in Kapitel 2.1 ein allgemeiner Überblick über verschiedene Cloud Computing-Methoden gegeben. Darauf aufbauend wird ein Einblick in die Grundlagen von DevOps und DevSecOps in Kapitel 2.2 gegeben und verschiedene Deployment-Strategien, Continuous Integration (CI) und Continuous Delivery (CD) Ansätze in Kapitel 2.3 erläutert. Eine kurze Übersicht über GitOps findet sich in Kapitel 2.4. Einen tiefen Einblick in die Architektur von Kubernetes gibt Kapitel 2.5. Dies ist ein wichtiger Bestandteil, um die Einschränkungen von Kubernetes in mandantenfähigen Clustern zu verstehen. Abschließend werden in Kapitel 2.6 wichtige DevOps-Tools vorgestellt.

2.1 Grundlagen zu Cloud Computing

In diesem Kapitel werden die Grundlagen des Cloud Computing behandelt. Dazu gehören die Definition des Begriffs selbst in Kapitel 2.1.1 sowie die verschiedenen XaaS-Begriffe in Kapitel 2.1.5, um das Prinzip der zu implementierenden Kubernetes-as-a-Service Plattform besser einordnen zu können.

2.1.1 Der Begriff „Cloud Computing“

Cloud Computing ist ein allgegenwärtiger Begriff. Der Begriff wird in der Literatur häufig als Schlüssel zur Überwindung von Kapazitäts- und Leistungsengpässen gesehen (vgl. Reinheimer 2018, S. 4). Für den Begriff selbst gibt es keine einheitliche oder standardisierte Definition, da es sich bei Cloud Computing um unterschiedliche Anwendungsszenarien handelt. In den Definitionen aus der Literatur finden sich jedoch typischerweise gemeinsame Merkmale wie „flexible und skalierbare Infrastruktur“, „Illusion unendlicher, auf Abruf verfügbarer IT-Ressourcen“ sowie „nutzungsbasierte Abrechnung“ (vgl. Reinheimer 2018, S. 4). Aus dem letzten Punkt kann u. a. geschlossen werden, dass Infrastruktur ohne großen Aufwand in einem Projekt beschafft werden kann.

Betrachtet man den Begriff „Cloud“ genauer, so könnte dies darauf hindeuten, dass die zur Verfügung gestellten Dienste über das Internet bereitgestellt werden, da das Symbol der Wolke historisch gesehen für das Internet steht (vgl., auch im Weiteren, Reinheimer 2018, S. 4).

Häufig wird auch die Definition des National Institute of Standards and Technology (NIST) zitiert. Sie beschreibt Cloud Computing als ein Modell, das den allgegenwärtigen und bequemen Zugriff auf einen Pool gemeinsam nutzbarer Ressourcen über ein Netzwerk ermöglicht. Dazu gehören Netzwerke, Speicherplatz, Rechenleistung, Anwendungen und andere Dienste,

die ohne menschliche Interaktion zwischen Nutzer und Anbieter sofort und bedarfsgerecht bereitgestellt werden (vgl. Mell und Grance 2011). Die Nutzer erwarten eine ständige Verfügbarkeit dieser Ressourcen.

Technisch ist Cloud Computing mit der Virtualisierung von Hardware und virtuellen Rechenzentren sowie den Servicemodellen „Software-as-a-Service“ (SaaS), „Platform-as-a-Service“ (PaaS) und „Infrastructure-as-a-Service“ (IaaS) verbunden. Diese werden in Kapitel 2.1.5 näher definiert.

Wirtschaftlich gesehen ist Cloud Computing eine besondere Form des IT-Outsourcings, bei dem spezialisierte Anbieter den Betrieb und die Wartung von IT-Diensten übernehmen.

2.1.2 Cloud Native Computing Foundation (CNCF)

Die Cloud Native Computing Foundation (CNCF) ist eine Organisation, die 2015 in Zusammenarbeit mit der Linux Foundation und Google gegründet wurde. Sie entstand durch die Veröffentlichung der Open-Source-Software Kubernetes im Jahr 2014. Diese wurde ursprünglich unter dem Namen Borg in den Rechenzentren von Google eingesetzt und im Zuge der Veröffentlichung in der Programmiersprache Go umgeschrieben und in Kubernetes umbenannt. Ziel der CNCF ist es, einen allgemeinen Leitfaden für die moderne Cloud-native Anwendungsentwicklung bereitzustellen, indem sie die Koordination und Überwachung von Open-Source-Technologien und Projekten rund um die Cloud-native Softwareentwicklung übernimmt. Heute nimmt die CNCF eine führende Rolle in der gesamten Cloud-Computing-Community ein und wird häufig als eine Art Gütesiegel für Open-Source-Technologien verwendet, wenn diese von der Organisation zertifiziert wurden (vgl. CNCF 2015, Aly und Murat 2021, Gigi 2020).

2.1.3 Cloud-native Ansatz

Neben der in Kapitel 2.1.1 erwähnten NIST-Definition von Cloud Computing definiert die CNCF den Begriff „Cloud-native Computing“ als skalierbare Anwendungen, die in modernen dynamischen Umgebungen laufen und Technologien wie Container, Microservices und deklarative APIs nutzen. Container-Orchestratoren wie z. B. Kubernetes spielen bei der Umsetzung dieses Ansatzes eine zentrale Rolle. Neben einer skalierbaren Infrastruktur müssen die darauf laufenden Applikationen die Anforderungen der 12-Factor-App-Methode (vgl. Wiggins 2017, Aly und Murat 2021, S. 16) erfüllen, um dem Cloud-nativen Zustand zu entsprechen. Dadurch können Probleme wie u. a. Skalierbarkeit, Erreichbarkeit, Ressourcennutzung bei Anwendungen in verteilten Systemen gelöst werden.

2.1.4 Organisationsformen von Clouds

Der Begriff „Cloud“ lässt sich im Zusammenhang mit Cloud Computing weiter untergliedern. In der Literatur wird zwischen Private, Public und Hybrid Clouds sowie einigen Mischformen unterschieden (vgl., auch im Weiteren, Reinheimer 2018, S. 7). Jede Form hat je nach Einsatzzweck ihre Vor- und Nachteile. Daher ist es wichtig, die passende Form für den individuellen Anwendungsfall auszuwählen. Um die richtige Wahl treffen zu können, wird häufig eine zweite Dimension, die sogenannte Sourcing-Option, hinzugezogen. Mit ihrer Hilfe können spezifische Mindestanforderungen definiert werden, inwieweit ein Unternehmen die Kontrolle über die eigene Cloud hat. In Abbildung 1 sind beide Dimensionen und deren Zusammenhang veranschaulicht.

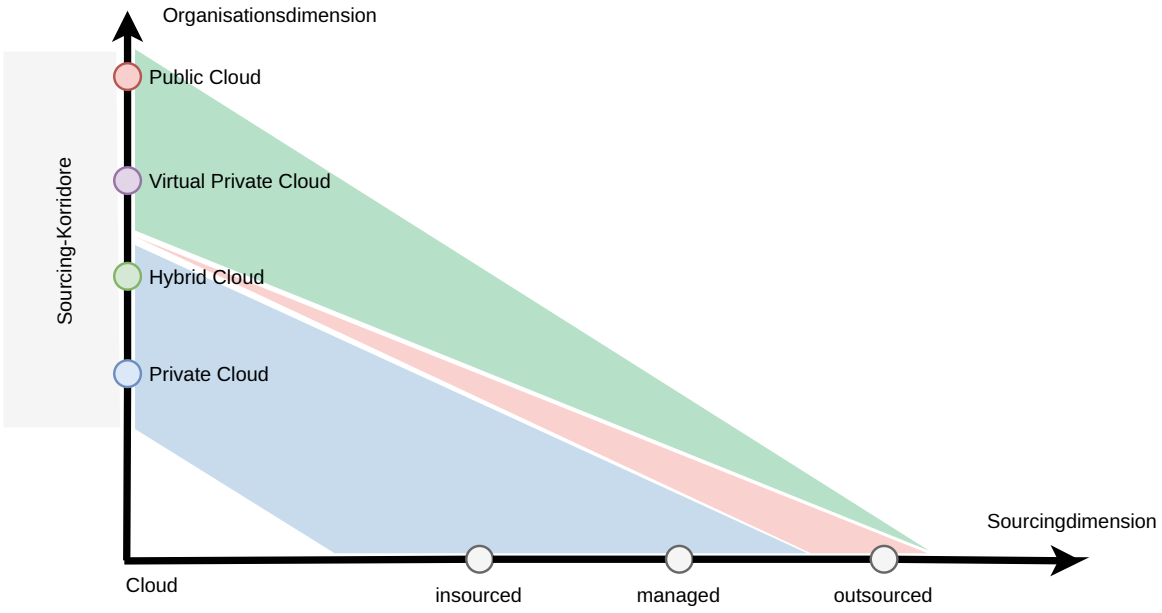

Abbildung 1: *Darstellung der Organisationsformen von Clouds (in Anlehnung an* BITKOM *2010, S. 17)*

So wird in einer Public Cloud die Infrastruktur vollständig von einem externen Dienstleister (outsourced) betrieben und verwaltet. Die individuelle Anpassbarkeit durch das Unternehmen ist sehr gering. In einer Private Cloud hingegen wird die Infrastruktur vollständig vom Unternehmen selbst (insourced) oder von einem externen Dienstleister nach den Vorgaben des Unternehmens betrieben (managed), was eine hohe Anpassungsfähigkeit bedeutet. Bei einer Hybrid Cloud werden bestimmte Teile an einen externen Dienstleister ausgelagert. Dadurch wird eine hohe Flexibilität und Skalierbarkeit erreicht. Allerdings müssen bestimmte Sicherheitsaspekte wie Datenschutz etc. genauer betrachtet werden (vgl. BITKOM 2010, S. 18 ff.).

- Unter einer **Public Cloud** versteht man eine Infrastruktur, die für die offene Nutzung von Cloud-Diensten bereitgestellt und von der Allgemeinheit genutzt wird. Sie wird in

der Regel von großen Unternehmen betrieben und verwaltet. Aus Sicht des Nutzers befindet sich die gesamte Hardware in externen Händen.

- Im Gegensatz dazu ist eine **Private Cloud** eine Cloud, die einer Organisation zur exklusiven Nutzung zur Verfügung steht. Der Zugang zu dieser Cloud ist auf bestimmte Nutzer beschränkt und kann sich auf dem Gelände der Organisation (on-premise) oder bei einem externen Dienstleister (managed) befinden. Eine Private Cloud bietet die höchste Form von Flexibilität und Datensicherheit.

- Eine weitere Form ist die **Hybrid Cloud**. Darunter versteht man eine Mischform aus einer Public, Private oder Community Cloud. Damit ist es z. B. möglich, Lastspitzen für kurze Zeit in eine externe Cloud auszulagern oder bestimmte Services von außen hinzuzukaufen, wenn sich beispielsweise eine On-Premise-Beschaffung nicht rechnet. Die verschiedenen Cloud-Typen werden über das Internet mit gängigen Technologien wie einem VPN verbunden.

- Des Weiteren gibt es die **Community-Cloud**, die meist von mehreren Communities genutzt und verwaltet wird. Dabei kann die Hardware selbst auf mehrere Standorte verteilt sein oder auch komplett off-premise gehostet werden. Die Communities selbst haben oft gemeinsame Anliegen wie z. B. eine Mission, Richtlinien oder Sicherheitsanforderungen, was eine solche Cloud-Lösung interessant macht.

2.1.5 Einordnung der XaaS-Begriffe

Unter „XaaS“ oder auch „Everything-as-a-Service“ versteht man einen Oberbegriff für verschiedene Servicemodelle, die beispielsweise über Cloud-Anbieter wie Amazon AWS, Microsoft Azure oder die Hetzner Cloud über das Internet bereitgestellt werden. Dabei werden drei Service-Ebenen unterschieden, denen eine Vielzahl von Geschäftsmodellen zugrunde liegen. Die drei Service-Ebenen sind in Abbildung 2 absteigend nach ihrem Abstraktionsgrad geordnet. Durch die Abstraktion ist es möglich, dass z. B. ein Service aus einer höheren Abstraktionsschicht auf einen realisierten Service in einer darunter liegenden Schicht zugreifen kann. Grundsätzlich gilt: Je höher die Abstraktionsebene, desto komplexer ist der auf dieser Ebene bereitgestellte Dienst (vgl. Reinheimer 2018, S. 9).

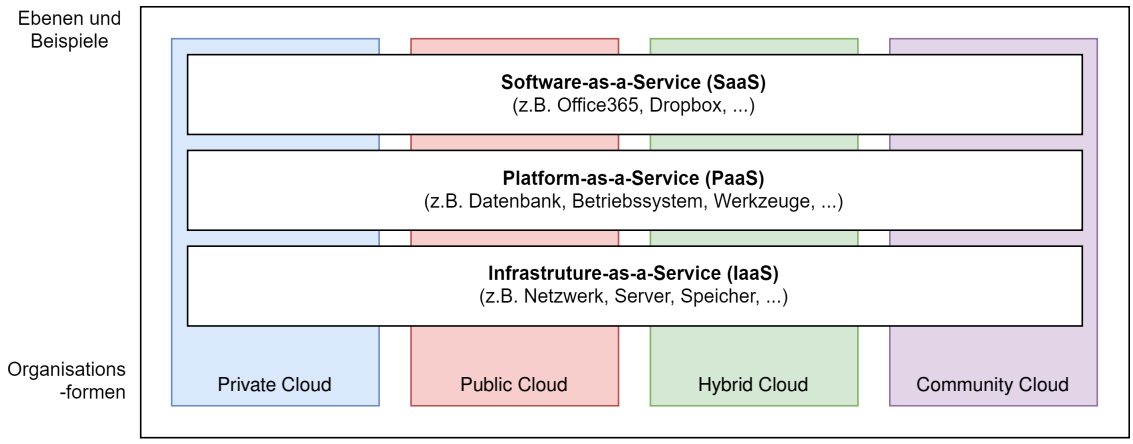

Abbildung 2: *Darstellung der Cloud-Stacks und deren Organisationsformen (in Anlehnung an* Reinheimer *2018, S. 9)*

Im weiteren Verlauf dieser Arbeit soll eine Kubernetes-as-a-Service (KaaS) Plattform in einer hybriden Cloud Organisationsform implementiert werden. Dies soll mit einem Service auf PaaS- oder SaaS-Ebene verglichen werden. Mehr dazu unter Kapitel 5.

2.2 Grundlagen zu DevSecOps

In diesem Abschnitt werden die Grundlagen verschiedener Begriffe rund um das Thema DevSecOps erläutert.

2.2.1 DevOps und die DevOps-Kultur

Um den Begriff „DevSecOps“ zu verstehen, muss zunächst die Herkunft und Bedeutung des Begriffs „DevOps“ geklärt werden. Konkret beschreibt DevOps ein Konzept zur Optimierung der Softwareentwicklung und -bereitstellung, welches durch eine verbesserte Zusammenarbeit von Entwicklungs- und Betriebsteams erreicht werden soll (vgl., auch im Weiteren, Kratzke 2023, S. 27-28). DevOps unterteilt sich dabei in Continuous Integration, Continuous Delivery und Continuous Deployment. Diese drei Praktiken stellen sicher, dass automatisierte Software-Releases in heutigen Entwicklungsprozessen besser optimiert, häufiger und qualitativ hochwertiger bereitgestellt werden können (vgl. Hall 2020). In Kapitel 2.3 werden diese Praktiken im Detail erläutert.

Darüber hinaus wird in der Literatur häufig von einer DevOps-Kultur gesprochen (vgl. Kratzke 2023, S. 28, Hall 2020). Dies ergibt sich aus den Anforderungen, die sich für Unternehmen ergeben, die DevOps erfolgreich einsetzen wollen. Dazu gehört neben der Anpassung vieler Softwaresysteme auch die Umstrukturierung und Organisation im Unternehmen. Ein hohes Maß an Transparenz, Kommunikation und Zusammenarbeit, aber auch gegenseitiges Vertrau-

en und Respekt sind notwendig, damit DevOps funktionieren kann (vgl., auch im Weiteren, Walls 2013, S. 7-8). Dies bezieht sich vor allem auf den Austausch zwischen Entwicklungs- und Betriebsteams, denn nur durch Kommunikation können Qualität, Verfügbarkeit und Erfolg in der Praxis sichergestellt werden. Diese Prämissen bilden quasi das Fundament von DevOps, weshalb es auch als Kultur bezeichnet wird, da diese immer individuell angepasst werden muss und jedes Team in der Umsetzung frei ist. Zusammenfassend lässt sich dies u.a. mit folgenden Ebenen beschreiben (vgl. Kratzke 2023, S. 28):

- Kultur und Arbeitsorganisation
- Architektur
- Deployment-Pipeline
- Deployment Environments
- Telemetriedaten

2.2.2 DevOps Challenges („Elephant in the room"-Problem)

Entscheidet sich ein Unternehmen für den DevOps-Ansatz, wird in der praktischen Umsetzung schnell klar, dass es keine allgemeingültige optimale Lösung für die Umsetzung von Continuous Integration, Continuous Delivery und Continuous Deployment gibt. Jedes Unternehmen bringt seine eigenen Anforderungen, Restriktionen, Budgets, Infrastrukturen und Softwareprodukte mit, was zu einer Vielzahl neuer organisatorischer Herausforderungen innerhalb der Teams führt. Nicht zu vernachlässigen ist auch ein möglicher Mangel an DevOps-Expertise in den Teams. Die Anpassung ist ein komplexer Prozess, der neue Team- und Entwicklungsprozesse erfordert. (Vgl. Walls 2013, S. 10 ff. und Hall 2020)

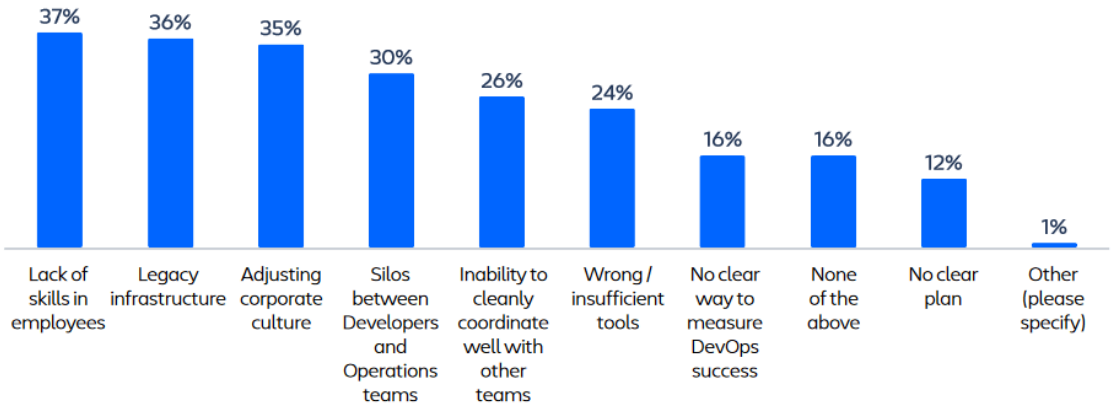

Abbildung 3: *Atlassian DevOps-Umfrage zu Hürden bei der Adaptierung von DevOps im Unternehmen (vgl.* Atlassian *2020)*

Abbildung 3 zeigt eine DevOps-Umfrage aus dem Jahr 2020 der Firma Atlassian. In dieser Online-Umfrage wurden mehr als 500 Personen aus der IT-Branche befragt. Dabei gaben 37%

der Befragten an, dass mangelnde Expertise der Mitarbeiter, dicht gefolgt von der Legacy-Infrastruktur mit 36%, die häufigsten Probleme bei der Adaption von DevOps im Unternehmen sind. Wie in Kapitel 2.2.1 beschrieben, erfordert die Einführung von DevOps signifikante kulturelle Anpassungen, vor allem im Bereich der Kollaboration, was häufig zu Widerständen innerhalb des Unternehmens führt, die die Adaption von DevOps maßgeblich beeinträchtigen. Die Umfrageergebnisse spiegeln diese Widerstände mit ca. 35% an dritter Stelle wider.

Im Rahmen dieser Arbeit wurde ebenfalls eine Umfrage beim betreuenden Unternehmen durchgeführt, um u. a. die in Abbildung 3 erzielten Ergebnisse in einem kleineren Umfeld zu validieren. Die Ergebnisse sind in Kapitel 3 zu finden.

2.2.3 Erweiterung auf DevSecOps

Während DevOps im klassischen Sinne darauf abzielt, die Zusammenarbeit zwischen mehreren Entwicklungs- und Betriebsteams zu optimieren, erweitert DevSecOps diesen Ansatz, indem verschiedene Sicherheitsaspekte von Anfang an in den Entwicklungsprozess integriert werden. Dies verbindet Entwicklung, Sicherheit und Betrieb und unterstreicht die Notwendigkeit, in der modernen Softwareentwicklung Sicherheitsmaßnahmen direkt in den DevOps-Zyklus zu integrieren, um die stetig steigenden Sicherheitsanforderungen zielgerichtet umsetzen zu können (vgl. Falk 2016).

Neben den technologischen Anpassungen erfordert die Einführung von DevSecOps ebenfalls einen kulturellen Wandel, wie in Kapitel 2.2.1 bereits beschrieben. Ein zentraler Aspekt ist dabei die sogenannte „Shift-Left“-Strategie, bei der Sicherheitsüberprüfungen sowohl zu Beginn als auch kontinuierlich während der Entwicklung durchgeführt werden, damit Sicherheitslücken möglichst früh erkannt und behoben werden können. Ziel ist es, qualitativ hochwertige und vor allem sichere Software zu entwickeln. Diese Strategie wird hauptsächlich durch automatisierte Tests innerhalb einer CI/CD-Pipeline umgesetzt. Dabei werden u.a. statische und dynamische Codeanalysen, Secret-Detections, Dependency-Checks etc. durchgeführt (vgl. Falk 2016). Im Idealfall wird jeder Code-Commit auf Sicherheitslücken überprüft, um eine durchgängige Sicherheit zu gewährleisten.

Grundsätzlich ist es den Entwicklungsteams freigestellt, welche Sicherheitsmaßnahmen sie einsetzen, aber gerade bei Sicherheitsfragen sollten wenig Kompromisse gemacht werden. Die Open Worldwide Application Security Project (OWASP) Foundation hat hierzu bereits eine Guideline mit Best Practices für eine sichere CI/CD-Pipeline veröffentlicht (vgl. OWASP 2024b). Danach sollen je nach Software Development Life Cycle (SDLC) bzw. Softwarearchitektur die Schritte aus Abbildung 4 umgesetzt werden.

Die Erläuterung der einzelnen Schritte aus Abbildung 4 sind im folgenden definiert (vgl. OWASP 2024b):

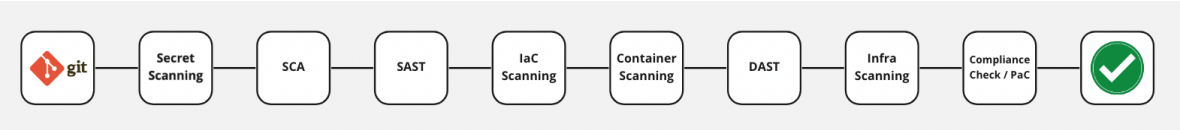

Abbildung 4: *Darstellung der Sicherheitsschritte für eine Basis DevSecOps-Pipeline (vgl.* OWASP *2024b)*

- **Secret Scanning**: Überprüft Git-Repositories auf versehentlich eingebettete Zugangsdaten. Dies hilft, Sicherheitsrisiken zu minimieren, die durch eine unbeabsichtigte Veröffentlichung sensibler Informationen entstehen können.
- **SCA (Software Composition Analysis)**: Analysiert Abhängigkeiten und Fremdbibliotheken auf bekannte Schwachstellen. Da viele Anwendungen auf externen Bibliotheken basieren, ist es wichtig, deren Sicherheit kontinuierlich zu überwachen.
- **SAST (Static Application Security Testing)**: Durchführung statischer Analysen des Quellcodes zur Identifizierung von Sicherheitslücken. Dieser Ansatz ermöglicht es, Schwachstellen frühzeitig im Entwicklungsprozess zu erkennen und zu beheben.
- **IaC Scanning (Infrastructure as Code Scanning)**: Überprüfung von Infrastruktur-Code (z. B. Terraform, Helm Charts) auf Fehlkonfigurationen. Da Infrastruktur zunehmend als Code verwaltet wird, ist es wichtig, diesen Code auf Sicherheitslücken und Fehlkonfigurationen zu prüfen.
- **IAST (Interactive Application Security Testing)**: Kombination von statischen und dynamischen Tests zur Laufzeit der Anwendung.
- **DAST (Dynamic Application Security Testing)**: Durchführung dynamischer Analysen von laufenden Anwendungen, um Sicherheitslücken zu identifizieren. Im Gegensatz zu statischen Tests überprüft DAST die Anwendung in ihrer Laufzeitumgebung.
- **Infrastructure scanning**: Überprüfung der Infrastruktur auf Sicherheitslücken und Konfigurationsfehler. Dies umfasst sowohl physische als auch virtuelle Infrastrukturen und zielt darauf ab, die gesamte Umgebung sicher zu halten.
- **Compliance check**: Überprüfung der Einhaltung von Sicherheitsrichtlinien und regulatorischen Anforderungen. Damit wird sichergestellt, dass Systeme und Prozesse den definierten Standards und gesetzlichen Vorgaben entsprechen.

2.3 Grundlagen zu Deployment-Pipelines

Die Adaption des DevOps-Ansatzes führt in der Regel dazu, dass bestehende Deployment-Mechanismen durch sogenannte Deployment-Pipelines oder auch CI/CD-Pipelines ersetzt werden. Diese sind ein wesentlicher Bestandteil von DevOps und sorgen dafür, dass Code,

der beispielsweise in Repositories gepusht wird, vollautomatisch gebaut, getestet und ausgeliefert wird. Dabei werden häufig verschiedene Umgebungen wie Test, Staging und Produktion genutzt (vgl. Kratzke 2023, S. 49). Damit wird auch sichergestellt, dass jeder neue Code in die bestehende Codebasis integriert werden kann. In der Regel ist es bei einer ordnungsgemäßen Integration von CI/CD-Lösungen nicht möglich, Code zu generieren, der ungetestet durch die Pipeline geschleust wird.

Grundsätzlich basieren CI/CD-Lösungen, egal ob in der Cloud oder on-prem gehostet, auf mehreren Phasen. Diese können in der Regel von den Betriebsteams je nach Einsatzzweck dynamisch definiert werden. Jede Phase kann einen oder mehrere so genannte „Jobs“ enthalten, die parallel, unabhängig und vollständig isoliert voneinander ausgeführt werden. Tritt in einem Job ein Fehler auf, so scheitert die Phase und der weitere Ablauf wird unterbrochen. Auf diese Weise wird sichergestellt, dass jede Phase vollständig und fehlerfrei beendet wird (vgl. Kratzke 2023, S. 49). Die eigentliche Implementierung erfolgt in der Regel nach einem festen Schema, das sich in der Praxis bewährt hat:

1. **Build-Phase:** Kompilieren des Quellcodes mit allen Abhängigkeiten und relevanten Daten. Dies wird meist mit Docker-Containern realisiert.
2. **Test-Phase:** Definierte Tests wie Unit-Tests etc. werden auf dem aktuellen Stand ausgeführt.
3. **Deploy-Phase:** Das fertige Image wird in der Regel in ein Image-Repository gepusht oder direkt in z. B. einen Kubernetes-Cluster deployt.

2.3.1 Phasen-Pipelines

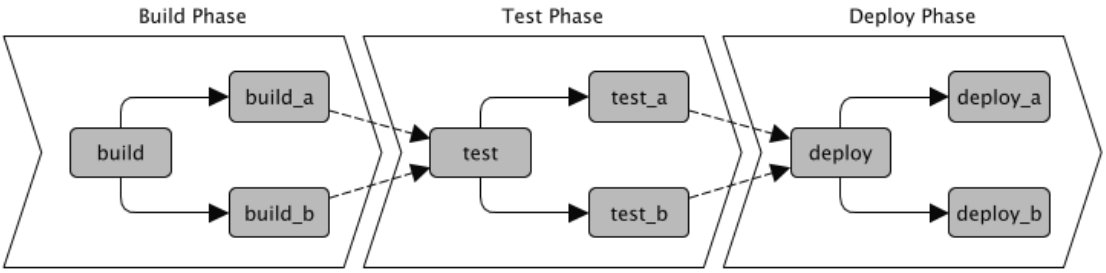

Abbildung 5: *Beispiel einer Phasen-Pipeline mit drei Phasen (vgl.* Kratzke *2023, S. 51)*

Die Abbildung 5 zeigt eine beispielhafte Darstellung eines Pipeline-Patterns der sogenannten Phasen-Pipeline. Bei diesem Pattern beginnt die nächste Phase erst, wenn alle Schritte der vorhergehenden Phase erfolgreich abgeschlossen wurden. Tritt ein Fehler auf, wird die gesamte Pipeline abgebrochen. Dieses Pattern wird häufig bei kleinen bis mittelgroßen Projekten eingesetzt, da es oft mit wenig Aufwand implementiert werden kann und noch eine relativ überschaubare Durchlaufzeit erreicht.

2.4 Grundlagen zu GitOps

GitOps spielt eine wesentliche Rolle im Cloud Computing. Gerade Themen wie DevOps oder Infrastructure as Code (IaC) profitieren maßgeblich davon. Ziel ist es, die gesamte Konfiguration und Verwaltung von z. B. Infrastruktur in Git-Repositories abzulegen, so dass jederzeit eine Versionierung und Nachvollziehbarkeit besteht. Diese Git-zentrierte Arbeitsweise ist bereits aus der Softwareentwicklung bekannt und wurde für Themen rund um Kubernetes übernommen und angepasst.

Kern von GitOps ist das Konzept der deklarativen Konfiguration, d. h. alle Systeme und Konfigurationen werden in Form von YAML- oder JSON-Dateien definiert und in einem Git-Repository gespeichert. Dadurch kann eine einzige Quelle der Wahrheit (engl. Single Source of Truth) erreicht werden. Änderungen werden in der Regel über Pull Requests angefordert und nach Validierung durch einen autorisierten Benutzer freigegeben. Moderne Continuous Delivery (CD)-Tools wie ArgoCD oder FluxCD können die Änderungen dann vollautomatisch in die gewünschten Systeme wie Kubernetes etc. einspielen. Ein weiterer Aspekt ist die kontinuierliche Überwachung und Synchronisation des Sollzustandes mit dem Istzustand der Laufzeitumgebung. Tools wie FluxCD und ArgoCD überwachen kontinuierlich die Laufzeitumgebung und vergleichen sie mit den Konfigurationsdateien im Git-Repository. Abweichungen werden automatisch korrigiert, um die Umgebung wieder in den gewünschten Zustand zu bringen. Manuelle Änderungen oder versehentliche Anpassungen können so automatisch rückgängig gemacht werden. Die Abweichung von Ist- und Soll-Zustand wird auch als „Drift“ bezeichnet. GitOps sorgt dafür, dass dieser in der Praxis nicht entsteht.

2.5 Grundlagen zu Kubernetes

In diesem Kapitel werden zunächst die Grundlagen der Orchestrierungsplattform Kubernetes in Kapitel 2.5.1 und Kapitel 2.5.2 erläutert. Anschließend wird das Konzept der Hochverfügbarkeit in Kapitel 2.5.3 und die Ansätze zur Mandantenfähigkeit in Kapitel 2.5.4 vorgestellt. Abschließend folgt noch die Beschreibung wichtiger Werkzeuge für den Betrieb und die Administration von Kubernetes.

2.5.1 Kubernetes Komponenten

Kubernetes ist eine Open-Source-Technologie zur Orchestrierung von Containern in einem verteilten System. Konkret bedeutet dies, dass verschiedene Workloads in Form von Containern systematisch auf physische oder virtuelle Maschinen verteilt werden. Diese müssen unter anderem die in Kapitel 2.1.3 beschriebenen Cloud-nativen Anforderungen erfüllen. Kubernetes übernimmt dabei die komplette Steuerung und Verwaltung der Ressourcen durch horizontale und vertikale Skalierung sowie die Sicherstellung der Verfügbarkeit durch Red-

undanz von Containern und Hardware. Eine Einheit aus mehreren Servern bzw. Nodes wird auch als Cluster bezeichnet (vgl. Kubernetes 2024n). Die Architektur hinter Kubernetes wird in Kapitel 2.5.2 beschrieben, zuvor sollen jedoch einige wichtige Grundkomponenten von Kubernetes erläutert werden, um die Zusammenhänge innerhalb eines Clusters zu verstehen.

Pod

Ein Pod ist die kleinste Arbeitseinheit in Kubernetes und kann aus einem oder mehreren Containern bestehen. Pods teilen sich Ressourcen wie Netzwerk und Speicher und stellen die grundlegende Laufzeiteinheit in Kubernetes dar. Die Container innerhalb der Pods kommunizieren über localhost oder über Interprocess Communication (IPC) (vgl. Kubernetes 2024m).

Namespace

Eine weitere wichtige Komponente sind die Namespaces. Diese bieten die Möglichkeit, Kubernetes-Objekte innerhalb eines Clusters zu partitionieren. Sie ermöglichen die logische Trennung und Verwaltung von Ressourcen. Namespaces liegen nicht auf den Nodes selbst, da Workloads innerhalb eines Namespaces auf verschiedene Nodes verteilt werden können. Sofern keine Netzwerkrichtlinien für den jeweiligen Namespace definiert sind, können die Pods bzw. Deployments untereinander und Namespace-übergreifend frei kommunizieren (vgl. Kubernetes 2024i).

Deployment

Ein Deployment ermöglicht die Bereitstellung und Verwaltung von Pods. Es bietet Funktionen zum Skalieren, Aktualisieren und Verwalten der gewünschten Anwendungszustände und stellt sicher, dass zu jedem Zeitpunkt die richtige Anzahl von Pod-Replikaten ausgeführt wird (vgl. Kubernetes 2024d).

Label

Labels sind Schlüssel-Wert-Paare, die an Kubernetes-Objekte angehängt werden. Sie werden verwendet, um Objekte nach bestimmten Kriterien zu organisieren und auszuwählen. Labels sind entscheidend für die Identifizierung und Filterung von Ressourcen im Cluster (vgl. Kubernetes 2024g).

Label-Selector

Label Selectors sind Abfragen, die verwendet werden, um eine Gruppe von Objekten anhand ihrer Labels auszuwählen. Sie sind entscheidend für das Filtern und Verwalten von Ressourcen (vgl. Kubernetes 2024g).

Persistentes Volume (Claim)

Ein Persistent Volume (PV) ist eine vom Cluster bereitgestellte Speichereinheit mit einer bestimmten Speicherkapazität, die unabhängig vom Lebenszyklus eines Pods existiert. Da Container zustandslos sind, müssen persistente Daten im PV gespeichert werden, wenn sie einen Neustart oder Absturz überleben sollen. Ein Persistent Volume Claim (PVC) ist eine von einem Benutzer angeforderte Speicheranforderung, die von einem PV erfüllt werden kann, wenn im Cluster eine Storage Class definiert ist. (siehe Kubernetes 2024l).

StorageClass

Eine Storage Class kann verwendet werden, um verschiedene Speichertypen zu definieren, die von Persistent Volumes verwendet werden können. Sie bietet eine Möglichkeit, verschiedene Speicheranforderungen wie Performance, Verfügbarkeit und Zugriffsmodi wie ReadWriteMany, ReadWriteOnce etc. zu spezifizieren. Normalerweise wird eine Storage Class als Standard festgelegt, die automatisch ausgewählt wird, wenn Speicher angefordert wird (vgl. Kubernetes 2024t).

Ingress

Ein Ingress ist ein API-Objekt, das HTTP- und HTTPS-Routen zu Diensten innerhalb des Clusters definiert. Es ermöglicht das Routing von externem Datenverkehr zu den richtigen Services auf Basis von URL-Pfaden oder Hostnamen (vgl. Kubernetes 2024e).

Service

Ein Service definiert eine logische Gruppe von Pods, die im Cluster eine feste IP-Adresse und einen DNS-Namen haben. Ein Service kann in Kombination mit einem Deployment erstellt werden. Services ermöglichen die Kommunikation zwischen Pods und bieten eine automatische Lastverteilung des eingehenden Traffics. Durch ein Port Mapping ist es auch möglich, diesen durch z. B. einen Ingress (vgl. Kubernetes 2024q) extern freizugeben.

StatefulSet

Ein StatefulSet ist eine Controller-Ressource in Kubernetes, die das Deployment und die Verwaltung von Pods mit Zustandsinformationen ermöglicht. Im Gegensatz zu Deployments, die stateless sind, bewahrt ein StatefulSet die Identität und den Status der einzelnen Pods wenn diese neugestartet oder gelöscht werden (vgl. Kubernetes 2024s).

ConfigMap

Mit Hilfe einer ConfigMap können Konfigurationsdaten in Form von Schlüssel-Wert-Paaren einem Deployment zur Verfügung gestellt werden. Dadurch ist es möglich, Konfigurationsdaten von Container-Images zu trennen und dynamisch zu verwalten. Dies ist vergleichbar mit lokalen Umgebungsdateien (.env) (vgl. Kubernetes 2024c).

Secret

Ein Secret ist eine spezielle Art von ConfigMap, die sensible Daten wie Passwörter, Tokens oder Schlüssel speichert. Secrets bieten eine sichere Möglichkeit, vertrauliche Informationen im Cluster zu verwalten (vgl. Kubernetes 2024p).

Service Account

Ein Service Account dient als API-Zugang zu Kubernetes innerhalb eines Pods. Er verfügt je nach Konfiguration über bestimmte Rechte und Einschränkungen. Wird einem Deployment ein Service Account zugewiesen, wird ein Access Token in Form eines JWT Tokens in den Container auf einen standardisierten Pfad gemountet. Über eine Kubernetes Library kann dann aus der Anwendung innerhalb des Containers in Echtzeit auf den Kubernetes Cluster zugegriffen werden, um z. B. Config Maps oder Secrets abzufragen (vgl. Kubernetes 2024r).

Annotation

Annotationen sind Schlüssel-Wert-Paare, die Metadaten zu Kubernetes-Objekten hinzufügen. Im Gegensatz zu Labels werden Annotations nicht zur Identifikation oder Selektion verwendet, sondern dienen dazu, Objekte mit zusätzlichen Informationen zu versehen (vgl. Kubernetes 2024a).

Ressource Quota

Ressource Quotas werden verwendet, um die Nutzung von Ressourcen innerhalb eines Namespaces einzuschränken. Mit Hilfe von Ressource Quotas können Administratoren sicherstellen, dass die verfügbaren Ressourcen gerecht verteilt werden und eine übermäßige Nutzung vermieden wird (vgl. Kubernetes 2024o).

Network Policy

Network Policies ermöglichen die Steuerung des Datenverkehrs zwischen einzelnen Pods. Sie bieten eine Möglichkeit, die Netzwerkkommunikation innerhalb des Clusters zu regeln und Sicherheitsrichtlinien durchzusetzen (vgl. Kubernetes 2024j).

2.5.2 Kubernetes Architektur

Neben den grundlegenden Basis-Komponenten, die in Kapitel 2.5.1 vorgestellt wurden, wird nun der Aufbau und das Zusammenspiel von Master- und Worker-Node beschrieben. Diese beiden Node-Komponenten bilden zusammen einen funktionsfähigen Cluster. Die sogenannte „Control Plane“, die auf den Master-Nodes läuft, ist für die Orchestrierung der Workloads über den Scheduler sowie für den Empfang von Befehlen über den API-Server verantwortlich.

In Abbildung 6 ist die grundlegende Architektur von Kubernetes mit ihren Basiskomponenten dargestellt. Direkt darunter befindet sich die Erklärung der einzelnen Komponenten.

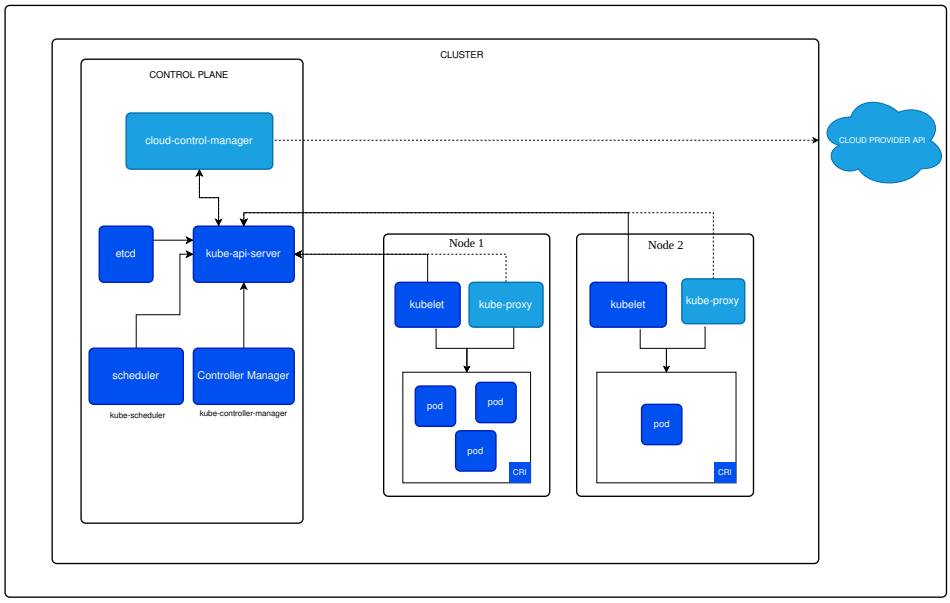

Abbildung 6: *Darstellung der Kubernetes Architektur (vgl.* Kubernetes *2024b)*

Control Plane Komponenten

etcd

etcd ist eine verteilte Key-Value-Datenbank, die den gesamten Cluster-Status speichert. Diese hochverfügbare Datenbank stellt sicher, dass alle anderen Kubernetes-Komponenten konsistente und zuverlässige Statusinformationen erhalten (vgl. Aly und Murat 2021, S. 4).

kube-api-server

Der kube-api-server ist die zentrale Management-Schnittstelle für den Kubernetes-Cluster. Er empfängt, validiert und verarbeitet RESTful API-Anfragen und dient als Kommunikationsschnittstelle zwischen allen Kubernetes-Komponenten (vgl. Aly und Murat 2021, S. 4).

Scheduler

Der Scheduler ist verantwortlich für die Verteilung neuer Pods auf die verfügbaren Worker im Cluster. Dabei berücksichtigt er Ressourcenanforderungen, Hardware-Spezifikationen und andere benutzerdefinierte Einschränkungen, um eine optimale Platzierung der Pods zu gewährleisten (vgl. Aly und Murat 2021, S. 4).

Controller Manager

Der Controller Manager enthält verschiedene Controller, die den gewünschten Zustand des Clusters aufrechterhalten. Dazu gehören unter anderem der Node-Controller, der Replication-Controller und der Endpoints-Controller, die alle gemeinsam dafür sorgen, dass der Cluster reibungslos funktioniert (vgl. Aly und Murat 2021, S. 4).

Worker Node Komponenten

Kubelet

Das Kubelet ist ein Agent, der auf jedem Worker Node läuft und für die Verwaltung der Pods auf diesem Node verantwortlich ist. Es kommuniziert mit der Control Plane und stellt sicher, dass die Pods gemäß ihren Spezifikationen ausgeführt werden (vgl. Aly und Murat 2021, S. 4).

Container Runtime

Die Container Runtime ist die Software, die die Container tatsächlich ausführt. Kubernetes unterstützt verschiedene Container Runtimes wie Docker, containerd und CRI-O, die für die Ausführung und Verwaltung der Container zuständig sind (vgl. Aly und Murat 2021, S. 5).

Kube-Proxy

Der Kube-Proxy ist für die Netzwerkkommunikation innerhalb des Clusters zuständig. Er verwaltet die Netzwerkregeln und ermöglicht die Kommunikation zwischen den verschiedenen Diensten im Cluster (vgl. Aly und Murat 2021, S. 5).

2.5.3 High-Availability (HA)

Ein zentraler Aspekt von Kubernetes ist die Ausfallsicherheit eines Clusters. Dies wird in der Literatur auch als „High Availability (HA)“ bezeichnet (vgl. Kubernetes 2024k). In der Praxis kann dieser Zustand durch Redundanzen erreicht werden. Dabei muss insbesondere die Control Plane vor einem Totalausfall geschützt werden, da sonst ein Ausfall des gesamten Clusters die Folge wäre. Die in Kapitel 2.5.2 dargestellte Architektur ist dagegen bereits so konzipiert, dass alle Komponenten der Control Plane automatisch auf weitere Master-Knoten repliziert werden. Mit Hilfe von sogenannten Load Balancern, die sich meist außerhalb des Clusters befinden, wird die Last auf einen oder mehrere Nodes verteilt. Fällt ein Knoten aus, wird der Traffic auf die noch vorhandenen Master-Knoten verteilt. Somit gibt es zu keinem Zeitpunkt einen Single Point of Failure. Bereits ab drei Master-Nodes innerhalb eines Clusters spricht man von einem High Availability Cluster (vgl. Aly und Murat 2021, S. 7).

Möchte man dies weiter absichern, so muss die Anzahl der Master Nodes in ungeraden Schritten mit folgender Formel weiter skaliert werden $Quorum = (n/2) + 1$ wobei n die Anzahl der

Master Nodes darstellt. Das Quorum wird für den im etcd-Server integrierten Konsensalgorithmus benötigt. Damit kann ein Datenverlust bei Ausfällen verhindert werden, da immer durch eine Mehrheit sichergestellt wird, dass Änderungen nur dann übernommen werden, wenn das Quorum erreicht ist.

Neben der Control Plane müssen auch die Worker Nodes entsprechend hochverfügbar ausgelegt werden. Dies wird durch die Verteilung der Pods auf verschiedene Nodes erreicht. Kubernetes verwendet dabei Mechanismen wie ReplicaSets und Deployments, um sicherzustellen, dass immer die gewünschte Anzahl von Pod-Replikaten läuft. Fällt ein Node aus, werden die betroffenen Pods automatisch auf andere verfügbare Nodes verschoben, so dass ein unterbrechungsfreier Betrieb der Anwendungen gewährleistet ist.

2.5.4 Mandantenfähigkeit (Multi-Tenancy)

Unter Multi-Tenancy versteht man im Zusammenhang mit Kubernetes ein Konzept, das es ermöglicht, dass sich verschiedene Akteure im selben Cluster befinden können, so dass diese isoliert und unabhängig voneinander agieren können. Der größte Vorteil dieses Ansatzes ist die Kostenreduktion bei der Anschaffung, dem Betrieb und der Administration des Clusters. Gerade bei physischen Servern können die Anschaffungskosten schnell alle Budgets sprengen (vgl., Kubernetes 2024h). Bei der Implementierung von Mandantenfähigkeit müssen verschiedene Aspekte wie Sicherheit, Isolation, Ressourcenkontrolle und Netzwerksegmentierung berücksichtigt werden, um den gewünschten Grad an Isolation zu erreichen. In der Literatur finden sich hierzu häufig Begriffe wie „Soft“ oder „Hard“ Multi-Tenancy, die bestimmte Eigenschaften der oben genannten Aspekte stärker oder schwächer betonen. So wird die „Soft“ Multi-Tenancy als eine schwächere Isolierung im Vergleich zur „Hard“ Multi-Tenancy dargestellt. Letztere kommt vor allem dann zum Einsatz, wenn sich die einzelnen Interessengruppen gegenseitig nicht vertrauen (vgl. Kubernetes 2024h). Vertrauen ist dabei ein wichtiger Indikator bei der Auswahl des Isolationslevels.

Da es, wie bereits erwähnt, keine native Implementierung von Mandantenfähigkeit gibt, müssen verschiedene Ansätze miteinander verglichen werden, um den gewünschten Grad an Isolation zu erreichen. In den letzten Jahren haben sich drei Methoden auf dem Markt etabliert. In Abbildung 7 sind diese dargestellt.

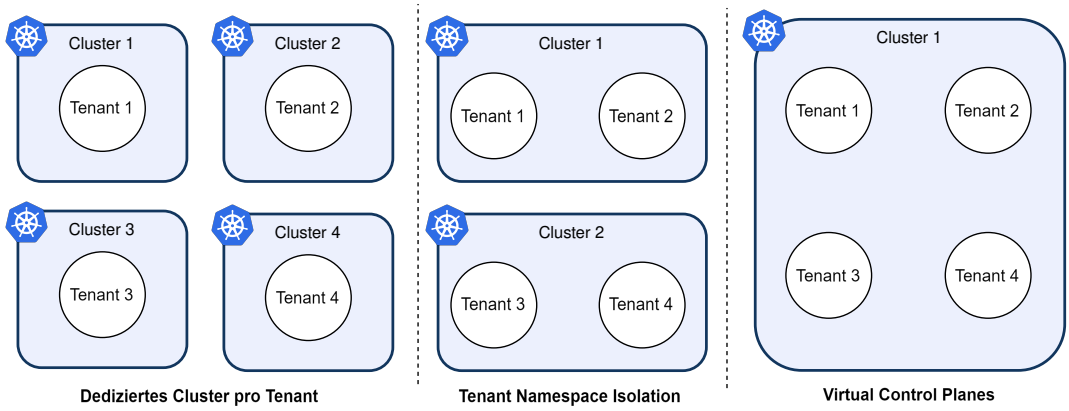

Abbildung 7: *Darstellung der Multi-Tenancy Ansätze (in Anlehnung an* Hwang *und* Newman *2023)*

Die erste Methode ganz links gehört zur Kategorie der harten Mandantenfähigkeit. Dabei wird für jeden Mandanten ein eigener Cluster eingerichtet, der physisch oder virtuell sein kann. Während der Vorteil die vollständige Isolation und maximale Sicherheit zwischen den Mandanten ist, ist dies mit hohen Kosten und Administrationsaufwand verbunden. Zudem kann es zum sogenannten Cluster-Sprawl-Problem kommen, bei dem so viele Maschinen verwaltet werden müssen, dass es irgendwann zu komplex bzw. zu aufwendig wird, alles zu warten. Dies kann spürbare Auswirkungen auf die Verfügbarkeit und Sicherheit haben (vgl. Hossain 2023). Jedoch ist dieser Ansatz mit am weitesten verbreitet, da häufig für z. B. Entwicklungs- und Produktivsysteme seperate Cluster zum Einsatz kommen (vgl., Hwang und Newman 2023, auch im Weiteren,).

