HAW HAMBURG

BACHELORTHESIS
Marlene Pfefferkorn

Evaluierung verschiedener Service Mesh Implementierungen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marlene Pfefferkorn

Evaluierung verschiedener Service Mesh Implementierungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 23. Oktober 2020

Marlene Pfefferkorn

Thema der Arbeit

Evaluierung verschiedener Service Mesh Implementierungen

Stichworte

Service Mesh, Istio, Linkerd2, Consul Connect, Kuma, Kubernetes, Microservices, Verteilte Systeme

Kurzzusammenfassung

Um den Herausforderungen großer Service Landschaften begegnen zu können ist die Bedeutung des Service Meshs in den letzten zwei Jahren immer weiter gestiegen. Das Ziel der vorliegenden Bachelorarbeit war es die Implementierungen der Service Meshs Istio, Linkerd 2, Consul Connect und Kuma zu evaluieren und miteinander zu vergleichen. Hierfür wurde eine Reihe an Kriterien aufgestellt. Diese wurden basierend auf einer auf Kubernetes laufenden Microservice Anwendung und der jeweiligen Dokumentation des Service Meshs praktisch und theoretisch analysiert und bewertet. Hierbei zeigte sich, dass Istio anhand der Kriterien am besten bewertet wurde, jedoch jedes Service Mesh andere Schwerpunkte besitzt und für einen entsprechenden Anwendungsfall besser geeignet ist.

Marlene Pfefferkorn

Title of Thesis

Evaluation of different service mesh implementations

Keywords

Service Mesh, Istio, Linkerd2, Consul Connect, Kuma, Kubernetes, Microservices, Distributed Systems

Abstract

In order to meet the challenges of large service landscapes, the importance of the service mesh has increased steadily over the past two years. The aim of this bachelor thesis was to evaluate and compare the implementations of the service meshes Istio, Linkerd 2,

Consul Connect and Kuma. A number of criteria have been established for this purpose. These were practically and theoretically analyzed and evaluated based on a microservice application running on Kubernetes and the respective documentation of the service mesh. The results showed that Istio was rated best based on the selected criteria, but that each service mesh has a different focus and is better suited for a corresponding application.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Eine modulare Architektur und eine flexible Infrastruktur sind die Basis des in den letzten Jahren immer relevanter werdenden cloudnativen Ansatzes geworden. Schnell erweiterbare und skalierbare Services, die flexibel an Wünsche und Bedürfnisse angepasst werden können [147], sowie die Frameworks, die ein Verwalten der Anwendung möglich machen, werden in der Entwicklung vorangetrieben. Besondere Aufmerksamkeit hat hierbei Kubernetes als System zur Container Orchestrierung erlangt [16].

Wenn man die Software Entwicklung cloudnativer Anwendungen der letzten Jahre betrachtet, bringt eine Microservice Architektur im Vergleich zu einem Monolithen einige Vorteile mit sich. So sind zum Beispiel schnellere und simplere Deployments sowie technologische Diversität zwischen den Services möglich. Zudem sorgen Microservices für klare Modulgrenzen [15]. Jedoch ergeben sich hieraus auch neue Herausforderungen. Unter anderem wächst die operative Komplexität[1] . Statt einer Anwendung müssen nun mehrere kleine betreut werden. Ein alleinstehender Service wird gegebenenfalls leichter verständlich. Jedoch wird die Komplexität einer Anwendung leicht auf die Kommunikation zwischen den Services verlagert [14].

Bei einer überschaubaren Menge an Services lassen sich die Kommunikation und Wege zwischen den Services noch nachvollziehen. Aber wie wird Zuverlässigkeit, Beobachtbarkeit und Sicherheit[2] bei wachsender Anzahl an Services sichergestellt? Bei einer Vielzahl von verteilten Services muss auch bedacht werden, dass das Netzwerk nicht zuverlässig oder latenzfrei und die Bandbreite nicht unendlich ist[9].