Der zweite Ansatz in der Mitte gehört zur Kategorie der weichen Mandantenfähigkeit. Hier befinden sich mehrere Mandanten innerhalb eines Clusters, die durch Kubernetes-eigene Funktionen wie Netzwerkrichtlinien, Ressource Quotas und Rollen voneinander getrennt sind. Die Mandanten liegen in dedizierten Namespaces. Für diese Methode wird kein zusätzliches Framework benötigt, das die Implementierung übernimmt. Dies kann sowohl ein Vor- als auch ein Nachteil sein, da bei einer Fehlkonfiguration die Isolation möglicherweise nicht mehr gegeben ist. Daher kann bzw. sollte dieser Ansatz nur in Umgebungen eingesetzt werden, in denen sich die Clients gegenseitig vertrauen. Ein mögliches Einsatzszenario wäre z. B. Teammitglieder für einen dedizierten Team-Cluster.

Der dritte und letzte Ansatz sind Virtual Control Planes (VCPs). Diese Methode ist relativ neu auf dem Markt und kombiniert die ersten beiden Ansätze. Es wird dabei nur ein großer physischer Cluster verwendet, der auch als Host- bzw. Management-Cluster bezeichnet wird. Innerhalb dieses Clusters werden für jeden Mandanten separate virtuelle Control Planes erichtet. Diese VCPs sorgen für eine vollständige Isolierung der verschiedenen Mandanten, während die zugrunde liegende Infrastruktur weiterhin gemeinsam genutzt wird. Dabei erhält jeder Mandant seine eigenen Komponenten wie API Server, Controller Manager oder etcd.

Er ist somit vollständig vom Host-Cluster entkoppelt. Hinzu kommt die Kombination aus hoher Sicherheit und effizienter Ressourcennutzung. Dies reduziert die Betriebskosten und den Administrationsaufwand gegenüber der ersten Methode erheblich. Nachteil dieses Ansatzes ist jedoch, dass er nur in Kombination mit einem Framework realisiert werden kann. Derzeit gibt es etwa eine Handvoll Lösungen auf dem Markt, wie Loft vCluster, Capsule, Kiosk und VirtualCluster (vgl. Hwang und Newman 2023).

2.5.5 Kubectl

Die „kubectl“ ist ein Kommandozeilentool (CLI) welches zur Verwaltung von Kubernetes-Clustern genutzt wird. Es ermöglicht alle Konfigurationen und Befehle direkt an den Kubernetes API Server zu senden. Zur Authentifizierung wird die lokale kubeconfig verwendet, die die Authentifizierungsdaten enthält.

2.6 Verschiedene DevOps-Tools

In diesem Kapitel werden wichtige Tools aus dem DevOps- und Kubernetes-Umfeld vorgestellt, die unter anderem zur Umsetzung des DevOps-Ansatzes eingesetzt werden.

2.6.1 Helm

Helm ist ein Paketmanager für Kubernetes, ähnlich wie NPM für JavaScript oder PIP für Python. Er ermöglicht die Installation und Konfiguration von Anwendungen in einem Kubernetes-Cluster mit nur einem Befehl. Die Steuerung erfolgt über die Helm-CLI oder über eine Deployment-Software wie Flux oder ArgoCD. Der Vorteil von Helm ist die nahtlose Integration in das Kubernetes-Ökosystem, da die für die Applikation benötigten Kubernetes-Definitionen in Form von YAML-Dateien erstellt werden. Dabei können innerhalb der YAML-Ressourcen sämtliche Kubernetes Definition verwendet werden, welche sonst auch manuell über die kubectl an den Kubernetes-API-Server gesendet werden könnten.

Helm verwendet sogenannte Charts, die vorgefertigte Kubernetes-Ressourcenpakete enthalten. Diese Charts bestehen aus einer Sammlung von YAML-Dateien, die die verschiedenen Komponenten und Abhängigkeiten einer Anwendung beschreiben. Dadurch wird es möglich, komplexe Anwendungen konsistent und wiederholbar zu deployen. Ein Helm-Chart kann einfach an die Bedürfnisse einer bestimmten Umgebung angepasst werden, indem Werte über eine Datei values.yaml oder direkt über die Kommandozeile überschrieben werden. Die Datei values.yaml enthält Konfigurationsparameter, die vom Chart-Ersteller frei gewählt werden können. Die dort eingetragenen Informationen werden dann in den eigentlichen Kubernetes-YAML-Dateien ersetzt, bevor diese an Kubernetes gesendet werden (vgl. Helm 2024). In

der Praxis werden häufig verschiedene values.yaml-Dateien für unterschiedliche Umgebungen gepflegt. Diese können z. B. values.prod.yaml, values.dev.yaml oder values.base.yaml heißen.

Grundsätzlich reduziert der Einsatz von Helm die Komplexität der Bereitstellung und Verwaltung von Kubernetes-Anwendungen erheblich. Die Integration mit CI/CD-Tools ermöglicht automatisierte und kontinuierliche Deployments, was insbesondere bei Kubernetes-as-a-Service von Vorteil ist, da so bestimmte Applikationen automatisch mitinstalliert werden können. Allerdings hat Helm auch einige Nachteile, die vor allem die Lesbarkeit bei großen Helm Chart Konfigurationen betreffen. So kommt es nicht selten vor, dass große Applikationen teilweise mehrere tausend Zeilen lange values.yaml Dateien ausliefern, die erst analysiert werden müssen, um dort die richtigen Parameter zu setzen.

2.6.2 Kustomization

Kustomization ist neben Helm ein weiteres wichtiges Werkzeug im Kubernetes-Ökosystem. Es dient der Verwaltung und Konfiguration von Kubernetes-Ressourcen, allerdings ohne Templates wie bei Helm. Kustomization setzt auf deklarative Konfigurationsdateien, um Kubernetes-Ressourcen zu definieren und anzupassen. Dies ist sowohl ein Vor- als auch ein Nachteil gegenüber Helm, da ohne Templating die Komplexität reduziert wird, aber die Flexibilität bei der dynamischen Erstellung und Anpassung von Konfigurationen eingeschränkt sein kann. Dennoch bietet Kustomization eine klare Strukturierung durch die Verwendung von kustomization.yaml Dateien, in denen Basisressourcen und Overlays definiert werden (vgl. Authors 2024b). In Abbildung 8 ist ein Beispiel dieser Struktur dargestellt.

- **Basisresourcen** sind die grundlegenden Kubernetes-Objekte, die in standardisierter Form vorliegen. Diese Objekte wie Deployments, Services, ConfigMaps und Secrets bilden die Ausgangskonfiguration, die in verschiedenen Umgebungen verwendet wird. Basisressourcen sind in der Regel unverändert und werden in der Datei kustomization.yaml als resources angegeben.

- **Overlays** sind Anpassungen, die auf Basisressourcen angewendet werden, um sie an spezifische Anforderungen oder Umgebungen anzupassen. Overlays können Änderungen an bestehenden Basisressourcen vornehmen, zusätzliche Ressourcen hinzufügen oder bestimmte Konfigurationen überschreiben. Diese Anpassungen werden ebenfalls in der Datei kustomization.yaml definiert und können z. B. über patchesStrategicMerge oder patchesJson6902 implementiert werden. Overlays ermöglichen es, Konfigurationen für verschiedene Umgebungen wie Entwicklung, Test und Produktion zu erstellen, ohne die Basisressourcen selbst ändern zu müssen.

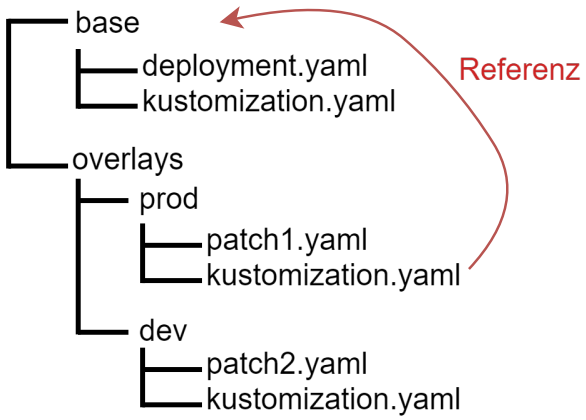

Abbildung 8: *Beispiel einer Kustomize-Dateistruktur (eigene Darstellung)*

Durch die Trennung von Basisressourcen und Overlays bietet Kustomization eine klare und wiederverwendbare Struktur für Kubernetes Konfigurationen. Die Basisressourcen bilden eine stabile Grundlage, während die Overlays flexible Anpassungen an spezifische Anforderungen und Umgebungen ermöglichen.

2.6.3 FluxCD

Zur Automatisierung von Continuous Deployment (CD) im Kontext von Kubernetes wird das Open-Source-Tool Flux bzw. FluxCD aus dem Jahr 2016 verwendet (vgl., auch im Weiteren, Authors 2024a). Es ermöglicht die Automatisierung und Bereitstellung von Anwendungen und deren Konfigurationen direkt aus einem Versionskontrollsystem wie Git. Dabei ist Flux vollständig kompatibel mit dem GitOps-Ansatz und nutzt das sogenannte Pull-Prinzip, um Änderungen in einem Git-Repository zu erkennen und mit einem Kubernetes-Cluster zu synchronisieren. Dabei wird das Quell-Repository ständig überwacht. Die Anwendung selbst läuft in einem eigenen Namespace innerhalb des Clusters. Durch die Verwendung von Service Accounts und Role-Based-Access-Control (RBAC) wird sichergestellt, dass Flux nur die notwendigen Berechtigungen erhält, um die verwalteten Kubernetes Ressourcen zu verändern. Dies minimiert das Sicherheitsrisiko und stellt sicher, dass nur autorisierte Bereiche verändert werden können.

Die Installation von Flux erfolgt typischerweise mit dem mitgelieferten Bootstrapping-Tool. Dieser Schritt verwendet intern Helm, um FluxCD im Cluster zu verteilen. Der Wartungs- und Konfigurationsaufwand geht bei einfachen Installationen gegen Null, was auch eine automatisierte Installation beim Cluster-Bootstrapping ermöglicht. Dies ist besonders bei mandantenfähigen Clustern von Vorteil, da so jeder Mandant eine eigene Instanz von Flux erhält und diese mit einem eigenen Git-Repository verwalten kann. Zugriffsbeschränkungen werden auch in diesem Fall durch RBAC sichergestellt. Die Steuerung von Flux erfolgt mittels de-

klarativer YAML-Dateien über das Git-Repository oder über die eigene Flux-CLI, die lokal beim jeweiligen Administrator ausgeführt wird.

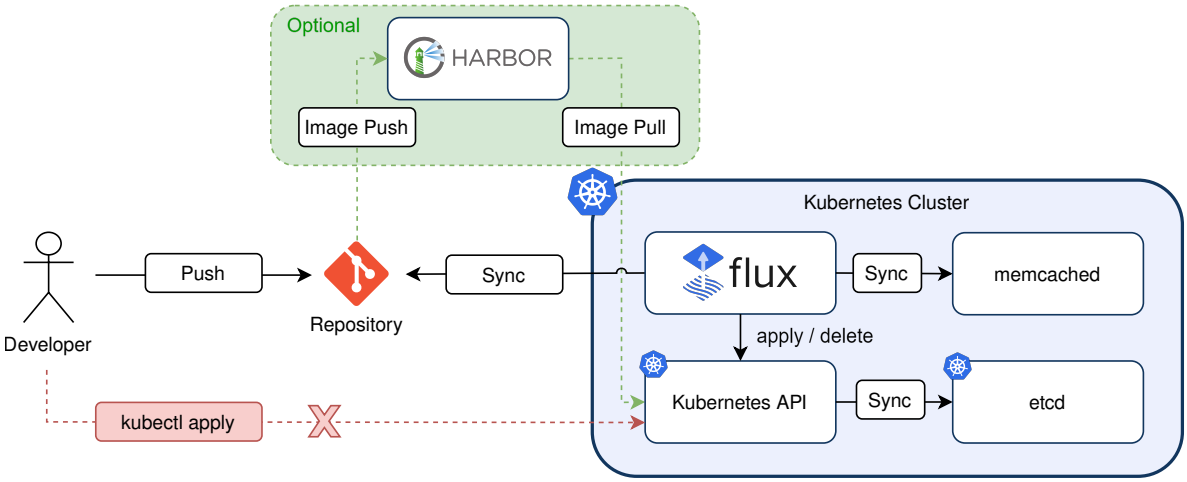

Abbildung 9: *Darstellung der Flux-Architektur (eigene Darstellung in Anlehnung an* Raju *2023)*

In Abbildung 9 wird die Architektur von Flux beschrieben. Diese besteht aus dem Flux-Daemon und dem Key-Value-Cache Memcached, der hauptsächlich zum Zwischenspeichern von Metainformationen über Docker-Images verwendet wird, wenn Flux so konfiguriert ist, dass es eine Docker-Image-Registry wie z. B. Harbor in Echtzeit überwachen soll. Die Verwendung von Memcached ist optional. Der Prozess wird von einem Entwickler durch einen Git-Push in das von Flux überwachte Repository gestartet. Dieser Commit wird nach einem definierten Zeitintervall von üblicherweise fünf Sekunden erkannt und Flux beginnt sofort mit der Synchronisation dieser Änderung im Cluster. Zur Erkennung der Änderungen wird ein Source-Controller verwendet, der im Daemon läuft. Die Änderungen werden dann über einen Service Account an den Kubernetes API Server weitergeleitet, der dann für die weitere Ausführung der Änderung sorgt. Dies funktioniert auch in die andere Richtung, sollte eine manuelle Änderung im Cluster durch das kubectl erfolgen, so erkennt FluxCD dies und verwirft diese Änderung wieder, so dass wieder der Stand aus dem Git-Repository gilt.

3 Umfrage zur Adaption der DevOps-Kultur im Unternehmen

In diesem Kapitel werden der Aufbau, die Durchführung sowie die Auswertung der qualitativen und quantitativen Online-Befragung mit dem Titel „Adopting DevOps Culture“ beschrieben, die sich unter anderem mit den Themen DevOps, Remote Development, Kubernetes und der Adaption der DevOps-Kultur befasst. Die Umfrage wurde innerhalb des betreuenden Unternehmens durchgeführt. In Kapitel 3.1 werden zunächst die verwendeten Forschungsmethoden näher erläutert, um mit diesen in Kapitel 3.2 geeignete Fragen zu formulieren. Die Durchführung der Online-Umfrage wird in Kapitel 3.3 beschrieben. Alle Ergebnisse und Interpretationen der Umfrage lassen sich in Kapitel 3.4 finden.

3.1 Vorgehensweise mittels qualitativer und quantitativer Forschungsmethoden

In Kapitel 2.2 wurden bereits einige Hürden bei der Adaption einer DevOps-Kultur in Unternehmen beschrieben. In dieser Online-Umfrage soll mit Hilfe von qualitativen und quantitativen Methoden in Form von Fragen ermittelt werden (vgl. Abuhamda, Ismail und Bsharat 2021, S. 73), inwieweit sich eine DevOps-Kultur bereits im Unternehmen etabliert hat. Des Weiteren soll überprüft werden, ob die Umfrageergebnisse aus Kapitel 2.2.2 auch auf dieses Unternehmen übertragbar sind. Dabei werden vor allem quantitative Fragen zu Häufigkeiten, Wichtigkeit und einfachen Ja-/Nein-Fragen für statistische Auswertungen formuliert, um ein Gesamtbild des Unternehmens und den dort beschäftigten Personen zeichnen zu können. Dabei werden sowohl allgemeine Daten wie Berufserfahrung und aktuelle Position als auch Kenntnisse über bestimmte Technologien erfasst, die zusätzlich bei der Definition der Anforderungen in Kapitel 4.2 helfen sollen.

Die qualitativen Fragen zielen in erster Linie darauf ab, den individuellen Wissensstand zu den einzelnen Themen der Befragung zu ermitteln. Dabei werden gezielt Fragen formuliert, die eine Freitexteingabe ermöglichen, um die Teilnehmer nicht in ihrem Denken zu beeinflussen und ein möglichst realistisches Bild des aktuellen Wissensstandes der Teilnehmer zu erfassen. Die Freitexteingaben haben zudem den Vorteil, dass sie eine differenziertere Analyse der unterschiedlichen Sichtweisen und Erfahrungen innerhalb des Unternehmens ermöglichen, die mit quantitativen Fragen möglicherweise nicht vollständig erfasst werden können.

3.2 Aufbau der Umfrage und der Fragen

Die Umfrage wird mit der Open-Source Lösung „Formbricks“ durchgeführt, welche bereits im Unternehmen für anderweitige Umfragen eingesetzt wird. Da der überwiegende Anteil der Zielgruppe im Home Office ansässig ist, wird eine Online-Befragung durchgeführt, um eine

größtmögliche Anzahl an Teilnehmern zu erzielen. Die Umfrage selbst ist in vier Themengebiete eingeteilt:

1. Allgemeine personenbezogene Fragen
2. Erfahrung und Integration von DevOps-Praktiken
3. Erfahrungen im Bereich Kubernetes
4. Kenntnisse im Bereich der Cloud-basierten Entwicklungsumgebungen

Die einzelnen Fragen selbst wurden bei den qualitativen Fragen auf „optional“ gesetzt, um die Abbruchquote (eng. drop-off rate) so gering wie möglich zu halten. Generell wurden die Themenbereiche so angeordnet, dass die meisten quantitativen Fragen zu Beginn gestellt wurden, um im Falle eines vorzeitigen Abbruchs der Befragung die statistischen Daten zu sichern, die in der Regel besonders schnell zu beantworten sind. Von dieser optimierten Anordnung erhofft man sich einen positiven Effekt auf das Antwortverhalten der Teilnehmer, da man es bei Online-Befragungen in der Regel nur mit Rücklaufquoten zwischen 5% und 30% zu tun hat (vgl. Le Masson 2023).

Beginnend mit dem ersten Abschnitt werden allgemeine Teilnehmerinformationen wie Berufserfahrung, Position, Beschäftigungsverhältnis, etc. erfragt. Die ersten vier Fragen lauten wie folgt:

1. **Wie sind Sie derzeit beschäftigt?**
   Antworten: Festanstellung; Absolvent; Werkstudent; Andere
2. **Was ist Ihre derzeitige Position im Team?**
   Antworten: Full-Stack Softwareentwickler/in; Software-Architekt/in; DevOps/Cloud Entwickler/in; Softwaretester/in; Keine Antwort; Andere
3. **Wie viele Jahre Erfahrung haben Sie in der Softwareentwicklung?**
   Antworten: Weniger als 1 Jahr; 1-3 Jahre; 3-5 Jahre; 5-10 Jahre; Mehr als 10 Jahre
4. **Wie oft stoßen Sie bei Ihrer Entwicklungsarbeit an die Leistungsgrenzen Ihres Laptops (CPU / GPU)?**
   Antworten: Nie; Selten; Monatlich; Wöchentlich; Täglich

Diese Informationen werden verwendet, um festzustellen, ob es einen Zusammenhang zwischen der Berufserfahrung und den verschiedenen Kenntnissen in den nachfolgenden Bereichen gibt. Außerdem können diese statischen Werte bei der Interpretation der Ergebnisse im Falle von Ausreißern o.ä. helfen. Die nachfolgenden Fragen stammen hauptsächlich aus dem DevOps-Bereich. Damit soll geprüft werden, inwieweit die alltägliche Entwicklungsarbeit von „operativen Aufgaben (Ops-Tasks)“ geprägt ist und was häufige Hürden dabei sind. Dieser Bereich besteht sowohl aus qualitativen als auch quantitativen Fragen. Diese sind wie folgt:

5. **Welche Tools kennen und/oder verwenden Sie?**
   Antworten: Docker; Docker Compose; FluxCD; ArgoCD; Helm; GitLab CI/CD; DevBox; DevSpace; DevPod; JetBrains Gateway; VSCode Server

6. **Wie oft müssen Sie bei Ihrer Arbeit „operative Aufgaben“ übernehmen?**
   Antworten: Nie; Selten; Gelegentlich; Häufig; Sehr häufig

7. **Finden Sie „operative Aufgaben“ (zeit-)aufwändig und würden diese gerne automatisieren, um mehr Zeit für die Softwareentwicklung zu haben?**
   Antworten: Ja; Nein

8. **Welche der folgenden, wenn überhaupt, waren Hindernisse bei der Implementierung von DevOps-Praktiken in Ihrem Team / Arbeitsalltag?**
   Antworten (Mehrfachauswahl): Komplexität und der damit verbundene Zeitaufwand; Fehlende Fähigkeiten bei mir oder Mitarbeitenden; Ältere Infrastrukturen (Legacy Code/Infra); Fehlende Ressourcen (Mitarbeiter, Zeit, Infrastruktur); Kein eindeutiger Plan; Falsche/unzureichende Tools/Infrastruktur; Fehlende Kommunikation mit anderen Teams oder Mitarbeitenden; Keiner der oben genannten Punkte; Andere

9. **Welche zusätzlichen Funktionen oder Automatisierungen im Bereich DevOps würden Ihnen helfen, effizienter zu arbeiten?**

10. **Welche Werkzeuge zur Umsetzung von Sicherheitszielen setzen Sie in Ihrem Team für CI/CD-Pipelines ein?**

Mit der achten Frage soll überprüft werden, ob die in Kapitel 2.2.2 beschriebenen Ergebnisse des Atlassian DevOps Reports auch für dieses Unternehmen zutreffen. Die Fragen neun und zehn zielen indirekt darauf ab, aktuelle Probleme in bestehenden teaminternen Prozessen zu identifizieren und Verbesserungsvorschläge zu benennen, die dann ggf. später als Anforderung umgesetzt werden können.

Im dritten Teil soll das allgemeine Interesse an Kubernetes und Cloud-nativer Softwareentwicklung abgefragt werden. Außerdem soll das bereits vorhandene Wissen über Multi-Tenancy in Kubernetes und die Vorteile von Kubernetes im Allgemeinen abgefragt werden. Die Fragen dazu sind wie folgt:

11. **Haben Sie Erfahrung in der Administration und/oder Nutzung von Kubernetes-Clustern?**
    Antworten: Nein, und ich bin nicht interessiert; Nein, aber ich bin interessiert; Ja, etwas Erfahrung; Ja, umfangreiche Erfahrung

12. **Angenommen, Sie und Ihr Team erhalten einen eigenen Kubernetes-Entwicklungs-Cluster, wofür würden Sie ihn einsetzen und was wäre Ihrer Meinung nach damit generell möglich?**

Der vierte und letzte Teil befasst sich mit der Integration neuer Teammitglieder und Cloud-basierter Entwicklungsumgebungen. Zu Beginn werden mit Frage 13 wieder statistische Werte erhoben. Frage 14 zielt auf die persönlichen Präferenzen bei der Softwareentwicklung in der Cloud ab. Diese kann individuell gewichtet werden.

Mit den Fragen 15, 16 und 17 soll herausgefunden werden, ob ein schnelles Onboarding neuer Teammitglieder notwendig ist und ob die in Frage 16 vorgestellte Lösung helfen würde, die Effizienz dieses Prozesses zu optimieren. Die Fragen 18 und 19 zielen darauf ab, bereits vorhandenes Wissen erneut abzufragen und mögliche Probleme aus der Vergangenheit aufzuarbeiten, um diese in Zukunft zu verbessern. Die Fragen des vierten Teils lauten wie folgt:

13. **Wie vertraut sind Sie mit dem Konzept von Cloud-basierten Entwicklungsumgebungen?**
    Antworten: Keine Kenntnisse; Gehört, aber nicht benutzt; Grundkenntnisse; Sehr vertraut

14. **Welche der folgenden Punkte sind für Sie bei einer Cloud-basierten Entwicklungsumgebung am wichtigsten?**
    Antworten: Backup; Viel Rechenleistung; Zwischen Lokal und Cloud zu wechseln; Schnelle Integration ins Team; Testumgebungen schnell einrichten; Einfache Konfiguration

15. **Wie wichtig ist es für Sie, dass neue Teammitglieder schnell und effizient in die Entwicklungsumgebung Ihres Teams integriert werden können?**
    Antworten: Unwichtig; Neutral; Wichtig; Sehr wichtig

16. **Würde Ihr Team von einer zentral verwalteten Projektkonfiguration in Git profitieren? Denken Sie dabei vor allem an neue Teammitglieder und den Aufwand jedes System manuell zu aktualisieren.**
    Antworten: Nein; Kann ich nicht sagen; Ja, das würde Zeit und Aufwand sparen

17. **Falls Sie die letzte Frage mit „Nein“ beantwortet haben. Wie begründen Sie diese Aussage?**

18. **Auf welche Herausforderungen sind Sie oder Ihre Teammitglieder bei der Einrichtung Ihres Systems in neuen Projekten gestoßen? Nennen Sie bitte auch zeitaufwändige Aufgaben.**

19. **Welche Herausforderungen oder Hindernisse sehen Sie bei der Einführung und Nutzung von Cloud-basierten Entwicklungsumgebungen im Allgemeinen oder in Ihrem Team?**

Die Auswertung und Interpretation der Ergebnisse lassen sich in Kapitel 3.4 finden. Diese dienen außerdem als Grundlage für die nachfolgenden Kapitel.

3.3 Durchführung im betreuenden Unternehmen

Der Durchführungszeitraum der Befragung betrug 14 Tage. Mit Unterstützung der internen IT wurde der Link zur Online-Umfrage an alle Mitarbeiter des Unternehmens verschickt. Zusätzlich wurde im internen Microsoft Teams auf die Umfrage aufmerksam gemacht. Insgesamt wurden 117 Festangestellte und 25 Studierende angeschrieben. Die Teilnehmerstatistik ist in der Abbildung 10 einsehbar. Nach Abschluss der Umfrage konnten insgesamt 68 Öffnungen registriert werden, von denen 46 Personen die Umfrage gestartet haben. Dies entspricht 68%. Nur 31 Personen beendeten die Umfrage vollständig, was 46% entspricht. Bei einer maximalen Teilnehmerzahl von 142 Personen liegt die Gesamtteilnehmerquote bei ca. 21,8% bzw. 32.4% mit den unvollständigen Rückmeldungen eingerechnet.

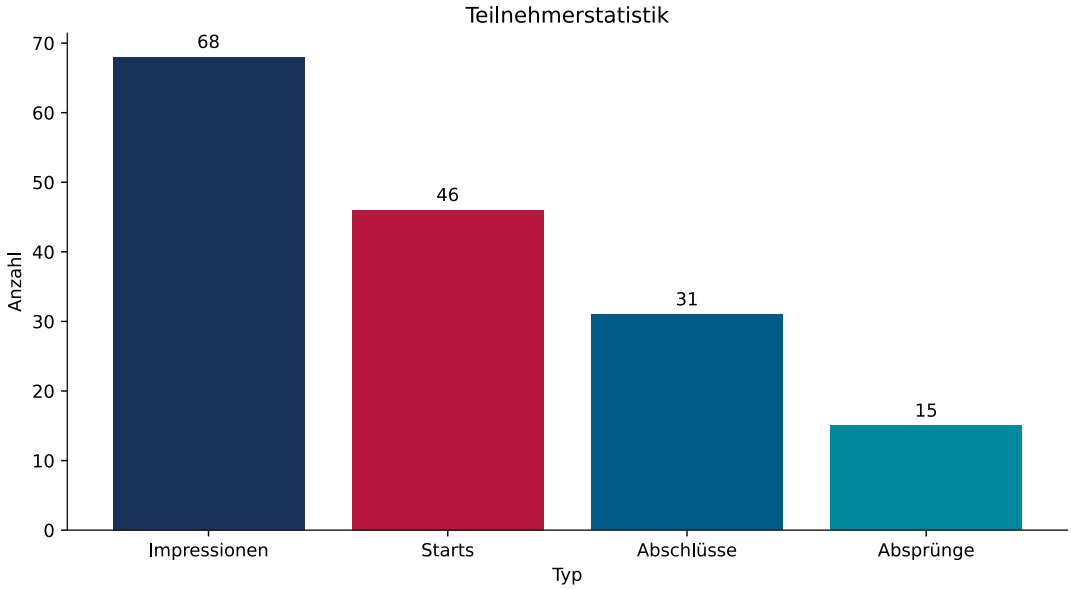

Abbildung 10: *Darstellung der Teilnehmerstatistik der Online-Befragung*

Für die Auswertungen in Kapitel 3.4 wurden nur vollständige Antworten berücksichtigt.

3.4 Auswertung und Interpretation der Befragungsergebnisse

In diesem Kapitel werden die Ergebnisse der Befragung anhand verschiedener Grafiken dargestellt. Auf jede Darstellung folgt eine Interpretation der Ergebnisse. Die Interpretation erfolgt abschnittsweise in der in Kapitel 3.2 festgelegten Reihenfolge.

3.4.1 Auswertung des ersten Abschnitts „Allgemeine personenbezogene Fragen“

Die Abbildungen 11 und 12 zeigen, dass ca. 87% der Antworten von festangestellten Mitarbeitern stammen. Davon sind 64,5% im Bereich der Full-Stack Softwareentwicklung tätig.

Da Themen des maschinellen Lernens derzeit eher in studentischen Projekten bearbeitet werden, ist der Anteil der Data Science Entwickler mit 6,4% sehr gering. Dieser Anteil könnte jedoch höher sein, da nur 16% bzw. vier von insgesamt 25 Studierenden an der Umfrage teilgenommen haben. Der überwiegende Teil der Teilnehmer liegt jedoch genau im Zielspektrum dieser Arbeit. Gerade die Webentwicklung profitiert am meisten von der Cloud-nativen Softwareentwicklung. Außerdem haben drei (9,7%) IT-Systemadministratoren an der Umfrage teilgenommen, so dass auch Erfahrungen außerhalb der Softwareentwicklung gesammelt werden konnten.

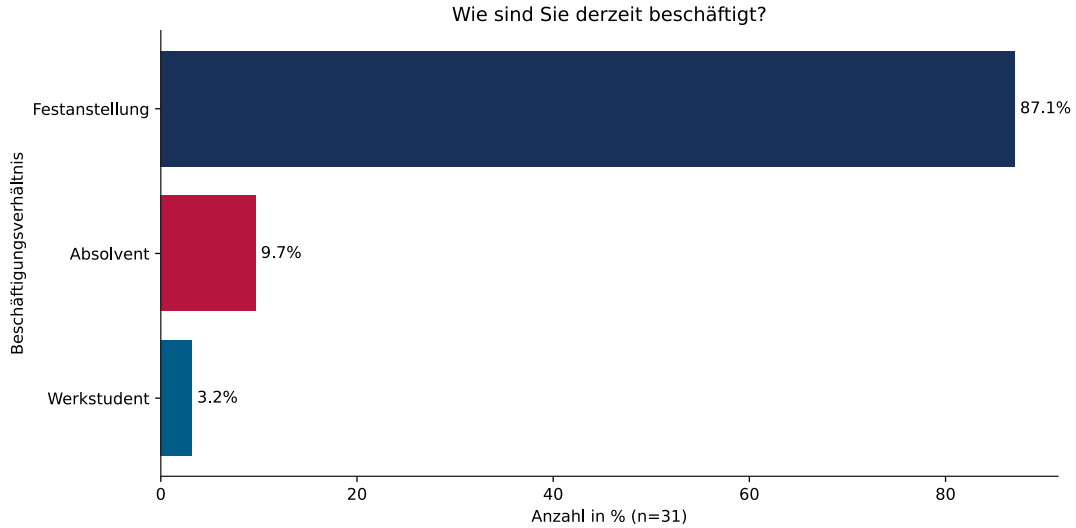

Abbildung 11: *Ergebnisse Frage 1: Wie sind Sie derzeit beschäftigt?*

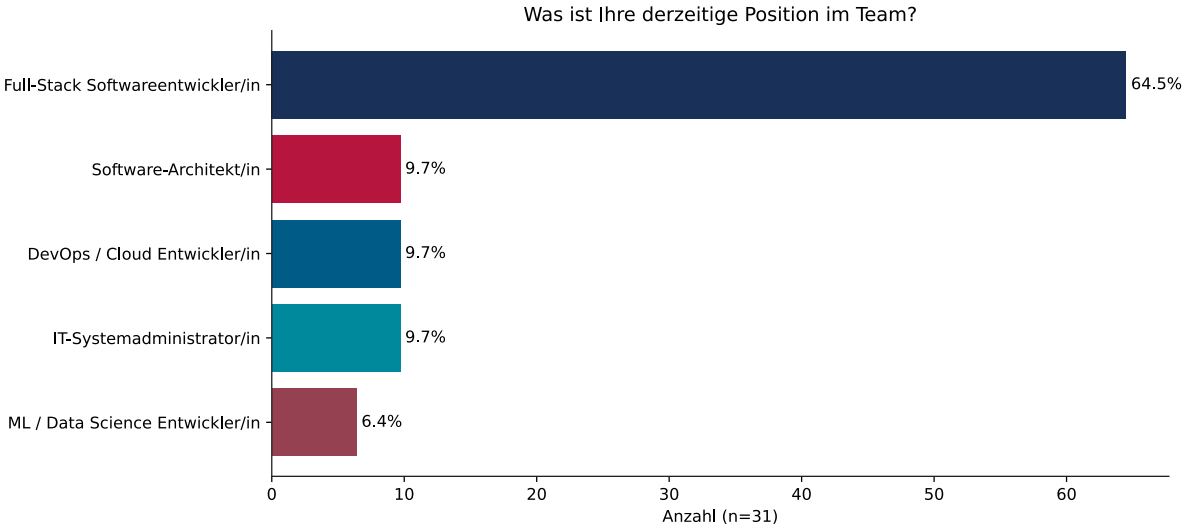

Abbildung 12: *Ergebnisse Frage 2: Was ist Ihre derzeitige Position im Team?*

Betrachtet man die Abbildung 13, so fällt auf, dass insgesamt ein breites Spektrum an Berufserfahrung in den Ergebnissen vertreten ist. Den größten Anteil mit 38,7% nehmen die Senior-Softwareentwickler ein, die auch die meiste Erfahrung mitbringen. Dies sollte sich im

weiteren Verlauf positiv auf die qualitativen Fragen auswirken. Hervorzuheben ist auch der Anteil der Junior-Softwareentwickler mit einer Berufserfahrung zwischen einem und drei Jahren von 22,6%. Damit können auch Erfahrungen und Meinungen von Personen ausgewertet werden, die frisch aus dem Studium o. ä. kommen. Hier könnte die Bereitschaft für neue Technologien und Prozesse, die mit der DevOps-Kultur einhergehen, auch größer sein.

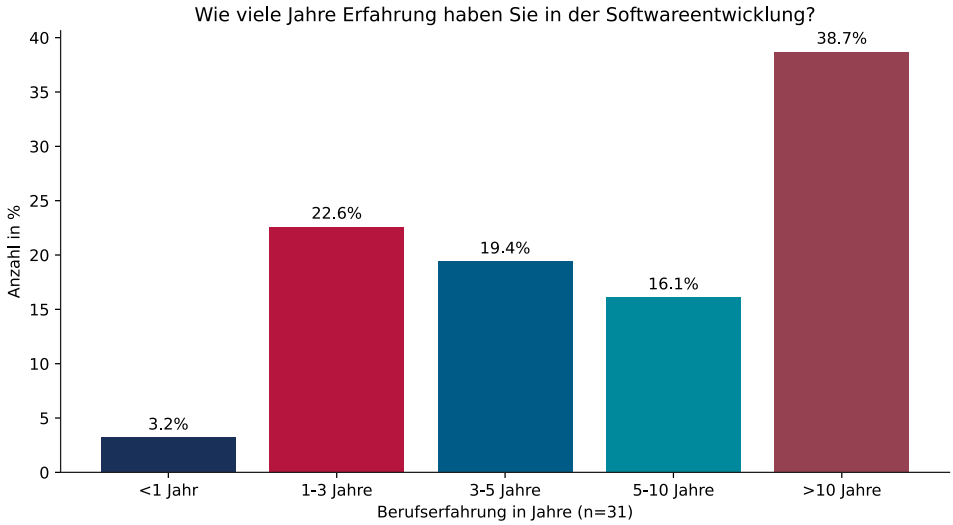

Abbildung 13: *Ergebnisse Frage 3: Wie viele Jahre Erfahrung haben Sie in der Softwareentwicklung?*

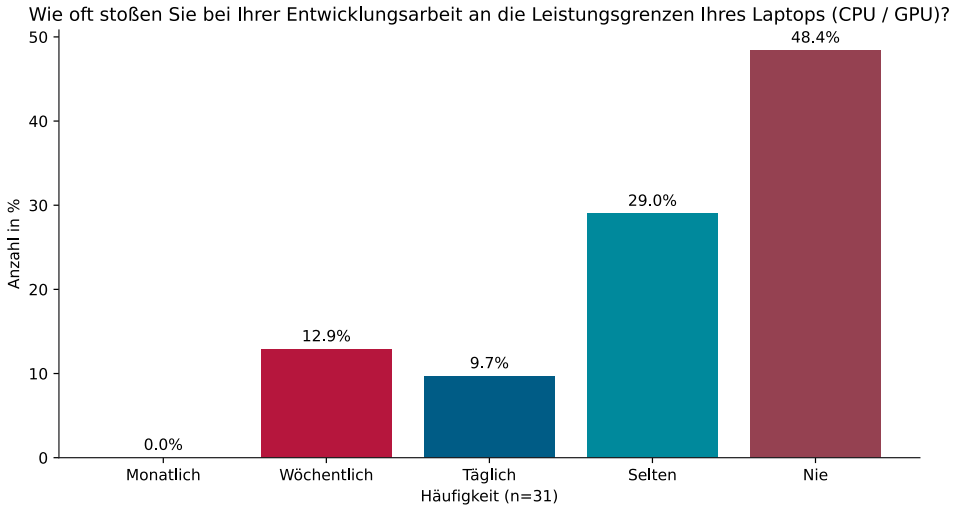

Abbildung 14: *Ergebnisse Frage 4: Wie oft stoßen Sie bei Ihrer Entwicklungsarbeit an die Leistungsgrenzen Ihres Laptops (CPU / GPU)?*

Mit Frage 6 in Abbildung 14 sollte ermittelt werden, inwieweit die Mitarbeiter mit den lokalen Ressourcen des Arbeitslaptops zufrieden sind. Etwa die Hälfte der Befragten, 48,4%, gab an, vollkommen zufrieden zu sein. Dass ca. 13% wöchentlich und sogar ca. 10% täglich an ihre Leistungsgrenzen stoßen, ist ein überraschendes Ergebnis, da die im Unternehmen eingesetzten Arbeitsgeräte weniger als ein Jahr alt sind und teilweise eine CPU mit bis zu 24 Kernen

und Arbeitsspeicher von bis zu 64 GB besitzen. Da dies jedoch nur die Erfahrungen von 31 Mitarbeitern widerspiegelt, kann keine allgemeingültige Aussage getroffen werden, es zeigt jedoch, dass eine Kubernetes-as-a-Service-Lösung hier positive Auswirkungen haben könnte, da Ressourcen aus der Cloud genutzt werden könnten.

Die Antworten aus dem ersten Abschnitt liefern bereits vielversprechende Ergebnisse. Die angestrebte Implementierung von Kubernetes-as-a-Service würde in diesem Unternehmen auf den ersten Blick eine sinnvolle Ergänzung darstellen.

3.4.2 Auswertung des zweiten Abschnitts „Erfahrung und Integration von DevOps-Praktiken“

In Abbildung 15 ist die prozentuale Verteilung der Bekanntheit verschiedener Werkzeuge aus der Softwareentwicklung dargestellt. Es fällt auf, dass sich die Erfahrung größtenteils zwischen Grundlagen und Fortgeschrittenen bewegt. Expertenwissen in DevOps-Tools wie ArgoCD, Flux oder GitLab CI/CD ist bis auf wenige Ausnahmen nicht vorhanden. Daher kann vermutet werden, dass ein unterstützendes Tool in diesen Bereichen für viele Mitarbeiter eine positive Unterstützung bei operativen Aufgaben sein kann.

Diese Vermutung lässt sich mit einem Blick auf die Abbildung 16 weiter erhärten, in der etwa die Hälfte der Befragten angibt, gelegentlich DevOps-Aufgaben übernehmen zu müssen. Bei gelegentlichen Aufgaben besteht zudem die Gefahr, dass Wissen bis zum nächsten Mal verloren geht, was ein effizientes Arbeiten erschwert. Auch hier könnte das oben genannte Tool Abhilfe schaffen. Dies bestätigen auch rund zwei Drittel der Befragten in Abbildung 17.

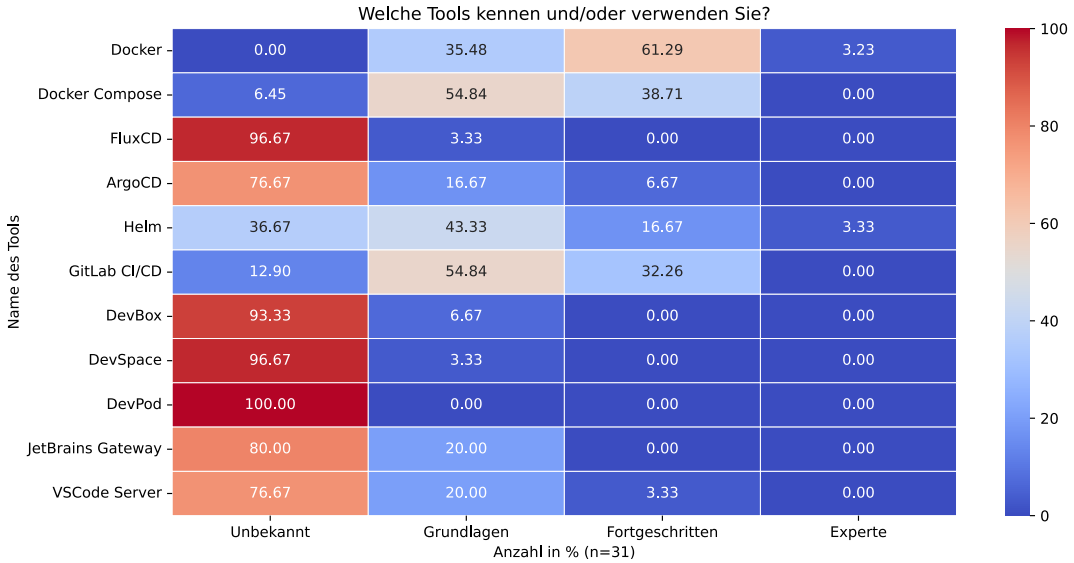

Abbildung 15: *Ergebnisse Frage 5: Welche Tools kennen und/oder verwenden Sie?*

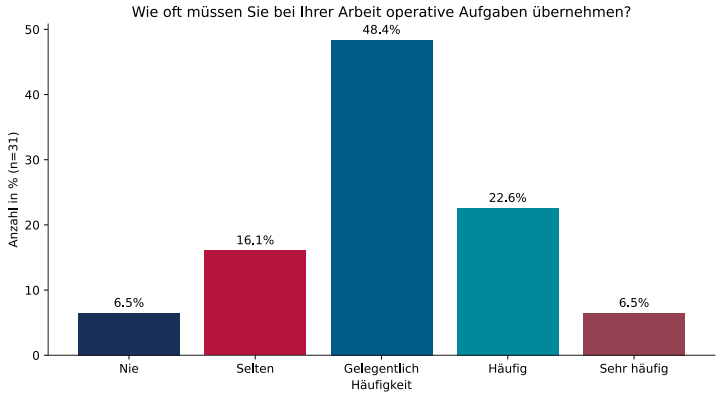

Abbildung 16: *Ergebnisse Frage 6: Wie oft müssen Sie bei Ihrer Arbeit operative Aufgaben übernehmen?*

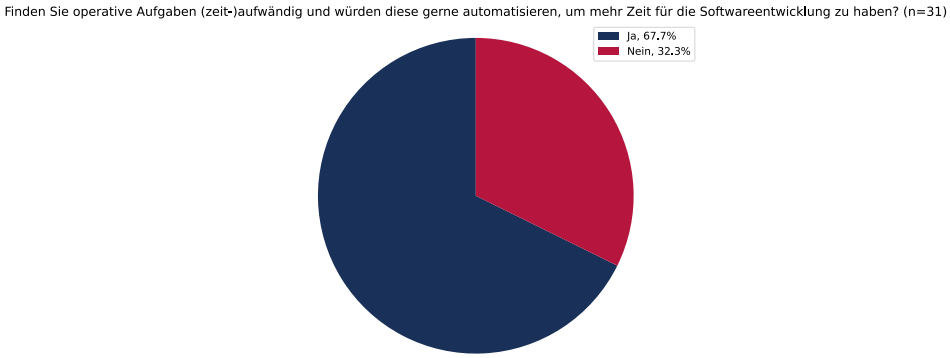

Abbildung 17: *Ergebnisse Frage 7: Finden Sie operative Aufgaben (zeit-)aufwändig und würden diese gerne automatisieren, um mehr Zeit für die Softwareentwicklung zu haben?*

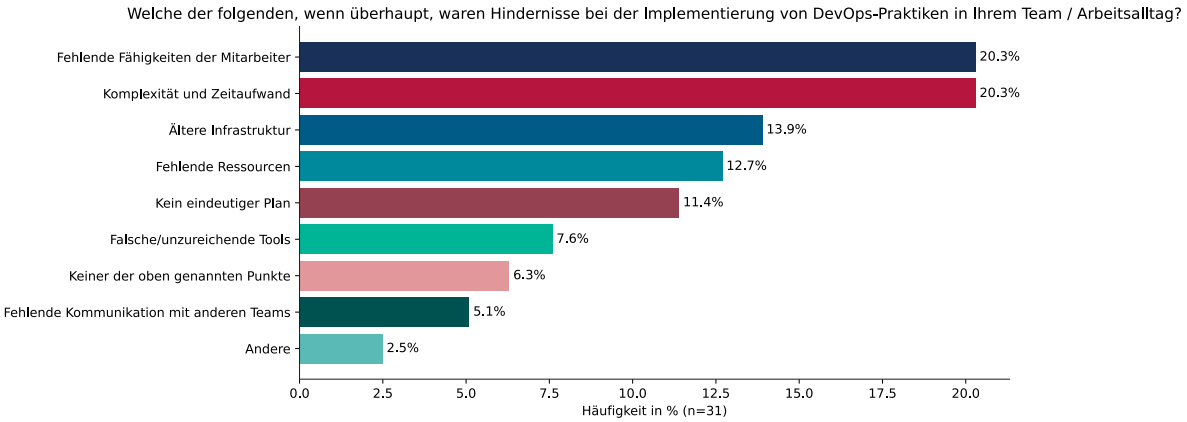

Abbildung 18: *Ergebnisse Frage 8: Welche der folgenden, wenn überhaupt, waren Hindernisse bei der Implementierung von DevOps-Praktiken in Ihrem Team / Arbeitsalltag?*

Die Frage 8 in Abbildung 18 zielte darauf ab, die Hindernisse zu identifizieren, die in der Praxis auftreten, wenn es um DevOps-Themen geht. Diese Frage orientiert sich an der in Kapitel 2.2.2 erwähnten DevOps-Umfrage von Atlassian im Jahr 2020. Es lassen sich eini-

ge Ähnlichkeiten feststellen, vor allem in den Bereichen fehlende Mitarbeiterfähigkeiten mit 20,3% und Legacy Infrastruktur mit 13,9%. Diese befinden sich in beiden Grafiken unter den ersten drei Positionen. Ebenfalls gaben 20,3% an, dass die Komplexität und der damit verbundene Zeitaufwand ein Hindernis darstellen. Diese Ergebnisse zeigen, wie wichtig eine Interaktion zwischen dem Dev-Bereich und dem Ops-Bereich ist. Nur so können Probleme erkannt und gelöst werden. Aktuell deutet dies darauf hin, dass automatisierte Deployments etc. aus den genannten Gründen oft nicht wie gewünscht umgesetzt werden können. Hier könnte Kubernetes-as-a-Service helfen.

Bei den Fragen 9 und 10 handelt es sich um qualitative Fragen. Die Antworten zu jeder Frage wurden aggregiert, so dass jeder genannte Punkt nur einmal vorkommt. Frage 9 zielte darauf ab, herauszufinden, welche Erweiterung einem persönlich helfen würde, effizienter im Bereich DevOps zu arbeiten. Die Ergebnisse stellen sich wie folgt dar

- Übersicht über funktionierende Pipelines in Projekten
- Vorlagen für wiederkehrende Aufgaben
- Automatische Erstellung von Deployments, vorzugsweise über die IDE
- Automatische Anpassung der NGINX vHost Konfigurationsdateien bei neuen Deployments
- Remote Debugging / Entwicklung
- Einfaches und sicheres Backup / Restore / Disaster Recovery von Umgebungen
- Klare Dokumentation mit Bildern zur Veranschaulichung
- Code-basierte Ansätze wie Pulumni
- GitLab Premium
- Darstellung von Containern im Netzwerk
- Erstellung von Laufzeitumgebungen für jede Funktionsentwicklung

Hier zeigt sich, dass der Wunsch nach automatisierten Tools zur Erstellung von Deployments und Testumgebungen bereits im Unternehmen vorhanden ist. Remote-Debugging und -Entwicklung sowie die automatische Anpassung von Webrouten sind ebenfalls erwünscht. Diese Themen bilden eine perfekte Schnittmenge mit der geplanten Kubernetes-Plattform. Diese Punkte können somit im weiteren Verlauf in die Anforderungen übernommen werden.