Diese Arbeit beschäftigt sich mit dem Vergleich verschiedener Service Meshes, die dieser Problematik Abhilfe schaffen sollen. Anhand eines Fallbeispiels werden die Implementierungen von Istio, Linkerd2, Consul Connect und Kuma in Bezug auf ihre jeweiligen Vor- und Nachteile verglichen und ihre Eignung in einer praktischen Anwendung analysiert.

[1] engl.: operational complexity
[2] engl.: reliability, observability, security

Hierfür wird im folgenden Kapitel zunächst definiert was ein Service Mesh ist, wie es aufgebaut ist und welche Möglichkeiten sich aus der Architektur ergeben.

Im dritten Kapitel werden die Bewertungskriterien vorgestellt. Diese werden in Unterkapiteln in die verschieden Bereiche Routing, Resilience, Security, Usability und Extensibility unterteilt.

Im vierten Kapitel wird das Fallbeispiel und dessen Architektur und Kommunikation, sowie Inbetriebnahme erläutert. Hierbei werden die verschiedenen Services und deren Schnittstellen vorgestellt.

In den darauf folgenden Kapiteln werden die Service Meshes anhand der Kriterien analysiert. Hierbei wird für jeden Bereich eine Tabelle mit der Analyse und dem Ergebnis erstellt.

Im neunten Kapitel werden die Ergebnisse einander gegenübergestellt und Tabellen mit dem Vergleich der verschiedenen Service Meshes dargestellt. Hier werden die jeweiligen Stärken und Schwächen der Service Meshes spezifiziert.

Im Fazit wird auf den Ergebnissen aus dem Vergleich aufgebaut, erläutert, welche Anwendungsmöglichkeiten sich für das jeweilige Service Mesh ergeben und dargelegt, warum es nicht ein Service Mesh gibt, dass alle Erwartungen erfüllt.

Im Ausblick werden weiterführende Gedanken zum Hype um die Service Mesh Architektur thematisiert und aufgezeigt, was noch fehlt.

2 Service Mesh

Im Folgenden werden der Aufbau und die Implikationen erläutert, die sich durch ein Service Mesh ergeben. Hierbei wird vor allem auf Kontrollmöglichkeiten des Netzwerkverkehrs[3], Beobachtbarkeit und Sicherheitsmöglichkeiten eingegangen, die sich durch ein Service Mesh ergeben.

2.1 Definition

Ein Service Mesh ist eine Infrastruktur Schicht, die mit Hilfe des Sidecar Entwurfsmusters Services vernetzt. Es übernimmt hierbei die Kommunikation zwischen den Services [135].

Das Sidecar Entwurfsmuster ist ein aus zwei Containern bestehendes single-node Pattern. In dem ersten Container befindet sich die Anwendung. Der zweite Container ist der Sidecar Container, der zusätzliche Funktionalitäten für die Anwendung bereitstellt. Beide Container werden auf der gleichen Maschine deployed und nutzen die gleichen Ressourcen[8]. Für das Service Mesh dienen die Sidecars in erster Linie als Proxys. Die Proxys bilden, so wie in Abbildung 1 zu sehen, ein Kommunikationsnetzwerk.

Hierdurch ergeben sich nun verschiedene neue Möglichkeiten. Die Konfiguration der Kommunikation wird externalisiert und ist somit unabhängig von der Service-spezifischen Sprache und des Frameworks. Die Kommunikation wird von der Geschäftslogik entkoppelt und lässt eine dezentralisierte Policy basierte Kontrollmöglichkeit zu, wie zum Beispiel welcher Service mit welchem sprechen darf[6].

Durch den Proxy erhält man zudem eine weitere Kontrollinstanz für den entsprechenden Service. Sollte dieser nicht erreichbar sein oder zu langsam antworten, können Anfragen an eine andere Instanz umgeleitet werden. Die Möglichkeit den Kommunikationsfluss zu

[3] engl.: traffic control

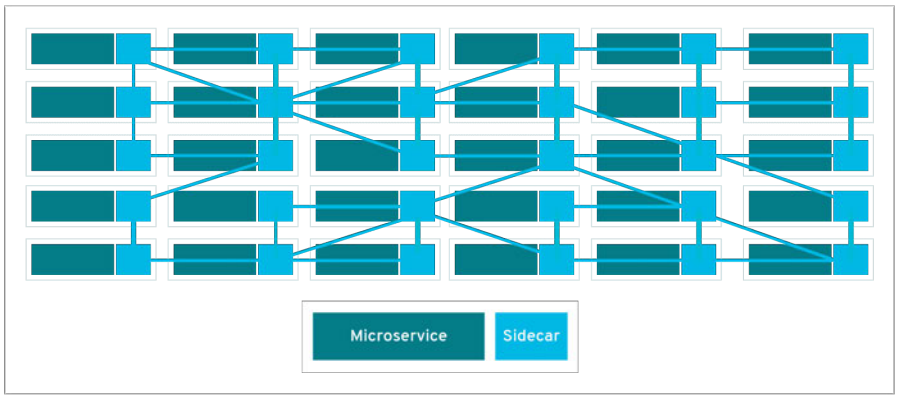

Abbildung 1: Sidecar Proxys im Service Mesh [148]

lenken ist auch für A/B beziehungsweise Canary Testing praktisch, bei dem ein gewisser Prozentsatz der Anfragen zu einer neueren Version des Services geleitet werden[9].

Weiterhin bietet sich nun die Möglichkeit die Kommunikation zwischen den Services zu beobachten. Erweiteres Logging der Requests, Konfiguration von Tracing in einem verteilten System und Metriken der Kommunikationswege werden nun einfacher[6].

Neben Kontrollmöglichkeiten des Netzwerkverkehrs und erweiterter Beobachtbarkeit, bieten sich auch weitere Sicherheitsmöglichkeiten. Die Sidecar Proxys bieten unter anderem mutual TLS in der Service-to-Service Kommunikation, verteilt und über eventuelle Clustergrenzen hinweg. Auch Identität, Zertifikate und Autorisierung können unabhängig von der Service Implementierung ermöglicht werden[2]. Die Ausfallsicherheit kann durch die Möglichkeit von Fault Injection[4] und Circuit Breaking[5] erhöht werden[9].

Häufig setzen Service Meshes ein darunter laufendes Kubernetes Cluster voraus. Dessen Funktionalitäten werden dann durch das Service Mesh erweitert. Wie beispielsweise erweiterte Service-Discovery, Load Balancing oder auch Kommunikationsstabilität[6][7].

Jedoch muss man neben den Vorteilen, die ein Service Mesh bietet, auch die Nachteile betrachten. So ergibt sich aufgrund der zusätzlichen Schicht eine höhere Komplexität des Systems. Höhere Latenzeinbußen können hier die Konsequenz sein. Darüber hinaus benötigt ein Service Mesh zusätzliche Ressourcen und ist in der Regel mit erheblichen Lernaufwand bei der Konfiguration verbunden[153].

[4] Fault Injection: Fehler werden in die Anwendung eingebaut, um das Verhalten testen zu können[12].
[5] Circuit Breaking: Sich wiederholende Verbindungsfehler werden identifiziert und können entsprechend behandelt werden. Beispielsweise werden Anfragen nicht mehr an diese Instanz weitergeleitet[112].
[6] engl.: Communication Resilience

2.2 Architektur

Auch wenn sich die anbieterspezifische Architektur der Service Meshes leicht unterscheidet, kann eine allgemeine Architektur spezifiziert werden. So unterteilt sich die Architektur eines Service Meshs generell in eine Data Plane und eine Control Plane (vgl. Abbildung 2).