Mit der Frage 10 wurde versucht, ein Überblick über eingesetze SecOps-Tools in den CI/CD-Pipelines der jeweiligen Teams zu erstellen. Dabei sind folgende Resultate erzielt worden:

- SBOM
- Trivy
- Sonarqube
- Datadog

- Black Duck
- Snyk
- Docker Content Trust
- OWASP Dependency-Check
- PIP-Audit
- Depdendency Track

Die Antworten verdeutlichen, dass in den Teams bereits eine aktive Auseinandersetzung mit Sicherheitsaspekten stattfindet. Dabei ist zu berücksichtigen, dass eine Anforderung später darin bestehen muss, dass die jeweiligen Teams diese Technologien weiterhin in ihrem CI/CD-Stack weiterverwenden können.

3.4.3 Auswertung des dritten Abschnitts „Erfahrungen im Bereich Kubernetes“

In diesem Abschnitt wurden die Erfahrungen mit Kubernetes abgefragt. So gaben bei Frage 11 in Abbildung 19 51,6% an, bereits einige Erfahrungen mit Kubernetes gesammelt zu haben, gefolgt von 19,4%, die über umfangreiche Erfahrungen mit Kubernetes verfügen. Damit sind fast drei Viertel der Antworten positiv gestimmt. Lediglich 9,7% gaben an, kein Interesse an Kubernetes zu haben. Auf diese Personen muss später gezielt Rücksicht genommen werden, damit sie weiterhin die Möglichkeit haben, z. B. lokal eigenständig weiterzuentwickeln.

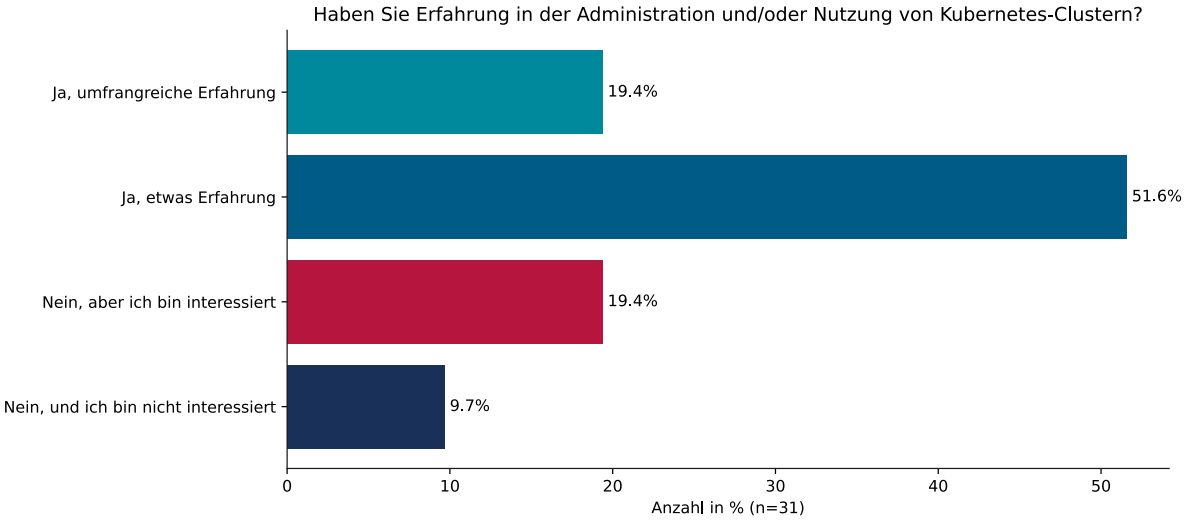

Abbildung 19: *Ergebnisse Frage 11: Haben Sie Erfahrung in der Administration und/oder Nutzung von Kubernetes-Clustern?*

Mit der qualitativen Frage 12 sollten mögliche Einsatzzwecke von einem Kubernetes Team-Cluster ermittelt werden. Dabei konnte indirekt zum einen ermittelt werden, was für Deployments in den einzelnen Teams ggf. erstellt werden, als auch die persönliche Einschätzung und

Vorstellung was mit solch einem Cluster möglich ist. Die Antworten sind wie folgt:

- Test- und Produktivumgebungen
- Ausfallsichere Werkzeuge / Hochverfügbarkeit
- Backups
- Integration von Kundenservern
- Entwicklungs-Cluster für Testdeployments
- Ablösung „lokaler“ Entwicklungsumgebungen
- Web-Anwendungen
- Schnelle Einarbeitung neuer (temporärer) Mitarbeiter/innen
- Visualisierung von Datenflüssen, Metriken für alle Pods etc.
- Automatische Deployments des Main Branches, um den aktuellen Code-Stand integriert testen zu können
- Die zentrale Bereitstellung von Kubernetes-Clustern wäre sehr hilfreich, um den Administrationsaufwand zu minimieren. Durch den Einsatz als Supporting Asset im Kontext SSEP hätten wir auch deutlich weniger Aufwand bei Security-Themen, wenn die Sicherheit zentral gewährleistet wäre
- Eigene Remote Entwicklungsumgebungen
- Wir nutzen bereits selbst gehostete und selbst administrierte Kubernetes-Cluster als Testumgebungen sowie hybrid für die Entwicklung (lokale Ausführung von Services mit Anbindung an Cluster-DBs)

Diese Ergebnisse zeigen durchweg die richtige Einstellung der Befragten zu Kubernetes. Viele der genannten Punkte werden bereits von Vanilla Kubernetes abgedeckt, wie z. B. die Integration von Kundenservern oder Hochverfügbarkeit. Auch Test- und Produktionsumgebungen lassen sich einfach aufsetzen. Die geplante Plattform sollte im Endzustand einen Großteil der Punkte abdecken können, auch die schnelle Einarbeitung von Mitarbeitenden sollte optimiert werden können. Mit dieser Frage können weitere wichtige Anforderungen formuliert werden.

3.4.4 Auswertung des vierten Abschnitts „Kenntnisse im Bereich der Cloud-basierten Entwicklungsumgebungen“

In diesem letzten Abschnitt sind die Ergebnisse rund um das Thema der Cloud-basierten Entwicklungsumgebungen zu finden.

Die Ergebnisse der Frage 13 in Abbildung 20 zeigen ein positiveres Bild als erwartet. Fast die Hälfte der Befragten hat Grundkenntnisse in dieser Technologie. Diese Ergebnisse stehen im Widerspruch zu denen in Abbildung 15. Bei den dort aufgeführten Technologien wie VSCode

Server, DevPod, DevSpace etc. gaben bis auf eine Ausnahme fast alle an, diese nicht zu kennen. Eine mögliche Erklärung könnte eine zu komplexe/schwammige Fragestellung sein, so dass die meisten nicht erkennen konnten, worum es sich bei Cloud-basierten Entwicklungsumgebungen genau handelt. Dies könnte vor allem durch den dritten Bereich „Erfahrungen im Bereich Kubernetes“ ausgelöst worden sein.

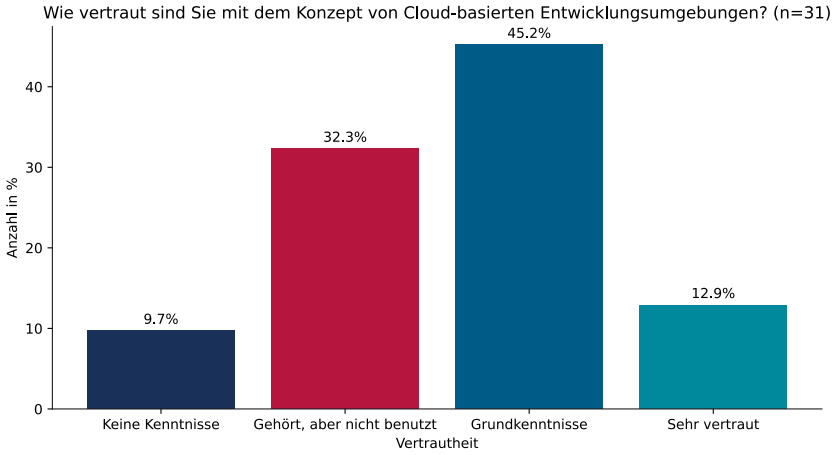

Abbildung 20: *Ergebnisse Frage 13: Wie vertraut sind Sie mit dem Konzept von Cloud-basierten Entwicklungsumgebungen?*

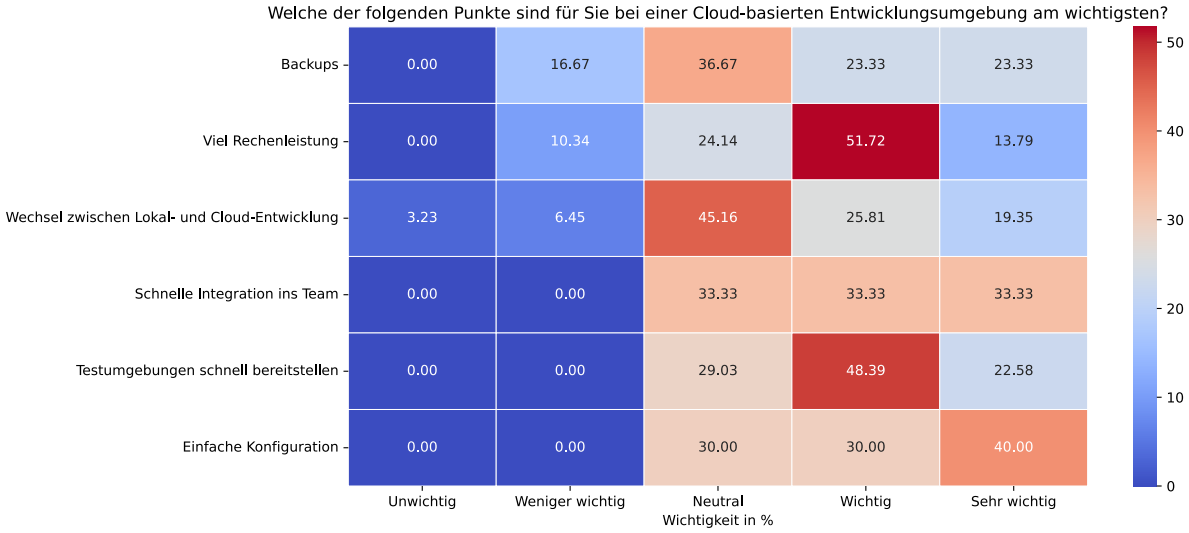

Abbildung 21: *Ergebnisse Frage 14: Welche der folgenden Punkte sind für Sie bei einer Cloud-basierten Entwicklungsumgebung am wichtigsten?*

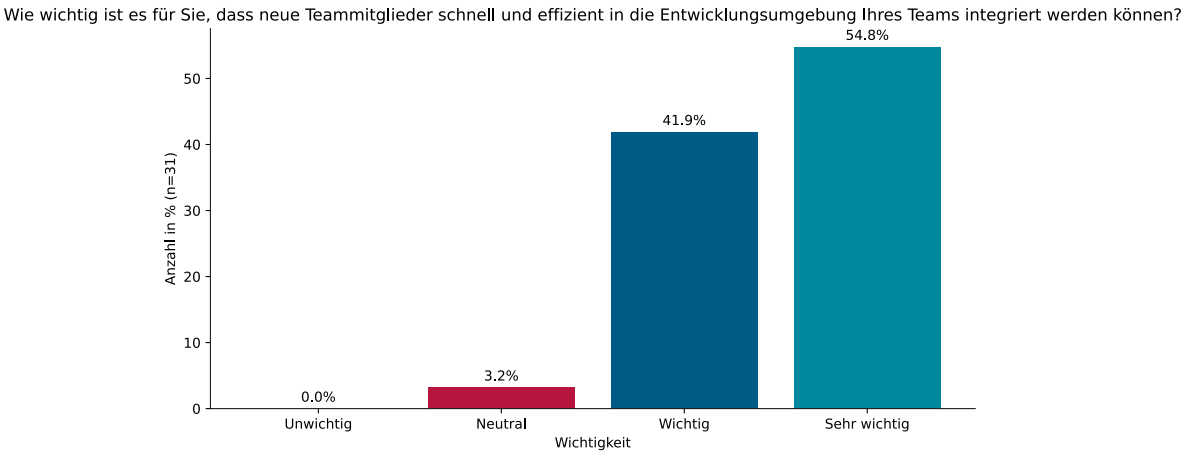

Abbildung 22: *Ergebnisse Frage 15: Wie wichtig ist es für Sie, dass neue Teammitglieder schnell und effizient in die Entwicklungsumgebung Ihres Teams integriert werden können?*

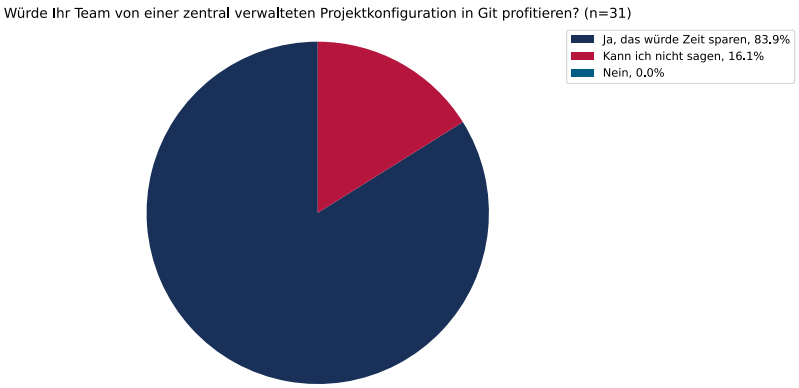

Abbildung 23: *Ergebnisse Frage 16: Würde Ihr Team von einer zentral verwalteten Projektkonfiguration in Git profitieren?*

Bei den Fragen 14, 15 und 16 in den Abbildungen 21, 22 und 23 konnte erneut bestätigt werden, dass eine schnelle Einarbeitung neuer Teammitglieder mit ca. 97% als wichtig bis sehr wichtig angesehen wird. Dies muss also bei den Anforderungen berücksichtigt werden. Mit der Frage 16 konnte dies nochmals bestätigt werden.

Des Weiteren gaben bei Frage 14 ca. 19,4% an, dass ein Wechsel zwischen lokaler und Cloud-Umgebung möglich sein muss, um weiterhin lokal entwickeln zu können. Dies wurde auch bereits bei Frage 11 in Abbildung 19 festgestellt. Ebenso muss eine einfache Konfiguration dieser Umgebungen möglich sein, dies gaben ca. 40% als sehr wichtig an.

Die qualitativen Fragen 18 und 19 zielten vor allem auf die Herausforderungen im Alltag ab. Unter anderem sollte angegeben werden, welche Probleme bei der Einrichtung bestehender oder neuer Projekte/Systeme auftreten. Darüber hinaus sollten Herausforderungen aufgezählt werden, die bei einer produktiven Einführung der Kubernetes-as-a-Service Plattform auftreten könnten. Die Ergebnisse der Frage 18 sind wie folgt:

- Cluster / Lens, Abhängigkeiten, Git, IDE, Datenbanken (Clients)
- Basisinstallation von Werkzeugen und Quellcodeabhängigkeiten auf verschiedenen Betriebssystemen.
- Kundenspezifische Änderungen der Cloud-Infrastruktur während der Projektlaufzeit.
- Zugriff auf Kubernetes Cluster ermöglichen
- Verfügbarkeit (inkl. Berechtigungen) von externen APIs, die für die lokale Ausführung benötigt werden.
- Keine Dokumentation der Infrastruktur
- Vorgänger machten vieles manuell, z. B. komplexer Rollout-Prozess, der nicht dokumentiert ist.
- Nur minimale Wartung, viele Module veraltet und mit vielen Sicherheitsrisiken
- Rollout auf verschiedenen Betriebssystemen, keine Linux-VM auf Windows erlaubt
- Unterschiedliche Betriebssysteme im Team (Windows, Linux), Konfigurationen wurden nicht eingecheckt, einige Konfigurationen konnten nicht automatisch verteilt werden, sondern mussten vom Entwickler manuell eingerichtet werden.
- Login mit Kurzzeit-Tokens kann zu Abbrüchen bei komplexen Prozessen führen
- Häufiger Wechsel von Umgebungen wird durch verschiedene VSCode Workspaces, Portforwardings etc. erschwert.
- Berücksichtigung spezifischer, von Projekt zu Projekt unterschiedlicher Anforderungen an Deployment-Umgebungen (z. B. On-Prem Cluster, Single-Node Deployments, Cloud Deployments)
- Trennung von Zugängen / Umgebungen bei gleichzeitiger gemeinsamer Nutzung von Ressourcen (z. B. aufgrund von Passwörtern / Umgebungsvariablen im Code bei gemeinsamer Codebasis zwischen Projekten)
- Herausforderungen bei der Gewährleistung von Sicherheitsanforderungen (BSI, SSEP): Dies erschwert die gemeinsame Nutzung zentraler Ressourcen, z. B. Harbor Image Registry, durch verschiedene Projekte, wenn die Sicherheitsanforderungen eine maximale Trennung anstreben.
- Verwaltung der Artefakte in JFrog, CI/CD für die Bibliotheken.
- Generell immer wieder IT-/Infrastrukturprobleme von IT-Seite, aber auch von Kundensystemen.

Viele der oben genannten Punkte sollten durch die Möglichkeit einer zentralen Verwaltung von Entwicklungsumgebungen mit z. B. Git gelöst werden können. Auch Probleme mit unterschiedlichen Betriebssystemen können durch einheitliche Umgebungen vermieden werden. Generell können fast alle Punkte mittelfristig durch die Einführung von Kubernetes-as-a-Service umgesetzt werden. Dies gilt auch für viele der Punkte in der folgenden Frage 19.

Die Ergebnisse der Frage 19 lauten wie folgt:

- Datenschutz
- GUI-Anwendungen in z. B. C++ oder Java
- Es muss sichergestellt werden, dass keine Unternehmensrichtlinien verletzt werden, z. B. kein Code auf fremden Servern.
- Remote-Arbeit (z. B. im Zug) nicht möglich.
- Unklarheit, ob der Kunde dies erlaubt
- Offline-Verfügbarkeit. Wenn nur online nutzbar problematisch, da Heimnetzwerk oft unterbrochen wird.
- Anpassung wie Schlüsselkonfiguration, Plugin für jeden Entwickler
- Editor ist Geschmackssache. Manche mögen VSCode, andere IntelliJ
- Zugänglichkeit / Verfügbarkeit von geschützten / sensiblen Kundendaten.
- Performance / Latenzen / Online-Zwang
- Ergänzung durch weiterhin volle Unterstützung lokaler Entwicklungsumgebungen könnte aufwendig sein

Bei den oben genannten Punkten ist zu erkennen, dass viele Bedenken hinsichtlich des Datenschutzes, der Flexibilität der IDE und der Offline-Verfügbarkeit bestehen. Diese Punkte müssen durch spezifische Anforderungen an die Plattform bestmöglich umgesetzt werden, damit ein späterer Produktivbetrieb von den Nutzern akzeptiert wird. Auf den ersten Blick sollte jedoch, wie bei Frage 18, ein Großteil der Herausforderungen gelöst werden können.

3.5 Fazit zur Auswertung der Online-Umfrage

Durch die Mitarbeiterbefragung konnten wertvolle Einblicke in den Arbeitsalltag der im Unternehmen beschäftigten Mitarbeiter gewonnen werden. Durch die quantitativen und qualitativen Fragen und die für eine Online-Befragung hohe Teilnahmequote von ca. 21,4% konnte bereits ein guter Einblick in das Unternehmen gewonnen werden. Die Tatsache, dass bereits Grundlagen in Kubernetes und auch Cloud-basierten Entwicklungsumgebungen vorhanden sind, wird sich in Zukunft positiv auf die DevOps-Kultur im Unternehmen auswirken. Sollte die in dieser Arbeit entstandene Plattform in diesem Unternehmen in den Produktivbetrieb gehen, sollte dies einen hohen Mehrwert für alle Beteiligten darstellen. Die für diese Arbeit wichtigen Erkenntnisse und Herausforderungen sollen durch gezielte Anforderungen in Kapitel 4 bestmöglich in die Konzeption und Umsetzung der Infrastruktur einfließen.

4 Analyse der Anforderungen an das Gesamtsystem

Dieses Kapitel beinhaltet in Kapitel 4.1 eine Ist-Analyse der aktuell bestehenden Kubernetes-Infrastruktur sowie der zugehörigen CI/CD-Pipeline. Des Weiteren erfolgt in Kapitel 4.2 die Definition der verschiedenen Stakeholder und Anforderungen an die Kubernetes-as-a-Service Plattform.

4.1 Ist-Analyse der derzeitigen Infrastruktur

Im Rahmen der Ist-Analyse erfolgt eine Evaluierung der aktuell verwendeten Infrastruktur sowie weiterer Systemkomponenten wie beispielsweise der GitLab CI/CD-Pipelines, der Wartbarkeit des Kubernetes-Clusters, der verwendeten Tools etc. Ziel ist es, potenzielle weitere Anforderungen zu identifizieren und Probleme des aktuellen Systems zu identifizieren, die nicht in die neue Infrastruktur übernommen werden sollten.

4.1.1 Analyse des Kubernetes-Clusters „Dieter“

Die folgende Kubernetes-Infrastruktur wird für aktive Entwicklungs- und Testumgebungen im Rahmen eines medizinischen Forschungsprojektes in Zusammenarbeit mit dem Klinikum Esslingen eingesetzt. Da es sich bei den einzelnen Kernpunkten des Projektes hauptsächlich um Themen aus dem Bereich des maschinellen Lernens (ML) sowie um Webanwendungen zur automatischen Erkennung von Schlafanomalien handelt, wurde von der Projektleitung zu Beginn des Projektes entschieden, Kubernetes als Basis im Projekt einzusetzen. Nach internen Informationen ist dies auch das erste Projekt, das Kubernetes im Unternehmen einsetzt. Da im medizinischen Bereich hohe Anforderungen an den Datenschutz bestehen, wurde die gesamte Hardware im Unternehmen bzw. im Schlaflabor des Klinikums Esslingen untergebracht. Ein Zugriff auf die Daten ist ausschließlich unter Verwendung eines Virtual Private Networks (VPN) möglich.

Die Abbildung 24 zeigt den aktuellen Aufbau des Kubernetes-Clusters mit der internen Bezeichnung „Dieter“. Die wesentlichen Komponenten, darunter die Control Plane und die Worker Nodes, wurden überwiegend mit virtuellen Maschinen (VM) auf Basis der bereits intern eingesetzten Virtualisierungsplattform Proxmox implementiert. In der Abbildung sind alle virtuellen Maschinen mit dem entsprechenden Proxmox-Logo gekennzeichnet und tragen zusätzlich die Abkürzung „svm-*“ im Hostnamen der jeweiligen Maschine. Ergänzt werden die VMs durch sogenannte Bare-Metal-Server, die aus physischer Hardware bestehen. Diese werden unter anderem für Zwecke des maschinellen Lernens und der Datenspeicherung für medizinische Daten eingesetzt. Zur Identifikation eignen sich die Hostnamen, die nach folgendem Schema aufgebaut sind: „*hw-*“. Für den Worker Node mit dem Hostnamen „uhw-0001“

gilt zusätzlich die Besonderheit, dass dieser physisch im Schlaflabor Esslingen untergebracht ist. Dieser Schritt musste aus Datenschutzgründen von Seiten des Klinikums Esslingen unternommen werden. Durch Zuweisung einer internen IP-Adresse über eine VPN-Verbindung konnte der Rechner dem Cluster „Dieter“ hinzugefügt werden.

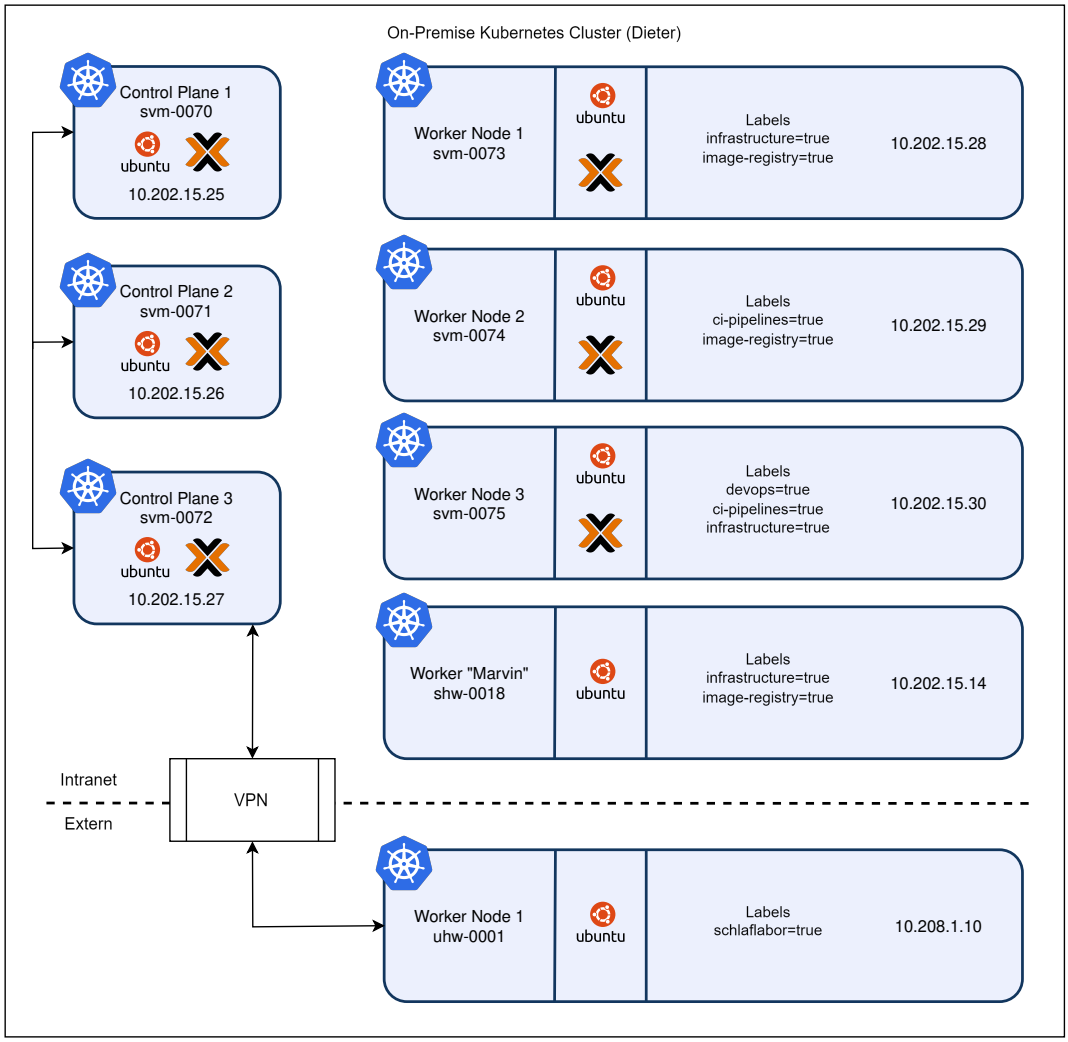

Abbildung 24: *Übersicht über den aktuell eingesetzten Kubernetes-Cluster mit dem Namen „Dieter“*

Gemäß der internen Dokumentation erfolgt die Zuweisung bestimmter Applikationen, wie beispielsweise ein GitLab Runner für CI/CD-Pipelines unter Zuhilfenahme von Kubernetes Node Labels (ci-pipelines=true), an die entsprechenden Worker. Auf diese Weise wird ein manueller Lastenausgleich vorgenommen, um eine Überlastung einzelner Worker zu vermeiden. Des Weiteren werden bestimmte Worker bestimmten Aufgaben zugewiesen. Die Zuweisung sämtlicher Labels erfolgte manuell über die kubectl. Die Labels werden dann in den jeweiligen Helm Charts manuell definiert.

In diesem Cluster werden keinerlei GitOps-Ansätze eingesetzt. Was bedeutet, dass sämtliche

Deployments, Anwendungen, Konfigurationen und ähnliches von den Teammitgliedern manuell im Cluster vorgenommen werden. Da zudem keine erweiterten Logging-Tools neben den standardisierten Kubernetes-Audit-Logs zum Einsatz kommen, ist die Wartbarkeit und Nachvollziehbarkeit von Änderungen nahezu unmöglich. Des Weiteren erfolgt der Zugriff aller Teammitglieder auf den Cluster über die gleiche Admin-Kubeconfig, wodurch automatisch jede Person Administrator mit uneingeschränkten Rechten ist. Aus sicherheitstechnischer Perspektive stellt so etwas ein großes Problem dar, das mit der neuen Kubernetes-as-a-Service-Architektur behoben werden muss.

Wirft man einen Blick auf die Maschinen selbst, so fällt auf, dass auf allen die Linux-Distribution Ubuntu in der Version 20.04 LTS zum Einsatz kommt. Aktuell ist bereits die Version 24.04 LTS auf dem Markt. Es scheint, dass Updates, Sicherheitsupdates oder Versionsupgrades bisher auf jeder Maschine manuell durchgeführt werden. Eine solche manuelle Administration bedeutet einen hohen Pflegeaufwand und kann unter Umständen auch zu Fehlern, Ausfallzeiten etc. führen. Dieser Zustand und die daraus resultierenden Folgen werden in der Literatur auch als „Cloud Sprawl“ (vgl. Stanham 2023) bezeichnet. Dies soll mit der neuen Plattform besser gelöst werden.

Im Folgenden werden die verwendeten Infrastrukturapplikationen näher analysiert. Solche Applikationen laufen in bestimmten Namespaces innerhalb des Clusters und sorgen dafür, dass z. B. Persistent Volumes durch eine StorageClass, Ingress durch eine IngressClass erstellt werden können oder auch Networking innerhalb des Clusters ermöglicht wird. Diese sind unabhängig von normalen Applikationen wie z. B. einer API für eine Webapplikation. Es handelt sich lediglich um Applikationen, die für den Betrieb des Clusters notwendig sind. Die Tabelle 1 gibt einen Überblick über diese manuell installierten Applikationen.

Nr.	Name	Namespace	Methode	Version (Ist)	Version (Soll)
1	NGINX Ingress-Controller	networking	Helm	v1.0.0	v1.10.1
2	Rook-Ceph	rook	Helm	v1.0.0	v1.14.7
3	Weave	kube-system	Helm	v1.0.0	v2.8.1
4	Prometheus	prometheus	Helm	v1.0.0	v2.53.0
5	Harbor Image-Registry	image-registry	Helm	v1.0.0	v1.15.0

Tabelle 1: *Tabelle der im Cluster Dieter eingesetzten Infrastrukturanwendungen*

Es fällt auf, dass die installierten Versionen teilweise mehrere Monate alt sind. Da keine automatisierten Updates durchgeführt werden, können solche Upgrades im Alltag oft vergessen werden, was ein mögliches Sicherheitsrisiko darstellen kann. Daraus lässt sich ableiten, dass in Zukunft z. B. Versionsupdates einfach über GitOps angestoßen werden können. Somit können Versionen nachvollziehbar über Git angepasst werden.

4.1.2 Analyse der GitLab CI/CD-Pipeline

Innerhalb des Projektteams werden verschiedene Anwendungen automatisiert über GitLab CI/CD Pipelines in den Cluster Dieter deployed. Dort kann dann auf diese mittels einem Ingress oder durch Port Forwarding über die kubectl zugegriffen werden. Das Gesamtsystem basiert dabei auf einem selbstgebauten generischen Template System, welches vorgefertige Templates für Deployments in den Programiersprachen Golang, Python und Node.js zur Verfügung stellt. Des Weiteren können automatisiert Helm Charts über die GitLab CI/CD-Pipeline in den Cluster deployt werden.

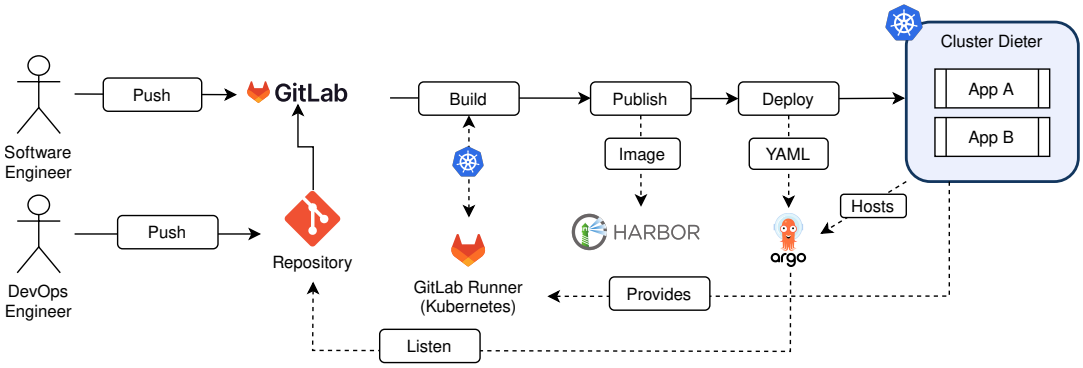

Abbildung 25: *Darstellung der aktuellen on-premise CI/CD Pipeline mit GitLab und Cluster Dieter*

In Abbildung 25 ist der schematische Aufbau der Pipeline inklusive der verwendeten Tools für CD (ArgoCD), Docker Image Registry (Harbor), CI/CD-Runner (GitLab Runner) sowie den Dieter Kubernetes Cluster dargestellt. Alle Werkzeuge außer GitLab werden innerhalb des Clusters gehostet.

Konkret gibt es je nach Akteur einen Workflow. Für einige Anwendungen gibt es ein zentrales Config Monorepository, das von ArgoCD verwaltet wird. Die Pipeline-Config-Dateien sind somit unabhängig vom Quellcode der einzelnen Projekte. Dieses Vorgehen ist nach dem GitOps-Ansatz prinzipiell erlaubt, in der Anwendung jedoch fragwürdig, da durch viele Ordner bzw. Commits Performance- und Sicherheitsprobleme entstehen können (vgl. Cobukcuoglu 2024, S. 140). Betrachtet man nun die CI/CD-Pipeline im Detail, so fällt auf, dass der Workflow des Akteurs „Software Engineer“ teilweise noch mit dem „klassischen“ CIOps-Ansatz arbeitet und nur teilweise ArgoCD integriert. Dieser hat sich in der Vergangenheit bewährt, ist aber seit dem GitOps-Ansatz nicht mehr zeitgemäß, da er u. a. gravierende Sicherheitslücken mit sich bringt (vgl. Cobukcuoglu 2024, S. 4 ff.).

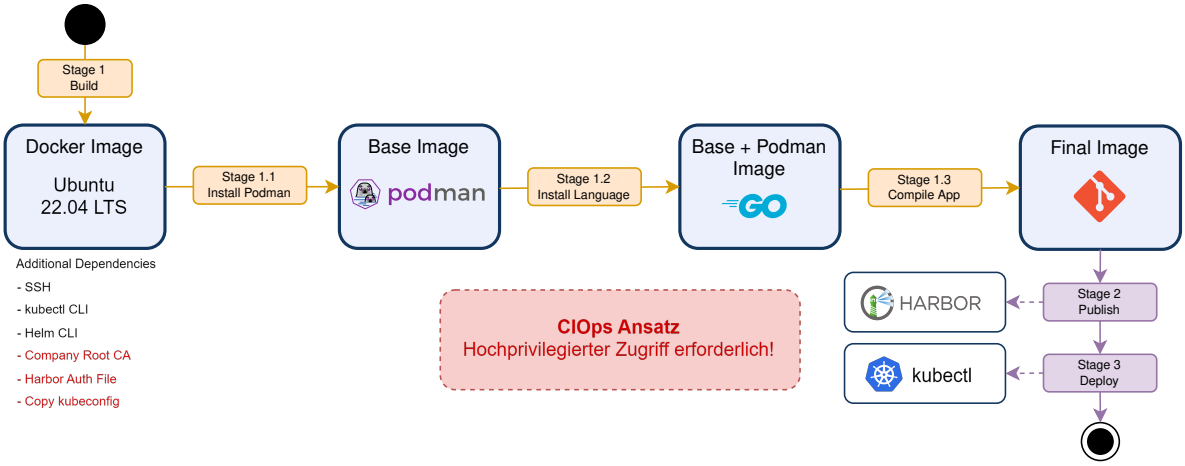

Abbildung 26: *Aufbau der verwendeten „klassischen“ CIOps-Pipeline (eigene Darstellung)*

Der in Abbildung 26 beschriebene Ansatz für Continuous Delivery (CD) wird vom CI-Server (GitLab Runner) übernommen. Dieser arbeitet nach dem Push-Prinzip, d.h. der CI-Server verbindet sich während des Build-Prozesses mit dem Kubernetes-Cluster (über kubectl) und rollt dort die Kubernetes-Manifeste aus. Dieses Vorgehen führt über kurz oder lang unweigerlich zum sogenannten Cluster Drift (vgl. Cobukcuoglu 2024, S. 5). Dabei entfernt sich der aktuelle Zustand des Clusters immer weiter von dem in Git. Wird über einen längeren Zeitraum kein Push durchgeführt, können in dieser Zeit manuelle Änderungen am Cluster vorgenommen werden, ohne dass ein automatisches Zurücksetzen auf den in Git definierten Zustand erfolgt.

Des Weiteren benötigt der Deploy-Prozess einen privilegierten Zugriff auf die Zielumgebung. Da dies in dieser Pipeline mittels kubectl umgesetzt wurde, wird eine kubeconfig benötigt, die im Klartext in einem separaten Git-Repository liegt, auf das die Deploy-Phase in Stage 3 zugreift. Diese Vorgehensweise stellt ein gravierendes Sicherheitsproblem dar, da die Config im Klartext vorliegt und die CI/CD-Pipeline darauf angewiesen ist. Außerdem kann der CI-Server die Anwendung nur dann bereitstellen, wenn ein Netzwerkzugriff auf den Cluster besteht. Diese Möglichkeit ist in Unternehmensnetzwerken durch verschiedene VLANs oder Firewalls oft nicht gegeben. Sollte der Cluster während des Deployments offline sein, kann es zu einem Drift von Anfang an kommen.

Ebenso ist es nicht möglich, eine deployte Applikation über Git zu löschen, dies muss manuell über kubectl erfolgen.

Zusammenfassend kann gesagt werden, dass die verwendete CI/CD Pipeline aufgrund der genannten Mängel nicht auf die neue Plattform übertragen werden kann. Es sind umfangreiche Anpassungen notwendig, um sie für den Kubernetes-as-a-Service-Betrieb mit GitOps kompatibel zu machen. Ebenso fehlen wichtige Sicherheitsmechanismen aus dem Bereich SecOps, wie sie in Kapitel 2.2.3 beschrieben sind.

4.2 Anforderungen an die neue Infrastruktur

In diesem Kapitel geht es um die grundsätzlichen Überlegungen zu der Kubernetes-as-a-Service Plattform, die realisiert werden soll, inklusive aller zu entwickelnden Tools. In Kapitel 4.2.1 wird zunächst die Plattform allgemein definiert, um eine ungefähre Vorstellung des Gesamtsystems zu erhalten. Anschließend werden in Kapitel 4.2.2 die potentiellen Stakeholder definiert. Die funktionalen und nicht-funktionalen Anforderungen werden in Kapitel 4.2.3 bzw. Kapitel 4.2.4 definiert.

4.2.1 Betrachtungen zum Gesamtsystem

Die Funktionalität des Gesamtsystems soll es ermöglichen, mit Hilfe von off- oder on-premise Hardware einen mandantenfähigen Kubernetes-Cluster bereitzustellen, den das Unternehmen seinen Mitarbeitern zur Verfügung stellt. Zielgruppe sind Softwareentwickler aus dem Full-Stack-Bereich. So werden vor allem Webanwendungen mit Front- und Backend in Form von APIs oder Microservices auf den Cluster deployt. Die Plattform soll aber so flexibel sein, dass nahezu jede andere Art von Software darauf deployt werden kann. Dies soll durch isolierte Umgebungen zwischen den einzelnen Nutzern erreicht werden, wobei der Administrationsaufwand und die Komplexität für den Nutzer so gering wie möglich gehalten werden soll, so dass jeder Nutzer unabhängig von seinem eigenen Kenntnisstand die Kubernetes-Plattform nutzen kann. Generell soll die Architektur in Form einer hybriden Cloud aufgebaut werden. Es soll möglich sein, Hardware aus allen Regionen in den Cluster einzubinden, um schnell neue Ressourcen zur Verfügung stellen zu können, aber auch um kundenspezifische Anforderungen abzudecken. Dies soll unter der Prämisse geschehen, dass die Plattform ausschließlich mit kostenfreien Open Source Technologien realisiert wird, um die Betriebskosten so gering wie möglich zu halten. Die Isolation zwischen den Nutzern soll sowohl auf Team- als auch auf Nutzerebene möglich sein. Das bedeutet, dass jedem Team ein eigener Cluster zur Verfügung gestellt wird, der auf Nutzerebene nochmals unterteilt wird. Dadurch wird eine maximale Unabhängigkeit und Flexibilität für alle Parteien erreicht. Im Folgenden werden diese beiden Abstraktionen als „Team-Cluster“ bzw. „Dev-Cluster“ bezeichnet.

Darüber hinaus soll jedes Team bzw. jedes Mitglied dieser Teams die Möglichkeit haben, Entscheidungen bezüglich der Infrastruktur bzw. der im Cluster laufenden Dienste völlig unabhängig von anderen Teams treffen zu können. Dadurch soll die Adaption der in Kapitel 2.2.1 beschriebenen DevOps-Kultur im Unternehmen schneller Akzeptanz finden. Zusätzlich wird sich dadurch erhofft, dass nur wenige Anfragen an das Plattform Operations Team gestellt werden, so dass sich das Team auf den Betrieb, sowie Fehler und Probleme konzentrieren kann. Langfristig wäre es optimal, wenn die gesamte Entwicklung auf dieser Plattform stattfinden würde. Dadurch könnten z. B. Kosten für Laptops etc. eingespart werden und alle Mitarbeiter würden mit einer gemeinsamen Basis arbeiten. Updates oder neue Tools können so einfach plattformweit verteilt werden.

Aus Sicht des Plattformbetriebsteams sollte die Konfiguration und Administration der gesamten Plattform mit dem GitOps-Ansatz umgesetzt werden. Damit kann auch langfristig sichergestellt werden, dass es zu keinem Cluster Drift kommt und die Wartbarkeit nicht durch undokumentierte manuelle Änderungen gefährdet wird. Der Management Cluster, der die einzelnen Tenant Cluster zur Verfügung stellt, soll mit einem „Top-Down“-Ansatz arbeiten, d. h. alle Änderungen an vorkonfigurierten Diensten, Konfigurationen etc. können durch das Betriebsteam angepasst oder überschrieben werden. Diese werden dann automatisch über Flux in die Tenant Cluster verteilt. So kann z. B. bei Supportanfragen schnell Hilfe angeboten werden und Updates auf der gesamten Plattform durchgeführt werden.

Außerdem sollen die einzelnen Dev-Cluster Umgebungen für Developer die Möglichkeit bieten, innerhalb des Clusters zu entwickeln. Dabei gelten die gleichen Voraussetzungen in Bezug auf Wissen und Unabhängigkeit wie bei den Tenant Team-Clustern. Das Bootstrapping und alle Konfigurationen sollten zentral verwaltet werden, jedoch weiterhin vollständige Unabhängigkeit, Flexibilität und einfache Bedienung bieten. Ebenso sollte die Möglichkeit bestehen, bei einem Ausfall der Cloud oder anderen Störungen lokal weiterzuentwickeln. Durch diese Flexibilität erhofft man sich, dass sich die Kubernetes-Plattform als nützliches und ergänzendes Werkzeug im Alltag der Nutzer etabliert, um langfristig die DevOps-Kultur und die damit verbundenen Vorteile zu festigen. Darüber hinaus ergeben sich völlig neue Möglichkeiten in Bezug auf die Ressourcenverfügbarkeit, da beispielsweise mit GPUs leistungsfähige Hardware problemlos eingebunden werden kann. Dies eröffnet völlig neue Horizonte im Bereich Machine Learning und MLOps.

Als weitere Unterstützung soll jedem Nutzer ein Tool in Form einer Command Line Interface (CLI) zur Verfügung gestellt werden. Dieses soll die Komplexität und die hohe Lernkurve von Kubernetes im Allgemeinen und im Zusammenhang mit CI/CD reduzieren, indem alltägliche Operationen wie das Anlegen neuer Deployments, die Verwaltung von Namespaces sowie die Erstellung von CI/CD-Dateien weitestgehend automatisiert werden. So dass die Kubernetes Plattform als Kubernetes-as-a-Service Dienst für den Endanwender nutzbar wird. Die genaue Konzeption und Implementierung dieses Tools ist in Kapitel 6 und Kapitel 7 beschrieben.

4.2.2 Stakeholder

Bevor die Anforderungen konkret definiert werden können, müssen die einzelnen Systemnutzer bzw. Stakeholder aus den verschiedenen Anwendungsbereichen identifiziert werden. Die Auswahl der Nutzerrollen erfolgt dabei mit Fokus auf die Softwareentwicklung, wobei eine spätere Änderung der Nutzerrollen die Anforderungen nicht oder nur geringfügig verändern sollte. Die Plattform selbst basiert auf einem generischen Konzept, das für alle Anwendungen zur Verfügung steht, die generell mit Kubernetes realisiert werden können. Des Weiteren stellen die folgenden Stakeholder nur eine mögliche Aufteilung der Aufgabenbereiche dar, dies kann je nach Anwendungsfall variieren.

Es werden folgende Stakeholder definiert:

- Der **Softwareentwickler** stellt den größten Teil der Anwender dar, er nutzt die Plattform, um Software zu entwickeln, zu testen und in verschiedenen Umgebungen wie Testing, Staging oder Productive zu deployen, um sie anderen zur Verfügung zu stellen. Während der Entwicklung werden in seinem Dev-Cluster verschiedene Tools zu Testzwecken installiert, um diese zu evaluieren. Für einige Anwendungen wird auch persistenter Speicher benötigt sowie eine IngressClass, um die Anwendungen im Web zur Verfügung zu stellen.

- Der **Gruppenleiter / Tenant-Admin** eines Entwicklungsteams wird vom Plattformbetriebsteam als Tenant-Admin definiert. Damit hat er die Möglichkeit, weitere Dev-Cluster anzulegen und bestimmte Applikationen wie z. B. Datenbanken jedem Teammitglied automatisiert zur Verfügung zu stellen. Diese Applikationen werden als Shared Apps bezeichnet und können nur vom Tenant Admin definiert werden. Da für das Anlegen neuer Dev-Cluster ein privilegierter Zugang zum Team-Cluster erforderlich ist, können Softwareentwickler nicht selbst neue Dev-Cluster anlegen. Dies kann jedoch je nach Anwendungsfall angepasst werden. Die Rolle des Tenant Admin kann auch weitergegeben werden, wenn ein anderes Teammitglied diese Rolle übernehmen möchte.

- Der **Plattformadministrator** ist für die Administration der gesamten Cloud und der damit verbundenen Kubernetes Plattform zuständig. Er ist verantwortlich für Updates, Supportanfragen, Ausfallsicherheit, Backups, Dokumentation etc. Er kann weitere Team-Cluster einrichten.

4.2.3 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen an das Gesamtsystem definiert. Um geeignete Anforderungen für den in Kapitel 4.2.1 definierten Use Case definieren zu können, werden neben den Vorstellungen des Autors weitere Quellen wie die durchgeführte Unternehmensbefragung in Kapitel 3 sowie die Analyse der aktuellen Infrastruktur in Kapitel 4.1 herangezogen. Auf diese Weise kann die Plattform so geplant und aufgebaut werden, wie es von den Nutzern gewünscht wird. Außerdem findet so bereits indirekt eine Verifikation der Anforderungen durch eine externe Instanz statt, da diese durch die qualitativen Ergebnisse der Befragung bereits im Vorfeld entstanden sind.

Konkret werden die Anforderungen in Form von User Stories definiert. Dabei wird jeweils die betroffene Rolle sowie deren Ziel definiert. Dies geschieht in einer standardisierten Form, um die Lesbarkeit und Qualität über alle User Stories zu gewährleisten. Dabei wird nach folgendem Schema vorgegangen (vgl. Salimi 2023):

Als [ROLLE/PERSONA] möchte ich [ZIEL], damit/um [GRUND/NUTZEN]

Die Anforderungen werden gemäß der „MoSCoW“-Priorisierung (vgl. Coley 2019) priorisiert. Diese Priorisierung ordnet die einzelnen User Stories einer von vier Kategorien zu und hilft, die Anforderungen klar zu priorisieren und sicherzustellen, dass die wichtigsten Elemente zuerst bearbeitet werden. In allen folgenden Kapiteln werden die funktionalen Anforderungen als „**FA [ID]**“ gekennzeichnet.

1. **Must have** (M): Diese Anforderungen sind für den Erfolg des Projekts unerlässlich. Ohne sie ist die Implementierung nicht vollständig.
2. **Should have** (S): Diese Anforderungen sind wichtig, aber nicht zwingend. Sie können bei Bedarf verschoben werden, ohne dass das Projekt scheitert.
3. **Could have** (C): Diese Anforderungen sind wünschenswert und würden den Wert des Projekts erhöhen, sind aber nicht entscheidend für den Projekterfolg.
4. **Won't have** (W): Diese Anforderungen werden in der aktuellen Phase nicht umgesetzt, könnten aber in zukünftigen Iterationen oder Projekten berücksichtigt werden.

In Tabelle 2 folgt nun die Definition der User Stories für die Kubernetes-as-a-Service Plattform:

Tabelle 2: *Funktionale Anforderungen an die Kubernetes-as-a-Service Plattform*

Nr.	User Story	Priorität
FA 1	Als Softwareentwickler möchte ich die Möglichkeit haben, alle Anwendungen in meinem Cluster selbstständig zu deployen, damit ich bei meiner Arbeit nicht blockiert werde.	M
FA 2	Als Softwareentwickler möchte ich vollen Zugriff auf mein(e) Dev-Cluster mittels kubeconfig haben, um Kubernetes Tools wie kubectl, Helm, etc. nutzen zu können.	M
FA 3	Als Gruppenleiter möchte ich neue Dev-Cluster in meinem Team erstellen, löschen und bearbeiten können, damit meine Teammitglieder arbeiten können.	M
FA 4	Als Gruppenleiter möchte ich die Kontrolle über vorkonfigurierte Anwendungen haben, um sie automatisch allen Teammitgliedern zur Verfügung zu stellen.	M
FA 5	Als Gruppenleiter möchte ich in der Lage sein, die Ressourcenverwaltung der einzelnen Dev-Cluster so anzupassen, dass ressourcenintensive Umgebungen möglich sind.	M
FA 6	Als Gruppenleiter möchte ich die Kontrolle über Worker-Knoten haben, um sie zu meinem Team-Cluster hinzuzufügen, zu aktualisieren und zu löschen.	M

Nr.	User Story	Priorität
FA 7	Als Gruppenleiter möchte ich eine zentrale Übersicht über alle Dev-Cluster und Shared Apps haben, um den Überblick nicht zu verlieren.	M
FA 8	Als Gruppenleiter möchte ich die Möglichkeit haben, meine Aufgaben als Tenant Admin abzugeben, damit sie von einer anderen Person übernommen werden können.	M
FA 9	Als Gruppenleiter möchte ich meine Team-Cluster in einem Git-Repository verwalten, um eine Änderungshistorie zu haben und alle Änderungen automatisch deployen zu können.	M
FA 10	Als Softwareentwickler möchte ich vorkonfigurierte Storage- und IngressClasses auswählen können, um meinen Anwendungen dynamisch persistenten Speicher und eine Webadresse zuzuweisen.	M
FA 11	Als Softwareentwickler möchte ich einen Überblick über die Dev-Cluster haben, die mir zur Verfügung stehen, damit ich schnell Anwendungen darauf deployen kann.	M
FA 12	Als Softwareentwickler möchte ich die Möglichkeit haben, zwischen lokaler und Cloud-basierter Softwareentwicklung zu wechseln, um bei Internetausfällen oder anderen Ausfallzeiten nicht blockiert zu sein.	M
FA 13	Als Softwareentwickler möchte ich meine vertraute Entwicklungsumgebung (IDE) weiterhin verwenden, um effizient arbeiten zu können.	M
FA 14	Als Softwareentwickler möchte ich die Möglichkeit haben, die Kubernetes-Plattform trotz meiner begrenzten Kenntnisse in diesem Bereich zu nutzen, um von der CI/CD-Pipeline und anderen Vorteilen zu profitieren.	M
FA 15	Als Plattformadministrator möchte ich in der Lage sein, die einzelnen Team-Cluster sowie Shared Apps in einem Git-Repository zu verwalten, um eine Änderungshistorie zu haben und alle Änderungen automatisiert deployen zu können.	M
FA 16	Als Softwareentwickler möchte ich die freie Wahl des Betriebssystems haben, um lokal und in der Cloud entwickeln zu können.	M
FA 17	Als Gruppenleiter möchte ich mit meinem Team vorkonfigurierte Devontainer erstellen, damit neue Mitarbeiter schnell eingearbeitet werden können.	M
FA 18	Als Gruppenleiter möchte ich eine Garantie dafür haben, dass die Kundendaten nicht außerhalb der Plattform gespeichert werden, um die Datenschutzanforderungen meiner Kunden zu erfüllen.	M

Nr.	User Story	Priorität
FA 19	Als Plattformadministrator möchte ich die Verwaltung der Hardware über Infrastructure-as-Code (IaC) steuern, um Cluster Sprawl zu vermeiden.	S
FA 20	Als Softwareentwickler möchte ich die aktuell genutzten und verfügbaren Ressourcen einsehen können, um das System nicht zu überlasten.	S
FA 21	Als Softwareentwickler möchte ich meine Anwendung in verschiedenen Dev-Clustern deployen können, um verschiedene Stages abzubilden.	S
FA 22	Als Gruppenleiter möchte ich in der Lage sein, mein Team-Cluster sowie Shared Apps in einem Git-Repository zu verwalten, um eine Änderungshistorie zu haben und alle Änderungen automatisiert deployen zu können.	M
FA 23	Als Softwareentwickler, möchte ich meinen Quellcode in der CI/CD-Pipeline mit SecOps-Tools überprüfen, um sicheren Code auszuliefern.	C
FA 24	Als Softwareentwickler möchte ich verschiedene Betriebssysteme im Cluster verwenden können, um meine verschiedenen Anwendungen zu testen.	C
FA 25	Als Plattformadministrator möchte ich über automatisierte Backup-Lösungen für den persistenten Speicher verfügen, um im Katastrophenfall alles wiederherstellen zu können.	C

Mit den „Must have“-Anforderungen soll zum Ende dieser Arbeit ein lauffähiges Minimum Viable Product (MVP) umgesetzt werden. Weitere Anforderungen einer niedrigeren Priorität werden bestmöglich umgesetzt jedoch besteht keine gewährleistung, dass diese im Zeitraum dieser Arbeit realisiert werden können.

4.2.4 Nicht-funktionale Anforderungen

In diesem Abschnitt werden die nicht-funktionalen bzw. Qualitätsanforderungen definiert. Diese werden ebenfalls gemäß der „MoSCoW“-Priorisierung priorisiert und in den folgenden Kapiteln mit dem Kürzel „**QA [ID]**“ gekennzeichnet. Die Qualitätsanforderungen sind in Tabelle 3 formuliert. Die Bewertung dieser ist in Kapitel 8 zu finden.

Tabelle 3: *Nicht-funktionale Anforderungen an die Kubernetes-as-a-Service Plattform*

Nr.	User Story	Priorität
QA 1	Das System muss eine ausreichende Isolation zwischen den Team- und Dev-Clustern bieten, um unabhängige und sichere Arbeitsumgebungen zu gewährleisten.	M
QA 2	Die Plattform muss ausschließlich mit kostenfreien Open Source Technologien realisiert werden, um die Betriebskosten so gering wie möglich zu halten.	M
QA 3	Das CLI-Tool muss eine einfache und intuitive Bedienung bieten, damit auch Nutzer mit wenig Erfahrung es erfolgreich verwenden können.	M

5 Realisierung der Kubernetes-as-a-Service Plattform

Dieses Kapitel beinhaltet die Konzeption und Implementierung der Kubernetes-Infrastruktur. Dabei werden zunächst in Kapitel 5.1 einige Tools miteinander verglichen, die später in den Kapiteln 5.2, 5.3, 5.4 und 5.5 bei der Realisierung der Plattform zum Einsatz kommen. Neben der Kubernetes Plattform wird auch eine CI/CD Pipeline in GitLab implementiert. Deren Struktur ist in Kapitel 5.6 beschrieben.

5.1 Analyse geeigneter Open-Source-Lösungen

Um die Kubernetes-as-a-Service-Plattform zu realisieren, müssen zunächst einige Tools miteinander verglichen werden, um die bestmögliche Lösung im Einklang mit den Anforderungen zu finden. Aufgrund der qualitativen Anforderung QA2 kommen für die Vergleiche nur Tools in Frage, die frei verfügbar sind. Generell werden in diesem Abschnitt Tools aus den wichtigsten Kernkomponenten der Kubernetes Plattform verglichen. Dazu gehören das Betriebssystem, die Mandantenfähigkeit von Kubernetes, das Container Storage Interface (CSI), das Container Network Interface (CNI), das Load Balancing sowie Lösungen, die eine Cloud-basierte Softwareentwicklung ermöglichen.

Im Rahmen der Entscheidungsfindung für die folgenden Vergleiche wurde für manche Entscheidungen eine Nutzwertanalyse durchgeführt, um die qualitativen und quantitativen Kriterien zu bewerten und zu gewichten.

Die Nutzwertanalyse basiert auf einer Skala von 0 „nicht erfüllt“ bis 10 „überragend“, die den Zielerfüllungsfaktor darstellt. Die folgenden Schritte wurden durchgeführt (vgl. Bundesministerium 2024):

1. **Festlegung der Kriterien:** Zunächst wurden die relevanten Kriterien identifiziert, die für die Entscheidungsfindung von Bedeutung sind.

2. **Gewichtung der Kriterien:** Jedes Kriterium wurde basierend auf seiner Wichtigkeit für den spezifischen Anwendungsfall gewichtet. Die Gewichtungen wurden in Prozent ausgedrückt, wobei die Summe aller Gewichtungen 100% ergibt.

3. **Bewertung der Lösungen:** Die verschiedenen Lösungen wurden hinsichtlich der festgelegten Kriterien auf einer Skala von 0 bis 10 bewertet. Diese Bewertungen basieren auf einer fundierten Analyse, die Literaturrecherche, Erfahrungsberichte und technische Dokumentationen umfassen kann.

4. **Berechnung der Teilnutzwerte:** Für jedes Kriterium wurde der Teilnutzwert berechnet, indem die Gewichtung des Kriteriums mit der Bewertung der Lösung multipliziert

wurde:

$$TN = Gf \times Zf$$

wobei TN der Teilnutzwert, Gf der Gewichtungsfaktor und Zf der Zielerfüllungsfaktor (Bewertung) ist.

5. **Berechnung des Gesamtnutzwertes:** Der Gesamtnutzwert für jede Lösung wurde durch die Summe der Teilnutzwerte berechnet:

$$GN = \sum TN$$

Dies ermöglicht eine ganzheitliche Bewertung jeder Lösung.

6. **Interpretation der Ergebnisse:** Anhand der Gesamtnutzwerte wurde die Eignung der verschiedenen Lösungen verglichen. Ein höherer Gesamtnutzwert deutet auf eine bessere Eignung im spezifischen Anwendungsfall hin.

5.1.1 Betriebssystem

Problemstellung

Als Basis für die verschiedenen physischen oder virtuellen Server, die die einzelnen Worker oder Control Plane Nodes für Kubernetes hosten, muss zunächst ein geeignetes Betriebssystem ausgewählt werden. Kubernetes selbst gibt Linux als Systemvoraussetzung an (vgl. Kubernetes 2024f), weshalb im Folgenden nur Linux-Distributionen miteinander verglichen werden können. Neben den klassischen Linux-Distributionen wie Debian, Ubuntu oder CentOS ergeben sich gerade im Zusammenhang mit Kubernetes völlig neue Möglichkeiten bei der Wahl des Betriebssystems. Das in Kapitel 2.5.4 erwähnte Problem des Cluster Sprawl lässt sich neben den Clustern selbst auch auf die Hardware bzw. Infrastruktur übertragen. So kann es bei einer großen Anzahl von Servern dazu kommen, dass durch die steigende Komplexität und den daraus resultierenden Wartungsaufwand der Überblick verloren geht. Dies betrifft vor allem automatisierte Updates und Konfigurationsänderungen, die mehr als einen Server betreffen. Da der GitOps-Ansatz auch die Infrastruktur mit einbezieht, muss auch hier eine geeignete Lösung gefunden werden, um im besten Fall alle Server zentral über Git steuern zu können. Dies vermeidet Server Sprawl und Configuration Drift und reduziert gleichzeitig die Komplexität. Hinzu kommen Punkte wie Zugriffskontrolle, sichere Kommunikation, Updates und Konfigurierbarkeit.

Kernkonzept von Talos

Talos Linux ist ein Betriebssystem mit der Besonderheit, dass es für den reinen Einsatz von Kubernetes entwickelt wurde und nicht wie z. B. Ubuntu aus einem Fork von Debian entstanden ist. Talos wurde mit Ausnahme des offiziellen Linux-Kernels vollständig in der Sprache

Go entwickelt und basiert auf einer minimalistischen Architektur mit den Schwerpunkten Sicherheit (Secure), Deklarative Konfiguration (Declarative), Kurzlebigkeit (Ephemeral), Minimal, Verteilt (Distributed) und Unveränderlich (Immutable) (vgl. Sidero 2024b). Alle Eingaben werden über einen gRPC-API-Endpunkt an das Betriebssystem selbst gemacht. Dadurch können manuelle Änderungen ausgeschlossen und eine Zugriffskontrolle realisiert werden. In der Regel erfolgt dies über ein eigenes CLI mit dem Namen „talosctl“. In Abbildung 27 ist die grundlegende Architektur von Talos dargestellt. Die orangefarbenen Kugeln sind Eigenentwicklungen von Sidero Labs und sorgen unter anderem für Zugriffskontrolle, Kommunikation und automatisches Bootstrapping von Kubernetes.

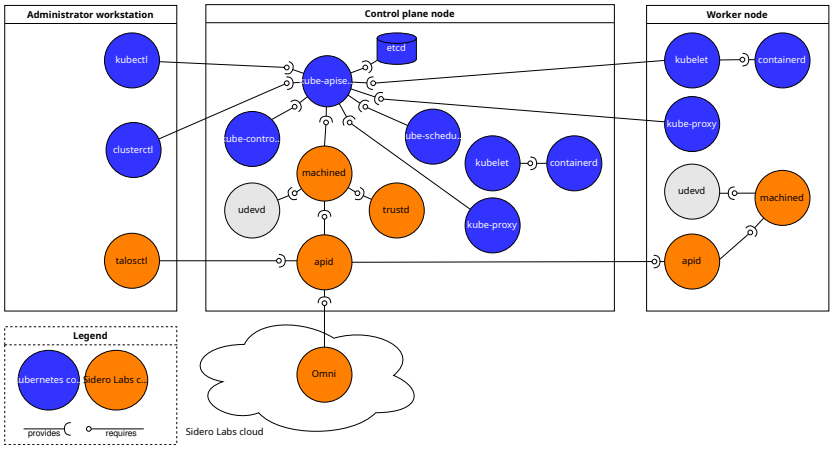

Abbildung 27: *Übersicht über die Talos Linux Architektur (vgl.* Sidero *2024b)*

Vergleich

Der Vergleich basiert auf einer klassischen Linux-Distribution und einer für Kubernetes optimierten Distribution. Erstere wird Debian in der LTS Version 12 „Bookworm“ sein, da diese vor allem auf Stabilität und eine breite Hardwareunterstützung setzt (vgl. Debian 2024). Für die zweite Kategorie wurde Talos Linux ausgewählt.

Für die Bewertung der einzelnen Kriterien der Nutzwertanalyse wurde für beide Lösungen der Vanilla-Zustand ohne Installation weiterer Pakete angenommen. Die Gewichtung der einzelnen Kriterien in Kapitel 5.1.1 erfolgte dabei nach ihrer Wichtigkeit, um die Anforderung FA 19 bestmöglich zu erfüllen und die oben genannten Probleme von vornherein zu umgehen bzw. zu lösen. Generell bietet Debian eine deutlich bessere Individualisierbarkeit, da eine große Auswahl an Paketen installiert werden kann und Anpassungen am Betriebssystem manuell vorgenommen werden können. Da der Fokus jedoch auf Kubernetes und dessen Administration und Installation liegt, konnten die Zielerfüllungsfaktoren gerade bei den Kriterien deklarative Konfiguration und Automatisierung bei Talos deutlich höher angesetzt werden, da es genau für diese Zwecke entwickelt wurde (vgl. Sidero 2024b).

		Debian 12		Talos Linux	
Kriterium	**Gf (%)**	**Zf**	**TN**	**Zf**	**TN**
Deklarative Konfiguration	30	0	0	10	300
Automatisierung	15	6	90	10	150
Integration von Kubernetes	15	0	0	10	150
Sicherheit	15	8	120	9	135
Zugriffskontrolle	10	6	60	9	90
Individualisierbarkeit	5	10	50	6	30
Hardwareunterstützung	5	10	50	10	50
Community und Support	5	10	50	5	25
SUMME	100		**420**		**930**

Tabelle 4: *Nutzwertanalyse: Vergleich von Debian und Talos Linux*

Fazit

Aufgrund der spezifischen Kriterien, die sich auf das Management von Kubernetes und die Verwaltbarkeit durch GitOps konzentrieren, konnte Talos Linux in diesem Vergleich überzeugen. Mit insgesamt 930 Punkten ist der Nutzen in diesem Vergleich mehr als doppelt so hoch wie bei Debian mit nur 420 Punkten. Aufgrund dieser Tatsache und den oben genannten Vorteilen von Talos Linux wurde dieses Betriebssystem für die Kubernetes-Plattform ausgewählt.

5.1.2 Multi-Tenancy

Problemstellung

Um die Mandantenfähigkeit innerhalb der Kubernetes-Plattform mit den entsprechenden Anforderungen FA1, FA2, FA3, FA5 und FA10 umsetzen zu können, müssen Lösungen verglichen werden, die sowohl „Soft-“ als auch „Hard“-Multi-Tenancy-fähig sind. In Kapitel 2.5.4 wurden die Begriffe sowie die derzeit etablierten Multi-Tenancy-Ansätze bereits definiert. Da die Self-Service-Plattform im Rahmen dieser Arbeit unter der Prämisse entwickelt wird, nur unternehmensintern zwischen Entwicklerteams genutzt zu werden, ist eine „harte“ Mandantenfähigkeit zwischen den Teams und eine „weiche“ Mandantenfähigkeit innerhalb der Teams erforderlich. Da davon ausgegangen werden kann, dass sich die Teammitglieder innerhalb eines Teams gegenseitig vertrauen, ist es nicht notwendig, extra isolierte Bereiche zu schaffen, die nur von bestimmten Personen eingesehen oder verwendet werden können. Dies ist dies sogar von Vorteil, da mehrere Personen so z. B. durch Pair Programming zusammenarbeiten können. Gleiches gilt für die Ressourcennutzung innerhalb des Teams. Da jedes Team-Cluster seine eigenen Worker-Nodes beherbergt, müssen die Teammitglieder die Ressourcen untereinander so aufteilen, dass alle damit arbeiten können, weshalb eine „schwache“ Mandantenfähigkeit ausreichend ist.

Da die Mandantenfähigkeit im Kubernetes-Umfeld immer häufiger zum Einsatz kommt, gibt

es auch immer mehr Frameworks, die versuchen, die verschiedenen Konzepte umzusetzen und dabei immer weniger komplex und benutzerfreundlicher werden. Im Rahmen dieser Arbeit wurde aus den drei Ansätzen jeweils ein Framework ausgewählt und auf seine Eignung für die Kubernetes-Plattform bewertet.

Kamaji

Für den Aufbau des in Kapitel 5.3 beschriebenen Management Clusters bzw. der Private Cloud wird ein Framework benötigt, das eine „harte Mandantenfähigkeit ermöglicht. Hierfür wird das Framework „Kamaji“ in Betracht gezogen. Das Konzept von Kamaji ist optimal für den Einsatz zur Realisierung einer Private oder Hybrid Cloud, da es klassische virtuelle oder physische Kubernetes Cluster in sogenannte „Management-Cluster“ umwandelt, die dann in der Lage sind, über Virtual Control Planes (VCPs) weitere Mandanten als vollwertige Kubernetes Cluster anzulegen. Man könnte das mit Managed Kubernetes von AWS oder Azure Cloud vergleichen. Dort ist man meistens auch nur ein Mandant in einem größeren physischen Cluster. Jeder VCP hat seine eigene Kubernetes API. Dies ermöglicht eine vollständige Isolation der administrativen Aktionen und stellt sicher, dass die Tenants nichts voneinander wissen und sich nicht gegenseitig beeinflussen können (siehe auch im Folgenden, Clastix 2020). Damit wäre bereits ein Teil der Anforderung FA18 erfüllt und der in Kapitel 4.2.1 beschriebene „Top-Down“-Ansatz umsetzbar, da das Management Cluster über eine unidirektionale Verbindung Änderungen in den einzelnen Tenant Clustern vornehmen kann. Die grundlegende Architektur von Kamaji ist in Abbildung 28 noch einmal bildlich dargestellt. Dort sind sowohl der Management Cluster als auch die einzelnen Tenants zu sehen und wie diese über den VCP miteinander verbunden sind.

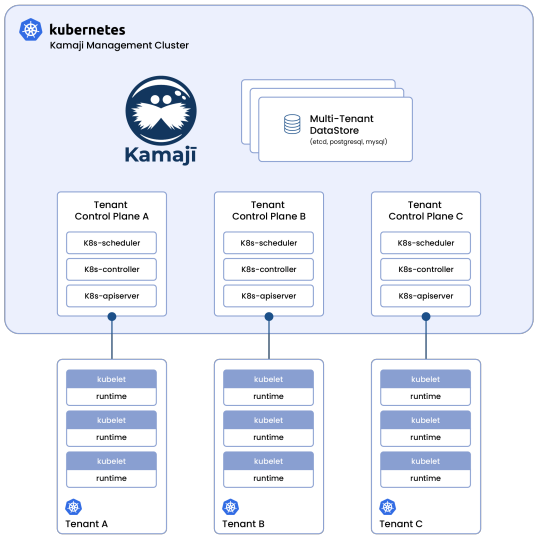

Abbildung 28: *Übersicht über die Kamaji Architektur (vgl.* Clastix *2020)*

Ein weiterer positiver Aspekt ist die Integration von GitOps. So können z. B. mit FluxCD automatisch Tenant Cluster erstellt und verwaltet werden. Damit könnte die Anforderung FA15 erfüllt werden. Eine weitere Besonderheit, die für diese Lösung spricht, ist der Umgang mit Workloads und die strikte Trennung der Worker-Nodes zwischen den Tenants. Da nur die Control Plane virtualisiert ist, muss jeder Tenant seine eigenen Ressourcen bereitstellen bzw. integrieren. Konkret bedeutet dies, dass jeder Tenant seine eigenen dedizierten Worker Nodes benötigt, wodurch das Noisy Neighbor Problem im Cloud Computing Kontext eliminiert wird. Somit kann eine hohe Auslastung eines „benachbarten Tenants die Performance im eigenen Tenant Cluster nicht beeinflussen. Dies würde sich auch positiv auf datenschutzrelevante Themen auswirken, da nur das Entwicklungsteam auf die Worker Nodes zugreifen kann. Durch die harte Isolation der einzelnen Cluster wird außerdem der Verwaltungsaufwand für die Plattformadministratoren gesenkt, welche sich sonst über Konzepte zur Isolation der Cluster gedanken machen müssten.

vCluster

Als weiterer Vergleich wird das Open Source Framework vCluster von Loft in den Vergleich einbezogen, da Kamaji und vCluster derzeit die einzigen größeren Open Source Lösungen am Markt sind, die das Konzept der Virtual Control Plane realisieren (vgl. Kubernetes 2024h und CNCF 2024). Bei näherer Betrachtung von vCluster zeigen sich jedoch deutliche Unterschiede in der Architektur im Vergleich zu Kamaji. Während Kamaji vollständig isolierte Cluster mit eigenen Worker Nodes erzeugt, erzeugt vCluster innerhalb des Management Clusters einen Cluster in einem neuen Namespace, der dort als Pod gestartet wird und die Ressourcen des Hostsystems nutzt. Dies hat unter anderem den Vorteil, dass es insgesamt schlanker und weniger ressourcenintensiv ist als Kamaji, dafür aber weniger strikte Isolation bietet. Die Architektur ist in Abbildung 29 dargestellt. Es ist zu erkennen, dass low-level Ressourcen, die von den Workloads der einzelnen Namespaces bzw. Tenants benötigt werden, über einen „syncer“ auf das Hostsystem synchronisiert und dort ausgeführt werden, da die Tenants über keine eigenen Worker-Nodes verfügen. Im Gegensatz zu High-Level-Ressourcen wie Deployments, Service Accounts oder CRDs, die im Key-Value-Speicher des jeweiligen Tenants in dessen Namespace abgelegt werden.

Diese Kombination aus der Nutzung von Ressourcen des Host-Systems und der Bereitstellung einer eigenen API für die Tenant-Benutzer vermittelt dem Benutzer den Eindruck, einen „echten“ dedizierten Cluster zu betreiben. Da jeder Tenant seine eigene Datenhaltung hat, ist ein Zugriff auf andere Tenants auf API-Ebene ausgeschlossen. Auch eine Kommunikation in das Hostsystem ist durch eine DNS-Isolation seitens vCluster ausgeschlossen. Dies ist jedoch in der kostenpflichtigen Variante „vCluster Pro“ möglich (vgl. Loft 2024a).

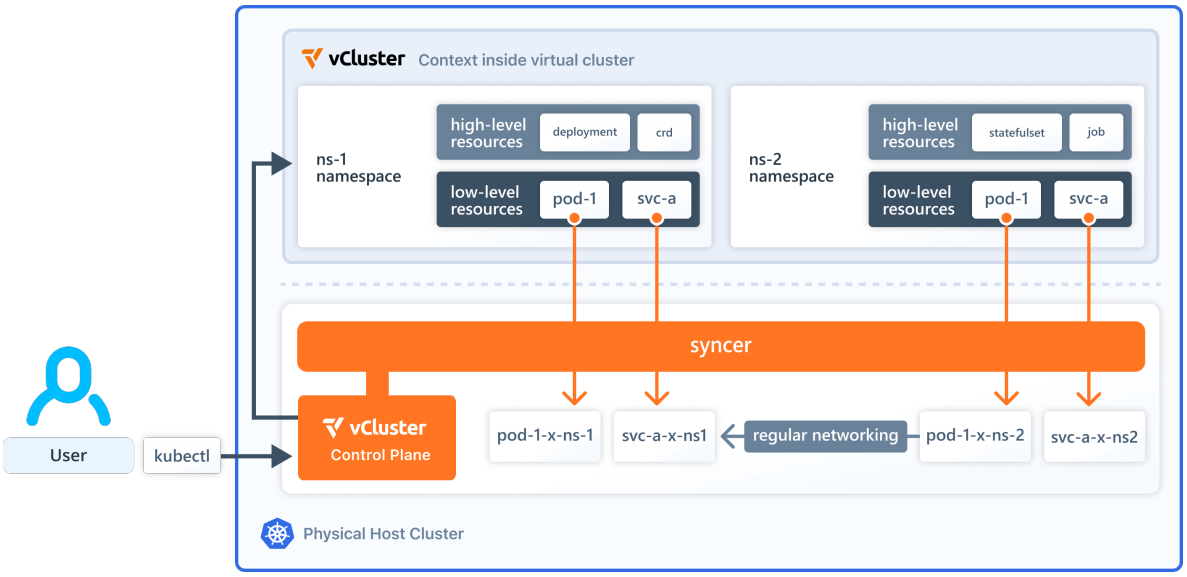

Abbildung 29: *Übersicht über die vCluster Architektur (vgl.* Loft *2024a)*

Die Integration von GitOps über YAML-Dateien, wie sie bei Kamaji möglich ist, funktioniert bei vCluster so nicht. Dort werden alle Konfigurationen und Tenants über eine eigene vCluster API gesteuert, die über eine eigene CLI angesprochen werden kann. Dies stellt ein erhebliches Problem dar, das gegen den Einsatz von vCluster auf dieser Kubernetes-Plattform spricht, da es im Widerspruch zu verschiedenen Anforderungen wie z. B. FA15 steht.

Capsule

Mit dem Open Source Framework Capsule aus dem Jahr 2022 kann das Konzept der Tenant Namespace Isolation umgesetzt werden. Wie in Kapitel 2.5.4 beschrieben, ist die Namespace Isolation ein Teil der „weichen“ Mandantenfähigkeit. Das Framework ist daher nicht für Team-Cluster, sondern nur für die in Kapitel 5.4 beschriebenen Dev-Cluster geeignet. Die Installation der Software selbst erfolgt in einem bestehenden Kubernetes Cluster mittels eines Helm Chart. Capsule selbst erlaubt es, einen oder mehrere Namespaces zu bündeln und einem bestimmten Mandanten zuzuordnen. Capsule selbst bezeichnet diese Mandanten als Tenants und kann diese über ein mitgeliefertes Custom Ressource Definition (CRD) erzeugen. Durch die Möglichkeit der CRDs ist es möglich, YAML-Dateien für einen Tenant zu erstellen, was wiederum bedeutet, dass GitOps mit diesem Framework möglich ist. Neben der Zuweisung von Namespaces ist es auch möglich, Rechenkapazitäten zu definieren und einem Tenant den Zugriff auf clusterweite Ressourcen wie Nodes, StorageClasses, IngressClasses, RuntimeClasses und PersistentVolumes zu erlauben. Alle weiteren Kubernetes Policies und Constraints, um die schwache Isolation zwischen den Tenants zu erreichen, werden von Capsule abstrahiert und verwaltet. Die Verwaltung der Tenants konzentriert sich somit nur auf die Tenant-Config als CRD, was einen administrativen Vorteil bietet. All dies sind Grundvoraussetzungen, um den Dev-Cluster wie gewünscht implementieren zu können und FA10 zu erfüllen. In Abbildung 30

ist die Architektur von Capsule nochmals bildlich veranschaulicht.

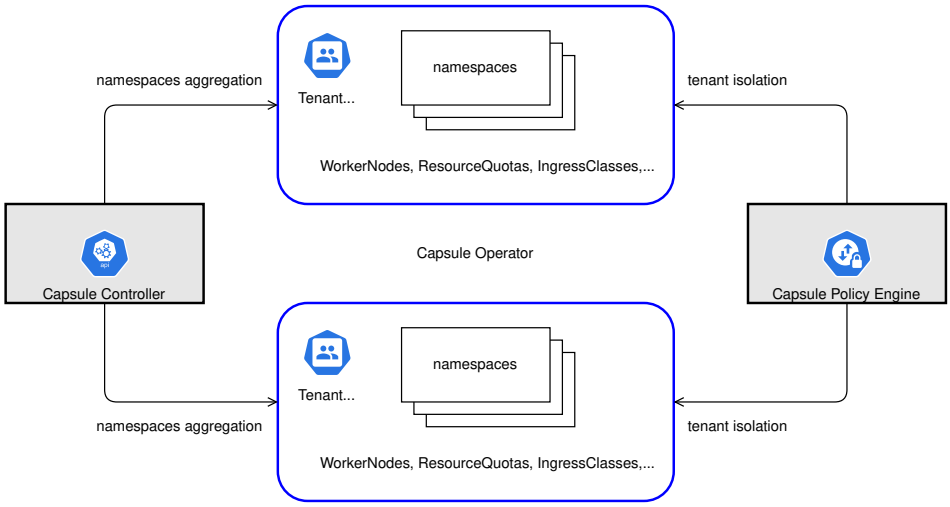

Abbildung 30: *Übersicht über die Capsule Architektur (vgl.* Clastix *2022)*

Ein weiterer zu prüfender Punkt ist das Rechtesystem und die Zugriffsmöglichkeiten auf die Tenant-Umgebung. Da die Anforderungen FA1 und FA2 voraussetzen, müssen die bisherigen CLI-Tools weiterhin funktionieren und der Zugriff über kubeconfig muss weiterhin möglich sein. Auch dies kann Capsule erfüllen, da über das Kubernetes-eigene RBAC für jeden Tenant eine kubeconfig erzeugt werden kann, wodurch die Nutzung bereits vorhandener Tools von außen problemlos möglich ist (vgl. Clastix 2022). Es wird unterschieden zwischen einem „Tenant-Owner“, der volle Rechte innerhalb des Tenants hat, und weiteren Benutzern, die vom Tenant-Owner angelegt werden können. Als Tenant-Owner hat man auch die Möglichkeit, weitere Namespaces anzulegen, die automatisch dem Tenant zugeordnet werden. Dies kann jedoch in der Tenant-Config eingeschränkt werden.

Neben den genannten Vorteilen gibt es jedoch auch Nachteile hinsichtlich der Nutzungsfreiheit innerhalb eines Tenants. So ist es z. B. nicht möglich, eigene CRDs zu definieren, da diese nur auf Clusterebene und somit nicht direkt von einem Tenant-Eigentümer erstellt werden können. Dies kann die Flexibilität der Entwickler innerhalb eines Tenants einschränken, wenn eine neue Anwendung installiert werden soll. Dies ist z. B. häufig bei Datenbanken wie MongoDB oder MariaDB und vielen anderen komplexen Anwendungen der Fall. Hier kann nur der Gruppenleiter als Clusteradministrator Abhilfe schaffen, indem er die CRDs im Cluster anlegt. Diese Einschränkung ist in diesem Kontext jedoch vertretbar, da in der Regel alle wichtigen Frameworks, die für das jeweilige Projekt benötigt werden, bereits im Team-Cluster installiert sind. Bei der Einführung eines neuen Tools muss dann immer die Zustimmung des Gruppenleiters eingeholt werden. Dies kann auch aus Sicherheits- und Datenschutzgründen sinnvoll sein, da so nicht einfach ungeprüft Anwendungen im Cluster gestartet werden können. Dies gilt jedoch nur für Anwendungen mit CRDs.

Capsule wird durch ein Add-On namens Capsule Proxy ergänzt, das einige RBAC-Probleme

in der Mandantenfähigkeit von Kubernetes löst, indem es Benutzern erlaubt, nur die Ressourcen clusterweit aufzulisten, die ihnen gehören. Das Problem mit Kubernetes RBAC ist, dass es keine ACL-gefilterten APIs gibt, was bedeutet, dass Benutzer ohne entsprechende Berechtigungen keine clusterweiten Ressourcen auflisten können (vgl. Clastix 2023). Capsule Proxy umgeht diese Einschränkungen, indem es einen Reverse Proxy implementiert, der bestimmte Anfragen an den Kubernetes API Server abfängt und die Zugriffsprobleme durch interne Logik im Capsule Controller auflöst und an den Nutzer zurücksendet. Dies sind Anfragen wie z. B. „/api/v1/namespaces“, „/api/v1/nodes“, „/api/v1/persistentvolumes“. Dadurch kann der Tenant-Benutzer über kubectl auf alle ihm zugewiesenen Ressourcen, Nodes, StorageClasses etc. zugreifen und diese verwalten. Ohne diesen Proxy wäre Capsule für den Dev-Cluster nicht geeignet, da dies im Widerspruch zu den Anforderungen FA11 und ggf. FA20 stünde.

Fazit

Bei der Analyse zur Findung des optimalen Frameworks zur Umsetzung der Mandantenfähigkeit wurden die derzeit am Markt populärsten Lösungen verglichen, die alle drei unterschiedliche Ansätze zur Umsetzung der Mandantenfähigkeit aufweisen. Für eine „harte“Mandantenfähigkeit kamen Kamaji und vCluster in die engere Wahl. Während vCluster trotz seiner effizienten und ressourcenschonenden Architektur auf den ersten Blick vielversprechend erschien, wurde aufgrund der eingeschränkten Integration von GitOps und der komplexeren Architektur und den damit verbundenen Problemen wie der Noisy Neighbor Problematik unter Last auf den Einsatz des Frameworks verzichtet.

Kamaji hingegen erwies sich als vielversprechende Lösung zur Umsetzung der geforderten „harten“Mandantenfähigkeit. Durch die vollständige Isolation und den Aufbau eigener Cluster ist eine strikte und saubere Trennung der Ressourcen gewährleistet, die auch im großen Maßstab wartbar und performant bleibt, da jeder Tenant auf eigene Worker-Nodes setzt und somit einige Ressourcen-, Datenschutz- und Isolationsprobleme bereits gelöst sind. Auch die Integration von GitOps ist optimal gelöst und ermöglicht eine Integration mit FluxCD. Für die weitere Umsetzung der Kubernetes-as-a-Service Plattform wird deshalb Kamaji als Basisframework für die Private Cloud eingesetzt.

Für die Umsetzung der „weichen“ Mandantenfähigkeit könnte in Theorie vCluster oder Capsule eingesetzt werden, da beide Lösungen die Ressourcen des Hostsystems nutzen, würde es aus Ressourcensicht keine Rolle spielen. Allerdings kann vCluster auch hier nicht überzeugen, da Probleme wie GitOps-Integration und RBAC-Mechanismen zur Authentifizierung bei Capsule deutlich besser gelöst sind und insgesamt eine rundere Oberfläche für die Administration und auch die Nutzung darstellen. Für die Umsetzung der Dev-Cluster innerhalb der Team-Cluster wird daher auf Capsule gesetzt.

5.1.3 Container Network Interface (CNI) und Load Balancing

Problemstellung

Um innerhalb der Cluster lokale Netzwerke aufzubauen wird ein Container Network Interface (CNI) benötigt. Hierzu gibt es zahlreiche kostenfreie Lösungen wie Cillium, Weave oder Calico (vgl. CNCF 2024). Während für die meisten Kubernetes Cluster jede dieser Lösungen bereits ausreichend wäre, da in den meisten Fällen der Standard Funktionsumfang wie Netzwerkrichtlinien, Netzwerke und Routing von all diesen Lösungen unterstützt wird, so gibt es bei dieser Plattform einige extra Anforderungen, welche eine spezifische Auswahl nötig macht. Da es sich bei der Plattform um eine Bare Metall integration also lokale physische Server handelt, müssen Themen wie Load Balancing für den API-Zugriff aufgegriffen werden. Da die Control Plane aus mehreren Nodes besteht, muss durch Load Balancing dafür gesorgt weren, dass es nur einen zentralen Zugriffspunkt für den Endnutzer in das Cluster gibt. Aufgrund der Komplexität eines CNI in Kubernetes wird hier nur auf die benötigten Funktionen näher eingegangen.

Es wird zwischen sogenannten Layer 2 (L2) Load Balancern und Border Gateway Protocol (BGP) Load Balancern unterschieden. Beide sind grundsätzlich unterschiedlich aufgebaut und verwenden unterschiedliche Protokolle. Ersterer basiert auf ARP-Abfragen auf der Netzwerkschicht Layer 2, während letzterer BGP-Routing verwendet. Der Vorteil von L2-LB ist die einfache und unkomplizierte Einrichtung sowie die sehr gute Kompatibilität mit jeglicher Netzwerkhardware. Bei BGP wird ein BGP-fähiger Router benötigt, was bei wenigen physischen Servern zu einem deutlichen Mehraufwand in der Administration führt.

Um die optimale Lösung zu finden, wurde eine Nutzwertanalyse zwischen den beiden Open Source Tools Cilium und MetalLB durchgeführt. Cilium wurde ausgewählt, da es von der CNCF als CNI empfohlen wird und seit kurzem auch L2 und BGP Load Balancer integriert mitliefert. MetalLB ist nur ein Load Balancer, der zusätzlich ein CNI benötigt (vgl. MetalLB 2024). Für den Vergleich werden daher nur die Load Balancing Eigenschaften miteinander verglichen. Die Gewichtung der Kriterien in Tabelle 5 wurde nach Wichtigkeit und Funktionsumfang angepasst.

Fazit

Die Gesamtpunktzahl zeigt, dass Cilium mit 835 Punkten eine umfassende Lösung insbesondere in den Bereichen Netzwerksicherheit und Load Balancing bietet, während MetalLB mit 770 Punkten eine spezialisierte Lösung für reines Load Balancing in Bare-Metal-Umgebungen darstellt. Ein wesentlicher Faktor für die Entscheidung zugunsten von Cilium war die überlegene Netzwerksicherheit. Cilium bietet eine tiefere Integration von Sicherheitsfunktionen durch eBPF (extended Berkeley Packet Filter), das eine granulare Kontrolle des Netzwerkverkehrs ermöglicht und Sicherheitsrichtlinien direkt im Kernel durchsetzt. Diese Fähigkeit minimiert die Angriffsfläche und bietet einen besseren Schutz gegen moderne Bedrohungen als herkömmliche

Ansätze, die MetalLB nicht in der gleichen Tiefe bietet. Da MetalLB nur in Kombination mit einem CNI wie Cilium eingesetzt werden kann, wurde die Entscheidung getroffen, Cilium sowohl als CNI als auch als Load Balancer einzusetzen, der aufgrund der in Kapitel 5.2 beschriebenen Testinfrastruktur voraussichtlich als L2 Load Balancer fungieren wird. Darüber hinaus reduziert die Verwendung eines einzigen Tools die Komplexität und den Wartungsaufwand für die Plattformadministratoren.

		Cilium		**MetalLB**	
Kriterium	**Gf (%)**	**Zf**	**TN**	**Zf**	**TN**
Load Balancing (L2)	30	8	240	9	270
Netzwerk-Sicherheit	25	10	250	5	125
BGP-Unterstützung	15	7	105	9	135
Skalierbarkeit	10	9	90	7	70
Einfache Integration	10	7	70	9	90
Community und Support	10	8	80	8	80
SUMME	100		**835**		**770**

Tabelle 5: *Vergleich von Cilium und MetalLB für Load Balancing*

5.1.4 Container Storage Interface (CSI)

Problemstellung

Die Wahl des richtigen Container Storage Interface (CSI) spielt bei dieser Kubernetes-Plattform eine wichtige Rolle. Da die Plattform später hunderte von Persistent Volumes verwalten wird, muss genau überlegt werden, welches CSI die Anforderungen an Backup, Verfügbarkeit, Disaster Recovery und Datenschutz erfüllt. Während dies in Managed Kubernetes Umgebungen wie AWS keine Rolle spielt, da der Storage von AWS zur Verfügung gestellt wird, muss dies bei On-Premise Lösungen selbst übernommen werden. Ohne den entsprechenden CSI wäre es nicht möglich, den Tenants eine StorageClass anzubieten, mit der PersistentVolumes dynamisch erzeugt werden können. Durch die Verwendung von Kamaji ergibt sich ein Sonderfall bei der Auswahl des CSI. Wie in Kapitel 5.1.2 beschrieben, benötigt jedes Team seine eigenen Worker Nodes, die ihren eigenen Speicher über interne Festplatten mitbringen. Theoretisch müsste also jeder Team-Cluster einen eigenen CSI installieren, der dann entweder die Speicherkapazität des Teams nutzt und als StorageClass zur Verfügung stellt und/oder einen externen Datenspeicher einbindet, der dann ebenfalls als StorageClass zur Verfügung steht. Hier gilt es eine Lösung zu finden, die durch eine schlanke Implementierung das Beste herausholt. Dabei ist vor allem auf die GitOps-Integration, den Konfigurationsaufwand und die Zuverlässigkeit zu achten, damit auch der Wartungsaufwand in diesem Bereich möglichst gering ist. Mit in den Vergleich fließen die Anforderungen FA10, FA18 und FA25.

Die aktuelle Infrastruktur verwendet als Speicherlösung einen Ceph-Cluster, der mit Hilfe von Rook Kubernetes-kompatibel gemacht wurde. Da dieser jedoch relativ aufwendig in der Konfi-

guration und Wartung ist, wird nach einer besseren Alternative gesucht, um die Verfügbarkeit des persistenten Speichers in den einzelnen Tenants zu optimieren. Im Folgenden werden LinSTOR und Rook Ceph beschrieben und evaluiert, um zu prüfen, ob LinSTOR besser geeignet ist als Rook Ceph. LinSTOR wurde wegen seiner Architektur ausgewählt. Diese und andere Vorteile werden im Folgenden beschrieben.

LinSTOR

LinSTOR ist eine Speicherverwaltungslösung, die es ermöglicht, die physischen Festplatten eines Systems in ein resilientes verteiltes System umzuwandeln, um Speicher über mehrere Server durch Replikation zur Verfügung zu stellen. Dadurch können Ausfallsicherheit und Datensicherheit gewährleistet werden. Die Architektur wird in Abbildung 31 beschrieben. Im Zentrum steht der LINSTOR Controller, der als zentrale Verwaltungseinheit fungiert. Der Controller koordiniert und verwaltet alle Operationen und Ressourcen innerhalb des Clusters, indem er mit den LINSTOR-Satelliten kommuniziert. Diese Satelliten sind die Knoten im Cluster, die die eigentlichen Speichereinheiten verwalten. Jeder Satellit hat Zugriff auf lokale Speichermedien wie NVMe, SSD und HDD und führt die vom Controller angeforderten Speicheroperationen aus. Die Partitionierung des physikalischen Speichers erfolgt mit Logical Volume Manager (LVM) oder ZFS. Eine zentrale Rolle spielt dabei Distributed Replicated Block Device (DRBD), das die Spiegelung von Block-Devices zwischen verschiedenen Knoten über ein RAID-1-ähnliches Verfahren ermöglicht. Jeder Knoten erhält eine vollständige Kopie der Daten und speichert die Metadaten von DRBD. Dies ermöglicht den Zugriff auf die Daten auf den LVM- oder ZFS-Partitionen auch dann, wenn alle LINSTOR- und DRBD-Komponenten entfernt wurden oder ausgefallen sind (vgl. Kereczman 2024).

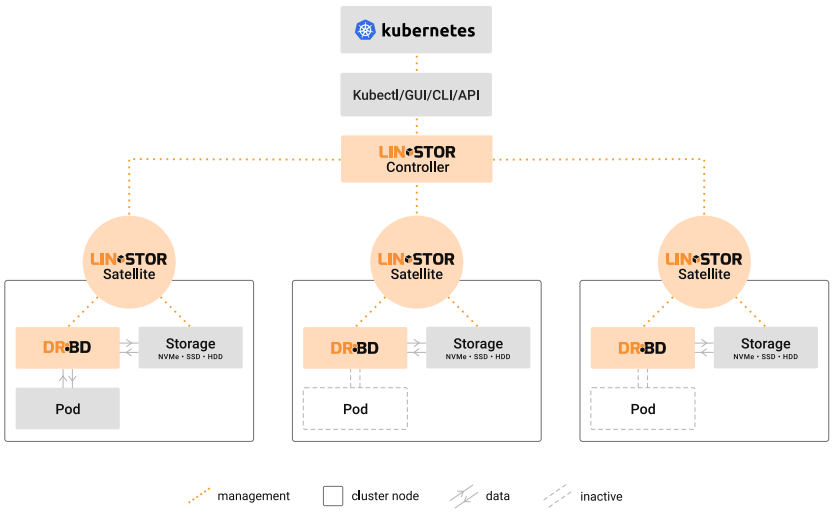

Abbildung 31: *Übersicht über die LinSTOR Architektur (vgl.* LINBIT *2024)*

Mit Hilfe eines zusätzlichen Operators namens „Piraeus Operator“ (siehe Piraeus 2024) kann LinSTOR nahtlos in Kubernetes integriert werden, was die Verwaltung von Speicherressour-

cen über kubectl, GUI, CLI oder API ermöglicht. Dies vereinfacht die Bereitstellung und Verwaltung von persistenten Speicherressourcen innerhalb eines Kubernetes-Clusters erheblich. Durch den Operator kann die Konfiguration des Controllers über GitOps automatisiert werden, da der zusätzliche Operator eigene CRDs zur Konfiguration des Controllers bereitstellt. Generell bietet LinSTOR folgende Vor- und Nachteile, die später für das Fazit relevant sind:

Vorteile

- Datenwiederherstellung ist selbst beim Totalausfall relativ einfach möglich
- Replikation des Speichers ist durch das RAID-1 ähnliche Verfahren ohne komplexe Berechnungen möglich und damit schneller
- Flexibles Dateisystem wie LVM oder ZFS
- Replikation durch DRBD ist bewährt und garantiert eine Verfügbarkeit von Daten

Nachteile

- Die Skalierung durch Hinzufügen von Replikaten ist speicherintensiv
- Eingeschränkter gleichzeitiger Zugriff, normalerweise nur von einem Knoten aus les- und schreibbar

Rook Ceph

Ceph ist eine Speicherlösung, die auf dem Objektspeichersystem RADOS basiert und häufig in großen verteilten Systemen eingesetzt wird. Zentraler Bestandteil der Architektur ist der CRUSH-Algorithmus, der die gespeicherten Daten pseudo-zufällig über den gesamten Ceph-Cluster verteilt. Dadurch wird Skalierbarkeit durch effiziente Nutzung des physischen Speichers ermöglicht. Im Vergleich zur RAID-1-ähnlichen Replikation von LinSTOR reduziert dies den Bedarf an redundantem Speicher. Außerdem ist das Satellitenprinzip bei Ceph nicht möglich. Der Ceph-Controller läuft innerhalb des Kubernetes-Clusters als eine große Anwendung, die auf mindestens drei Knoten verteilt sein sollte, um Hochverfügbarkeit zu gewährleisten. Da nicht jeder Knoten eine Kopie aller Daten enthält, ist dies im Kontext der Mandantenfähigkeit mit den Teamclustern eher von Nachteil, da somit jeder Tenant mindestens drei Worker Nodes benötigt (vgl. Rook 2024). Steht also kein extern gemanagter Ceph-Cluster zur Verfügung, muss jeder Tenant seinen eigenen Ceph-Cluster erzeugen, da durch Kamaji eine „harte“ Mandantenfähigkeit besteht und keine Kommunikation zum Management Cluster möglich ist. Dieser Ressourcen-Overhead muss bei der Auswahl der Speicherlösung berücksichtigt werden.

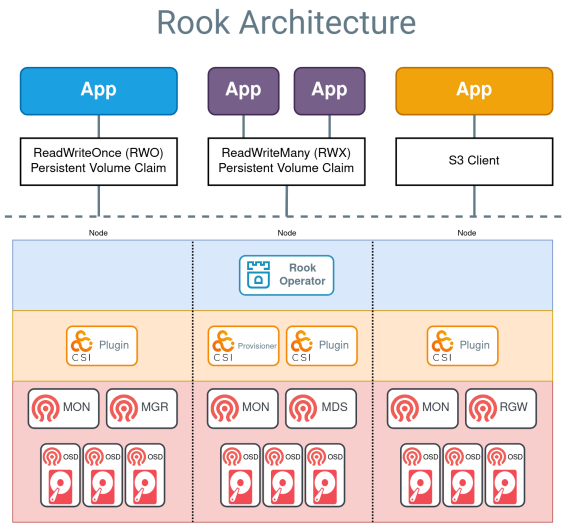

Abbildung 32: *Übersicht über die Rook-Ceph Architektur (vgl.* Rook *2024)*

In Abbildung 32 ist die Architektur von Ceph inklusive Rook zu sehen. Rook übernimmt dabei die Verwaltungsebene, indem es die Automatisierung und Orchestrierung der Speicherressourcen übernimmt. Die gesamte Konfiguration und Verwaltung des Ceph-Clusters wird von Rook übernommen. Der Ceph Cluster selbst läuft dabei in Pods innerhalb des Kubernetes Clusters. Die Verwaltung der StorageClass, die von den Workloads genutzt wird, wird ebenfalls von Rook übernommen. Rook dient quasi als Wrapper um den Ceph-Cluster. Dadurch entstehen hauptsächlich Administrative Vorteile.

Vorteile

- Effizientere Nutzung des physischen Speichers durch den CRUSH-Algorithmus
- Gleichzeitiger Zugriff von mehreren Knoten möglich
- Hohe Skalierbarkeit durch verteilte Datenhaltung
- Kontinuierliches Monitoring und Management des Ceph-Clusters durch Rook für hohe Verfügbarkeit und Ausfallsicherheit

Nachteile

- Ressourcenintensiv und weniger flexibel
- Datenwiederherstellung ohne laufendes Ceph-System schwierig
- Weniger geeignet für Anwendungen mit häufigen kleinen Schreiboperationen
- Erheblicher Aufwand bei der Datenverteilung, was sich auf Latenz und Durchsatz auswirkt

Fazit

Aufgrund der genannten Vor- und Nachteile der beiden Speicherlösungen fiel die Entscheidung nach sorgfältiger Abwägung und unter Berücksichtigung der funktionalen Anforderungen auf LinSTOR. Durch die Besonderheit der Mandantenfähigkeit und der Prämisse, dass kein externer Ceph-Cluster zur Integration zur Verfügung steht, bietet LinSTOR durch die einfache Konfiguration und Architektur mehr Vorteile. Durch die freie Wahl des Dateisystems und die einfache Replikation in einem RAID-1-ähnlichen Verfahren kann LinSTOR besser in die einzelnen Team-Cluster integriert werden, da theoretisch auch nur ein starker Worker Node einen LinSTOR-Cluster bilden kann und im Falle eines Totalausfalls die Daten sicher wären. Die Datenwiederherstellung ist aufgrund der Architektur mit wenig Linux-Grundkenntnissen möglich. Dies ist wichtig, da sich in den Dev-Clustern theoretisch nicht gepushter Quellcode oder andere wichtige Daten befinden könnten, die eine schnelle Wiederherstellung erfordern. Außerdem profitiert LinSTOR von niedrigen Schreiblatenzen, da die Speicherung weniger komplex ist als bei Ceph. Dies ist wichtig, da in den Team-Clustern wahrscheinlich Datenbanken gehostet werden, die einen schnellen persistenten Speicher benötigen. Bei beiden Lösungen ist es möglich, sowohl ReadWriteOnce als auch ReadWriteMany als Zugriffsmodus zu definieren. Dies ist für bestimmte Workloads notwendig, damit mehrere Pods auf ein PV gleichzeitig zugreifen können. Der Nachteil der ineffizienten Speichernutzung bei LinSTOR wurde weniger stark gewichtet, da davon auszugehen ist, dass ein einzelner Team-Cluster nur wenige hundert Gigabyte benötigt, die bei Bedarf relativ kostengünstig erweitert werden können. Sollte jedoch ein externer Datenspeicher angeschafft werden, auf den mehrere Tenants zugreifen, ist Rook-Ceph in einem dedizierten Cluster besser geeignet. Diese Option wird im Rahmen dieser Arbeit jedoch nicht weiter vertieft.

5.1.5 Cloud-basierte Softwareentwicklung

Problemstellung

Für die einzelnen Dev-Cluster wird eine Lösung benötigt, um innerhalb dieser Cluster eine Cloud-basierte Softwareentwicklungsumgebung zu starten. Die einzelnen Nutzer sollen sich im Idealfall mit ihrer lokal installierten Entwicklungsumgebung mit ihrer entfernten Entwicklungsumgebung verbinden können, um dort in Echtzeit zu entwickeln und die Anwendungen mit den Ressourcen des Clusters zu starten und zu testen. Die gewählte Lösung muss die Anforderungen FA12, FA13 und FA16 erfüllen. Nach einer ausführlichen Internetrecherche wurden die beiden Open Source Frameworks DevPod und DevSpace von Loft für den Vergleich ausgewählt. Diese eigneten sich auf den ersten Blick optimal für den verwendeten Anwendungsfall. Durch den Vergleich soll die beste Lösung für die gestellten Anforderungen gefunden werden.

DevPod

Mit DevPod können über Provider reproduzierbare Entwicklungsumgebungen auf unterschiedlichen Zielinfrastrukturen erstellt werden. Dabei verfügt jede Umgebung über einen isolierten Container, der über den Devontainer-Standard mittels einer „devontainer.json“ spezifiziert wird. DevPod selbst läuft als CLI oder GUI Anwendung auf dem lokalen System und verbindet sich über die lokale kubeconfig mit dem Zielsystem. Durch den Container-Ansatz ist es also völlig egal, auf welchem Zielsystem entwickelt wird, da es sowohl lokal über Docker Desktop als auch remote in einem Kubernetes-Cluster gestartet werden kann. Die Entwickler selbst bezeichnen DevPod als eine Art Bindeglied zwischen der lokalen IDE und der entfernten Maschine, die für die Entwicklung genutzt werden soll. Folgende Vorteile ergeben sich aus der Dokumentation (vgl. Loft 2024b)

- Bedienung mittels grafischer Oberfläche (GUI) oder CLI möglich
- Keine Installation im Cluster notwendig, da von DevPod übernommen
- Vielfältige IDE-Unterstützung wie VSCode, JetBrains und ggf. SSH
- Lokale und entfernte Entwicklung weiterhin möglich
- Verwendet den Open-Source-Standard Devcontainer
- Beliebiger Zielprovider möglich wie on-premise Kubernetes oder Public Cloud Provider wie AWS

DevSpace

DevSpace ist ein Client-seitiges Open-Source-Entwicklungswerkzeug für Kubernetes, mit dem Anwendungen direkt in Kubernetes erstellt, getestet und debuggt werden können. Im Gegensatz zu DevPod erstellt DevSpace nur für das aktuelle Projekt eine Remote-Umgebung, in der der Quellcode kontinuierlich synchronisiert und zum Testen gestartet wird. Darüber hinaus automatisiert es repetitive Aufgaben wie das Erstellen von Images und das Deployment. Dieser Anwendungsfall ist sicherlich vorteilhaft, aber für diesen Kontext weniger geeignet. Bei DevPod werden beispielsweise komplette Umgebungen auf Basis von z. B. Debian erzeugt, in denen alle Programmiersprachen und Tools automatisch mitinstalliert werden. Bei DevSpace beschränkt sich dies auf die aktuelle Anwendung. Außerdem bleibt der Quellcode lokal, so dass ein Gerätewechsel nicht so einfach wie bei DevPod möglich ist. Ebenso kann DevSpace nur über die CLI bedient werden, was die Lernkurve gerade am Anfang steiler macht. Für das abschließende Fazit konnten folgende Vorteile in der Dokumentation von DevSpace gefunden werden (vgl. Loft 2024c):

- Hot Reloading: Aktualisierung laufender Container ohne Neuaufbau von Images oder Neustart der Container
- Vereinheitlichung der Deployment-Workflows für Teams über verschiedene Umgebungen hinweg

- Speichern aller Workflows in einer deklarativen Konfigurationsdatei: devspace.yaml
- Automatisierung repetitiver Aufgaben wie Image-Erstellung und Deployment
- Keine serverseitige Komponente notwendig, läuft direkt als CLI-Tool
- Hohe Performance durch bi-direktionale Dateisynchronisation zwischen lokaler Entwicklungsumgebung und Kubernetes-Containern

Fazit

Die auf den ersten Blick ähnlichen Frameworks unterscheiden sich bei näherer Betrachtung deutlich voneinander. Aufgrund der in der Problemstellung genannten Anforderungen fiel die Wahl für die weitere Implementierung auf DevPod. Dieses stellt aufgrund der Vor- und Nachteile die nahezu perfekte Lösung für den gewünschten Anwendungsfall dar und kann durch die Trennung von Konfiguration und Quellcode von den Plattformadministratoren optimal vorkonfiguriert werden, so dass lediglich die Provider heruntergeladen werden müssen und jeder Nutzer der Plattform frei entscheiden kann, welche Umgebung mit welchem Repository und welcher Konfiguration in seinem Dev-Cluster gestartet werden soll. Zusätzlich wird für jede Umgebung ein Persistent Volume angelegt, in dem der Quellcode gespeichert wird. Dadurch geht nicht gespeicherte Arbeit bei einem Neustart nicht verloren. Durch die einfache Bedienung mittels einer interaktiven GUI wird zudem eine schnelle Adaption der Kubernetes-Plattform erwartet, da auch Benutzer mit wenig Erfahrung im Devcontainer-Bereich die Anwendung nutzen können. Die Adaption von DevPod ist unter Kapitel 5.5 zu finden.

5.2 Realisierung der on-premise Testinfrastruktur

Für die Realisierung der Kubernetes-Plattform wird zunächst die passende Hardware sowie die passende Netzwerkinfrastruktur benötigt. Diese ist nachfolgend beschrieben.

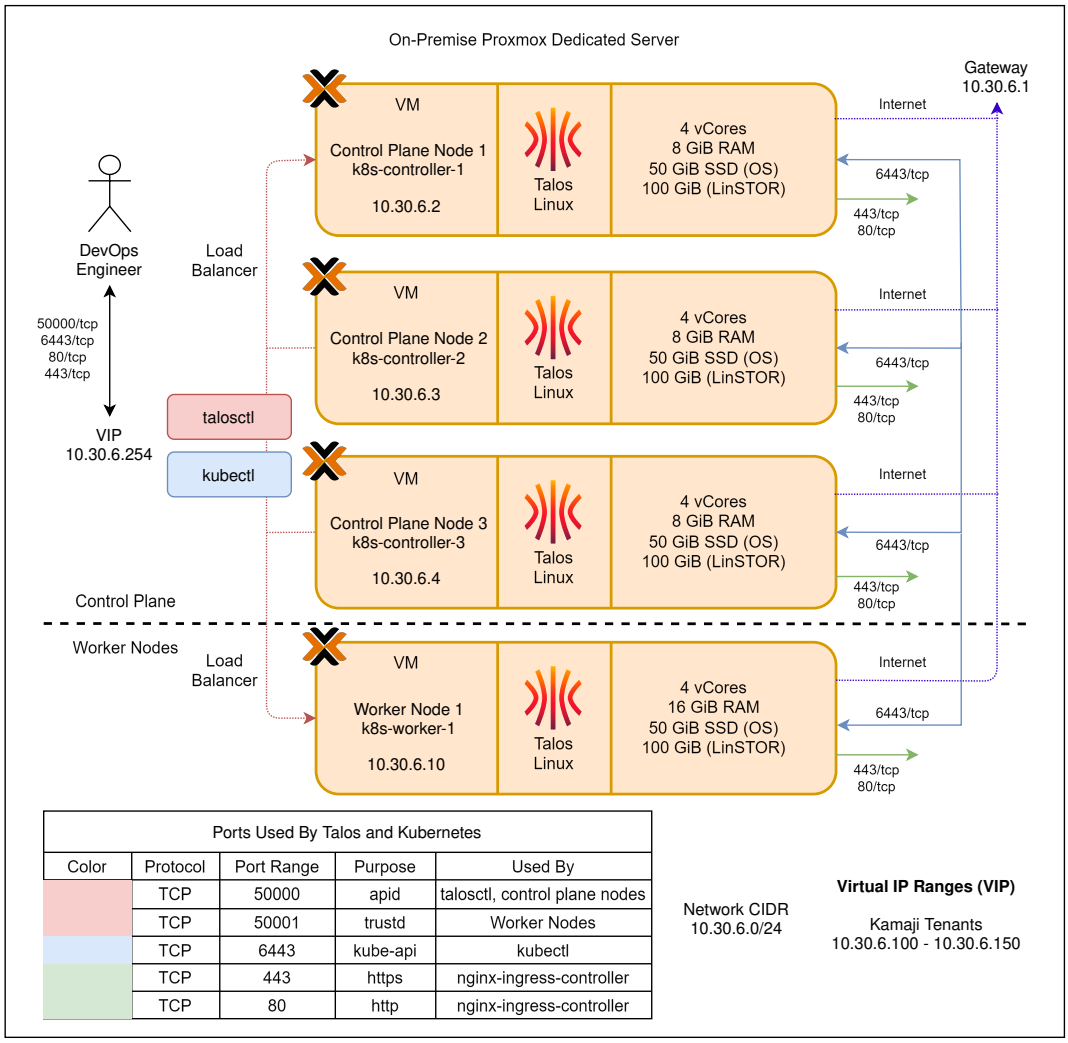

Abbildung 33: *Übersicht der on-premise Kubernetes Infrastruktur (eigene Darstellung)*

Virtuelle Maschinen

Auf den Einsatz von Bare-Metal-Servern für die Control Plane sowie die Worker Nodes wird aus Kostengründen verzichtet. Die gesamte Testinfrastruktur basiert auf virtuellen Maschinen, die auf einem dedizierten Proxmox Server des Autors gehostet werden. In Abbildung 33 ist der Aufbau grafisch dargestellt. Da es sich um eine Testinfrastruktur handelt, wurden die virtuellen Maschinen nach den offiziellen Hardwareempfehlungen von Talos Linux und Kubernetes konfiguriert (vgl. Sidero 2024c). Neben der Systemfestplatte erhält jeder Node zusätzlich eine emulierte 100 GiB SSD, die für den LinSTOR-Satelliten benötigt wird. Diese ist auf allen Nodes gleich groß, um eine 1:1 Replikation der Daten zu ermöglichen. Als Be-

triebssystem wird, wie in Kapitel 5.1.1 beschrieben, Talos Linux verwendet, welches als ISO per CD/ROM an die virtuelle Maschine angehängt wird. Die Bootpriorität wurde so gewählt, dass im ersten Schritt immer von der Systemfestplatte gebootet wird. Falls dort kein Betriebssystem installiert ist, wird das Talos Linux ISO gestartet, welches automatisch in den Wartungsmodus bootet. So konnte die VM automatisch über ihre IP konfiguriert werden. Alle notwendigen Kernelmodule, die für den virtuellen Betrieb von Proxmox benötigt werden, wie z. B. der QEMU Guest Agent für ACPI-Kommandos wie Reboot, Shutdown, etc. wurden der ISO hinzugefügt. Die Konfiguration des Fleet-Repositories welches für die Talos Linux Konfigurationsdateien verwendet wird ist in Kapitel 5.3.2 zu finden.

Netzwerk

Da es sich um eine On-Premise-Implementierung handelt, müssen alle Netzwerkkonfigurationen im Vorfeld durchgeführt werden, damit die automatische Installation von Talos Linux reibungslos funktioniert und nach Abschluss der Installation ein Zugriff auf den Kubernetes-Cluster möglich ist. Zu Beginn wurde jeder VM eine eigene virtuelle Netzwerkschnittstelle hinzugefügt, die sich im Netzwerk „10.30.6.0/24“ befindet. Dies ist ein dediziertes Netzwerk nur für diese Kubernetes-Plattform, damit die Server miteinander kommunizieren können. Auf eine Firewall innerhalb dieses Netzwerkes wurde aus Zeit- und Komplexitätsgründen verzichtet. Alle Geräte können ungehindert miteinander kommunizieren. Ebenso können alle Knoten über das Gateway 10.30.6.1/32 das Internet erreichen. Damit der abgebildete DevOps Engineer über die „talosctl“ mit Talos kommunizieren bzw. später über die „kubectl“ auf den Cluster zugreifen kann, müssen die in der Tabelle dargestellten Ports geöffnet sein. Über die Talos-Konfiguration wird außerdem die virtuelle IP-Adresse (VIP) „10.30.6.254“ als Loadbalancer-IP für den Kube-API-Server festgelegt. Über diese kann später auf Port 6443 mit dem Cluster kommuniziert werden. Zusätzlich wurde der IP-Bereich 10.30.6.100 bis 10.30.6.150 als VIP-Bereich für die einzelnen Tenants von Kamaji definiert. Jeder Tenant erhält eine dedizierte IP-Adresse aus diesem Bereich, über die dann auf den Team-Cluster zugegriffen werden kann. Die Nodes können auch über HTTP und HTTPS erreicht werden. Dies ist später für den NGINX Ingress Controller wichtig, um später innerhalb der einzelnen Cluster eine IngressClass zu erstellen.

Namenskonvention für Nodes

Jeder Server erhält einen einzigartigen Hostnamen, welcher später in der Talos Konfiguration eingetragen wird. Die Konvention ist vorallem später im Produktivbetrieb wichtig, um eine klare und skalierbare Infrastruktur aufbauen zu können, welche auch noch beim Einsatz von mehreren hundert Servern funktioniert. Es wird folgendes Schema definiert:

[DOMAIN]-[CONTINENT]-[REGION]-[NODE_TYPE]-[NODE_NUMBER]

Beispiel für einen Worker Node: **itd-eu-central-1-worker-01**

5.3 Systemkonzeption und Realisierung des Management Clusters (Private Cloud)

In diesem Kapitel werden alle wichtigen Maßnahmen zur Umsetzung des Management Clusters bzw. der Private Cloud beschrieben. Zunächst wird in Kapitel 5.3.1 der Aufbau der Private Cloud beschrieben. Anschließend wird in Kapitel 5.3.2 und 5.3.3 die Integration von GitOps für Talos Linux und Kamaji erläutert. Außerdem wird die Integration von LinSTOR als Standard-StorageClass in Kapitel 5.3.4 beschrieben, die das Kamaji-Framework benötigt. Abschließend wird ein Lösungsansatz zur automatischen Erzeugung von SSL-Zertifikaten für die Team-Cluster in Kapitel 5.3.5 vorgestellt.

5.3.1 Übersicht der Systemkomponenten

In Abbildung 34 ist eine Übersicht der einzelnen Systemkomponenten der Private Cloud bzw. des Management Clusters zu sehen. Diese Komponenten werden benötigt, um das in Kapitel 4.2.1 beschriebene Minimum Viable Product (MVP) zu implementieren und die Anforderungen von GitOps sowie die Isolationskriterien zu erfüllen. Dabei wird FluxCD als Kernkomponente eingesetzt, um Konfigurationen und Tenants in Echtzeit aus dem entsprechenden Git-Repository (siehe Kapitel 5.3.3) ins Cluster zu synchronisieren. Darüber hinaus verwaltet der Cluster einen zentralen LinSTOR-Cluster, der für die persistente Speicherung von Kamaji für die einzelnen etcd-Datenbanken benötigt wird. Dessen Konfiguration ist in Kapitel 5.3.4 beschrieben. Um mögliche Webapplikationen für Monitoring etc. zur Verfügung zu stellen, wurde zusätzlich ein NGINX Ingress Controller inklusive eines Cert Managers installiert, der sich um die in Kapitel 5.3.5 beschriebene SSL-Zertifikatserstellung sowie die Zertifikatsverwaltung von Kamaji kümmert.

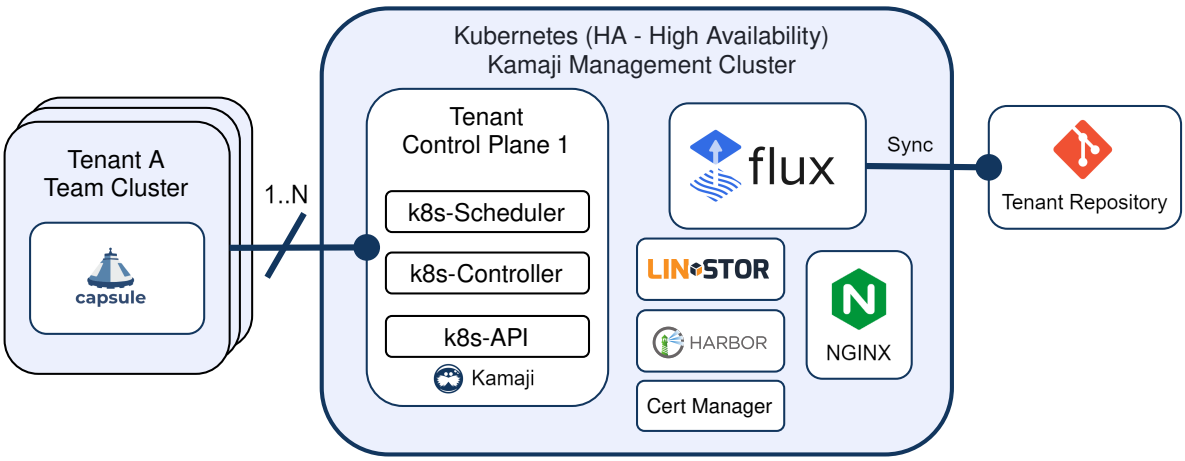

Abbildung 34: *Übersicht der Systemkomponenten des Management Clusters (eigene Darstellung)*

5.3.2 GitOps: Aufbau des Fleet-Repositories für Talos Linux

Um die Anforderung FA19 zu erfüllen, wird ein Git-Repository benötigt, das alle Konfigurationen von Talos Linux enthält, so dass die Plattformadministratoren die physischen Server über die Talos-CLI „talosctl“ verwalten können. Der Autor empfiehlt die Verwaltung einer Private Cloud pro Repository Branch, da es sonst zu unübersichtlich wird. Der Zweig sollte einen eindeutigen Namen haben. Das Besondere an Talos ist, dass neben der Serverkonfiguration auch die Kubernetes-Konfiguration enthalten ist, da beim Bootstrapping der einzelnen Talos-Knoten automatisch Worker- und Control-Plane-Knoten erzeugt werden, die dann einen vollwertigen Kubernetes-Cluster bilden. Da sich jeder Talos Node beim ersten Boot im Maintenance Mode befindet, wird erst beim Bootstrap entschieden, welche Rolle die einzelnen Nodes erhalten. Dabei wird die Maschinenkonfiguration per CLI an die jeweilige IP-Adresse übermittelt. Die Strukturierung der einzelnen YAML-Dateien für Node- und Kubernetes-Konfigurationen erfolgt nach einem festen, von Talos vorgegebenen System. Die Anordnung in Ordnern ist jedoch frei wählbar. Dieser Vorteil wird genutzt, um wie in Abbildung 35 ein Fleet-Repository aufzubauen, das auch für mehrere Private Clouds funktioniert.

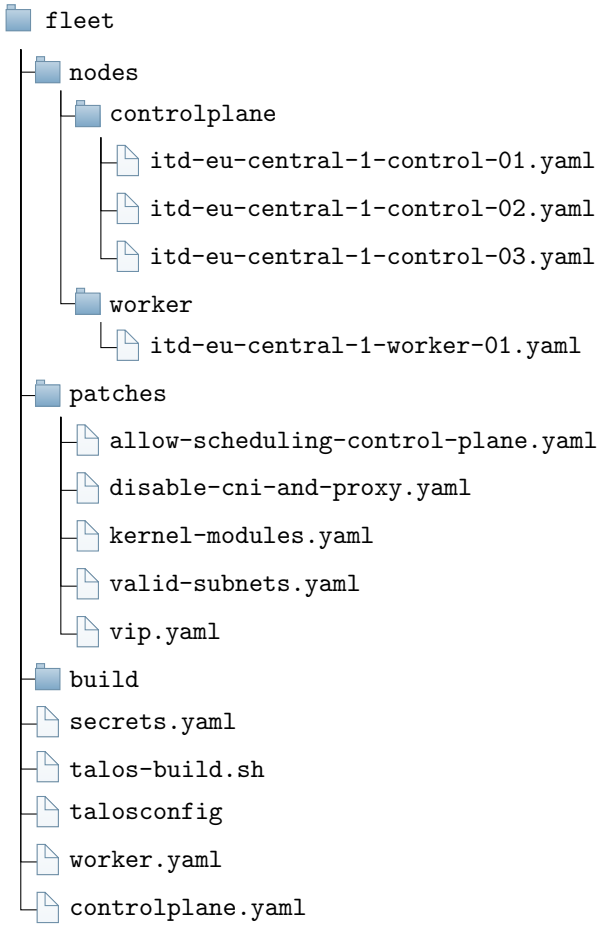

Abbildung 35: *Dateistruktur des Fleet-Repositories*

Talos erstellt die folgenden Basisdateien mit Hilfe von talosctl mit einem Initialbefehl „talosctl gen config“:

- **talosconfig**: Speichert die Zugangs- und Verbindungsdaten, um sich nach dem Bootstrapping der Talos-Nodes an der Talos-API der jeweiligen Nodes zu authentifizieren.
- **secrets.yaml**: Listet alle öffentlichen und privaten Schlüssel der jeweiligen Kubernetes-Zertifikate auf. Damit können mehrere Cluster mit den gleichen Zertifikaten erstellt werden. Sinnvoll für eigene Zertifikate.
- **worker.yaml**: Beispielhafte Maschinenkonfiguration für Worker Nodes, die Informationen wie Hostnamen, Kernelmodule, Zertifikate etc. enthält.
- **controlplane.yaml**: Beispiel einer Maschinenkonfiguration für Control Plane Nodes, die Informationen wie Hostnamen, Kernel-Module, Zertifikate etc. enthält.

Die Maschinenkonfigurationen müssen nun an den jeweiligen Anwendungsfall angepasst werden. Hierbei verfolgt Talos eine sogenannte Multi-Document-Merge-Strategie, die durch das Zusammenführen einzelner kleiner Patches wieder eine große Maschinenkonfiguration erzeugt (vgl. Sidero 2024a), da jeder Node nur mit einer einzigen Datei provisioniert wird. So kann z. B. die Konfiguration eines Nodes auf mehrere kleine Patchdateien aufgeteilt werden und später selektiv Patch X, Y, Z ausgewählt werden. Es wurde daher entschieden, einerseits globale Patches und andererseits Node-spezifische Patches zur Verfügung zu stellen. Die globalen Patches befinden sich im Ordner „patches“ und sollten keine Node-spezifischen Informationen wie IP-Adressen oder Node-Namen enthalten, da sie auf alle Nodes gleichzeitig angewendet werden. Außerdem sollten diese nach dem Single-Responsibility-Prinzip aufgebaut und benannt werden. Die Node-spezifischen Patches befinden sich im Ordner „nodes“, der wiederum in Worker- und Control-Plane-Nodes unterteilt ist. In diesen YAML-Dateien, die nach dem Hostnamen des Nodes benannt sind, befinden sich nur Anpassungen, die beim Zusammenführen der Basis-Maschinenkonfiguration überschrieben werden.
In Listing 1 ist der Inhalt der Datei „itd-eu-central-1-control-01.yaml“ dargestellt.

```
machine:
  network:
    hostname: itd-eu-central-1-control-01
```

Listing 1: *Talos-Node-Patch zum überschreiben des Hostnamens*

Implementierung eines Helferskripts für Maschinenkonfigurationen

Das Zusammenführen von Konfigurationen wird normalerweise von talosctl übernommen, bevor diese an die API gesendet werden. Dies erfordert jedoch umständliche CLI-Befehle, bei denen jeder Patch mit dem Argument „–patch @patchName.yaml“ angegeben werden muss. Dies kann bei einer großen Anzahl von Patches schnell zeitaufwendig und unübersichtlich werden. Aus diesem Grund wurde ein Bash-Skript entwickelt, das die einzelnen globalen Patches mit den Node-spezifischen Patches und der Basismaschinenkonfiguration zusammenführt und im „build“-Ordner ablegt. Dabei wird die Dateistruktur des „nodes“-Ordners gespiegelt.

Diese YAML-Dateien können dann mit nur einem Befehl über talosctl an den entsprechenden Node gesendet werden. Das Skript befindet sich im Hauptverzeichnis unter dem Namen „talos-build.sh“. Der Autor erhofft sich von diesem Skript ein weniger fehleranfälliges und effizienteres Arbeiten bei Änderungen an den Talos-Nodes. Das Skript befindet sich im Anhang unter Listing 7.

5.3.3 GitOps: Aufbau des Cloud-Repositories für Kamaji

Das Cloud-Repository, das erstmals in Abbildung 34 vorgestellt wurde, dient einem ähnlichen Zweck wie das Fleet-Repository, mit dem Unterschied, dass es den Management Cluster und die Kamaji Tenants verwaltet. Dieses Repository erfüllt die Anforderung FA15. Die Dateistruktur ist im Abbildung 36 dargestellt. Im Folgenden werden die einzelnen Ordner und Dateien des Repositories erläutert und auf eventuelle Besonderheiten hingewiesen.
Das Verzeichnis „clusters/management“ enthält die Konfiguration von FluxCD, der darin enthaltene Ordner „flux-system“ wird von Flux selbst verwaltet und sollte nicht editiert werden. Die Dateien „infrastructure.yaml“ und „tenants.yaml“ bilden den Ausgangspunkt, der von Flux als erstes aufgerufen wird. Die darin enthaltenen Kustomizations referenzieren als Ressourcen den jeweiligen Ordner eine Ebene darüber. Da das Tool Kustomize im Zielordner automatisch nach einer „kustomization.yaml“ sucht, wurde diese entsprechend in jedem Ordner angelegt, welche die nächsten Referenzen enthält. Man kann sich das wie eine Baumstruktur vorstellen, in der jede kustomization.yaml einen Knoten bildet, der weitere Blätter in Form von Dateien referenziert. Im Ordner „infrastructure“ befinden sich Helm Charts welche automatisch von Flux im Management Cluster deployt und verwaltet werden. Dort befinden sich die Komponenten welche in Abbildung 34 dargestellt sind. Dort können nach belieben weitere Apps von den Plattformadministratoren hinzugefügt werden.

Im Ordner „tenants“ befinden sich die Konfigurationen für die einzelnen Team-Cluster sowie ein selbst entwickeltes generisches System von Shared Apps in Form von Helm Charts, die nach der Erstellung des Kamaji-Tenants über Flux automatisch vom Management Cluster installiert werden. Diese Shared Apps können von den Tenant Administratoren nicht gelöscht werden. Dies entspricht dem gewünschten „Top-Down“Ansatz aus Kapitel 4.2.1. Auch diese Anwendungen können beliebig erweitert oder verändert werden. Im Ordner „projects“ werden die Team-Cluster von den Administratoren der Plattform verwaltet. Die darin enthaltene Datei „kamaji-tenant.yaml“ enthält die Custom Resource Definition (CRD) von Kamaji mit allen möglichen Parametern wie Cluster-Name, Ressourcen-Verfügbarkeit, Netzwerk-Konfiguration, Kubernetes-Version etc. Eine besondere Rolle spielt die Datei „helmrelease-patch.yaml“, die zusammen mit der kustomization.yaml dafür sorgt, dass die Helm-Releases der Shared Apps für die Tenants überschrieben werden. Da Flux standardmäßig im eigenen Cluster deployen möchte, muss über diesen Patch der Zielcluster in Form einer kubeconfig angegeben werden. Da Kamaji diese als Secret im Management Cluster im Namespace „tenants“ anlegt, kann

damit auf den Team-Cluster zugegriffen werden. Flux überwacht ständig den Ist-Zustand aller Tenants und vergleicht diesen mit dem Soll-Zustand aus Git. Abgerundet wird das Ganze durch die „kamaji-tenant-tasks.yaml“, die dafür sorgt, dass verschiedene Aufgaben wie das Anlegen von Namespaces im Team-Cluster direkt nach dem Anlegen des Tenants erzeugt werden, da sonst die Helm Chart Installationen der Shared Apps fehlschlagen. Die Datei verwendet die Datei „namespaces.yaml“ als Referenz.

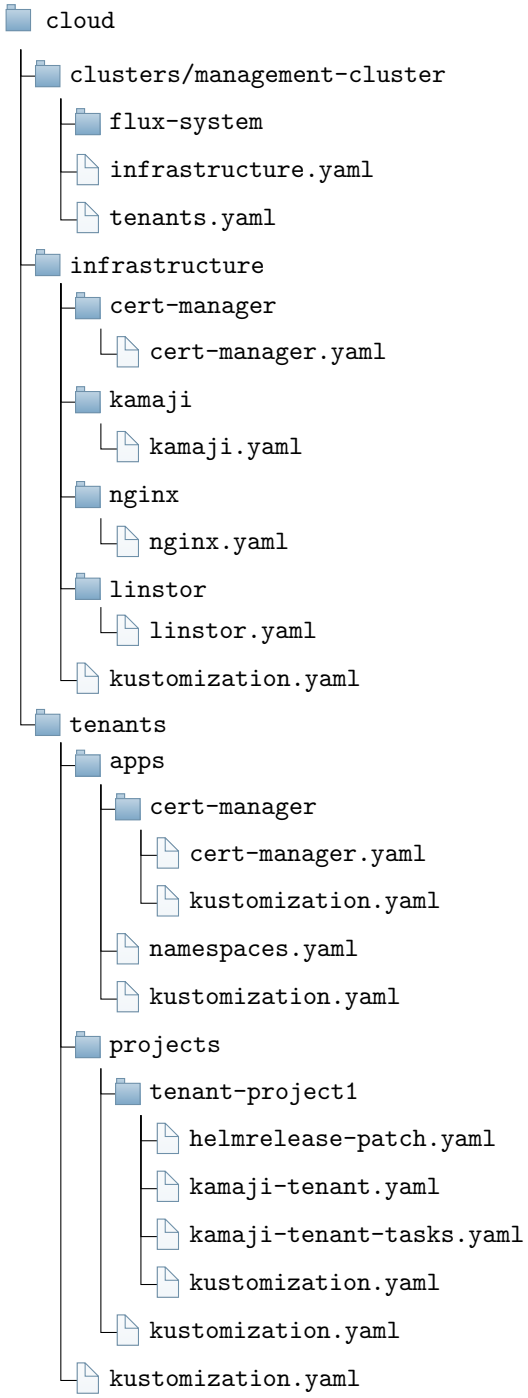

Abbildung 36: *Dateistruktur des Cloud-Repositories*

5.3.4 Integration von LinSTOR als CSI

LinSTOR wird von Piraeus mit dem entsprechenden Kubernetes-Operator über einen Helm Chart bereitgestellt. Da Talos Linux bereits die notwendigen Kernel-Module für die DRBD-Synchronisation mitbringt, ist die Erstkonfiguration in wenigen Minuten erledigt. Nachdem LinSTOR im Management Cluster deployt ist, kann über ein LinSTOR Add-On für die kubectl der Storage Pool mit dem Namen „ssd_pool_01“ manuell angelegt werden. Dieser verwendet als physikalischen Speicher die dedizierten LinSTOR SSDs wie in Kapitel 5.2 beschrieben. Da auf allen Control Plane Nodes eine SSD vorhanden ist, kann dadurch eine vollständige Replikation zwischen allen drei Nodes erreicht werden, so dass ein Ausfall von bis zu zwei Control Plane Nodes abgefangen werden kann. Im Cluster selbst wird der SSD-Pool über eine StorageClass bereitgestellt. Dabei wird der von Piraeus mitgelieferte Storage-Controller verwendet. In Listing 10 ist die Konfiguration mit Kommentaren dargestellt.

5.3.5 Automatische Generierung von Wildcard-Zertifikaten für Team-Cluster

Um die Anforderung FA10 zu erfüllen, benötigt jeder Team-Cluster eine eigene IngressClass, um über den NGINX Ingress Controller einen Ingress zu generieren, um Deployments über eine URL nach außen zu veröffentlichen. Der NGINX Webserver fungiert dabei als Reverse Proxy, der den eingehenden HTTP/HTTPS-Traffic auf den internen Deployment-Port (z. B. 8000) umleitet. Damit der NGINX korrekt arbeiten kann, benötigt er ein signiertes Wildcard SSL Zertifikat wie z. B. „*.tenant1.itd-intern.de“. Dieses erlaubt die beliebige Generierung weiterer Subdomains über die IngressClass. Das Problem ist nun, dass beim Anlegen eines Tenants dieses Zertifikat noch nicht generiert und signiert wurde. Der Administrator der Plattform könnte dies theoretisch im Vorfeld manuell erledigen und dann umständlich per Flux im Tenant Cluster als Secret deployen. Dies soll nun aber im Bootstrapping-Prozess bei der Erstellung eines Kamaji-Tenants automatisiert werden. Dabei müssen einige Sicherheitsregeln beachtet werden. Zum einen benötigt man zum Signieren eines Zertifikats ein (firmeninternes) Root-Zertifikat, welches im Idealfall bereits auf allen Geräten der Mitarbeiter installiert ist. Da der Private Key dieses Zertifikats mit äußerster Sorgfalt behandelt werden muss, kommt es nicht in Frage, dass der Signiervorgang in den Team-Clustern stattfindet oder ein Zugriff auf den Klartext Private-Key möglich ist.
Da der Zugriff auf den Management Cluster nur bestimmten Personen erlaubt ist, sollte es vertretbar sein, dass das Root-Zertifikat mit Public und Private Key als Secret im Management Cluster liegt. Mit Hilfe des Cert Managers kann nun über GitOps ein Trigger erstellt werden, der nach dem Anlegen eines Kamaji Tenants automatisch eine Zertifikatserstellung und Signierung mit dem Root-Zertifikat durchführt. Das resultierende Wildcard Server Zertifikat wird dann als Secret im „tenants“ Namespace gespeichert und kann per Flux in den entsprechenden Team-Cluster synchronisiert werden. Dort steht es dann dem NGINX Ingress Controller zur Verfügung. Der konkrete Ablauf ist in Abbildung 37 beschrieben. Nach der

Erstellung der Kubernetes Manifeste durch Flux überwacht der Cert Manager in festgelegten Intervallen das Ablaufdatum des jeweiligen Zertifikats und erneuert es entsprechend. Flux erkennt dabei den Drift im Team-Cluster und aktualisiert das Secret entsprechend.

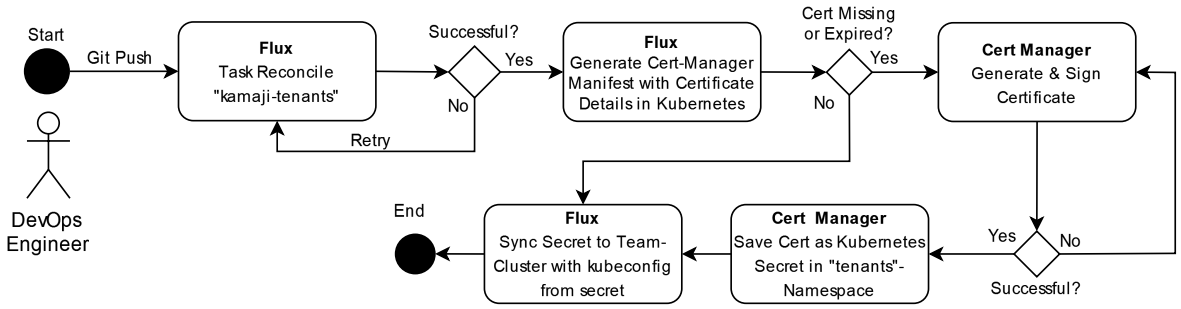

Abbildung 37: *Ablauf einer Zertifikaterstellung durch Flux und Cert Manager (eigene Darstellung)*

5.4 Systemkonzeption und Realisierung der Team-Cluster

In diesem Kapitel werden der Aufbau und die Umsetzung der Team-Cluster beschrieben. Der Schwerpunkt liegt dabei auf der Administration sowie einigen Besonderheiten bei der Implementierung. Zunächst werden die Systemkomponenten des Clusters in Kapitel 5.4.1 vorgestellt. Anschließend wird das Team-Repository in Kapitel 5.4.2 vorgestellt, über das der Cluster mit GitOps verwaltet werden kann.

5.4.1 Übersicht der Systemkomponenten

Unter Abbildung 38 sind die Systemkomponenten der mit Kamaji virtualisierten Team-Cluster zu sehen. Da sich diese nicht von herkömmlichen Kubernetes-Clustern unterscheiden, müssen auch hier einige Basiskomponenten wie CNI und CSI installiert werden, um sie lauffähig zu machen. Auch hier kommen Cilium und LiNSTOR zum Einsatz. Die Installation dieser ist identisch zum Management Cluster mit Flux. Die Punkte, in denen sich der Team-Cluster vom Management Cluster unterscheidet, liegen hauptsächlich in den verschiedenen Zugriffsebenen. Da mehrere Parteien wie der Gruppenleiter (Tenant-Admin) und die einzelnen Entwickler auf diesen Cluster zugreifen, unterteilt sich dieser einmal in den Plattformbereich sowie in den Tenant-Bereich. Der Zugriff auf den Plattformbereich ist dabei auf den Tenant-Admin beschränkt, der die Kontrolle über die Dev-Cluster mit Capsule sowie die Verwaltung der „Shared Apps“ hat, die über FluxCD mit z. B. Helm Charts deployt werden können.
Die eigene FluxCD-Instanz ist dabei mit dem in Kapitel 5.4.2 beschriebenen Team-Repository verbunden, das dediziert für diesen Team-Cluster erstellt wird. Ebenso bietet Capsule die Grundlage für die spätere Cloud-basierte Softwareentwicklung, die in Kapitel 5.5 beschrieben ist. Zusätzlich wird, wie in Kapitel 5.3.5 beschrieben, eine eigene NGINX Ingress Controller

Instanz installiert, um eine unabhängige Verwaltung des Webservers zu ermöglichen.

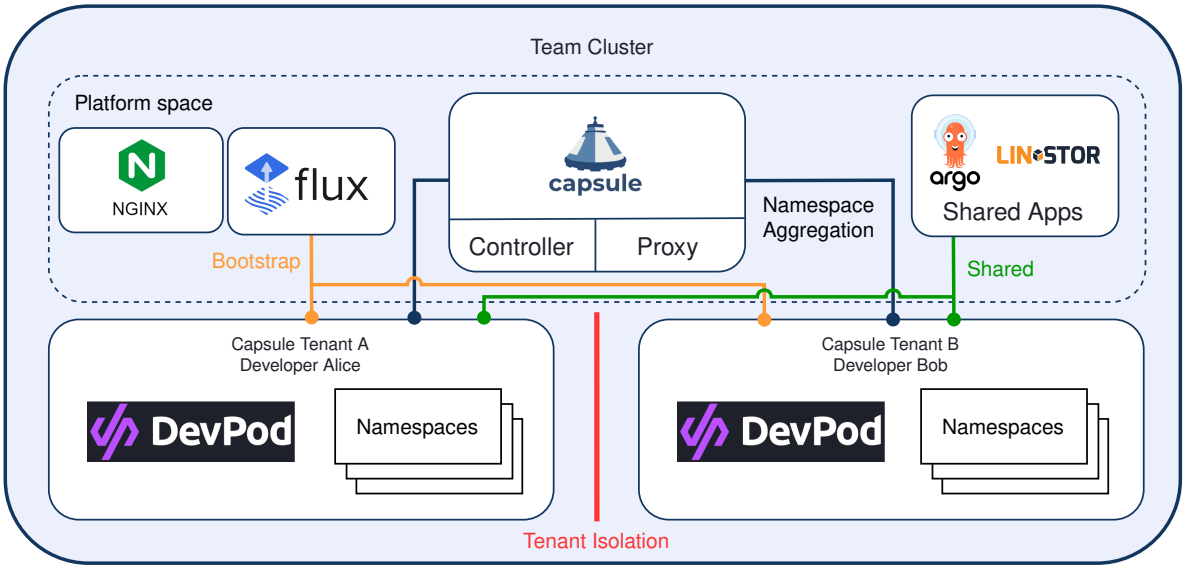

Abbildung 38: *Übersicht über die Systemkomponenten eines Team-Clusters (eigene Darstellung)*

Die einzelnen Dev-Cluster werden von den jeweiligen Softwareentwicklern verwaltet. Diese haben in ihrem Cluster volle Administrationsrechte und können über kubectl weitere Namespaces ihrem Dev-Cluster zuweisen lassen. Dadurch kann unter anderem die Anforderung FA1 erfüllt werden. Außerdem ist so keine Bestätigung in Git durch den Team-Cluster-Admin notwendig, da normale Entwickler keine Rechte haben, das Team-Repository zu verändern.

5.4.2 GitOps: Aufbau des Team-Repositories

Da das Team-Cluster ein Flux-Repository ist, hat es eine ähnliche Struktur und Funktionsweise wie das Cloud-Repository von Kapitel 5.3.3. Die Besonderheit besteht darin, dass es das erste Repository ist, das nicht von den Plattformadministratoren verwaltet wird. Umso wichtiger ist eine einfache Bedienung und volle Flexibilität, damit die Teams selbstständig agieren können. Hier spielen vor allem die Anforderungen FA3, FA4, FA5, FA6, FA7, FA8 und FA9 eine Rolle, die mit diesem einzelnen Repository weitgehend erfüllt werden können.

In Abbildung 39 ist die Grundstruktur des implementierten Team-Repositorys zu sehen. Dort befindet sich die FluxCD-Sync-Konfiguration im Ordner „clusters/tenant-project1“. Der Name des Clusters wird vom Plattformadministrator festgelegt. Sobald eine Verbindung zum Cluster besteht, beginnt FluxCD sofort mit der Synchronisation und installiert alle ausgewählten Infrastruktur- bzw. Shared Apps aus dem Ordner „infrastructure“. In diesem Ordner werden die in Abbildung 38 dargestellten Shared Apps installiert. Auch hier kann frei zwischen Helm Charts, Git Repositories oder OCI-Registries gewählt werden. In diesem Repository ist der Ordner „tenants“ für die Verwaltung der Capsule Tenants zuständig. Dabei symbolisiert jeder Ordner wie z. B. „developer-alice“ einen eigenständigen Dev-Cluster.

Die Konfiguration des Tenants erfolgt ähnlich wie bei Kamaji über eine CRD mit allen notwendigen Konfigurationsparametern wie Tenant-Name, Tenant-Namespace und Rollenverwaltung. In dieser Datei werden auch die Capsule Proxy Freigaben definiert. So werden z. B. über ein Regex-Pattern die Storage- und IngressClasses vom Team-Cluster an die Dev-Cluster weitergegeben, was für die Umsetzung der Anforderung FA10 notwendig ist. Außerdem wird dort noch einmal explizit konfiguriert, ob das Anzeigen, Editieren oder Löschen von Persistent Volumes, IngressClasses, StorageClasses und PriorityClasses möglich ist (vgl. Clastix 2023). Aus Sicherheitsgründen wurde entschieden, außer für PVCs nur das Recht „List“ zu vergeben. Dies kann jedoch bei Bedarf angepasst werden. In Listing 8 ist eine Beispielkonfiguration eines Capsule Tenants zu sehen.

Zusätzlich erhält jeder Team-Cluster eine eigenständige ArgoCD-Instanz, die über die individuellen Kubeconfigs der Tenants beliebige Anwendungen über CI/CD-Pipelines im Cluster starten kann. Grundsätzlich funktioniert ArgoCD ähnlich wie FluxCD, also mit einem zusätzlichen Git-Repository und dem GitOps-Ansatz. Dieses kann jedoch über eine GUI im Webbrowser verwaltet werden, was es Entwicklern erleichtern soll, ihre Deployments zu verwalten.
Um die Anforderung FA18 zu erfüllen, hat jeder Team-Cluster seinen eigenen LinSTOR-Controller, der die physische Speicherkapazität der hinzugefügten Worker-Nodes nutzt. Dadurch kann eine vollständige Trennung der gespeicherten Daten zwischen den Team-Clustern gewährleistet werden. Die Konfiguration ist die gleiche wie in Kapitel 5.1.4 und 5.3.4 beschrieben.

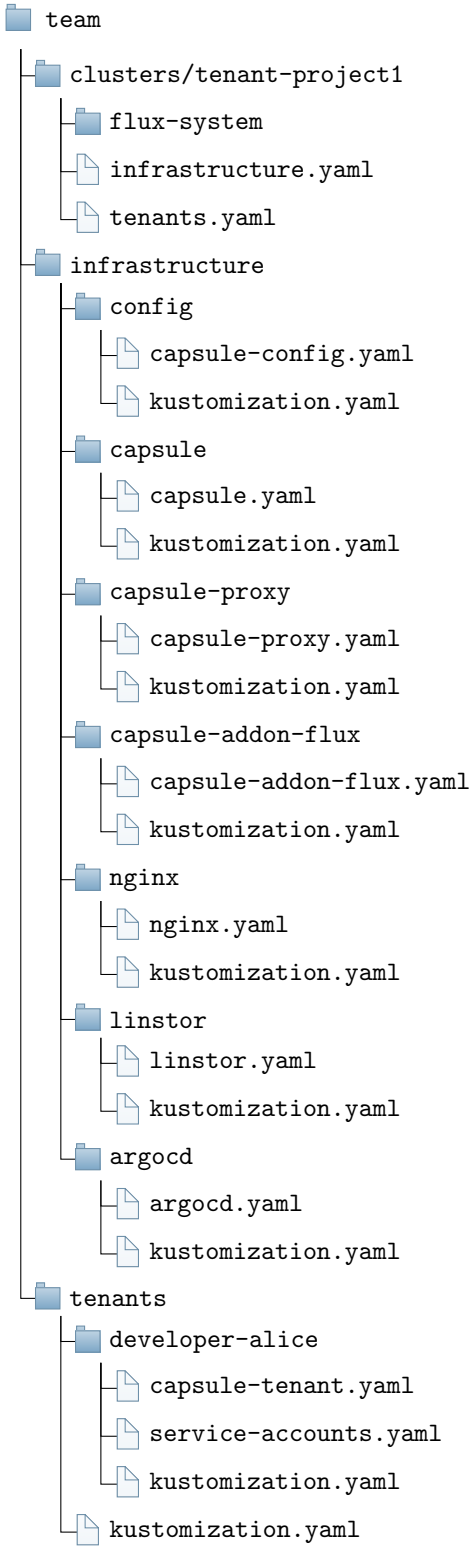

Abbildung 39: *Dateistruktur des Team-Repositories*

5.5 Integration eines Cloud-basierten Softwareentwicklungskonzepts

In diesem Kapitel wird die Umsetzung der Cloud-basierten Softwareentwicklung mit DevPod beschrieben. Zunächst werden in Kapitel 5.5.1 die einzelnen Komponenten des Systems sowie deren Zusammenspiel erläutert. Anschließend wird in Kapitel 5.5.2 der Aufbau eines zentral verwalteten Repository beschrieben, auf das jeder Entwickler mit DevPod zugreifen kann.

5.5.1 Übersicht der Systemkomponenten

Aufgrund der funktionalen Anforderungen FA12, FA13, FA16 und FA17 muss ein Konzept entwickelt werden, um die Kubernetes-as-a-Service-Plattform auch für Entwicklungszwecke nutzen zu können. Gleichzeitig darf aber die lokale Entwicklung nicht beeinträchtigt werden. Es muss also ein hybrides Modell entwickelt werden, das dem Anwender volle Freiheit in der Nutzung lässt. In einem ersten Schritt wurden dazu Kapitel 5.1.5 erste Vergleiche von Open Source Tools durchgeführt. Das dabei ausgewählte Framework „DevPod“ muss nun sinnvoll in die Kubernetes-Plattform integriert werden, um dem Self-Service-Ansatz weiterhin gerecht zu werden.

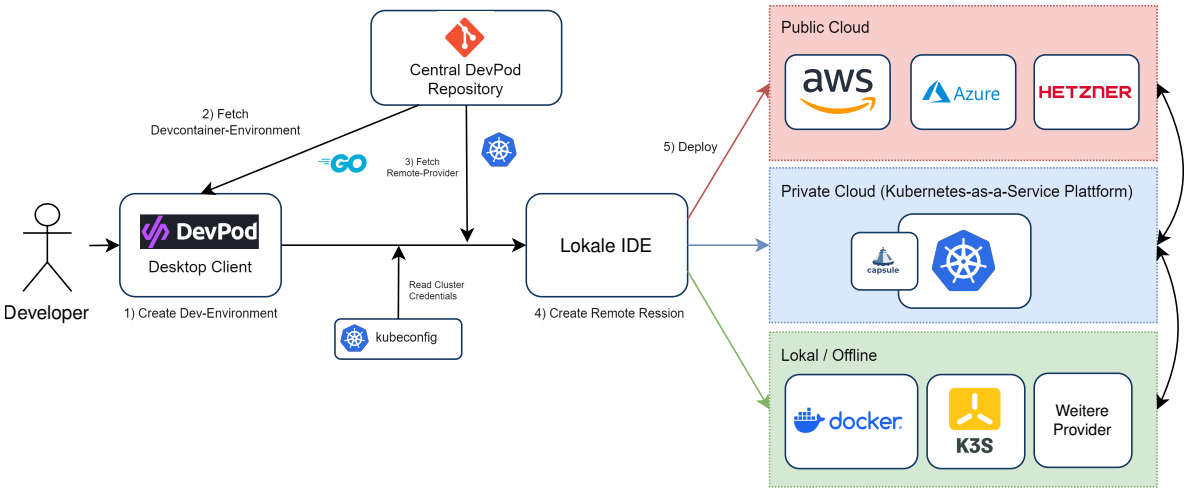

Abbildung 40: *Übersicht der Systemkomponenten zur Cloud-basierten Softwareentwicklung (eigene Darstellung)*

In Abbildung 40 ist der grundsätzliche Aufbau sowie das Zusammenspiel der einzelnen Komponenten dargestellt. Nach der Installation der DevPod-Desktop-Anwendung auf dem lokalen System des Nutzers, das auf Windows, Linux oder Mac basieren kann, muss der Client noch konfiguriert werden. Dies geschieht über eine YAML-Konfigurationsdatei, die ähnlich wie die kubeconfig im lokalen Benutzerverzeichnis im Ordner „.devpod“ abgelegt wird. Diese synchronisiert sich 1:1 mit der GUI von DevPod, was den Vorteil hat, dass die YAML-Konfigurationsdatei über das zentrale DevPod-Git-Repository zur Verfügung gestellt werden kann. Der Entwickler muss dazu lediglich das zentral verwaltete Git-Repository, z. B. über

das firmeninterne GitLab, auf sein lokales System klonen. Dieses sollte im Workspace-Bereich abgelegt werden, in dem sich auch die anderen Repositories für Projekte, usw. befinden. Neben der manuellen Installation der Konfigurationsdatei könnte diese im Unternehmenskontext auch automatisiert über ein Verwaltungstool wie Puppet auf den Systemen installiert werden. Dies würde den gesamten Prozess noch benutzerfreundlicher gestalten.
Um die Cloud-Ressourcen nutzen zu können, müssen dann nur noch die folgenden Schritte durchgeführt werden:

- In **Schritt 1** erstellt der Entwickler nun mit Hilfe der DevPod GUI eine neue Dev-Umgebung, für die er lediglich eine Git-Repository URL oder einen Pfad zu einem lokalen Ordner, in dem sich sein Quellcode befindet, angeben muss.
- In **Schritt 2** muss noch die passende „devcontainer.json“ ausgewählt werden, die sich im zuvor heruntergeladenen Git-Repository befindet. Diese sollte zur Programmiersprache des zuvor ausgewählten Projekts passen. In der Devcontainer-Konfiguration wird die Container-Umgebung mit allen notwendigen Tools, Programmiersprachen etc. vordefiniert.
- In **Schritt 3** wählt der Entwickler seine Zielumgebung aus, in der die Cloud-Umgebung als Container gestartet werden soll. Durch das zentrale Repository ist der Provider für die Kubernetes-as-a-Service-Plattform bereits vorkonfiguriert. Dieser verwendet die lokale kubeconfig, um sich mit dem Capsule Dev-Cluster zu verbinden.
- In **Schritt 4** startet DevPod die ausgewählte Entwicklungsumgebung (IDE) wie z. B. VSCode. Die IDE baut dann eine Remote-Sitzung mit der Cloud-Umgebung auf, normalerweise über SSH. Sobald die Verbindung hergestellt ist, gibt es praktisch keinen Unterschied zwischen lokaler und Remote-Entwicklung. Die IDE kann wie gewohnt verwendet werden. Es werden lediglich die Ressourcen der entfernten Umgebung genutzt.
- **Schritt 5** wird meist parallel zu Schritt 4 ausgeführt. Dabei installiert DevPod im Hintergrund die Cloud-Umgebung über die devcontainer.json und die kubeconfig über verschiedene kubectl-Befehle. Wird eine lokale Option wie Docker Desktop gewählt, erfolgt dies über die jeweiligen CLI-Tools der Containerlösung. Bei Public-Cloud-Anbietern wie AWS erfolgt die Generierung über API-Befehle. Dazu müssen in DevPod weitere Parameter zur Authentifizierung etc. eingegeben werden.

5.5.2 Zentral verwaltetes Repository für DevPod

In diesem Abschnitt werden die Dateistruktur des Repositories sowie eine beispielhafte Devcontainer.json und das zugehörige Dockerfile näher erläutert. Das Git-Repository selbst wird unabhängig von der Self-Service Plattform verwaltet. Im Kontext dieser Arbeit sollte dieses dann unternehmensweit schreibgeschützt zugänglich sein. Somit kann jeder Entwickler davon profitieren. Um die in Kapitel 5.5.1 genannten Anforderungen zu erfüllen, wurde das Repository, wie in Abbildung 41 zu sehen, so aufgebaut, dass es den Anspruch eines Minimum Viable Products (MVP) erfüllt. Dieses könnte dann je nach Anwendungsfall in Zukunft von

den Teams erweitert werden, z. B. durch einen Fork des Repositories. Damit wäre gleichzeitig die Anforderung FA17 erfüllt.

Dateistruktur des Git-Repositories

Der Ordner „environments“ enthält die einzelnen devcontainer Konfigurationsdateien für die unterstützten Programmiersprachen. Diese Datei muss in Schritt 2 ausgewählt werden. In der zugehörigen Docker-Datei wird die eigentliche Container-Umgebung konfiguriert. Diese kann z. B. auf Debian basieren, auf dem nützliche Tools wie Git, Nano, Wireshark etc. vorinstalliert sind. Später wird dort auch die CLI aus Kapitel 6 mit ausgeliefert.

Im Ordner „providers“ wird ein vorkonfigurierter Kubernetes Provider mit dem Namen „itd-capsule-kubernetes“ ausgeliefert. Dieser verwendet die lokale kubeconfig des Entwicklers, damit dieser sich direkt mit seinem Dev-Cluster verbinden kann. Über DevPod selbst können beliebig viele Provider angelegt werden, dies ist z. B. notwendig, wenn ein Entwickler mehrere Dev-Cluster besitzt. Die Icon Vector Datei wird zur einfachen Identifikation des Providers in der GUI verwendet.

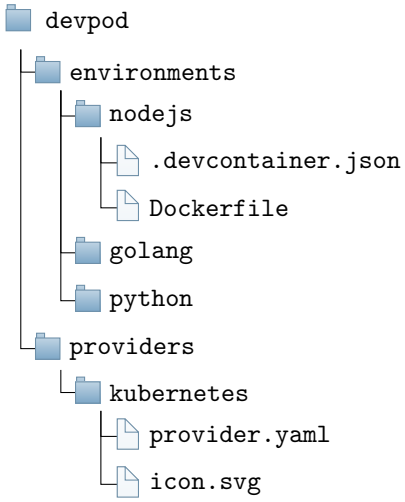

Abbildung 41: *Dateistruktur des DevPod-Repositories*

Beispiel einer devcontainer.json

In Listing 2 ist eine beispielhafte .devcontainer.json dargestellt, die speziell für VSCode geschrieben wurde. Dort sind im Feature-Objekt spezifische Tools für Node.js vorinstalliert sowie im Customization-Objekt verschiedene VSCode-Plugins wie GitHub Copilot, ESLint und das WSL-Plugin für Remote Sessions. Damit soll vor allem noch einmal gezeigt werden, dass für die Anforderungen FA13 und FA17 eine flexible eigenständige Lösung gefunden wurde, auf die problemlos migriert werden kann, ohne viel neues Know-how in den Teams aufbauen zu müssen.

```jsonc
{
    "name": "ITD DevPod - Node.js",
    "build": {
        "dockerfile": "Dockerfile"
    },
    "features": {
        "ghcr.io/devcontainers/features/common-utils:2": {
        "installZsh": "true",
        "username": "devpod",
        "upgradePackages": "true"
        },
        "ghcr.io/devcontainers/features/node:1": {
        "version": "none"
        }
    },
    "customizations": {
        // Configure properties specific to VS Code.
        "vscode": {
        "extensions": [
            "dbaeumer.vscode-eslint",
            "GitHub.copilot",
            "GitHub.copilot-chat",
            "ms-vscode-remote.remote-wsl"
        ]
        }
    },
    "forwardPorts": [],
    [...]
}
```

Listing 2: *Node.js-Devcontainer Konfigurationsdatei*

5.6 Realisierung einer generischen CI/CD-Pipeline

Um die Kubernetes-Plattform zu kompletieren, muss diese noch um eine generische CI/CD-Pipeline ergänzt werden, welche während der Entwicklung eingesetzt werden kann, um das in Kapitel 6 beschriebene Unterstüzungswerkzeug verwenden zu können und um die Anforderung FA1 umsetzten zu können. Da bereits in Kapitel 4.1.2 das derzeit bestehende CI/CD-Pipeline-System analysiert wurde, können die gefunden Probleme wie z. B. den CIOps-Ansatz in der neuen Pipeline besser umgesetzt werden. Für das neue System wird ebenfalls wieder auf Open-Source Lösungen gesetzt. Somit wird das neue System wie das alte auch, auf GitLab und dem zugehörigen GitLab CI/CD-Runner basieren. Dieser wird per Helm Chart über das Team-Cluster-Repository automatisch deployt und konfiguriert. So kann jedes Team selbst entscheiden wie viele GitLab Runner für ihr Projekt benötigt werden.

Systemkonzept und Realisierung der neuen Pipeline

Es muss sichergestellt werden, dass die neue CI/CD-Pipeline auf einem generischen Konzept basiert, welches insbesondere im Build-Prozess unabhängig von der verwendeten Programmiersprache die Projekte bauen und als fertige Container über eine Container Registry wie z. B. Harbor dem Kubernetes Cluster zur Verfügung stellen kann. Da in Kapitel 4.1.2 teilweise gravierende Sicherheitsmängel entdeckt wurden und generell kein generischer Ansatz im Build-Prozess verwendet wird, wurde ein komplett neues Konzept entworfen, um das alte System vollständig abzulösen.
Da ArgoCD für Continuous Delivery (CD) verwendet wird, muss dies nicht als zusätzlicher Schritt in das Konzept aufgenommen werden. Die Erstellung der „Apps“ in ArgoCD wird später durch das CLI-Unterstützungstool übernommen. Für diesen Kontext reicht es also aus, wenn beliebige Projekte in beliebigen Programmiersprachen als Container erstellt werden können und automatisch mit entsprechenden Tags in einer Container Registry zur Verfügung gestellt werden. Das alte System, wie in Abbildung 26 dargestellt, verwendet den Docker-in-Docker-Ansatz, bei dem über ein Dockerfile im Quellcode die Container über einen Container-Daemon wie Docker oder Podman gebaut werden, der innerhalb des bereits existierenden Docker-Containers des CI-Runners läuft. Dies geschieht typischerweise durch die Ausführung des „docker build“ Befehls innerhalb der Pipeline. Da dieser Ansatz jedoch Probleme wie schlechtere Performance mit sich bringt und einen privilegierten GitLab-Runner-Container voraussetzt, was unter anderem gravierende Einfallstore für Hackerangriffe öffnet (vgl. GitLab 2024), wird in dem in Abbildung 42 vorgestellten neuen System das Framework „kaniko“ verwendet. Dieses kann ohne externen Daemon verwendet werden, um Container zu bauen. Es benötigt keinen privilegierten Host-Container mehr und unterstützt auch das Taggen und Pushen von Containern in externe Registries. Da Kaniko nur ein bereitgestelltes Dockerfile aus dem Quellcode benötigt, wurden alle Schritte und Abhängigkeiten, die zur Ausführung des Containers notwendig sind, auf die Dev-Seite ausgelagert. Dies schafft Unabhängigkeit und gleichzeitig eine generische vielseitige CI-Pipeline.

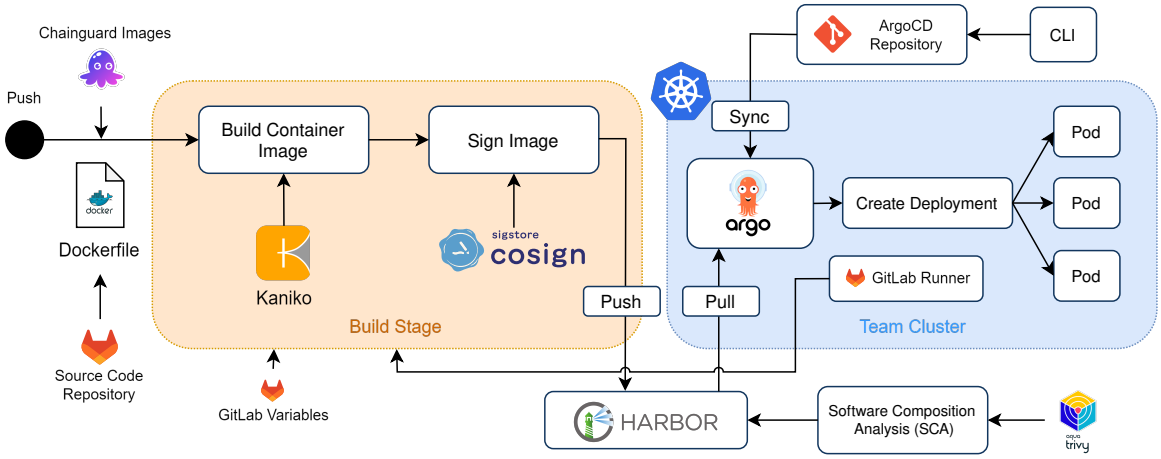

Abbildung 42: *Aufbau der neuen sicheren CI/CD-Pipeline (eigene Darstellung)*

Gleichzeitig wurde die Pipeline um einige SecOps-Tools erweitert, wie sie in Kapitel 2.2.3 beschrieben sind. So werden neue Container-Images mit „cosign“ und einem privaten Schlüssel signiert, was eine Manipulation des Images verhindert. Darüber hinaus wird in der Harbor Container Registry zukünftig eine Software Composition Analysis (SCA) mittels Trivy durchgeführt, um CVE-Sicherheitslücken zu finden. Diese werden jedoch bereits durch die Verwendung von sogenannten „hardened“ und „minimal“ Base Images der Firma Chainguard auf ein Minimum reduziert. Diese Images werden täglich aktualisiert, so dass sie keine der bekannten CVEs mehr enthalten. In Listing 9 ist ein exemplarisches Dockerfile dargestellt.

Fazit

Insgesamt konnte durch das neue Konzept eine Phasen-Pipeline erstellt werden, welche den bisherigen CIOps-Ansatz vollständig ablöst. Es wurde eine strikte Trennung von CI und CD erreicht, indem zukünftig ArgoCD für die Deployments zuständig ist. Die Pipeline sowie die Images selbst enthalten keine kritischen Zugriffsdaten mehr und konnten durch die Chainguard-Images und den Kaniko-Buildprozess in ihrer Gesamtgröße auf ein Minimum reduziert werden. Zum Vergleich: Mit der alten Pipeline hatte ein Go-Container-Image durchschnittlich ca. 600 MiB und 30-200 Sicherheitslücken. Ein neues Go-Container-Image basierend auf einem Chainguard-Static-Image und kompiliertem Quellcode hat ca. 2-3 MiB und keine bekannten Sicherheitslücken. Durch den Einsatz von Gitlab-Variablen konnten zudem alle Tokens und Zertifikate aus den YAML-Dateien der Git-Pipeline entfernt werden. Zukünftig können weitere SecOps-Tools wie z. B. die automatische Erkennung von Geheimnissen etc. in weiteren Schritten integriert werden.
Da sich die Pipeline primär um die Bereitstellung des Container-Images kümmert, wird nun ein unterstützendes Werkzeug benötigt, um das Gesamtsystem der Self-Service Plattform, wie in Kapitel 4.2.1 beschrieben, zu vervollständigen. Aus diesem Grund wurde in diesem Konzept bewusst auf CD-Stages verzichtet, um die Basis-Pipeline als generisches System den einzelnen Teams zur Verfügung zu stellen, damit diese ihre Container-Images ohne Konfigurationsaufwand erstellen können. Alles weitere ist nachfolgend in Kapitel 6 beschrieben.

6 Konzeption des CLI-Unterstüzungswerkzeugs

Dieses Kapitel beschreibt das Design des CLI-Unterstützungswerkzeugs zur Durchführung spezifischer Ops-Aufgaben über das Terminal innerhalb der IDE, um das Self-Service-Modell der Kubernetes-Plattform zu vervollständigen.

6.1 Grundlegende Funktionsweise der CLI

In Kapitel 5 wurde die Implementierung der allgemeinen Infrastruktur der Kubernetes-Plattform beschrieben. Der Schwerpunkt lag dabei auf der Administration und Verwaltung der Plattform durch die Benutzergruppe „Plattformadministrator“. Wie in Kapitel 4.2.1 beschrieben, beinhaltet die Plattform jedoch auch eine Komponente zur Unterstützung der Endanwender, also der Benutzergruppe „Softwareentwickler“. Dies soll durch ein Unterstützungswerkzeug in Form eines Command Line Interfaces (CLI) erfolgen. Wie die Beispiele Helm, FluxCD, ArgoCD und Kubernetes (kubectl) zeigen, ist dies eine gängige Methode lokale Systeme mit dem Cloud-nativen Ökosystem zu verbinden. Bei der Umsetzung wird außerdem besonders Wert auf eine einfache und selbsterklärende Bedienung gelegt. Die Ergebnisse in Kapitel 3 haben gezeigt, dass überwiegend nur Grundkenntnisse im Bereich Docker, Kubernetes und CI/CD vorhanden sind. In Kapitel 6.4 wird das Konzept zur Optimierung der User Experience näher beschrieben.

Konkret soll die CLI in der Lage sein, den Entwickler bei der Entwicklung und dem anschließenden Testen durch ein Deployment im Cluster zu unterstützen. Die entsprechenden Grundlagen wurden bereits in Kapitel 5.5 und 5.6 gelegt, indem eine geeignete Entwicklungsumgebung sowie die entsprechende CI-Pipeline geschaffen wurden. Die CLI selbst wird die Continuous Delivery (CD)-Bausteine übernehmen, indem sie entsprechende Manifeste für Kubernetes und ArgoCD erstellt, die auf Basis vordefinierter Template-Dateien sichere Deployments erzeugen, die u.a. die Kubernetes-Sicherheitsrichtlinien des Open Worldwide Application Security Project (OWASP) einhalten (vgl. OWASP 2024a). Die CLI selbst soll eine geführte Konfiguration erhalten, die nach Abschluss durch den Benutzer eine Konfigurationsdatei erzeugt, in der alle Projekt- und Authentifizierungsinformationen enthalten sind, um diese auf mehrere Geräte verteilen zu können. Dies folgt einem ähnlichen Prinzip wie es die kubectl mit der „kubeconfig“-Datei im Benutzerverzeichnis handhabt.

Für die Authentifizierung und Autorisierung im Cluster und Git wird auf eine Eigenentwicklung verzichtet, da dies aufgrund der zur Verfügung gestellten APIs von Kubernetes und GitLab nicht notwendig ist. Da zudem die Plattform selbst eine Abhängigkeit zu GitLab und Kubernetes aufweist, bietet sich diese Integration optimal an.

6.2 Architektur

Für die spätere einfache Nutzung über das Terminal wird die kompilierte Anwendung in die Umgebungsvariable PATH des Betriebssystems aufgenommen. Dazu benötigt das CLI selbst einen einprägsamen Namen. Im Rahmen dieser Arbeit erhält die CLI den Namen „itdcli“. Dieser setzt sich aus dem Kürzel der Firma und dem Wort CLI zusammen. So kann später im Terminal durch Eingabe von „itdcli“ ohne Angabe eines Pfades o. ä. die CLI verwendet werden.

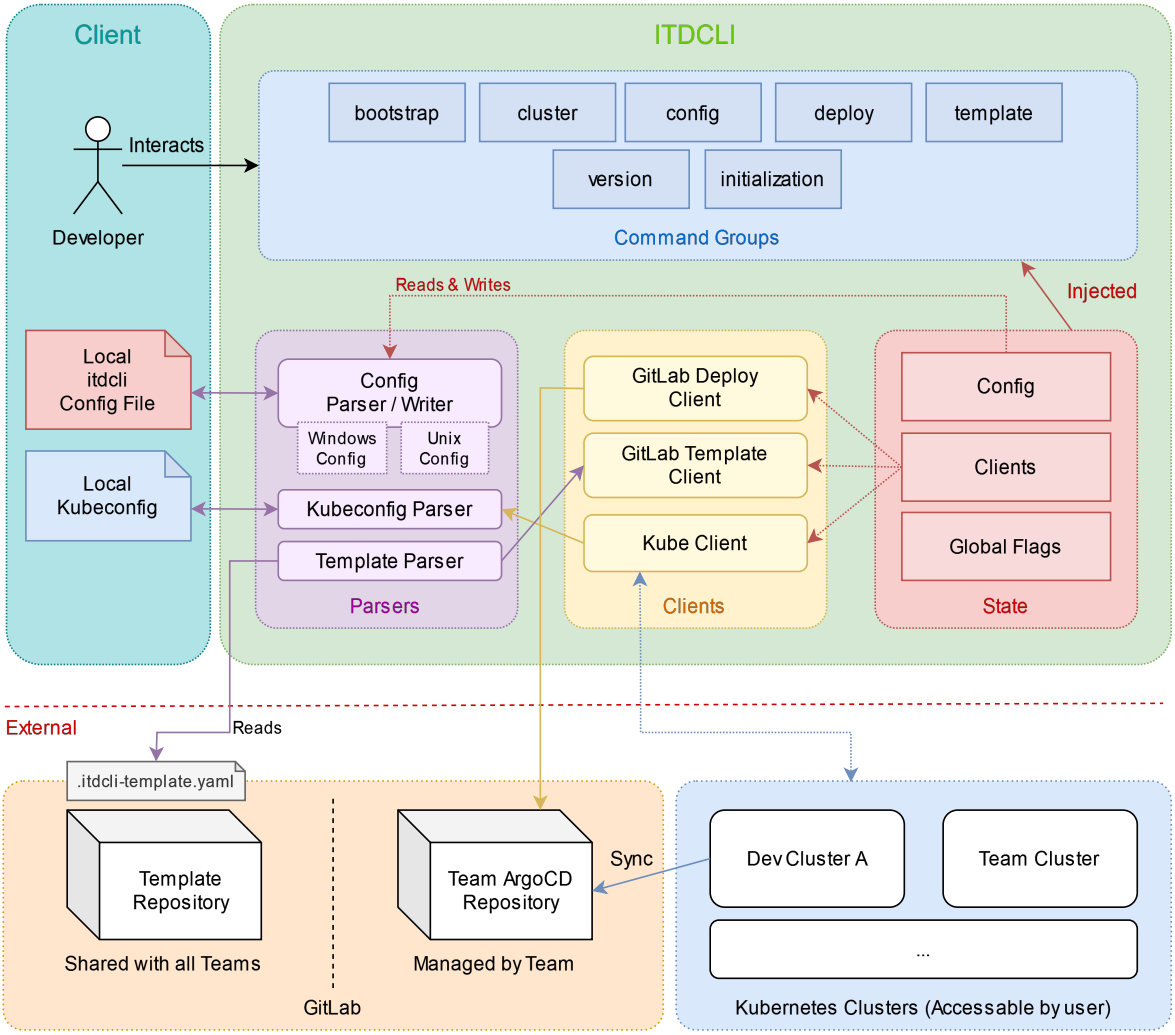

Abbildung 43: *Darstellung der Architektur der CLI (eigene Darstellung)*

In Abbildung 43 ist die high-level Architektur der CLI dargestellt. Die CLI selbst besteht aus den vier Modulen Parsers, Clients, Command Groups und dem State, die in der grünen Box „ITDCLI“ dargestellt sind. Die Parser kümmern sich um alle dateibezogenen Aufgaben, wie das Lesen und Schreiben von Konfigurationsdateien sowohl von lokalen als auch von entfernten Dateien aus z. B. Git-Repositories. Dabei greifen sie auf bestehende Clients der jeweiligen Anbieter wie GitLab oder Kubernetes zu. Diese Clients befinden sich im „State“

der CLI, der vor der Ausführung eines Kommandos durch die lokale Konfigurationsdatei itdcli im Benutzerverzeichnis des Benutzers initialisiert wird. Sie enthält alle notwendigen Informationen über Provider, Repositories, Benutzerdaten etc. Die Konfigurationsdatei wird bei jedem Kommando vom Config Parser eingelesen und im State gespeichert. Die Clients verbinden sich dann automatisch mit den Providern.

Der Benutzer selbst agiert mit der CLI ausschließlich über die Befehle der Befehlsgruppen. Diese Gruppen enthalten kontextabhängige Befehle, die für den jeweiligen Anwendungsfall benötigt werden. In Kapitel 6.3 werden diese im Detail erläutert. Die Logik innerhalb der Kommandos hat immer Zugriff auf den State, da dieser an jedes Kommando übergeben wird. Dadurch kann sichergestellt werden, dass zur Laufzeit immer nur ein State existiert. Dies ist vergleichbar mit einer Singleton-Klasse. Der State enthält auch sogenannte „Global Flags“, diese können vom Benutzer als Parameter im Terminal übergeben werden, welche dann zur Laufzeit Variablen überschreiben, die sonst beispielsweise über die Konfigurationsdatei standardmäßig gesetzt werden. Ein Beispiel hierfür wäre der „–context“. Dieser ist Standardmäßig auf den aktuell ausgewählten Kubernetes Cluster gesetzt. Durch die Angabe dieses Flags kann dieser aber auch nur für den aktuellen Befehl überschrieben werden, ohne dass eine Anpassung der Konfiguration notwendig ist.

Neben den GitLab-Clients gibt es auch einen Kube-Client, der für den Abruf von Echtzeitdaten aus dem Kubernetes-Cluster zuständig ist. Dieser Client verbindet sich über die vorhandenen Verbindungsinformationen und Zugangsdaten aus der kubeconfig-Datei mit dem aktuell ausgewählten Cluster. Dies macht eine zusätzliche Anpassung und Verwaltung der Zugangsdaten zu Kubernetes überflüssig und bietet gleichzeitig den Vorteil, dass Anpassungen an der Config weiterhin über kubectl möglich sind. Somit besteht immer eine perfekte Synchronisation zwischen kubectl und itdcli. Wird also über das kubectl der aktuell aktive Cluster geändert, so wird dies beim nächsten Kommando mit der itdcli in diesem ausgeführt.

6.3 Übersicht der Befehlsgruppen und Befehle

Eine moderne CLI verwendet so genannte Kommandos oder Befehle, die oft in Gruppen angeordnet sind. Die Befehle sind in der Regel mit detaillierten Informationen zu ihrer Verwendung versehen. Dies macht eine externe Dokumentation in der Regel überflüssig. Um den Anwendungsfall dieser Arbeit abzudecken, wurden folgende Gruppen und Befehle erstellt, die die Funktionalitäten des MVP abdecken sollen. Die Namen wurden so gewählt, dass sie möglichst selbsterklärend sind. Eine Übersicht der Befehle ist in Tabelle 6 dargestellt. Die fett markierten Gruppen verfügen über einen Guided Mode.

Befehlsgruppe	Befehl	Erklärung
Bootstrap	bootstrap	Zeigt alle verfügbaren Kommandos dieser Gruppe an.
	bootstrap deploy-repo	Startet den Bootstrap-Prozess für die Einrichtung des Deployment-Repositories.
	bootstrap pipeline	Setzt die CI/CD-Pipeline im GitLab-Repository auf.
	bootstrap template	Wendet Konfigurationstemplates an.
Deploy	deploy	Zeigt alle verfügbaren Kommandos dieser Gruppe an.
	deploy kubernetes	Führt das Deployment auf einem Kubernetes-Cluster durch.
Initialization	initialization	Initialisiert die CLI mit Benutzerinformationen.
Cluster	cluster	Zeigt alle verfügbaren Kommandos dieser Gruppe an.
	cluster list-contexts	Listet alle verfügbaren Kubernetes-Kontexte auf.
	cluster current-context	Zeigt den aktuell verwendeten Kubernetes-Kontext an.
	cluster use-context	Wechselt zu einem bestimmten Kubernetes-Kontext.
Config	config	Zeigt alle verfügbaren Kommandos dieser Gruppe an.
	config deploy-repo add	Fügt ein neues Deployment-Repository hinzu.
	config deploy-repo list	Listet alle vorhandenen Deployment-Repositories auf.
	config deploy-repo remove	Entfernt ein bestehendes Deployment-Repository.
	config deploy-repo set	Setzt ein Deployment-Repository als aktiv.
Template	template	Zeigt alle verfügbaren Kommandos dieser Gruppe an.
	template list	Listet das aktuelle Template-Repository auf.
	template add	Setzt ein neues Template-Repository.
	template remove	Entfernt das Template-Repository.
Version	version	Zeigt die aktuelle Version der CLI an.

Tabelle 6: *Übersicht über die Befehlsgruppen und Befehle der CLI*

6.4 Optimierung der Benutzererfahrung (UX)

Herkömmliche CLIs arbeiten in der Regel ausschließlich mit festen Argumenten, die meist nach dem Schema „–PARAMETER [WERT]“ aufgebaut sind. Diese müssen vom Benutzer angegeben werden, um die entsprechenden Daten korrekt zuzuordnen. Für erfahrene Benutzer

oder Automatisierungen ist diese Methode optimal, um schnell und effizient mit der CLI arbeiten zu können. Für Anfänger kann dies jedoch eine größere Hürde darstellen, da die Benennung der Parameter oft nicht eindeutig ist. Aufgrund der Umfrageergebnisse in Kapitel 3.4 ist davon auszugehen, dass wenig Erfahrung mit Kubernetes vorhanden ist. Die CLI muss daher so gestaltet werden, dass sie den Benutzer durch Erklärungen, Fragen und einen selbsterklärenden Aufbau unterstützt. Gleichzeitig muss sie aber auch parametrisierbar bleiben, um die oben genannten Vorteile weiterhin nutzen zu können.

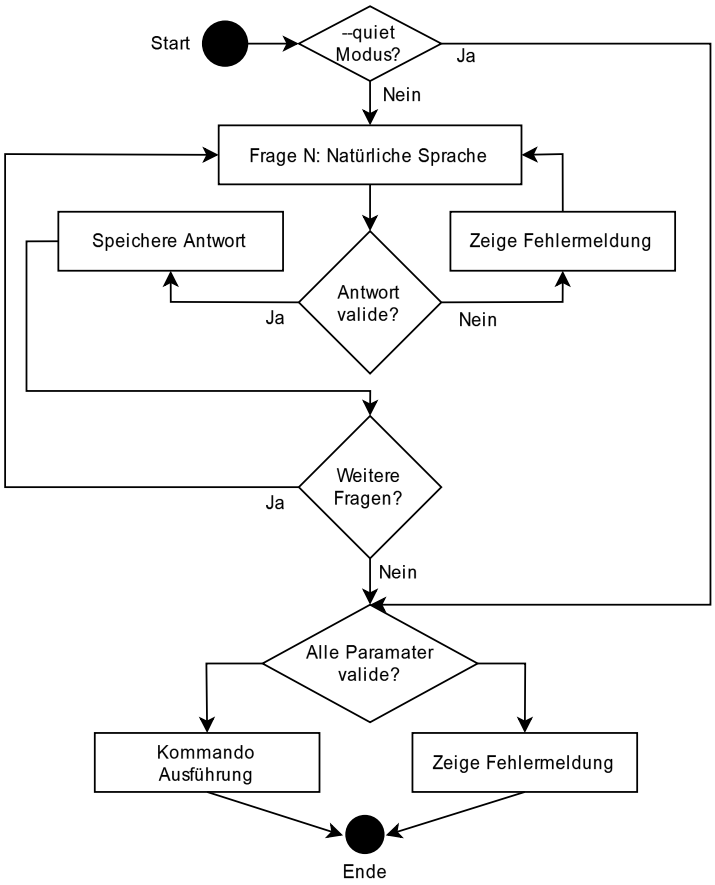

Abbildung 44: *Ablaufdiagramm für den geführten Modus eines Kommandos (eigene Darstellung)*

Konkret bedeutet dies, wie in Abbildung 44 zu sehen, dass wichtige Befehle der CLI zusätzlich durch einen „Guided Mode“ ergänzt werden, der den Benutzer Schritt für Schritt durch Fragen in natürlicher Sprache zu einer Antwort auffordert, die dann durch einen Filter auf Plausibilität geprüft wird. So kann bei jeder Frage direkt auf falsche Eingaben reagiert werden, indem z. B. die Frage wiederholt wird und der Benutzer durch eine Fehlermeldung darauf hingewiesen wird, dass seine Eingabe falsch war. Außerdem können durch Ja/Nein-Fragen bestimmte Themen direkt übersprungen werden, z. B. „Möchten Sie einen Ingress für Ihre Anwendung erstellen? ([Y]/n)“. Dabei ist jeweils die Option vorausgewählt, die voraussichtlich häufiger gewählt wird. Dadurch können die Fragen schneller beantwortet wer-

den. Dieser Modus kann durch Setzen des Parameters „–quiet“ oder „-q“ abgeschaltet werden.

Außerdem ist eine Priorisierung zwischen Fragen und Parametern vorgesehen, um das Überspringen von Fragen zu ermöglichen. So überschreiben die Parametereingaben vor dem Start des Geführten Modus die gemeinsamen Variablen für die Benutzerantworten, so dass vor jeder Frage geprüft werden kann, ob bereits eine Antwort vorliegt. Ist dies der Fall, werden diese ohne Rückfrage übersprungen. Ziel ist es, dem Benutzer die freie Wahl bei der Bedienung der CLI zu lassen.

6.5 Konzeption des Template Repositories

Die CLI soll für den Deployment-Modus vorgefertigte YAML-Template-Dateien verwenden, die Platzhalter enthalten. Durch Benutzerabfragen sollen diese Platzhalter durch reale Daten ersetzt werden. Für Kubernetes werden z. B. Dateien wie „Deployment.yaml“, „Service.yaml“, „Ingress.yaml“ und „PersistentVolumeClaim.yaml“ benötigt. Diese können später beliebig erweitert werden, wenn weitere Deployment-Typen oder andere Funktionalitäten in der CLI hinzugefügt werden. Da sich das Template Repository in Git befindet, muss beim Bootstrapping der CLI in der lokalen Konfigurationsdatei eine Verbindung zu diesem Repository hergestellt werden. Dies sollte frei konfigurierbar sein, so dass ein Benutzer / Team nicht auf ein Repository angewiesen ist, wenn es eigene Templates verwenden möchte. Normalerweise können normale CLI-Benutzer keine Anpassungen am Template Repository vornehmen, da dieses unternehmensweit zugänglich ist und sich daher im Read-Only-Modus befindet.

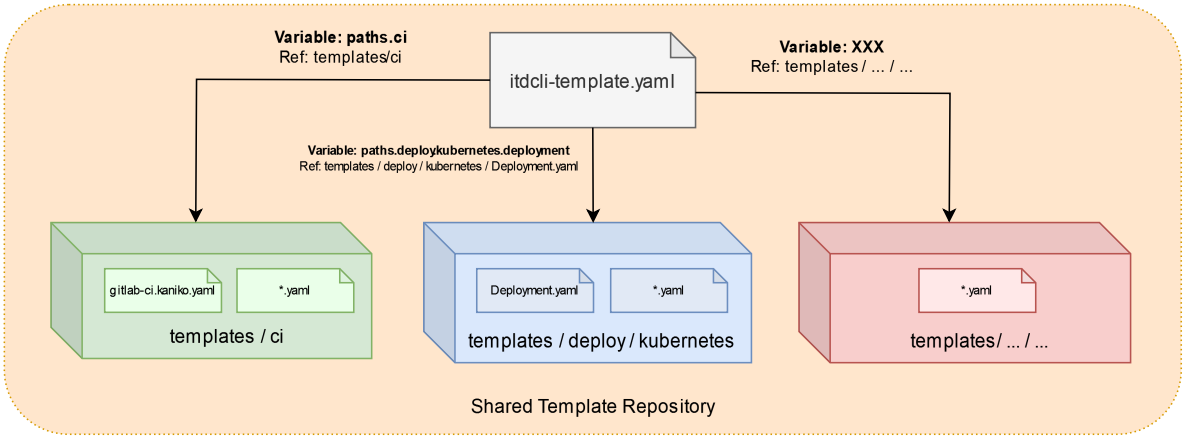

Abbildung 45: *Darstellung der Index-Datei des Template-Repositories (eigene Darstellung)*

Die Flexibilität, eine Referenz auf ein beliebiges Template Repository zu setzen, reicht jedoch nicht aus. Da Teams eigene Repositories erstellen können, wird, wie in Abbildung 45 zu sehen, eine Indexdatei benötigt, die die CLI einlesen kann, um sich in der Dateistruktur des Repositories zurechtzufinden. Dies hat den Vorteil, dass nicht alle Pfade im Quellcode hardcodiert werden müssen, sondern über eine Abstraktionsschicht mit Variablen gearbeitet werden kann.

Der in Abbildung 43 vorgestellte Template Parser soll diese Aufgabe in Echtzeit übernehmen. Er soll die Datei „itdcli-template.yaml“ im Hauptverzeichnis des Repositories suchen und einlesen. Dabei soll die YAML-Struktur in eine geeignete Datenstruktur überführt werden, so dass über diese die Pfade im Quellcode ausgelesen werden können. So kann beispielsweise als Referenz für die Datei „Deployment.yaml“ die Variable „paths.deploy.kubernetes.deployment“ verwendet werden. Diese hat dann als Wert den internen Repository-Pfad als String. Mit dieser Lösung soll ein vollständiges generisches Template-System ermöglicht werden, das von den Anwendern je nach Bedarf und Anwendungsfall genutzt und angepasst werden kann.

6.6 Beispiel: Ablauf zur Erstellung eines Kubernetes Deployments

In Abbildung 46 ist ein beispielhafter, vereinfachter Ablauf dargestellt, den ein Benutzer durchläuft, wenn er seine Applikation in der IDE mit Hilfe der itdcli in seinem Dev-Cluster deployen möchte. Für dieses Beispiel hat der Benutzer bereits alle Bootstrapping- und Initialisierungsschritte der CLI vollständig durchlaufen. Ebenso befindet sich in der kubeconfig bereits der passende Kubernetes Cluster.

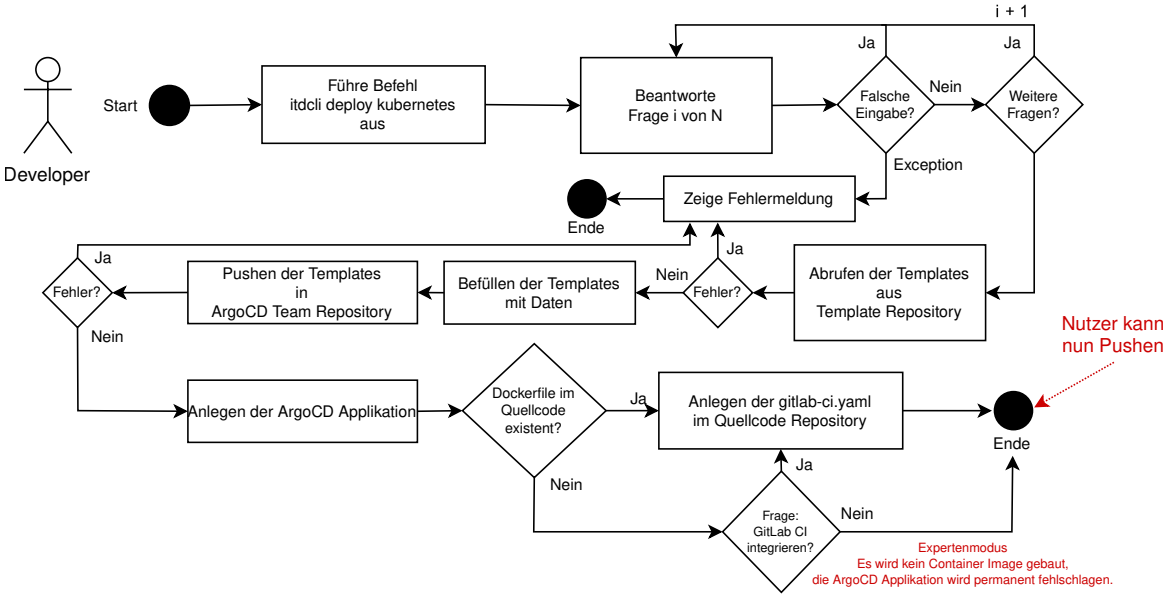

Abbildung 46: *Ablaufdiagramm zur Erstellung eines Kubernetes Deployments mit der CLI (eigene Darstellung)*

Der Ablauf ist nur eine grobe Darstellung der Abläufe, im Hintergrund laufen noch weitere Schritte ab, vor allem in Bezug auf die Benutzeranfragen. Mehr dazu unter Kapitel 7.

7 Implementierung des CLI-Unterstüzungswerkzeugs

In diesem Kapitel wird die eigentliche technische Implementierung des CLI-Unterstützungstools beschrieben.

7.1 Auswahl der Programmiersprache und Frameworks

Als Programmiersprache für die itdcli wird Go in der Version 1.23 verwendet. Diese Programmiersprache hat sich in den letzten Jahren als Standard für alle Cloud-nativen Anwendungen und CLIs durchgesetzt (vgl. Google 2024). Go bietet den Vorteil, dass der mitgelieferte Compiler für alle Plattformen und OS-Distributionen lauffähige Anwendungen erstellen kann. Außerdem gibt es dank der großen Open-Source-Community für alle großen Anwendungen wie Kubernetes oder GitLab SDKs, die die aufwendige Integration ihrer APIs erleichtern. Go selbst ist relativ ähnlich zu C, bietet aber einige zusätzliche Features und Verbesserungen wie Concurrency, Memory Safety und eine einfache Syntax, was diese Sprache ideal für verteilte Systeme macht. Aus diesen Gründen hat sich Go auch als primäre Sprache für Kubernetes und Co. etabliert. In Sachen Geschwindigkeit kann Go mit C mithalten, erreicht aber nicht die native Geschwindigkeit von C. Dafür sind die Kompilierzeiten von Go im Vergleich zu anderen Sprachen wie Java oder JavaScript deutlich schneller.
Als Basis für itdcli wird das Open Source CLI Framework „cobra“ des Autors spf13 verwendet. Es ist derzeit das größte Framework in diesem Bereich mit über 37 Tausend Sternen auf GitHub und wird von fast allen wichtigen CLIs wie kubectl, helmcli, fluxcli etc. verwendet. Das Framework bietet eine relativ flache Architektur und versucht, dem Entwickler den Großteil des Boilerplate Codes abzunehmen, damit er sich auf die Erstellung der Geschäftslogik konzentrieren kann. Es wird durch die Erweiterungen „viper“ und „pflag“ desselben Autors ergänzt. Viper wird für die Konfigurationsdatei verwendet, die in Kapitel 7.2.1 näher beschrieben wird. Es enthält wichtige 12-Faktor-Regeln, die für verteilte Systeme und deren Konfiguration wichtig sind. Das Paket pflag wird von cobra benötigt, um die POSIX/GNU-Style Flags (–flag) zu verwalten. Außerdem wird das offizielle GitLab und Kubernetes SDK verwendet, um mit den REST-APIs dieser Anwendungen zu kommunizieren.

Name	Version	Lizenz	URL
spf13/cobra	v1.8.1	Apache-2.0	https://github.com/spf13/cobra
spf13/viper	v1.19.0	Apache-2.0	https://github.com/spf13/viper
spf13/pflag	v1.0.5	Apache-2.0	https://github.com/spf13/pflag
xanzy/go-gitlab	v0.107.0	Apache-2.0	https://github.com/xanzy/go-gitlab
k8s/client-go	v0.30.3	Apache-2.0	https://github.com/kubernetes/client-go

Tabelle 7: *Tabelle der eingesetzen Frameworks für die CLI*

7.2 Konfiguration und Zustand (State)

7.2.1 Konfigurationsdatei

Wie in Abbildung 43 beschrieben, verfügt das CLI über eine eigenständige Konfigurationsdatei mit dem Namen „config.yaml“, die im lokalen Benutzerverzeichnis im Ordner „.itdcli“ abgelegt wird. Durch den Punkt als Präfix wird dieser Ordner je nach Betriebssystem als versteckter Ordner behandelt. Dadurch wird z. B. ein versehentliches Löschen durch den Linux-Befehl „rm *“ verhindert. Innerhalb dieser Konfigurationsdatei befinden sich, wie in Abbildung 47 dargestellt, hauptsächlich die Zugangsdaten zu den einzelnen Providern sowie die Informationen zum Template- und Deployment-Repository. Zusätzlich werden bestimmte Benutzerinformationen wie Vorname, Nachname, Benutzername, E-Mail gespeichert, die für spätere Git-Commits verwendet werden. Zusätzlich erhält jede Konfiguration eine Instanz-ID, um jede Instanz von itdcli einem Gerät zuordnen zu können. Diese ID kann bei Bedarf in Git-Commits o.ä. hinzugefügt werden, um Benutzer eindeutig zu identifizieren. Die Kubernetes-Konfiguration wird hauptsächlich für die globalen Flags verwendet, die über die Methode „AddFlags(...)“ zu cobra hinzugefügt werden.

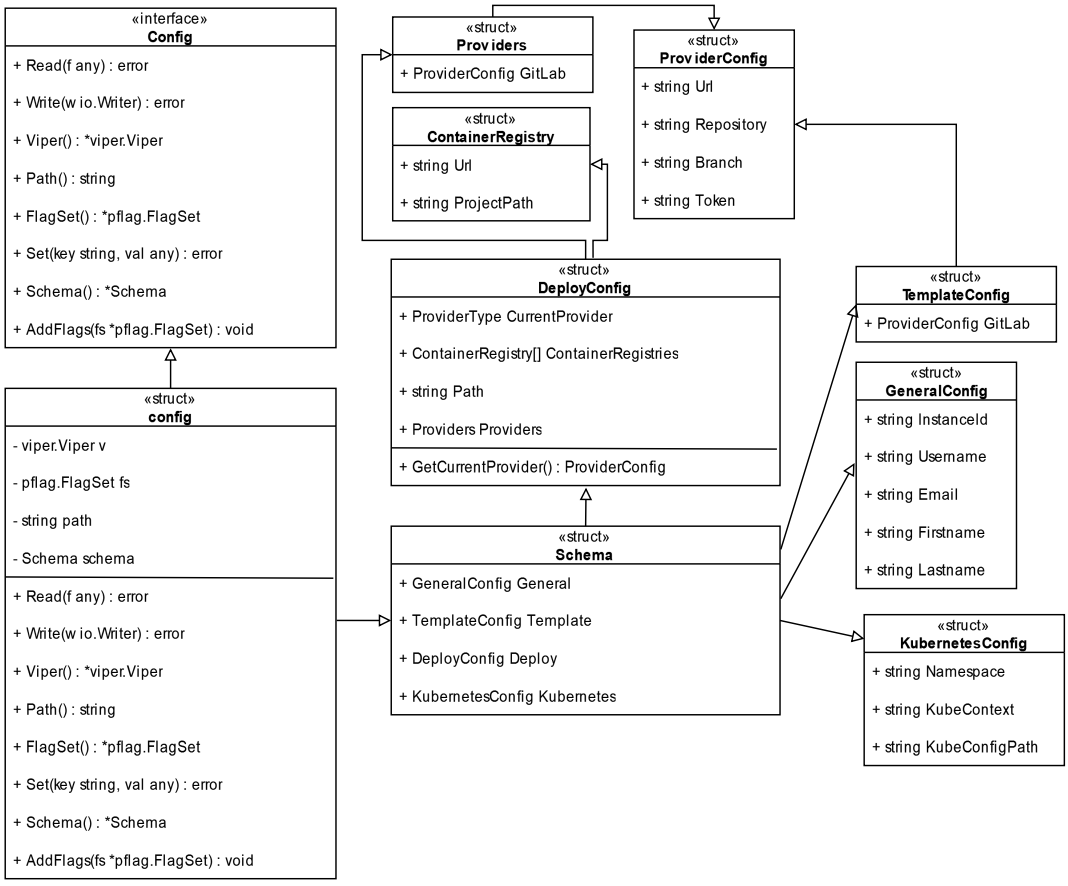

Abbildung 47: *Darstellung der Go-Interfaces für die Konfigurationsdatei (eigene Darstellung)*

Die Konfigurationsdatei verwendet das YAML-Format, was ein einfaches Schreiben und Lesen der Konfiguration in Echtzeit ermöglicht. Außerdem können so auch manuelle Änderungen an der Datei durch den Benutzer vorgenommen werden, da der Aufbau von YAML-Dateien durch die einfache Syntax für jeden Entwickler selbsterklärend sein sollte. Im Quellcode werden sogenannte Interfaces und Structures verwendet. Mit ihrer Hilfe können komplexe Datenstrukturen erstellt werden. Dieses Verfahren wird auch als Marshal bzw. Unmarshal bezeichnet. Damit können alle Datenstrukturen von Go in gängige Formate wie YAML, JSON, XML, etc. umgewandelt und gespeichert werden. Der Vorteil ist, dass bei der Konvertierung automatisch alle Daten aus der Konfigurationsdatei in die entsprechenden Datentypen der Structs konvertiert werden. Dies sorgt für sauberen Code und verhindert ständige Konvertierungen in der Geschäftslogik, wodurch das DRY-Prinzip (Don't Repeat Yourself) nicht verletzt wird. Die einzelnen Variablen in den jeweiligen Structs müssen, wie in Go üblich, als Public oder Global definiert werden, damit sie außerhalb des aktuellen Packages (config) verwendet werden können. Dies wird durch Großschreibung der Variablen erreicht.

Durch die Implementierung der Config als Interface ist es möglich, diese als Struct-Variable dem in Kapitel 7.2.2 beschriebenen State hinzuzufügen. Da dieser, wie in der Architektur beschrieben, an alle Befehle übergeben wird, kann so in den jeweiligen Controllern über den Aufruf „s.Config()“ auf das Config-Objekt zugegriffen werden. Alle anderen Methoden dieser Schnittstelle stehen dann zur Verfügung.

7.2.2 Zustand (State)

Die CLI verfügt über einen temporären Zustand (State), der zu Beginn eines Befehls initialisiert wird. Innerhalb des Zustands befinden sich, wie in der Architektur beschrieben, die aktuelle Konfiguration sowie die aktiven Clients für GitLab und Kubernetes. Diese bauen nach dem Laden der Konfiguration automatisch eine Verbindung zu den jeweiligen Template- und Deployment-Repositories auf. Ebenso baut der Kubernetes Client mit Hilfe der lokalen kubeconfig, die sich im Benutzerverzeichnis befindet, eine Verbindung zum aktuell ausgewählten Cluster auf. Dieser verwendet immer synchron zur kubectl den aktiv gesetzten Context. Unter einem Context versteht man ein bestimmtes Cluster, welches über den Namen ausgewählt werden kann.

Dieser zentrale Zustand wurde implementiert, um Code-Duplikate innerhalb der Command Controller zu minimieren. Dadurch wird ein sauberer Code gewährleistet und die Größe der Controller reduziert. Diese können sonst durch den Transaction Script ähnlichen Aufbau schnell unübersichtlich werden.

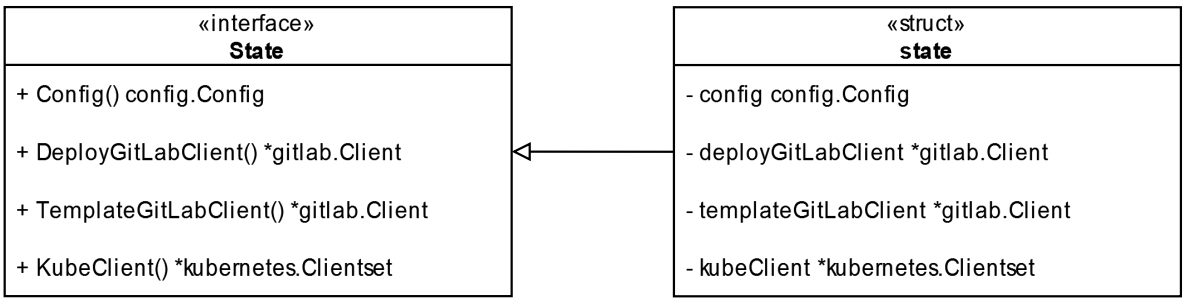

Abbildung 48: *Klassendiagramm der Go-Interfaces und Structs des Zustands (eigene Darstellung)*

In Abbildung 48 ist das Klassendiagramm für den State zu sehen. Dieses verwendet ebenfalls eine Abstraktion der Structs mit Interfaces, so kann dieser einfach als Objekt an beispielsweise die Commands übergeben werden. Über die implementierten Methoden des States kann dann in der Geschäftslogik darauf zugegriffen werden.

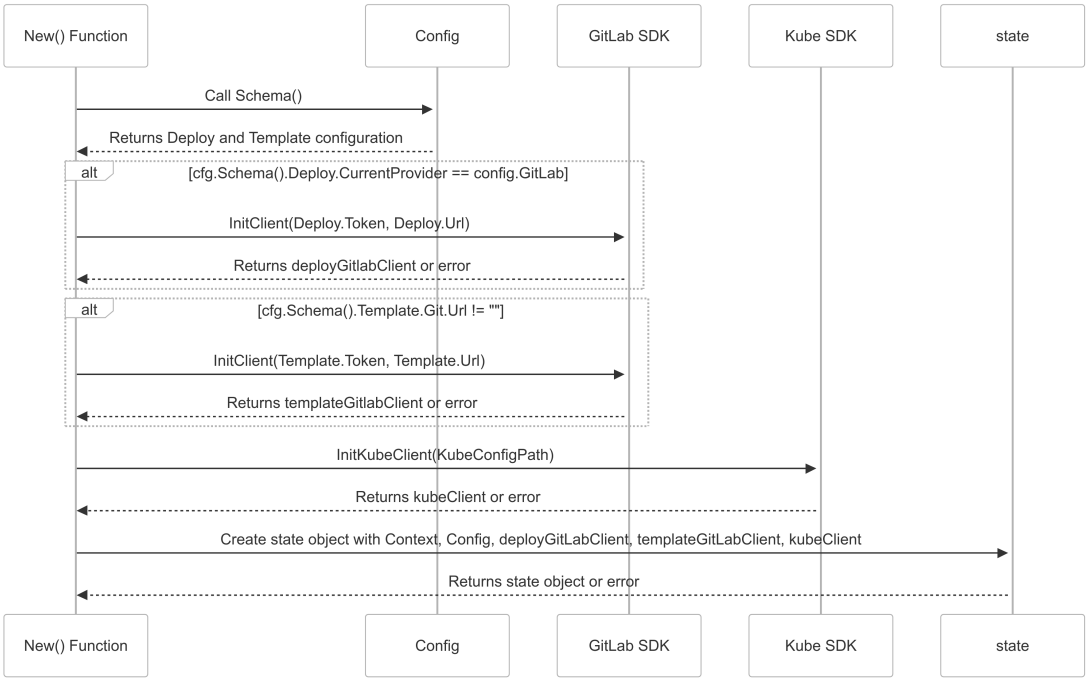

Abbildung 49: *Sequenzdiagramm für die Initialisierung des Zustands (eigene Darstellung)*

In Abbildung 49 ist die Initialisierung des Zustands mit der Methode „New() (State, error)“ zu sehen, die in der Main-Methode von Go aufgerufen wird. Dort werden die oben erwähnten Client-Verbindungen, etc. aufgebaut.

Das zurückgegebene State-Objekt wird dann in der Main-Methode als Zeigerreferenz per Übergabeparameter an alle Befehle und Befehlsgruppen weitergegeben. Damit ist sichergestellt, dass es immer nur ein State-Objekt geben kann.

7.3 Realisierung des Template Repositories

Die in Kapitel 6.5 beschriebene Konzeption des Template Repository wird in diesem Abschnitt nun technisch umgesetzt. Es wurde ein Go-Package mit dem Namen „template“ erstellt, das die in Abbildung 50 dargestellten Methoden und Schnittstellen enthält.

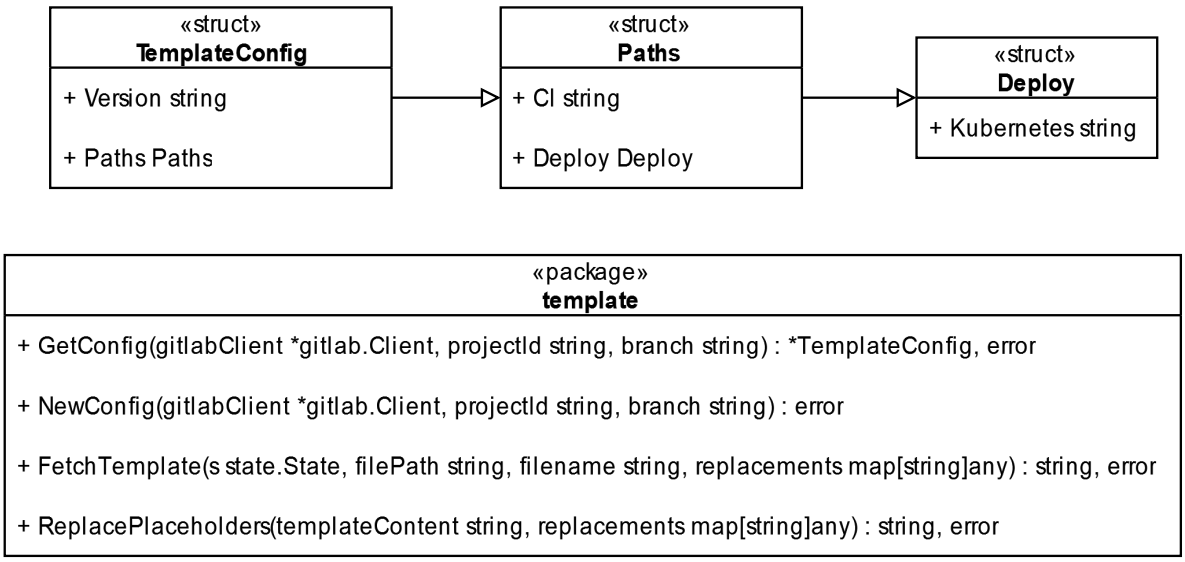

Abbildung 50: *Klassendiagramm der Go-Interfaces und Structs des Template Pakets (eigene Darstellung)*

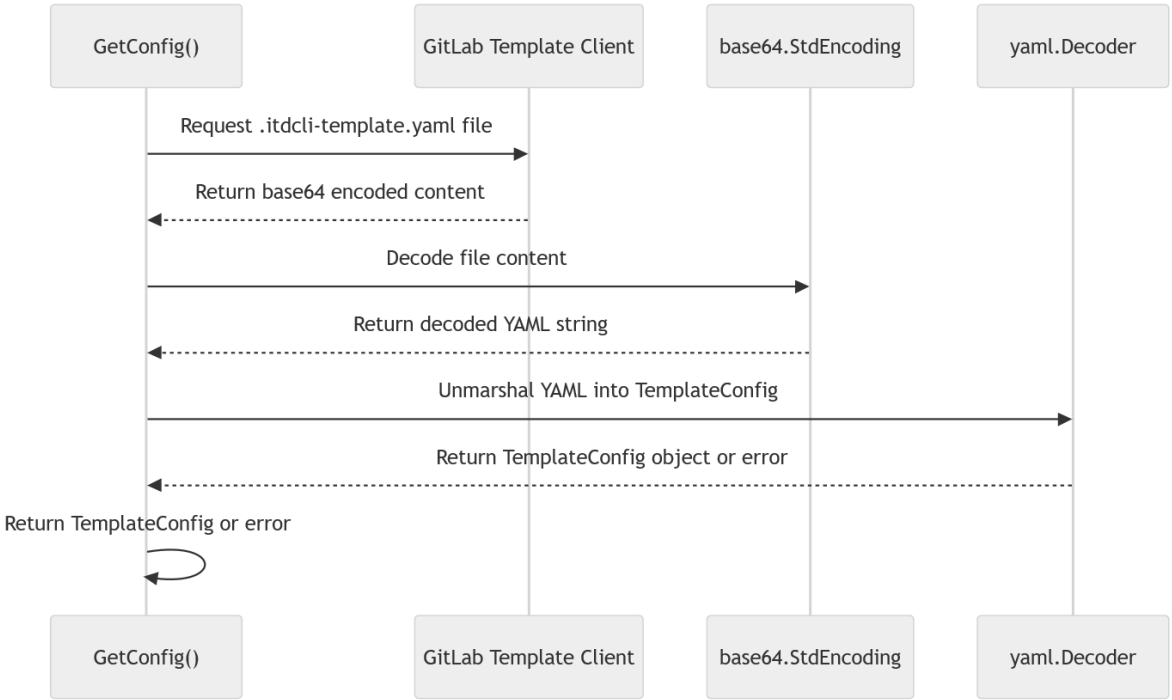

Abbildung 51: *Sequenzdiagramm zum Abrufen der itdcli-template.yaml-Datei aus dem Template Repository (eigene Darstellung)*

Der in Kapitel 7.4.4 dargestellte Befehl zum Erstellen einer Kubernetes-Applikation verwendet das Template-Paket, um die vorgefertigten Templates aus dem Git-Repository zu holen und mit Daten zu versehen, damit sie in das Deployment-Repository gepusht werden können. Da der Benutzer im Bootstrapping-Prozess das Template-Repository wie in Kapitel 7.4.2 dargestellt bereits konfiguriert hat, kann mit Hilfe der im Konzept definierten Datei „itdcli-tempate.yaml“ (Index-Datei) mit der Methode „template.GetConfig()“ auf das Template-Repository zugegriffen und die Datei abgerufen werden.

Der genaue Ablauf ist in Abbildung 51 beschrieben. Die Besonderheit hierbei ist, dass wie bei der Konfigurationsdatei der Inhalt der YAML-Datei in die entsprechende TemplateConfig-Struktur aus Abbildung 50 unmarshallt wird. Dadurch kann in der Business-Logik einfach durch die Datenstruktur navigiert werden und gleichzeitig ist die Typsicherheit der einzelnen Attribute gewährleistet.

Mit Hilfe der Indexdatei aus dem Template Repository können nun die einzelnen Templates geholt werden. Dazu wird die Methode „template.FetchTemplate(...) (string, error)“ verwendet. Das zugehörige Sequenzdiagramm ist in Abbildung 52 dargestellt. Dabei spielen vor allem die übergebenen „replacements“ eine wichtige Rolle, da die Platzhalter in den von GitLab gelieferten Template-Dateien mit dem Schema „%%PLACEHOLDER_NAME%%“ durch reale Daten ersetzt werden. Die Methode liefert am Ende eine vollständig gefüllte Template-Datei zurück, die direkt weiterverarbeitet werden kann. Ein Beispiel für die Platzhalter ist in Listing 3 visualisiert. Die Dort dargestellten Platzhalter werden dann z. B. durch den App Namen „Demo Go App 1“ ersetzt. Weitere Details dazu finden sich in Kapitel 7.4.4.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4      name: %%PLACEHOLDER_APP_NAME%%
5      namespace: %%PLACEHOLDER_NAMESPACE%%
6  spec:
7      replicas: %%PLACEHOLDER_REPLICAS%%
8      selector:
9      matchLabels:
10         app: %%PLACEHOLDER_APP_NAME%%
11     template:
12     metadata:
13         labels:
14         app: %%PLACEHOLDER_APP_NAME%%
15     spec:
16         containers:
17         - name: %%PLACEHOLDER_CONTAINER_NAME%%
18             image: %%PLACEHOLDER_CONTAINER_IMAGE%%
19             imagePullPolicy: Always
20     [...]
```

Listing 3: *Template-Datei für ein Kubernetes Deployment*

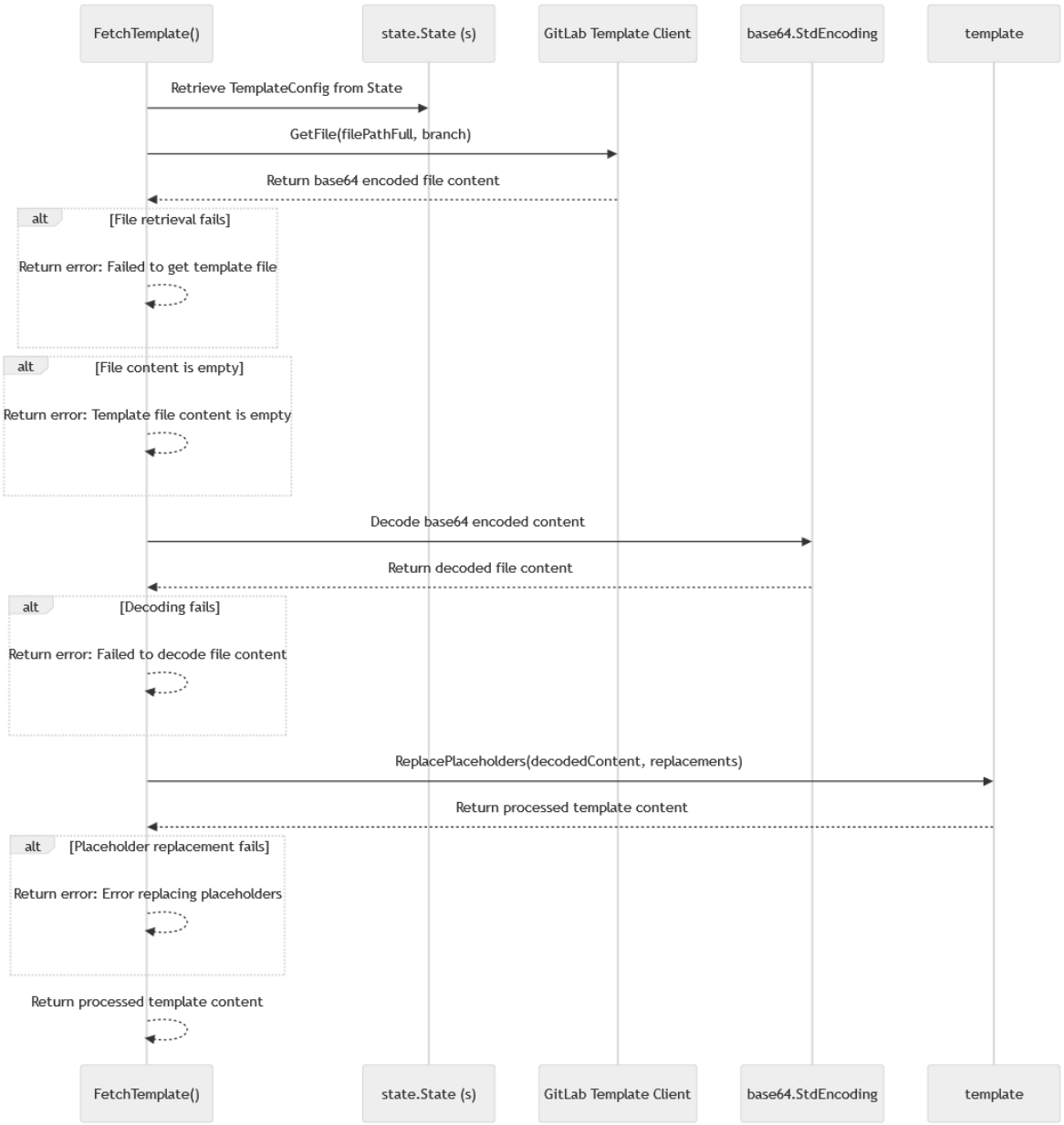

Abbildung 52: *Sequenzdiagramm zum Abrufen und Ausfüllen eines Templates aus dem Template Repository (eigene Darstellung)*

7.4 Befehlgruppen und Befehle

In diesem Abschnitt wird die Implementierung der Kernbefehle aus Tabelle 6 (fett markiert) der CLI eingegangen. Neben der Bootstrapping-Befehlsgruppe wird auch noch kurz auf die CRUD-ähnlichen Befehle (CREATE, READ, UPDATE und DELETE) wie template, deploy, cluster und config eingegangen.

7.4.1 Bootstrap: Initialisierung der CLI-Instanz

Mit dem Befehl „itdcli init“ wird die CLI für die erste Benutzung vorbereitet. Dazu werden, wie in Kapitel 7.2.1 beschrieben, der Vor- und Nachname sowie der Benutzername und die E-Mail-Adresse des Benutzers im geführten Modus mit Eingabeaufforderungen abgefragt. Dazu werden die in Tabelle 8 beschriebenen Flags definiert, die vom Benutzer entweder direkt oder über den geführten Wizard-Modus eingegeben werden können. Zusätzlich wird automatisch eine Instanz-ID im UUIDv4-Format generiert und in der Konfigurationsdatei gespeichert. Diese kann später zur eindeutigen Identifizierung von Benutzern oder Maschinen verwendet werden.

Flag	Kurz	Standard	Beschreibung
`--username`	`-u`	-	Your username
`--email`	`-e`	-	Your work email address
`--firstname`	`-f`	-	Your first name
`--lastname`	`-l`	-	Your last name

Tabelle 8: *Beschreibung der Kommandozeilen-Flags für den Bootstrap-Befehl Init*

Nachdem alle Eingaben gemacht wurden, werden die Daten mit der Methode „Set()“ der Konfiguration im State gesetzt und anschließend mit der Methode „Write()“ in der Konfigurationsdatei persistiert. Diese Daten werden später von den GitLab Clients verwendet, um die Benutzerinformationen für Git-Commits mit denen des Benutzers zu füllen. Dadurch wird verhindert, dass in der Commit-Historie z. B. nur „itdcli pushed to [branch]“ steht.

7.4.2 Bootstrap: Deployment- und Template-Repository

Die Befehle „itdcli bootstrap deploy-repo“ sowie „itdcli bootstrap template-repo“ starten den geführten Wizard-Modus zur Definition des Deployment- und Template-Repositories, die anschließend in der Konfigurationsdatei persistiert werden. Da beide Befehle nahezu identisch aufgebaut sind, werden sie nicht separat erläutert. Die folgende Implementierung gilt daher für beide Befehle. Das folgende Beispiel beschreibt den Befehl „itdcli bootstrap template-repo“.

Damit cobra den Befehl registrieren kann, muss als nächstes eine Methode erstellt werden, die bestimmte Parameter wie den Befehlsnamen, die Ausführungsmethode sowie eine Beschreibung des Befehls definiert. Dies geschieht in der Methode „NewBootstrapTemplateCmd(s state.State) *cobra.Command“ die im Main aufgerufen wird. In dieser Methode werden zusätzliche Flags definiert, um eine Dateneingabe über die Kommandozeile zu ermöglichen. In Tabelle 9 sind die Flags inklusive einer Kurzform aufgelistet.
Wenn der Benutzer den Befehl ausführt, werden ihm im geführten Modus die folgenden Fragen

Flag	Kurz	Standard	Beschreibung
`--url`	`-u`	`-`	URL of the Git instance
`--repository`	`-r`	`-`	Git Repository name or ID
`--branch`	`-b`	`-`	Repository branch to be used
`--token`	`-t`	`-`	GitLab Group/Personal Access Token
`--initialize`	`-i`	`false`	Initialize the template repository with the default itdcli-template.yaml configuration (default: `false`)

Tabelle 9: *Beschreibung der Kommandozeilen-Flags für die Bootstrap-Befehle Deployment- und Template-Repository*

bzw. Eingabeaufforderungen am Terminal angezeigt:

1. Please enter the url of the Git instance
2. Please enter the Git repository name or the ID
3. Please enter the Git branch name
4. Please enter the GitLab Group/Personal Access Token
5. Do you want to initalize the Template Repository with a new itdcli-template.yaml index file? (nur für „bootstrap template-repo“)

Sind alle Eingaben korrekt, wie z. B. die Überprüfung des korrekten URL-Formats, werden die Daten mit der Methode „Set()“ der Konfiguration im State gesetzt und anschließend mit der Methode „Write()“ in der Konfigurationsdatei persistiert. Dabei wird die Provider Config in der Kategorie „Template.Git“ mit den soeben gesetzten Daten gefüllt. Da derzeit nur GitLab implementiert ist, hat der Benutzer keine Auswahl, für welchen Provider er das Template Repository konfigurieren möchte. Dies wird aber in Zukunft möglich sein, wenn weitere Provider wie z. B. GitHub etc. benötigt werden.

7.4.3 Bootstrap: Setup der CI/CD-Pipeline für das Team

Das in Kapitel 5.6 vorgestellte CI-Pipeline-Konzept mit dem dazugehörigen ArgoCD-Team-Repository kann mit diesem Befehl konfiguriert werden. Dabei werden dem Benutzer verschiedene teamspezifische Fragen gestellt, um den von FluxCD installierten GitLab Runner, der sich im Team-Cluster befindet, zu konfigurieren. In Abbildung 42 wurden die GitLab Variablen erstmals in der Build Stage angezeigt. Diese werden nun automatisch von der CLI erzeugt. Dieses Kommando unterstützt nur das vorgestellte CI/CD Konzept, bei dem ein Team die zur Verfügung gestellten Pipeline Templates verwendet, die auch die entsprechenden Variablen verwenden.

Auch hier werden Flags definiert, die in Tabelle 10 zu sehen sind. Dieser Befehl unterstützt auch den Guided-Modus, indem interaktiv Fragen gestellt werden, um die oben genannten Flags auszufüllen. Dabei wird jede Frage direkt nach der Eingabe validiert, so dass eine fehlerhafte Eingabe direkt korrigiert werden kann und nicht der gesamte Prozess wiederholt

Flag	Kurz	Standard	Beschreibung
`--group`	`-g`	-	GitLab group name or ID
`--container-file-path`	`-C`	-	Path to the container file in your source code repositories
`--registry-password`	`-p`	-	Password for the container registry (WARNING: Must be at least 8 characters long and cannot contain spaces)
`--registry-project`	`-r`	-	Project name for the container registry
`--registry-url`	`-u`	-	URL of the container registry
`--registry-user`	`-U`	-	Username for the container registry
`--root-ca-path`	`-a`	-	Path to your Root CA file (*.pem or *.crt)
`--overwrite-variables`	`-o`	`false`	Overwrite existing variables

Tabelle 10: *Beschreibung der Kommandozeilen-Flags für den CI-Pipeline Setup Befehl*

werden muss.
Sind alle Fragen erfolgreich validiert und ausgefüllt, wird über die vom Benutzer eingegebene Gruppen-ID die entsprechende Gruppe über den GitLab Deployment Client gesucht und die Variablen automatisch über REST erstellt. Dies sieht dann so aus wie in Abbildung 53 dargestellt. Wenn dann eine Pipeline innerhalb dieser Gruppe ausgeführt wird, werden die Container-Images mit Hilfe von Kaniko automatisch in die dort angegebene Image-Registry hochgeladen.

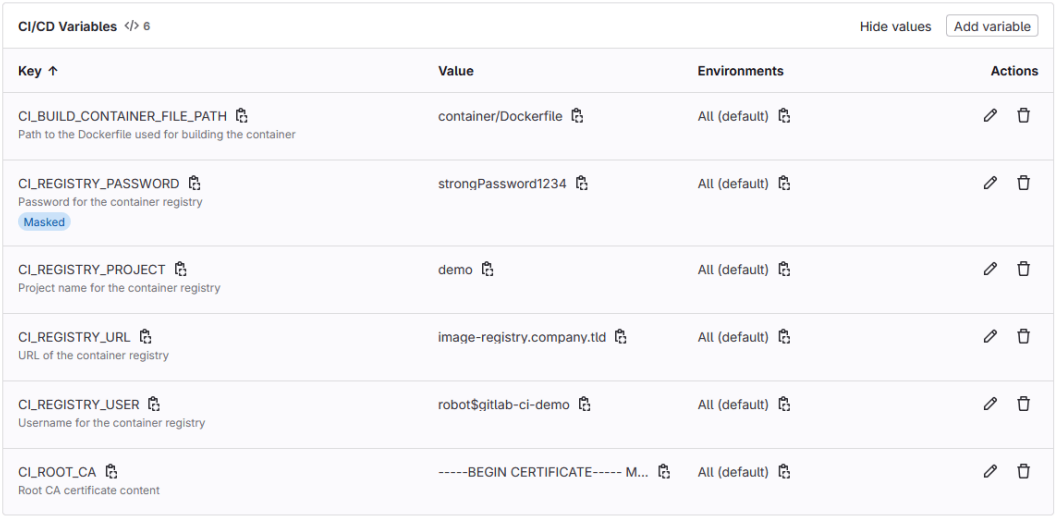

Abbildung 53: *Darstellung der Pipeline GitLab Runner Variablen (eigene Darstellung)*

Mit diesem Kommando wurde nun die Lücke geschlossen, um die CI/CD Pipeline in Verbindung mit der Kubernetes Plattform und der CLI zu nutzen. Der Befehl kann innerhalb weniger Minuten ausgeführt werden und stellt sicher, dass das Team arbeitsfähig ist. Durch die Verwendung von GitLab-Variablen gibt es auch keine Sicherheitsbedenken, da diese sensiblen Informationen nicht wie üblich in den Template-Dateien hartkodiert werden müssen.

Außerdem wird der Benutzer gewarnt, wenn die Variablen bereits für die angegebene Gruppe definiert sind. Ein versehentliches Überschreiben kann somit ausgeschlossen werden.

7.4.4 Deploy: Erstellen einer Kubernetes Applikation

Die Befehlsgruppe „deploy“ bildet den Kern der CLI. Im aktuellen Zustand hat der Benutzer die Möglichkeit, den Befehl „itdcli deploy kubernetes“ auszuführen, der es ihm erlaubt, jede Anwendung, die per Container gestartet werden kann, in einem Dev- oder Team-Cluster zu deployen. Dabei wurden die Kubernetes Manifest Templates gezielt so erstellt, dass sie durch eine minimale generische Struktur den größtmöglichen Nutzen ohne manuelle Anpassungen bieten. Gerade für Einsteiger, die noch nie ein Deployment mit Kubernetes ohne Helm Chart o.ä. erstellt haben, könnte dies sonst eine Herausforderung darstellen. Wie bei den anderen Befehlen der Bootstrap-Befehlsgruppe stehen sowohl der interaktive als auch der Terminal-Modus zur Verfügung. Die verfügbaren Flags sind in Tabelle 11 definiert.

Flag	Kurz	Standard	Beschreibung
`--path`	`-p`	-	Path to the application source code which will be deployed
`--name`	-	-	Name of the deployment
`--replicas`	-	`1`	Amount of replicas to be created
`--namespace`	-	-	Namespace where the application should be deployed
`--node-selector-key`	-	-	Node selector key
`--node-selector-value`	-	-	Node selector value
`--ingress-url`	-	-	URL where the application should be accessible
`--ingress-class`	-	-	Ingress class for the application
`--volume-size`	-	-	Size of the volume (e.g., 1Gi, 500Mi)
`--storage-class`	-	-	Storage class for the volume
`--storage-access`	-	-	Storage access mode for the volume
`--mount-path`	-	-	Mount path for the volume
`--service-port`	-	-	Port where the application listens on
`--quiet`	`-q`	`false`	Disable interactive mode where the user is prompted for deployment details

Tabelle 11: *Beschreibung der Kommandozeilen-Flags für den Kubernetes Deployment Befehl*

Der Benutzer hat die Möglichkeit, den interaktiven Modus über das „–quiet“-Flag vollständig zu deaktivieren. Dadurch kann der Nutzer gezielt Fragen überspringen, wodurch automatisch der Standardwert für das jeweilige Flag genommen wird. Dieser wird dann 1:1 in die Vorlage übernommen. Dieser Modus ist vor allem für erfahrene Anwender sinnvoll, da oft nicht alle Parameter für einen Einsatz benötigt werden und so Zeit gespart werden kann. Für alle anderen Benutzer werden im Geführten Modus Fragen und Eingabeaufforderungen am Terminal

angezeigt. Dabei werden auch alle Eingaben direkt überprüft und bei Fehleingaben wiederholt.

Folgende Fragen und Anweisungen werden dem Benutzer gestellt:

1. Enter the path to the source code of the application to be deployed (default: .):
2. Enter the name of the deployment (e.g. My Demo App 1):
3. Enter the amount how many pods (replicas) should be created for your application (default: 1):
4. **Select a namespace where your application should be deployed or enter a new one:**
5. **Select a node selector key for your application:**
6. **Found the following ports in your Dockerfile: [PORT]. Would you like to use it?**
7. Should an ingress be created? (y/N):
8. Enter the URL where your application should be accessible in your browser:
9. **Select an IngressClass for your application:**
10. Should a volume be created? (y/N):
11. Enter the volume size (e.g., 1Gi, 500Mi):
12. Enter the mount path for the volume where you can access it within the container later (e.g. /mnt/app):
13. **Select a storage access mode for your volume:**
14. **Select a storage class for your volume:**

Eine Besonderheit stellen die fett markierten Ausgaben dar. Neben der Frage selbst werden im Hintergrund über den vorkonfigurierten Kubebernetes-Client in Echtzeit die verschiedenen Storage- und Ingress-Classes sowie alle verfügbaren Nodes und Namespaces abgerufen und in einer Liste zusammen mit der Frage ausgegeben. Der Benutzer hat dann die komfortable Möglichkeit, aus einer nummerierten Liste die passende Option auszuwählen. Für die Auswahl des Namespaces würde dies wie folgt aussehen:

```
Select a namespace where your application should be deployed or enter a new one:
    [1]: cert-manager
    [2]: default
    [3]: flux-system
    [4]: kamaji-system
    [5]: kube-node-lease
    [6]: kube-public
    [7]: kube-system
    [8]: piraeus-datastore
    [9]: tenants
Select an option by number or enter a new one:
```

Listing 4: *Frage zur Namespace Auswahl in der CLI*

Man erhofft sich dadurch eine höhere Effizienz und weniger Tippfehler. Außerdem ist es für

Anfänger einfacher zu verstehen, was z. B. ein Namespace ist. Wie in Listing 4 in der letzten Zeile zu sehen ist, hat der Benutzer auch die Möglichkeit, direkt einen neuen Namespace anzulegen, so dass auch hier die volle Flexibilität gewährleistet ist.

Automatische Portsuche für den Service Port

Eine weitere Besonderheit ist die Angabe des Service Ports. Da die Team Cluster einen NGINX Reverse Proxy verwenden, der die Ports für HTTP und HTTPS auf die lokal im Cluster verfügbaren Service Ports der Pods weiterleitet, muss dieser bei jedem Deployment angegeben werden. Da die Nutzerbefragung gezeigt hat, dass viele Mitarbeiter Docker kennen oder nutzen, kann davon ausgegangen werden, dass diese mit der Handhabung von Dockerfiles vertraut sind. Dort wird häufig über den Befehl „EXPOSE [PORT]“ der entsprechende Applikationsport angegeben. Da der Benutzer in der ersten Frage den Pfad zum Quellcode angegeben hat, wurde ein Skript entwickelt, welches rekursiv alle Verzeichnisse nach den Dateien „Dockerfile“ und „.dockerfile“ durchsucht. Da diese Datei für die CI/CD-Pipeline benötigt wird, ist davon auszugehen, dass zum Zeitpunkt der Ausführung dieses CLI-Kommandos eine solche Datei bereits existiert. Daher wird, wie im Sequenzdiagramm in Abbildung 54 dargestellt, mit Hilfe des Regex-Patterns „(?i)^EXPOSE\s+(\d+)“ versucht, einen oder mehrere Ports zu finden. Dieses von der Methode „SearchPortsInDockerfile()“ zurückgegebene Array wird dann dem Benutzer wie in Listing 4 gezeigt angezeigt. Wenn kein Port gefunden wird, muss der Benutzer den Port manuell angeben.

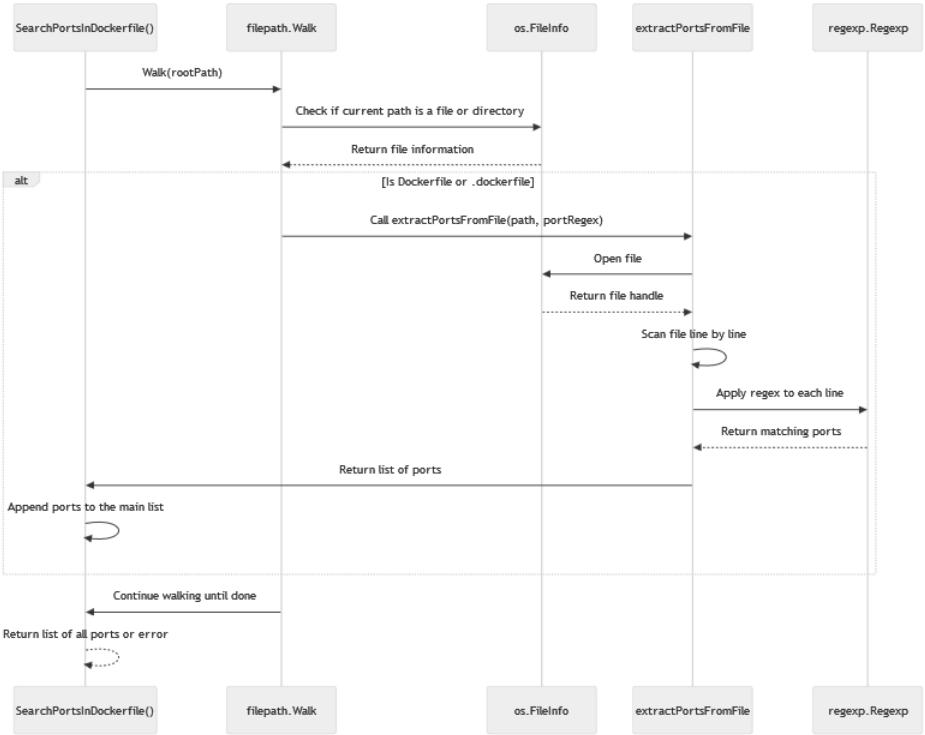

Abbildung 54: *Sequenzdiagramm zur Ermittlung der Docker Ports aus einem Dockerfile (eigene Darstellung)*

Erstellen des ArgoCD Deployments im Deployment Repository

Nachdem alle Fragen erfolgreich beantwortet wurden, werden die entsprechenden Kubernetes Manifeste aus dem Template Repository geholt und mit den Daten gefüllt. Diese werden dann in das entsprechend konfigurierte Deployment Repository, welches das ArgoCD Team Repository ist, hochgeladen. Dort wird anhand des Benutzernamens aus der Konfigurationsdatei ein Ordner mit dessen Namen angelegt, in dem wiederum ein weiterer Ordner mit dem Namen des Deployments angelegt wird. In diesem werden die Kubernetes Manifeste abgelegt. Zusätzlich werden für ArgoCD noch die Application Manifeste auf Basis von Templates erzeugt, die auf den soeben angelegten Ordner im Git-Repository zeigen. Sobald der Push durch die CLI erfolgt ist, versucht ArgoCD diese Applikation im Cluster zu erzeugen. Da jedoch noch kein Container-Image durch die Pipeline erzeugt wurde, schlägt dieser Vorgang auf unbestimmte Zeit fehl, bis das passende Image in der Registry verfügbar wird.

Die CLI legt daraufhin eine generische Datei „.gitlab-ci.yml“ im Hauptverzeichnis des angegebenen Quellcodes ab. Das Kommando meldet daraufhin eine erfolgreiche Statusmeldung zurück, die dem Benutzer signalisiert, dass die Anwendung nun zum Pushen bereit ist. Die CI/CD Pipeline startet dann automatisch in GitLab und pusht das Container Image in die definierte Container Registry aus dem Bootstrapping Prozess.

7.4.5 CRUD-Befehle zur Steuerung und Konfiguration der CLI

Neben den Kernkomponenten verfügt die CLI auch über einige administrative Befehle, die in der Tabelle 6 nicht fett markiert sind. Diese basieren auf dem CRUD-Prinzip, bei dem einzelne Konfigurationsblöcke direkt per Terminal verändert werden können. Da diese Kommandos im Wesentlichen die bereits vorgestellten Pakete und Methoden verwenden, wird auf eine genaue Beschreibung der Implementierung verzichtet. Dies gilt für die Befehlsgruppen *template*, *config* und *cluster*. Insbesondere die Befehlsgruppe cluster stellt im Wesentlichen nur eine Kopie der Befehle der kubectl dar. So würde beispielsweise der Befehl „itdcli cluster list-contexts“ das in Listing 5 dargestellte Ergebnis liefern, das mit dem Befehl „kubectl config get-contexts“ identisch ist. Verwendet wird dazu der Kubernetes Client aus dem State.

```
1 CURRENT   NAME                          CLUSTER     AUTHINFO          NAMESPACE
2 *         kubernetes-admin@project1  project1    kubernetes-admin  default
3 [...]
```

Listing 5: *Anzeigen der verfügaren Kubernetes Contexte mit der CLI*

7.5 Automatische Cross-Plattform Unterstützung durch goreleaser

Damit die CLI problemlos auf verschiedenen Plattformen ausgeliefert werden kann, wird das Tool „goreleaser“ eingesetzt. Dieses Tool ermöglicht eine automatische Versionsverwaltung sowie die Bereitstellung verschiedener kompilierter Versionen der CLI. Dies ist unter anderem auch zur Erfüllung der Anforderung FA16 notwendig. Dazu wird im Hauptverzeichnis des CLI-Quellcodes eine Datei „.goreleaser.yml“ erstellt, mit der der goreleaser die CLI für die folgenden Architekturen und Betriebssysteme kompilieren soll:

- **OS**: FreeBSD, Windows, Linux
- **Architektur**: amd64, arm, arm64, i386

Außerdem wird beim Kompilieren die aktuelle Versionsnummer in die Datei „version.go“ eingetragen, so dass beim Anzeigen der Version mit dem Befehl „itdcli version“ immer die aktuell verwendete Go-Version und CLI-Version angezeigt wird. Eine manuelle Anpassung ist somit nicht mehr notwendig. Die fertigen Builds können dann optional automatisch in getaggter Form in Git veröffentlicht und von den Nutzern heruntergeladen werden. Auch eine kryptographische Signierung mit einem Private Key ist für den späteren produktiven Einsatz möglich.
In der kompilierten Version wird dem Benutzer also folgendes angezeigt, wenn er den Befehl „itdcli version“ aufruft:

```
1 version.BuildInfo{Version:"v1.0.0", GoVersion:"go1.23.0"}
```

Listing 6: *Anzeigen der aktuellen Versionsnummer in der CLI*

8 Evaluation

In diesem Kapitel werden die implementierte Kubernetes-as-a-Service Plattform evaluiert und die Forschungsfragen beantwortet.

8.1 Vorgehen und Einschränkungen

Da im Rahmen dieser Arbeit aufgrund der Komplexität und Vielfalt der zu bearbeitenden Themen aus zeitlichen Gründen keine User-Study mit der implementierten Kubernetes-Plattform durchgeführt werden konnte, besteht die folgende Bewertung der Kubernetes Self-Service Plattform in erster Linie aus den umgesetzten bzw. nicht oder nur teilweise umgesetzten Anforderungen sowie einem Rückblick auf die ausgewählten Technologien und umgesetzten Konzepte, welche für die Umsetzung entworfen wurden. Dabei wird geprüft, ob alle mit „Must have (M)“ priorisierten Anforderungen vollständig umgesetzt werden konnten. Die daraus resultierenden Ergebnisse werden dann in Kapitel 9.1 zur Beantwortung der Leitfragen verwendet.

8.2 Technologieentscheidungen

In diesem Abschnitt werden die ausgewählten Technologien und Frameworks, die in der Kubernetes-Plattform zum Einsatz kommen, retrospektiv bewertet.

8.2.1 Infrastruktur und Multi-Tenancy

Die Auswahl der Technologien für den Aufbau der Infrastruktur erfolgte durch Vergleiche in Kapitel 5.1 zwischen den derzeit modernsten Technologien in den jeweiligen Bereichen. Da bei den Vergleichen auch die funktionalen Anforderungen berücksichtigt wurden, konnte die technische Umsetzung auf der in Kapitel 5.2 beschriebenen Testinfrastruktur nahezu problemlos erfolgen. Die Entscheidung für die Open-Source-Lösung von GitLab in Verbindung mit der Integration von FluxCD und ArgoCD ermöglichte den Einsatz von GitOps und ein harmonisches Zusammenspiel zwischen der Kubernetes-Plattform und der Steuerung durch GitOps.

Die Integration von Kamaji und Capsule im Zusammenspiel erwies sich in der Umsetzung zunächst herausfordernd, durch den Anwendungsfall mit Team- und Dev-Cluster in einem mussten gezielt Konzepte entwickelt werden, um eine saubere GitOps-Struktur zu schaffen, damit eine strikte Trennung der einzelnen Bereiche vom Management Cluster bis hin zum Dev Cluster möglich ist. Insgesamt konnte dies mit den ausgewählten Technologien in sehr

gutem Maße erreicht werden, was u. a. auch dem GitOps zentrierten Konzept von Kamaji und Capsule zu verdanken ist. Die geforderte „harte“ und „weiche“ Mandantenfähigkeit konnte mit Kamaji und Capsule erreicht werden, was durch die derzeit aktuellen Konzepte wie Virtual Control Planes und Namespace Isolation umgesetzt wurde. Ein ausführlicher Praxistest konnte im Rahmen dieser Arbeit nicht durchgeführt werden, jedoch zeigte sich bereits bei der Implementierung das Potential dieser beiden Technologien, weshalb einem zukünftigen Produktivbetrieb nichts im Wege stehen sollte.

8.2.2 LinSTOR als Container-Storage-Interface (CSI)

Der Einsatz von LinSTOR hat sich durch die einfache und zufriedenstellende Integration mit Kubernetes für den in dieser Arbeit definierten Rahmen bewährt. Die bereitgestellten Speichermedien der einzelnen Master Nodes konnten erfolgreich in einen LinSTOR Cluster integriert werden, sodass eine gewisse Daten- und Ausfallsicherheit gewährleistet werden konnte. Auch das Anlegen einer StorageClass war mit dem piraeus-Operator für Kubernetes problemlos möglich. Insgesamt fügte sich LinSTOR optimal in diesen Anwendungsfall ein. Auch der zukünftige produktive Einsatz sollte weiterhin in dieser Art und Weise möglich sein, da durch die Architektur von LinSTOR weitere Storage Pools ohne großen Aufwand hinzugefügt werden können. Allerdings muss für den produktiven Einsatz nochmals geprüft werden, ob nicht ein externes Speichermanagement mit einem dedizierten Storage-Cluster sinnvoller wäre, da man Cluster, die Workloads verarbeiten, grundsätzlich stateless halten sollte. Im Falle eines Totalausfalls könnte so schneller auf eine Alternative ausgewichen werden.

8.2.3 CI/CD-Pipeline

Durch die Implementierung der CI/CD-Pipeline in Kapitel 5.6 mit dem GitLab Runner und Kaniko innerhalb der jeweiligen Kubernetes-Cluster wurde die Möglichkeit geschaffen, ein generisches Pipeline-Konzept zu erstellen, das mit nahezu jeder Container-fähigen Anwendung funktioniert. Da den Konfigurationsmöglichkeiten des Kubernetes GitLab Runners keine Grenzen gesetzt sind, wurde für diesen komplexen Anwendungsfall eine stabile und zukunftssichere Lösung gewählt, die sich optimal in die technische Umsetzung der Kubernetes Plattform einfügt. So können in Zukunft problemlos weitere Stages für z. B. Testing, SecOps, etc. hinzugefügt werden, die dann automatisch über das Pipeline Template Repository allen Tenants zur Verfügung stehen.

8.2.4 Cloud-basierte Softwareentwicklung

Um die Vorteile der Hybrid Cloud auch für Entwickler nutzbar zu machen, wurde das Open-Source-Tool DevPod in die Self-Service-Plattform integriert, das sich durch seine interak-

tive und leicht verständliche Desktop-GUI deutlich von anderen Lösungen abhebt. Da die Kubernetes-Plattform für den Endanwender einfach zu bedienen sein sollte, eignete sich DevPod optimal, da mehrere Anforderungen mit einer Lösung abgedeckt werden konnten. Die wichtigste Anforderung war die Kompatibilität mit allen gängigen Betriebssystemen und Entwicklungsumgebungen. Dies konnte DevPod über die nativen Remote Development Provider der einzelnen IDEs lösen, so dass die aktuell verwendete IDE problemlos weiterverwendet werden konnte. Ein weiteres wichtiges Feature war eine einfache und im besten Fall automatische Konfiguration, so dass diese von den Plattformadministratoren vorkonfiguriert werden kann. Dies konnte durch die integrierte YAML-Konfigurationsdatei von DevPod zufriedenstellend gelöst werden, da diese in generischer Form über Git allen Teams zur Verfügung gestellt werden kann. Zusammenfassend konnte auch hier die derzeit beste Lösung am Markt für diesen speziellen Anwendungsfall ausgewählt werden. Die Praxis muss jedoch zeigen, ob die Vielzahl der unterschiedlichen Softwareprojekte in den verschiedenen Sprachen mit diesem Tool harmoniert.

8.3 Evaluation der funktionalen Anforderungen

In diesem Abschnitt werden die einzelnen funktionalen Anforderungen aufgegriffen und auf ihre korrekte Umsetzung überprüft. Dazu werden die umgesetzten technischen Maßnahmen den jeweiligen Anforderungen zugeordnet. Erfüllt eine Maßnahme mehrere Anforderungen, werden diese entsprechend zusammengefasst und mit einer geeigneten Überschrift versehen, die die Anforderungsgruppe beschreibt.

8.3.1 Umgesetzte Anforderungen

FA1, FA10 und FA21: Nutzung des Clusters für Anwendungen

Mit diesen Anforderungen sollte sichergestellt werden, dass die Entwickler in ihrer täglichen Arbeit nicht behindert werden und ohne fremde Hilfe Deployments in ihren Clustern durchführen können. Dabei sollten ihnen die im Cluster vorhandenen Ressourcen wie Ingress- und StorageClasses zur Nutzung zur Verfügung stehen. Ebenso sollte es möglich sein, Applikationen in bestimmten Clustern zu deployen, wenn ein Entwickler mehr als einen Dev Cluster besitzt. Diese Anforderungen konnten durch die Entwicklung des itdcli mit dem darin integrierten „deploy“-Modus, der in Kapitel 7.4.4 beschrieben ist, umgesetzt werden. Durch die Auswahl des Kubernetes Contexts vor der Ausführung des Deployment-Kommandos kann ein bestimmter Cluster ausgewählt werden. Während des Guided Wizards kann der Benutzer auch in Echtzeit die entsprechende Ingress- und StorageClass des jeweiligen Clusters auswählen. Dies wurde in Listing 4 veranschaulicht. Da die CI/CD Pipeline nur für den Build und Push des Container Images auf die vom Team definierte Container Registry zugreift, ist diese clusterunabhängig, da jeder Dev Cluster automatisch darauf zugreifen kann.

FA2: Zugriff auf den Cluster als Entwickler

Ziel dieser User Story war es, den Zugriff auf alle Cluster mit dem standardisierten RBAC von Kubernetes mittels kubeconfig zu ermöglichen. Durch die Integration von Capsule und Kamaji konnte diese Anforderung vollständig erfüllt werden, da jeder Tenant Admin eines Team Clusters sowie die Tenants von Capsule jeweils eine eigene kubeconfig erhalten. Durch den Capsule Proxy werden alle RBAC-Probleme, die durch Multi-Tenancy in Kubernetes entstehen, eliminiert. So kann ein Dev Cluster wie ein Dedicated Cluster verwaltet werden. Gleiches gilt für Tenant Admins, die über die Admin Kubeconfig einen vollwertigen Kubernetes Cluster steuern können.

FA3, FA4, FA5, FA6, FA7, FA9, FA11, FA15 und FA22: Steuerung über GitOps

Diese Anforderungen machten den größten Teil dieser Arbeit aus, um die gesamte Steuerung der Kubernetes-Plattform mit dem GitOps-Ansatz zu ermöglichen. Dazu wurden die entsprechenden Repositories in Kapitel 5.3.3, 5.3.2 und 5.4.2 angelegt, welche eine Trennung in Infrastrukturadministration (Fleet Repository), Plattformadministration (Cloud Repository) und Team Cluster Administration (Team Repository) ermöglichen. Auf die ersten beiden Repositories haben nur die Plattformadministratoren Zugriff, was über das Rollenmanagement von GitLab gesteuert wird. Jedes Team erhält für den dafür angelegten Team Cluster ein eigenständiges Repository, auf das nur die Teammitglieder Zugriff haben. Dort können die Teammitglieder die Rollen in Git nach Belieben vergeben, wobei ausdrücklich empfohlen wird, einen Tenant Admin zu definieren, der die Verwaltung des Team Clusters übernimmt. Innerhalb dieser Repositories wird durch die Integration von FluxCD sichergestellt, dass jede auf dem Markt erhältliche Kubernetes Applikation automatisch deployt werden kann. Dies geschieht in der Regel über einen Helm Chart. Im Kontext der Kubernetes Plattform werden diese Applikationen als „Shared Apps“ bezeichnet und stehen sowohl auf Management- als auch auf Team-Ebene zur Verfügung. So wie es die Anforderungen FA15 und FA22 verlangen. Mit Hilfe von Capsule können innerhalb des Team-Repositories in wenigen Minuten neue Tenants erstellt werden, die automatisch von FluxCD mit den Shared Apps provisioniert werden.

FA14, FA24: CI/CD-Pipeline und Anwendungsmöglichkeiten

Durch das enge Zusammenspiel zwischen itdcli und der CI/CD-Pipeline in GitLab, die von itdcli auch automatisch konfiguriert werden kann, wurde eine Möglichkeit geschaffen, auch Kubernetes-Einsteigern die Nutzung der Kubernetes-Plattform zu ermöglichen. Die Pipeline wurde, wie in Kapitel 5.6 erläutert, nach einem generischen Konzept entwickelt, wodurch auch die Verwendung beliebiger Betriebssysteme, wie in der Anforderung FA24 gefordert, möglich ist. Es gilt jedoch die Einschränkung, dass es sich bei dem Betriebssystem um ein containerfähiges Betriebssystem handeln muss, das über herkömmliche Dockerfiles gestartet werden kann. Denkbar wäre in Zukunft aber auch die Integration der Technologie „KubeVirt“, die innerhalb von Kubernetes vollwertige virtuelle Maschinen erzeugen kann. Auf diesen könnten

dann z. B. Windows Server gestartet werden. Dies wurde jedoch im Rahmen dieser Arbeit nicht behandelt. Da es sich aber nur um eine Anforderung mit der Priorisierung „Could have (C)“ handelt, ist diese Anforderung zumindest teilweise erfüllt.

FA8: Weitergabe der Tenant Admin Rolle

Wenn ein Teamleiter nicht die Rolle des Tenant Admins für einen Kamaji-Cluster übernehmen möchte, kann er die Rolle des Tenant Admins einem bestimmten Teammitglied zuweisen, indem er einfach die Rolle „Owner“ im GitLab-Rechtesystem zuweist. Dieses kann dann Änderungen am Team-Repository vornehmen und erhält über die Admin-Kubeconfig auch die notwendigen Rechte, um Capsule Tenants zu verwalten oder neue Worker Nodes zum Cluster hinzuzufügen. Theoretisch könnte also jeder Entwickler Tenant Admin werden, wenn dies im Team gewünscht wird. Allerdings steigt dadurch die Gefahr eines Totalausfalls des Team-Clusters durch fehlerhafte Konfiguration oder manuelle Änderungen, die von FluxCD nicht ohne weiteres durch Reconciliation rückgängig gemacht werden können.

FA12, FA13, FA14, FA16, FA17: Cloud-basierte Softwareentwicklung

Cloudbasierte Softwareentwicklung war ein wesentlicher Bestandteil dieser Self-Service-Plattform. Aus den gestellten Anforderungen, die sich unter anderem aus den Umfrageergebnissen in Kapitel 3.4 ergaben, und den Überlegungen, wie dies mit GitOps umgesetzt werden könnte, wurde in Kapitel 5.5 ein umfassendes Konzept entwickelt. Dadurch wurde es möglich, die Vorteile, die Kubernetes und die Cloud im Allgemeinen mit sich bringen, wie Ausfallsicherheit, mehr Ressourcen und das Arbeiten von Thin Clients aus, den Entwicklern als optionales Modell zur Verfügung zu stellen. Besonderes Augenmerk wurde darauf gelegt, eine einfache Adoption dieses Systems zu ermöglichen, indem ein hybrides Arbeiten weiterhin möglich ist und die bevorzugte IDE weiterhin verwendet werden kann. Darüber hinaus ermöglicht die Kompatibilität von DevPod mit Windows, Linux und Mac die freie Wahl des Betriebssystems. Durch die Implementierung eines Shared Repository in Kapitel 5.5.2 ist eine einfache Konfiguration und Installation von DevPod auf der lokalen Maschine gewährleistet. Dieses Konzept kann, wie in Kapitel 5.5.2 beschrieben, auf Team-Ebene erweitert werden, indem das Repository geforkt wird und eigene Devcontainer-Dateien erzeugt werden.

FA15, FA19: Verwaltung der Infrastruktur für das Management Cluster

Durch die GitOps-Integration können Plattformadministratoren die gesamte Bare-Metal-Infrastruktur über GitOps und die talosctl verwalten. Das in Kapitel 5.3.2 gezeigte Fleet-Repository bildet dabei die erste Grundlage für IaC, jedoch ist durch die manuelle Verwendung von talosctl zur Konfiguration der Nodes noch keine hundertprozentige IaC möglich. Hier sollte in Zukunft noch eine Anpassung von Ansible in Verbindung mit Talos oder Proxmox erfolgen, um eine vollständige Verwaltung durch Git zu ermöglichen. Anders verhält es sich mit dem in Kapitel 5.3.3 dargestellten Cloud Repository, das mit FluxCD vollständig in Git verwaltet werden kann. Damit konnte die Anforderung FA19 vollständig umgesetzt

werden. FA19 ist prinzipiell ebenfalls umgesetzt, jedoch wie oben erwähnt noch mit Raum für Verbesserungen. Da diese Anforderung jedoch eine niedrigere Priorität hat, ist dies noch vertretbar.

FA18: Datenschutz für Team Cluster

In den Umfrageergebnissen wurden viele Bedenken bezüglich des Datenschutzes geäußert. Diese Anforderung sollte daher sicherstellen, dass die persistenten Daten innerhalb eines Team Clusters nicht nach außen gelangen können. Dies konnte durch die Integration von LinSTOR innerhalb jedes Team Clusters erreicht werden. Dadurch ist es möglich, wie in Kapitel 5.1.4 beschrieben, die dedizierten Speichermedien der Worker Nodes eines jeweiligen Team Clusters für die StorageClass zu verwenden. Somit ist es bei einer normalen Installation und Nutzung des Kamaji Clusters unmöglich, dass Daten mit anderen Tenants in Kontakt kommen. Wichtig zu erwähnen ist, dass gewisse Netzwerkisolationen und Pod Security Policies eingehalten werden müssen, damit ein Angriff über das Netzwerk keinen Zugriff auf sensible Inhalte ermöglicht. Dies wurde jedoch im Rahmen dieser Arbeit nicht weiter behandelt, da es auch stark von der Nutzung des Clusters und den darauf laufenden Workloads abhängt.

8.3.2 Nicht oder nur teilweise umgesetzte Anforderungen

Dieser Abschnitt beschreibt die nicht umgesetzen funktionalen Anforderungen. Diese konnten aus zeitlichen Gründen nicht oder nur teilweise in die Kubernetes-Plattform integriert werden.

FA20: Darstellung der verfügbaren Ressourcen des Dev Clusters

Zum aktuellen Stand ist es nur möglich die zur Verfügbaren Ressourcen über die kubectl oder mit einer GUI Anwendung wie Lens darzustellen. Ebenfalls wäre es möglich diese noch manuell in der jeweiligen Tenant-Konfiguration von Capsule in Erfahrung zu bringen. Da dies jedoch nur mit umwegen und externen Tools möglich ist, wurde diese Anforderung als nicht umgesetzt gesehen. Es wäre denkbar, diese Anforderung in die itdcli mitaufzunehmen, welche durch den Kubernetes Client die vorhandenen Ressourcen ausließt und dem Nutzer entweder grafisch oder per Terminal zu Verfügung stellt.

FA23: SecOps-Integration in der CI/CD-Pipeline

Durch den generischen Aufbau der CI/CD Pipeline wurden bereits erste Bausteine geschaffen, um die klassischen SecOps Funktionalitäten, die in den Grundlagen in Kapitel 2.2.3 aufgelistet sind, mit relativ überschaubarem Zeitaufwand zu ergänzen. Die Anforderung konnte jedoch teilweise umgesetzt werden, da in der Container Registry u. a. durch Trivy die hochgeladenen Container in regelmäßigen Abständen auf Sicherheitslücken überprüft werden. Außerdem werden die Container-Images mit cosign signiert (siehe Abbildung 42), so dass diese später bei Bedarf in Kubernetes verifiziert werden können. Durch die Verwendung des GitLab Runners

und der Pipeline Templates wäre es in Zukunft denkbar, weitere Stages vor der Build Stage zu implementieren, die klassische Tests wie Secret Scanning oder statische Codeanalysen durchführen.

FA25: Automatisierte Backup-Lösung für LinSTOR

Automatisierte Backups der Persistent Volumes der einzelnen LinSTOR-Cluster konnten aus Zeitgründen nicht umgesetzt werden. Da dieses Thema nicht in direktem Zusammenhang mit der Self-Service Plattform steht, wurde es auf eine niedrige Priorität gesetzt. Sollte die Plattform jedoch in den Produktivbetrieb gehen, muss im Vorfeld eine sichere Backup-Strategie erarbeitet werden. Hier könnte die 321-Backup-Strategie eingesetzt werden, so dass die Daten auch außerhalb des Clusters/Gebäudes gesichert werden. Dies sollte jedoch mit dem von LinSTOR verwendeten DRBD problemlos möglich sein. Beispielsweise könnten zusätzliche Backup-Cluster eingerichtet werden, auf die in regelmäßigen Abständen der gesamte Content über das Netzwerk übertragen wird.

8.4 Evaluation der nicht-funktionalen Anforderungen

Dieser Abschnitt umfasst die Evaluation der definierten nicht-funktionalen bzw. Qualitäatsanforderungen. Diese wurden so gewählt, dass diese eine gewisse Richtung vorgeben, welche Technologien und Ziele die Kubernetes-Plattform letztendlich erfüllen soll.

QA1: Umgesetze Isolation innerhalb eines Team Clusters

Ziel dieser Anforderung war es, eine ausreichende Isolation zwischen den Team- und Dev-Clustern zu gewährleisten, damit sensible Daten jederzeit geschützt sind. Es wurden zunächst keine konkreten technologischen Operationen vorausgesetzt, um durch Vergleiche und Tests die optimale Lösung zu evaluieren. Wie in Kapitel 5.1.2 dargestellt, konnte dies mit den Technologien Kamaji und Capsule für dieses Anwendungsszenario optimal erreicht werden. Zwischen den Team-Clustern besteht eine vollständige „harte“Isolation, die mit zwei dedizierten Bare-Metal-Clustern vergleichbar ist. Eine Kommunikation zwischen den Clustern über konventionelle Methoden ist nicht möglich. Innerhalb der Teamcluster wurden bestimmte Prämissen vorausgesetzt, wie z.B. dass sich die Teammitglieder untereinander kennen und vertrauen. Aufgrund dieser Designentscheidung wurde auf eine harte Isolation verzichtet und stattdessen auf eine Lösung mittels Namespace Isolation gesetzt. In diesem Fall Capsule. Damit kann eine ausreichende Isolation für Entwicklungs- und Testzwecke innerhalb eines Teams gewährleistet werden.

QA2: Umsetzung mit kostenlosen Open-Source-Technologien

Durch diese Vorgabe war es nicht möglich, kommerzielle Softwarelösungen o.ä. für die Implementierung der Kubernetes-Plattform zu verwenden. Grund für diese gewisse Einschränkung

war die Frage, ob eine derart komplexe Plattform zum jetzigen Zeitpunkt mit freien Mitteln zufriedenstellend umgesetzt werden kann. Da Cloud-Ressourcen in Public Clouds wie AWS sowie für On-Premise-Hosting in der Anschaffung bereits sehr kostenintensiv sind, musste eine Lösung gefunden werden, um auch kleinen und mittelständischen Unternehmen die Möglichkeit zu geben, in Kubernetes zu investieren, da die Einstiegshürde grundsätzlich bereits sehr hoch ist. Im Rahmen dieser Arbeit konnte eine komplett kostenfreie Plattform implementiert werden, die zur Gänze auf Open-Source-Technologien basiert, die zum jetzigen Zeitpunkt alle in regelmäßigen Abständen Updates erhalten.

QA3: Einfache und intuitive Bedienung des CLI-Tools

Ein primäres Ziel dieser Plattform war die autonome und einfache Bedienung. Dies sollte daher auch für das implementierte CLI-Unterstützungstool gelten, da es Bestandteil dieser Plattform ist. Wie die Umfrageergebnisse in Kapitel 3.4 gezeigt haben, waren bei den teilnehmenden Personen größtenteils nur Grundkenntnisse im Bereich Kubernetes und DevOps vorhanden. Daher war es notwendig, die Anforderungen so zu gestalten, dass diese Personen nicht z. B. durch eine komplexe Bedienung abgeschreckt werden. Daher wurde ein Guided Mode in die CLI integriert, der es, wie in Kapitel 7 gezeigt, ermöglicht, ohne viel Erfahrung in den Bereichen Kubernetes, CI/CD und DevOps eine Anwendung auf der Kubernetes-Plattform zu deployen. Durch die eingebauten Usage-Informationen für jeden Befehl und jedes Parameterflag konnte die Dokumentation der CLI bereits in das Tool selbst integriert werden. Dies ist eine gängige Methode im Bereich der CLIs. Insgesamt konnte so eine einfach zu bedienende CLI entwickelt werden, die den Anforderungen gerecht wird.

8.5 Beantwortung der Leitfragen

In diesem Abschnitt werden die in Kapitel 1.2 aufgestellten Leitfragen abschließend beantwortet.

LF1: Inwieweit erfüllen bestehende multimandantenfähige Lösungen für Kubernetes die Anforderungen an eine „harte“ Isolation von Entwicklungs- und Produktionsumgebungen?

Die derzeit bekanntesten Softwarelösungen in diesem Bereich sind vCluster und Kamaji. Diese sind in der Lage, die Kubernetes API zu abstrahieren, indem sie virtuelle Control Planes verwenden, um neue virtuelle Cluster innerhalb eines dedizierten Clusters zu erzeugen. Dadurch kann eine „harte“Isolation erreicht werden. Dies wurde in Kapitel 5.1.2 ausführlich gezeigt. In Bezug auf Entwicklungs- und Produktionsumgebungen muss zunächst eine Definition von Multi-Tenancy im Zusammenhang mit verschiedenen Produktionsbereichen erstellt werden. Da Multi-Tenancy, wie diese Arbeit gezeigt hat, sowohl für die Trennung von Teams als auch von Unternehmen geeignet ist, kann dies sowohl in kleiner Form auf Projektbasis als auch

auf Applikationsbasis erfolgen. Während die Trennung in Form von Clustern für produktive Bereiche in der Theorie einfach und logisch erscheint, erfordert die praktische Umsetzung eine sorgfältige Abwägung und Implementierung zusätzlicher Maßnahmen. Insbesondere das Management gemeinsamer Ressourcen, die Einhaltung von Compliance-Anforderungen und die Sicherstellung der Performance stellen große Herausforderungen dar. Ein wichtiges Element zur Erfüllung solcher Compliance-Anforderungen kann die Mandantenfähigkeit mit ihren verschiedenen Isolationsschichten sein. Im Rahmen dieser Arbeit wurde ein Beispiel für einen spezifischen Anwendungsfall gezeigt, die verwendeten Technologien können jedoch aufgrund ihrer offenen Konfiguration für beliebige andere Anwendungsszenarien genutzt werden. Zusammenfassend bedeutet dies, dass aktuelle mandantenfähige Lösungen für Kubernetes die Anforderung der klassischen „harten"-Isolation erfüllen können, jedoch weitere Schritte unternommen werden müssen, die u.a. den Einsatz weiterer Softwarelösungen für z.B. Firewall-Regeln, Logging, etc. erfordern.

LF2: Welchen Wissensstand haben Entwickler in den Bereichen sichere Entwicklung, Cloud-native Entwicklung, Continuous Delivery (CD) und Remote Development?

Kapitel 3, welches sich mit der Online-Nutzerbefragung und Auswertung beschäftigt, konnten wertvolle Ergebnisse zu den in der Frage formulierten Themen gewonnen werden. Die Ergebnisse zeigen, dass Entwickler in den grundlegenden Prinzipien der Cloud-native Entwicklung und Continuous Delivery vertraut sind. Obwohl ein großer Teil der Befragten über eine umfangreiche Berufserfahrung von über 10 Jahren verfügt, zeigte die Umfrage, dass das tiefere Verständnis für fortgeschrittene Konzepte der Cloud-native Entwicklung und Continuous Delivery überraschend gering ist. Diese Diskrepanz zwischen Berufserfahrung und Fachwissen könnte auf verschiedene Faktoren zurückzuführen sein. Ein wesentlicher Erklärungsfaktor für die ernüchternden Ergebnisse könnte der spezifische Branchenfokus des Unternehmens sein, in dem die Umfrage durchgeführt wurde. Das Unternehmen ist überwiegend im Dienstleistungsbereich für die deutsche Automobilindustrie tätig, eine Branche, die traditionell auf stabile, bewährte Technologien setzt und oft langsamer bei der Adaption neuer Technologien ist.

Ein positiver Aspekt der Umfrageergebnisse betrifft den Bereich der sicheren Softwareentwicklung. Die Befragung hat gezeigt, dass das Bewusstsein und die Praxis für IT-Sicherheit im Unternehmen stärker ausgeprägt sind als in anderen technischen Bereichen. Dies spiegelt sich in der aktiven Nutzung von Sicherheitsmaßnahmen in den CI/CD-Pipelines wider. Automatisierte Abhängigkeitsprüfungen, Schwachstellenanalysen und die Erstellung von SBOM-Bäumen (Software Bill of Materials) werden regelmäßig durchgeführt (siehe Kapitel 3.4.2). Diese Maßnahmen sind vermutlich Reaktionen auf die steigenden Anforderungen der Kunden, sowie die ständig wachsende Bedeutung der IT-Sicherheit in der Softwareentwicklung allgemein.

Im letzten Bereich der Remote-Entwicklung, die generell ein relativ neuer Ansatz in der modernen Software-Entwicklung ist, vor allem im Zusammenhang mit Kubernetes und der Cloud, konnte festgestellt werden, dass die Befragten mehrheitlich schon davon gehört haben und über einige Grundlagen verfügen (ca. 45%). Auf der anderen Seite hatten fast 40% noch keinen Kontakt mit diesem Konzept. Bei den Vorteilen von Remote Development waren sich die meisten Befragten jedoch wieder einig und stellten vor allem die erhöhte Rechenleistung sowie die schnelle Bereitstellung von Testumgebungen in den Vordergrund. Ein grundlegendes Verständnis, was Remote Development ist und welche Vor- und Nachteile es mit sich bringt, scheint also vorhanden zu sein.

Zusammenfassend kann also gesagt werden, dass in den definierten Bereichen doch weniger Wissen vorhanden ist als ursprünglich angenommen. Mögliche Gründe dafür könnten traditionelle Toolstacks etc. sein. Viele der modernen DevOps-Tools waren den Befragten unbekannt, was darauf schließen lässt, dass in den jeweiligen Bereichen vor allem Basiswissen und weniger komplexe Aufgaben ausgeführt werden.

LF3: Können unterstützende Werkzeuge die Bewältigung von operationellen Herausforderungen (Ops Challenges) in multimandantenfähigen Cluster-Umgebungen erleichtern, insbesondere für diejenigen Entwickler, die keine umfangreiche Erfahrung in der Verwaltung solcher Systeme haben?

Die Implementierung der Kubernetes-Plattform in Kapitel 5 sowie der itdcli in Kapitel 6 und Kapitel 7 haben gezeigt, dass dies möglich ist. Klassische Ops-Herausforderungen wie das Aufsetzen und Verwalten von Clustern, CI/CD-Pipelines etc. stellen Entwicklungsteams oft vor große Herausforderungen, weshalb häufig auf einfachere Mittel zurückgegriffen wird. Das Gesamtkonzept der Kubernetes-Plattform mit all ihren Komponenten zeigt, dass es auch ohne großen administrativen Aufwand möglich ist, eine solche Plattform zu betreiben. Durch die Architektur der Plattform ist es gelungen, die administrativen Aufgaben den Plattformadministratoren, die über Expertenwissen verfügen, zuzuweisen und damit die Endanwender zu entlasten. Durch die itdcli wurde die Bedienung für die Endanwender weiter vereinfacht, indem wiederkehrende Aufgaben wie das Deployment von Anwendungen in Kubernetes sowie Onboarding-Prozesse bei der Einrichtung der Plattform im Team teilweise automatisiert wurden. Durch den Einsatz von anwenderoptimierten Tools wie der itdcli ist auch eine Nutzung ohne große Erfahrung möglich, wie der Guided Mode der CLI zeigt.

LF4: Wie kann die Entwicklung einer kostengünstigen Self-Service-Plattform gestaltet werden, um die täglichen Entwicklungsprozesse innerhalb eines bestehenden Teams mit hoher Fluktuation zu unterstützen, und wie wirkt sich dies auf die Effizienz und Effektivität aus?

Durch die im Rahmen dieser Arbeit implementierte Kubernetes Self-Service Plattform konnte gezeigt werden, dass dies auch mit freien Methoden möglich ist und diese durch den Einsatz von unterstützenden Tools wie der in Kapitel 6 und Kapitel 7 erstellten CLI eine hilfreiche Ergänzung in der täglichen Entwicklungsarbeit sein können. Darüber hinaus können Teams durch die Nutzung von zentralen Umgebungen wie Team-Clustern und den damit verbundenen Dev-Clustern die anfallenden Entwicklungsprozesse eigenständiger bearbeiten, da keine Abhängigkeiten nach außen bestehen. Überträgt man diese Erkenntnisse nun auf Teams mit hoher Fluktuation, so kann sich dies positiv auf die Effizienz und Effektivität auswirken, da durch die zentralen Repositories und den hohen Automatisierungsgrad in Bereichen der Cloud-basierten Softwareentwicklung, wie in Kapitel 5.5 gezeigt, die Einarbeitungszeit eines neuen Teammitglieds und die Installation von Projekten und Umgebungen verkürzt wird. Dadurch kann ein aufwändiges Onboarding entfallen, was wiederum die Effizienz der anderen Teammitglieder erhöht.

Darüber hinaus kann durch den Self-Service-Ansatz der Kubernetes-Plattform eine Arbeitsumgebung geschaffen werden, die weniger abhängig von personellen Ressourcen ist, da Ops-Aufgaben nun weitgehend selbstständig und ohne große Erfahrung durchgeführt werden können, wie das Beispiel der itdcli gezeigt hat. Zusammenfassend lässt sich also sagen, dass die implementierte Kubernetes-Plattform ein mögliches Beispiel dafür darstellt, wie die Effizienz und Effektivität dynamischer Teams gesteigert und der Arbeitsalltag durch unterstützende Werkzeuge vereinfacht werden kann.

9 Zusammenfassung und Ausblick

Abschließend wird in diesem Kapitel eine kurze Zusammenfassung der Arbeit sowie ein Fazit und ein Ausblick auf mögliche Erweiterungen gegeben.

9.1 Zusammenfassung

Ziel dieser Arbeit war die Konzeption und Implementierung einer mandantenfähigen Kubernetes DevSecOps Plattform, die speziell für Remote Development optimiert ist und bestehende Entwicklungsteams bei der Anwendungsentwicklung effektiv unterstützt. Um dieses Ziel zu erreichen, wurden zunächst vier Leitfragen definiert, die eine grundlegende Richtung für die spätere Implementierung der Kubernetes-as-a-Service Plattform vorgaben. Darüber hinaus sollte durch das eingebaute Self-Service-Modell eine einfache Bedienbarkeit und Nutzbarkeit für verschiedenste Softwareprojekte gegeben sein, um in Zukunft eine breite Adoption dieser Plattform zu ermöglichen. Um dies zu erreichen, wurde zunächst ein umfangreicher Grundlagenteil zu den verschiedenen Technologien rund um Kubernetes sowie einigen weiteren wichtigen Cloud Computing- und DevOps-Grundlagen erstellt. So wurden beispielsweise Themen wie Kubernetes inklusive Multi-Tenancy sowie die DevOps-Kultur und die damit verbundenen Probleme und wichtige Tools wie FluxCD zur Ermöglichung von GitOps behandelt.

Im nächsten Schritt wurde eine qualitative und quantitative Online-Befragung im betreuenden Unternehmen durchgeführt, um den aktuellen Wissensstand zu Themen wie „sichere Softwareentwicklung“, „Cloud-native Softwareentwicklung“, Continuous Delivery (CD) und Cloud-basierte Softwareentwicklung (Remote Development) abzufragen. Darüber hinaus sollte in Erfahrung gebracht werden, inwieweit die DevOps-Kultur bereits im Unternehmen manifestiert ist, was die Adaption der in dieser Arbeit geplanten Kubernetes-Plattform erleichtern würde. Dazu wurden zunächst die allgemeine Vorgehensweise sowie die Fragen der Umfrage erläutert, um diese anschließend allen Mitarbeitern des Unternehmens zur Verfügung zu stellen. Anschließend wurden die Ergebnisse der Umfrage ausgewertet, um im nächsten Schritt die Anforderungen an die Kubernetes-Plattform sowie an ein CLI-Unterstützungstool, welches später Teil der Plattform werden soll, zu definieren, um ein durchdachtes und geplantes weiteres Vorgehen zu ermöglichen.

Vor der eigentlichen Anforderungsdefinition wurde parallel zur Erhebung eine Ist-Analyse der aktuell bestehenden Kubernetes-Infrastruktur sowie der generischen CI/CD-Pipeline durchgeführt. Ziel dieser Analyse war es, Schwachstellen und Probleme zu identifizieren sowie notwendige bestehende Funktionalitäten zu identifizieren, die in der neuen Kubernetes-Infrastruktur weiterhin zur Verfügung stehen müssen. Durch diese beiden Arten der Analyse konnten sowohl funktionale als auch nicht-funktionale Anforderungen sowie drei Stakeholder

(Plattformadministrator, Softwareentwickler und Gruppenleiter/Tenant-Admin) definiert werden. Die Anforderungen wurden anschließend in Form von User Stories aufbereitet und nach der MosCoW-Methode priorisiert.

Danach konnte mit der eigentlichen Konzeption und Implementierung der Kubernetes-as-a-Service Plattform begonnen werden. Zunächst wurden verschiedene Open Source Lösungen für Mandantenfähigkeit, Storage, Betriebssystem, Netzwerk und Cloud-basierte Softwareentwicklung recherchiert, um einen geeigneten Toolstack zur Verfügung zu haben. Die eigentliche Implementierung erfolgte auf einem Proxmox-Testsystem, das aus mehreren virtuellen Maschinen bestand. Nach dem Aufbau des grundlegenden Kubernetes-Clusters wurde mit der Entwicklung von Konzepten begonnen, um die Plattform vollständig mit GitOps-Ansätzen kontrollierbar zu machen und eine strikte Trennung zwischen Infrastruktur und Endbenutzer zu ermöglichen, um später eine einfache Bedienbarkeit zu gewährleisten. Dazu wurden drei Repository-Konzepte entwickelt, die verschiedenen Stakeholdern zur Verfügung stehen, um sowohl die Infrastruktur als auch den Management Cluster und die darauf laufenden Team Cluster in Form von virtuellen Clustern (Tenants) zu verwalten. Ziel war es, durch die Mandantenfähigkeit von Kubernetes eine harte Isolation und Trennung auf Team-Ebene (Team-Cluster) zu realisieren, die innerhalb der Teams weiter unterteilt wird, um auf Entwickler-Ebene eigene sogenannte Dev-Cluster zu generieren, die im Besitz der jeweiligen Entwickler sind. Diese Trennungen erforderten den Einsatz von Kamaji und Capsule als mandantenfähige Frameworks. Darüber hinaus musste ein Konzept entwickelt werden, um innerhalb der Dev-Cluster Remote-Entwicklung zu ermöglichen, so dass Vorteile wie Cloud-Ressourcen und das Arbeiten von beliebigen Geräten etc. genutzt werden können. Dazu wurde ein Konzept auf Basis von DevPod entwickelt. Zusätzlich wurde eine neue generische CI/CD Pipeline auf Basis von GitLab entwickelt, die sich optimal in das Self-Service Modell einfügt und einige SecOps Komponenten enthält bzw. in Zukunft ermöglichen wird.

In den nächsten beiden Kapiteln wurde mit dem Design und der Implementierung eines Support-Tools in Form einer Golang CLI begonnen, die innerhalb der IDE den Entwickler konkret bei der Entwicklung und dem anschließenden Testen durch ein Cluster-Deployment unterstützen soll. Darüber hinaus wurden einige Bootstrapping-Methoden implementiert, die das Onboarding neuer Teams in die Plattform erleichtern sollen. So wird z. B. die CI/CD Pipeline halbautomatisch initialisiert und für das jeweilige Team konfiguriert, so dass auch Neulinge in den Bereichen DevOps und Kubernetes die Plattform nutzen können. Der Fokus bei der Entwicklung lag aufgrund der Umfrageergebnisse auf einer einfachen Bedienbarkeit sowie einem Modus, der den Benutzer bei bestimmten Befehlen durch geführte Fragen zum Ziel führt. Die CLI erhielt den Namen itdcli und ist dank Golang auf allen gängigen Plattformen wie Windows, Linux und Mac nutzbar. Schließlich wurde eine Bewertung durchgeführt, die sich in erster Linie auf einen Rückblick auf die verwendeten Technologien konzentrierte, gefolgt von einer Bewertung der funktionalen und nicht funktionalen Anforderungen. Anforderungen, die nicht oder nur teilweise umgesetzt wurden. Dabei wurden Anforderungsgruppen definiert,

um Anforderungen, die der gleichen Funktionalität dienen, zusammenzufassen. Anschließend wurden noch die nicht-funktionalen Anforderungen bzw. auch Qualitätsanforderungen evaluiert, indem die umgesetzte Plattform nochmals genau analysiert wurde, ob das eigentlich angestrebte Ziel durch die Leitfragen und Anforderungen erfüllt wurde. Außerdem wurden die Leitfragen nochmals aufgegriffen und beantwortet.

9.2 Fazit

Das eigentliche Ziel der Arbeit, nämlich die Konzeption und Implementierung einer mandantenfähigen Kubernetes DevSecOps Plattform, die speziell für Remote Development optimiert ist und bestehende Entwicklungsteams bei der Anwendungsentwicklung effizient unterstützt, gepaart mit den Forschungsfragen, konnte durch den Einsatz moderner State-of-the-Art Lösungen und Technologien wie Kubernetes, Kamaji, Capsule, FluxCD und Golang mit einem zufriedenstellenden Ergebnis abgeschlossen werden. Im Rahmen dieser Arbeit wurde eine Kubernetes-as-a-Service Plattform entwickelt, die es ermöglicht, die Kosten und die Einstiegshürde von Kubernetes durch die Generierung einer eigenen Private Cloud zu minimieren und die strengen Datenschutz- und IT-Sicherheitsanforderungen der heutigen Zeit zu erfüllen. Gleichzeitig unterstützt und steigert es die Effizienz und Produktivität von Entwicklungsteams durch den Einsatz zentraler GitOps-Lösungen, die vor allem repetitive Aufgaben automatisieren, die sonst häufig manuell durchgeführt werden. Durch die Beantwortung aller aufgestellten Forschungsfragen mit durchwegs positiven Ergebnissen gilt das Ziel dieser Masterarbeit als erfüllt.

9.3 Ausblick

Für die Zukunft wäre die Integration der noch fehlenden funktionalen Anforderungen denkbar. Darüber hinaus wäre eine erweiterte Beta-Testphase mit anschließender Live-Schaltung der Plattform in absehbarer Zeit möglich. Hierfür müssten jedoch noch weitere Schritte zur Absicherung der Cluster und der Backup-Strategien der eingesetzten Speicherlösungen unternommen werden. Darüber hinaus ergaben sich Ideen und Erweiterungsmöglichkeiten für die Umsetzung der Plattform. So wäre die Integration einer grafischen Benutzeroberfläche anstelle der CLI denkbar, da diese die Benutzerfreundlichkeit weiter erhöhen würde. So könnte z. B. neben der Terminalsteuerung auch ein Webinterface angeboten werden. Darüber hinaus wäre auch die Prüfung einer externen persistenten Speicherlösung wie Rook-Ceph oder LinSTOR in einem dedizierten Cluster inklusive Backup-Möglichkeiten denkbar. Damit könnte die Plattform selbst quasi zustandslos gestaltet werden, was einige Vorteile bei der Verteilung von Ressourcen und Workloads mit sich bringt. Ebenso könnte das Infrastrukturmanagement mit Ansible oder Terraform weiter automatisiert werden, so dass auch Cloud-Lösungen schnell in den Cluster integriert werden können. Generell werden in Zukunft immer mehr Ressourcen in externe Public Clouds ausgelagert, was die Plattform unterstützen muss.

Literaturverzeichnis

ABUHAMDA, E., ISMAIL, I. und BSHARAT, T. 2021. Understanding quantitative and qualitative research methods: A theoretical perspective for young researchers. *International Journal of Research*. Jg. 8, S. 71–87. Verfügbar unter: DOI: 10.2501/ijmr-201-5-070.

ALY, S. und MURAT, K. 2021. *Kubernetes in Production Best Practices : Build and Manage Highly Available Production-ready Kubernetes Clusters.* Packt Publishing. ISBN 9781800202450. Auch verfügbar unter: http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d2757909%26site%3dehost-live.

ATLASSIAN 2020. *Atlassian Survey 2020 - DevOps Trends* [online]. [Letzter Zugriff: 12. April 2024]. Verfügbar unter: https://www.atlassian.com/de/whitepapers/devops-survey-2020.

AUTHORS, T. F. 2024a. *Core Concepts of Flux* [online]. [Letzter Zugriff: 4. Mai 2024]. Verfügbar unter: https://fluxcd.io/flux/concepts/.

AUTHORS, T. K. 2024b. *Kustomize* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://kubectl.docs.kubernetes.io/guides/introduction/kustomize/.

BITKOM 2010. *Cloud Computing: Was Entscheider wissen müssen* [online]. [Letzter Zugriff: 28. April 2024]. Verfügbar unter: https://www.bitkom.org/sites/default/files/file/import/BITKOM-Leitfaden-Cloud-Computing-Was-Entscheider-wissen-muessen.pdf.

BUNDESMINISTERIUM 2024. *Qualitative Bewertungsmethoden* [online]. [Letzter Zugriff: 18. Juli 2024]. Verfügbar unter: https://www.orghandbuch.de/Webs/OHB/DE/Organisationshandbuch/6_MethodenTechniken/65_Wirtschaftlichkeitsuntersuchung/652_Qualitative/qualitative-node.html.

CLASTIX 2020. *Kamaji: Concepts* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://kamaji.clastix.io/concepts/.

CLASTIX 2022. *Capsule: Getting started* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://capsule.clastix.io/docs/general/getting-started.

CLASTIX 2023. *Capsule Proxy* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://capsule.clastix.io/docs/general/proxy/.

CNCF 2015. *Cloud Native Computing Foundation* [online]. [Letzter Zugriff: 25. April 2024]. Verfügbar unter: https://www.cncf.io/.

CNCF 2024. *Cloud Native Landscape* [online]. [Letzter Zugriff: 25. April 2024]. Verfügbar unter: https://landscape.cncf.io/.

COBUKCUOGLU, B. e. a. 2024. *GitOps : Grundlagen und Best Practices*. dpunkt.verlag. ISBN 9783988900128. Auch verfügbar unter: https://dpunkt.de/produkt/gitops/.

COLEY, C. 2019. *MoSCoW Prioritisation* [online]. [Letzter Zugriff: 12. Juli 2024]. Verfügbar unter: https://www.coleyconsulting.co.uk/moscow.htm.

DEBELLIS, D. und PETERS, C. 2022. *2022 Accelerate State of DevOps Report: A deep dive into security* [online]. [Letzter Zugriff: 20. März 2024]. Verfügbar unter: https://cloud.google.com/blog/products/devops-sre/dora-2022-accelerate-state-of-devops-report-now-out?hl=en.

DEBIAN 2024. *Gründe für den Einsatz von Debian* [online]. [Letzter Zugriff: 18. Juli 2024]. Verfügbar unter: https://www.debian.org/intro/why_debian.

FALK, A. 2016. *SecDevOps – Kontinuierlich sichere Software bauen* [online]. [Letzter Zugriff: 20. April 2024]. Verfügbar unter: https://www.informatik-aktuell.de/betrieb/sicherheit/secdevops-kontinuierlich-sichere-software-bauen.html.

GIGI, S. 2020. *Mastering Kubernetes*. Bd. Third edition. Packt Publishing. ISBN 9781839211256. Auch verfügbar unter: http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d2514611%26site%3dehost-live.

GITLAB 2022. *Breakdown of software development methodologies practiced worldwide in 2022* [online]. [Letzter Zugriff: 19. März 2024]. Verfügbar unter: https://www.statista.com/statistics/1233917/software-development-methodologies-practiced/.

GITLAB 2024. *Use kaniko to build Docker images* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://docs.gitlab.com/ee/ci/docker/using_kaniko.html.

GOOGLE 2024. *Go for Cloud and Network Services* [online]. [Letzter Zugriff: 22. August 2024]. Verfügbar unter: https://go.dev/solutions/cloud.

HALL, T. 2020. *Was ist DevOps-Kultur?* [online]. [Letzter Zugriff: 12. April 2024]. Verfügbar unter: https://www.atlassian.com/de/devops/what-is-devops/devops-culture.

HELM 2024. *Charts* [online]. [Letzter Zugriff: 27. April 2024]. Verfügbar unter: https://helm.sh/docs/topics/charts/.

HOSSAIN, R. 2023. *Managing Kubernetes Cluster Sprawl* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://loft.sh/blog/managing-kubernetes-cluster-sprawl/.

HWANG, Y. und NEWMAN, A. 2023. *Kubernetes multi-tenancy: three key approaches* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://www.spectrocloud.com/blog/kubernetes-multi-tenancy-three-key-approaches.

KERECZMAN, M. 2024. *Comparing LINSTOR and Ceph Storage Clusters* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://linbit.com/blog/how-does-linstor-compare-to-ceph/.

KRATZKE, N. 2023. *Cloud-native Computing*. 2., updated edition. München: Carl Hanser Verlag GmbH & Co. KG. Verfügbar unter: DOI: 10.3139/9783446479258.

KUBERNETES 2024a. *Annotations* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/.

KUBERNETES 2024b. *Cluster Architecture* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/architecture/.

KUBERNETES 2024c. *ConfigMaps* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/configuration/configmap/.

KUBERNETES 2024d. *Deployments* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

KUBERNETES 2024e. *Ingress* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/services-networking/ingress/.

KUBERNETES 2024f. *Installing kubeadm* [online]. [Letzter Zugriff: 17. Juli 2024]. Verfügbar unter: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#before-you-begin.

KUBERNETES 2024g. *Labels and Selectors* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/.

KUBERNETES 2024h. *Multi-tenancy* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/security/multi-tenancy/.

KUBERNETES 2024i. *Namespaces* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

KUBERNETES 2024j. *Network Policies* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/services-networking/network-policies/.

KUBERNETES 2024k. *Options for Highly Available Topology* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/.

KUBERNETES 2024l. *Persistent Volumes* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

KUBERNETES 2024m. *Pods* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/workloads/pods/.

KUBERNETES 2024n. *Production-Grade Container Orchestration* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/.

KUBERNETES 2024o. *Resource Quotas* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/policy/resource-quotas/.

KUBERNETES 2024p. *Secret* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/configuration/secret/.

KUBERNETES 2024q. *Service* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/services-networking/service/.

KUBERNETES 2024r. *Service Accounts* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/security/service-accounts/.

KUBERNETES 2024s. *StatefulSets* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/.

KUBERNETES 2024t. *Storage Classes* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://kubernetes.io/docs/concepts/storage/storage-classes/.

LE MASSON, V. 2023. *The magic number: how to optimise and improve your survey response rate* [online]. [Letzter Zugriff: 23. Juni 2024]. Verfügbar unter: https://www.kantar.com/inspiration/research-services/what-is-a-good-survey-response-rate-pf.

LINBIT 2024. *LinSTOR: Kubernetes Persistent Container Storage* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://linbit.com/kubernetes/.

LOFT 2024a. *vCluster: Overview* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://www.vcluster.com/docs/v0.19/architecture/overview.

LOFT 2024b. *What is DevPod?* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://devpod.sh/docs/what-is-devpod.

LOFT 2024c. *What is DevSpace?* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://www.devspace.sh/docs/getting-started/introduction.

MELL, P. und GRANCE, T. 2011. *The NIST Definition of Cloud Computing* [online]. [Letzter Zugriff: 26. April 2024]. Verfügbar unter: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

METALLB 2024. *Network Addon Compatibility* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://metallb.universe.tf/installation/network-addons/.

OWASP 2024a. *Kubernetes Security Cheat Sheet* [online]. [Letzter Zugriff: 14. August 2024]. Verfügbar unter: https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html.

OWASP 2024b. *OWASP DevSecOps Guideline* [online]. [Letzter Zugriff: 20. April 2024]. Verfügbar unter: https://owasp.org/www-project-devsecops-guideline/.

PIRAEUS 2024. *GitHub: Piraeus Operator* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://github.com/piraeusdatastore/piraeus-operator.

RAJU, S. 2023. *FluxCD: Introduction and Installation Demo* [online]. [Letzter Zugriff: 30. April 2024]. Verfügbar unter: https://medium.com/@selvamraju007/fluxcd-introduction-and-installation-demo-fb9fe0cb7555.

REINHEIMER, S. 2018. *Cloud Computing: Die Infrastruktur der Digitalisierung*. Springer Fachmedien Wiesbaden. Edition HMD. ISBN 9783658209674. Auch verfügbar unter: https://books.google.de/books?id=M99YDwAAQBAJ.

ROOK 2024. *Storage Architecture* [online]. [Letzter Zugriff: 20. Juli 2024]. Verfügbar unter: https://rook.io/docs/rook/latest-release/Getting-Started/storage-architecture/#object-storage-s3.

SALIMI, S. 2023. *Nichtfunktionale Anforderungen als User Stories* [online]. [Letzter Zugriff: 12. Juli 2024]. Verfügbar unter: https://www.agile-academy.com/de/product-owner/nichtfunktionale-anforderungen-als-user-stories/.

SIDERO 2024a. *Talos Linux: Configuration Patches* [online]. [Letzter Zugriff: 25. Juli 2024]. Verfügbar unter: https://www.talos.dev/v1.7/talos-guides/configuration/patching/.

SIDERO 2024b. *Talos Linux: Philosophy* [online]. [Letzter Zugriff: 18. Juli 2024]. Verfügbar unter: https://www.talos.dev/v1.2/learn-more/philosophy/.

SIDERO 2024c. *Talos Linux: System Requirements* [online]. [Letzter Zugriff: 25. Juli 2024]. Verfügbar unter: https://www.talos.dev/v1.7/introduction/system-requirements/.

STANHAM, L. 2023. *What is Cloud Sprawl?* [online]. [Letzter Zugriff: 25. Juni 2024]. Verfügbar unter: https://www.crowdstrike.com/cybersecurity-101/secops/cloud-sprawl/.

WALLS, M. 2013. *Building a DevOps Culture*. O'Reilly Media. ISBN 9781449368371. Auch verfügbar unter: https://books.google.de/books?id=L1BZ0w0g-v4C.

WIGGINS, A. 2017. *The Twelve-Factor App* [online]. [Letzter Zugriff: 20. April 2024]. Verfügbar unter: https://12factor.net.

Anhang

A Infrastruktur

```
#!/bin/bash

# Function to check if talosctl is installed
check_talosctl() {
  if ! command -v talosctl &> /dev/null; then
    echo "talosctl could not be found. Please install it to proceed."
    exit 1
  fi
}
check_talosctl

if [ $# -ne 2 ]; then
  echo "Usage: $0 <CLUSTER_NAME> <API_ENDPOINT>"
  exit 1
fi

CLUSTER_NAME=$1
API_ENDPOINT=$2

# Create the build directory if it does not exist
mkdir -p build/nodes/controlplane
mkdir -p build/nodes/worker

# Function to generate configs for a specific node type
generate_configs() {
  local node_type=$1
  local node_folder="nodes/$node_type"
  local output_folder="build/$node_folder"

  # Iterate over each node configuration file for the given node type
  for node_file in "$node_folder"/*.yaml; do
    node_name=$(basename "$node_file")
    config_patches=""

    # Collect all patch commands
    for patch_file in patches/*.yaml; do
      config_patches+="--config-patch @$patch_file "
    done

    # Generate config command
    echo "Generating config for $node_name with all patches"
    talosctl gen config \
      --output "$output_folder/$node_name" \
```

```
        --output-types "$node_type" \
        --with-secrets secrets.yaml \
        $config_patches \
        --config-patch "@$node_file" \
        --force \
        $CLUSTER_NAME \
        $API_ENDPOINT

      echo "Config generated: $output_folder/$node_name"
    done
  }

  # Generate configs for controlplane and worker nodes
  echo "Starting config generation for controlplane nodes..."
  generate_configs "controlplane"
  echo "Starting config generation for worker nodes..."
  generate_configs "worker"
  echo "Config generation completed."
```

Listing 7: *Talos Merge Skript für Patches*

```
apiVersion: capsule.clastix.io/v1beta2
kind: Tenant
metadata:
  name: developer-alice
spec:
  additionalRoleBindings:
  - clusterRoleName: cluster-admin
    subjects:
    - name: gitops-reconciler
      kind: ServiceAccount
      namespace: tenant-alice
  owners:
  - name: system:serviceaccount:tenant-alice:gitops-reconciler
    kind: ServiceAccount
  - name: alice
    kind: User
    proxySettings:
    - kind: PriorityClasses
      operations:
      - List
    - kind: StorageClasses
      operations:
      - List
    - kind: IngressClasses
      operations:
      - List
    - kind: Nodes
      operations:
```

```
        - List
      - kind: PersistentVolumes
        operations:
        - List
        - Update
        - Delete
    storageClasses:
      allowedRegex: "^linstor-.*$"
    ingressOptions:
      allowedClasses:
        allowedRegex: "^nginx-.*$"
```

Listing 8: *Capsule Tenant Konfigurationsdatei*

```
# Use the official Golang image as a base
FROM cgr.dev/chainguard/go:latest as builder

# Set the working directory inside the container
WORKDIR /app

# Copy the Go source code into the container
COPY . .

# Build the Go application
RUN go build -o build .

# Use a minimal Docker image for the final image
FROM cgr.dev/chainguard/static:latest

# Set the working directory inside the container
WORKDIR /app

# Copy the compiled Go binary from the builder stage
COPY --from=builder /app/build .

# Command to run the executable
CMD ["./main"]
```

Listing 9: *Exemplarisches Dockerfile mit Chainguard Images*

```
  apiVersion: storage.k8s.io/v1
  kind: StorageClass
  metadata:
    name: linstor-lvm01-ssd # Name der StorageClass für Workloads
    annotations:
      "storageclass.kubernetes.io/is-default-class": "true" # Festlegung als
          Standard-StorageClass
  parameters:
```

```
8       csi.storage.k8s.io/fstype: xfs
9       linstor.csi.linbit.com/autoPlace: "3" # Replizierung auf drei Nodes
10      linstor.csi.linbit.com/storagePool: "ssd_pool_01" # Pool-Name
11    provisioner: linstor.csi.linbit.com # Provisioner von Piraeus
12    allowVolumeExpansion: true # Erlaubt die nachträgliche erweiterung eines Volumes
13    volumeBindingMode: WaitForFirstConsumer # Wartet bis ein Deployment den Speicher mountet
```

Listing 10: *Konfiguration der StorageClass für LinSTOR*

Ehrenwörtliche Erklärung

Name: Efremidis
Vorname: Alexander
Matrikel-Nr.: 771514
Studiengang: Angewandte Informatik

Hiermit versichere ich, Alexander Efremidis, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebene Literatur und Hilfsmittel verwendet habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ort, Datum

Unterschrift