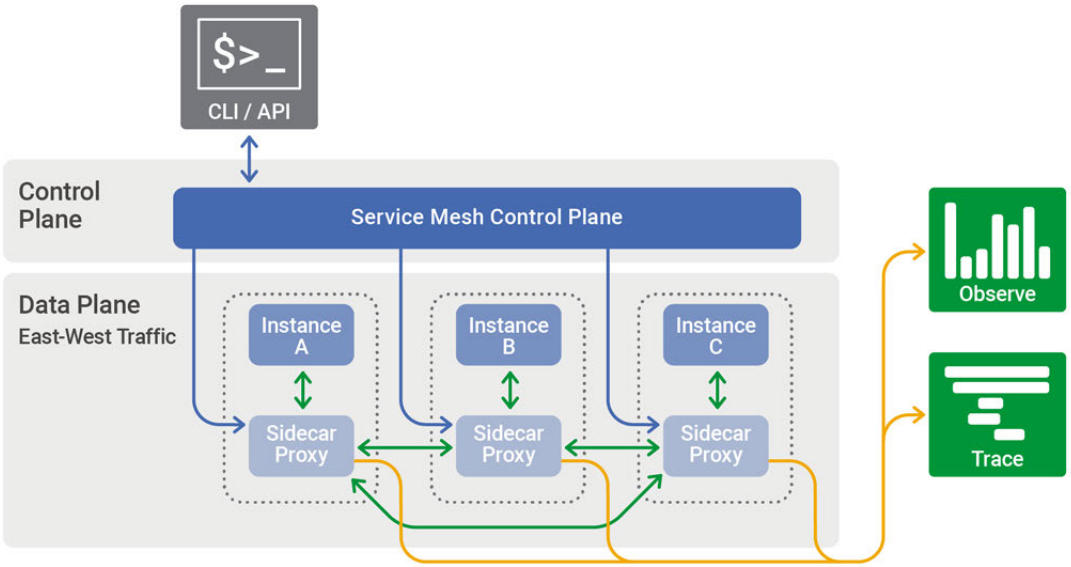

Abbildung 2: Aufbau eines Service Meshs [150]

Die Data Plane besteht aus den Sidecar Proxys. Sie ist für die Kommunikation und die Umsetzung der im vorangegangenen Abschnitt erwähnten Features verantwortlich. So gesehen ist die Data Plane für die Übersetzung, Weiterleitung und Beobachtung jedes Netzwerk Pakets zuständig, das von oder zu einem Service fließt[90].

Die Control Plane ist für die Übersetzung der zustandslosen Sidecar Proxys in ein verteiltes System zuständig. Über sie läuft die Policy und Konfiguration der Data Plane[90].

3 Bewertungskriterien

Um die verschiedenen Service Meshes miteinander vergleichen zu können, werden im Folgenden die gewählten Bewertungskriterien erklärt. Diese orientieren sich in erster Linie an den im Service Mesh verfügbaren und in Kapitel 2 genannten Features. Diese wurden in die Bereiche Routing, Observability, Resilience und Security unterteilt. Darüber hinaus sollen die durch die Implementierung erworbenen Erkenntnisse in die Bewertung einfließen. Hieraus ergeben sich die Bereiche Usability und Extensibility. Jedes Kriterium K wird auf einen Bewertungsmaßstab B abgebildet und einer Analysetechnik A zugewiesen.

3.1 Bewertungsmaßstab

Die Bewertung der Kriterien erfolgt über die im folgenden aufgelisteten Maßstäbe. Die Wahl des Maßstabs orientiert sich zum einen an der Formulierung und zum anderen am Härtegrad des Kriteriums. Weiche Kriterien werden ordinal mit einer Skala bewertet. Härtere Kriterien, die ein klares Ergebnis liefern, werden hingegen mit Ja, Nein oder der absoluten Häufigkeit bewertet.

- B1: Ja/Nein
- B2: Niedrig/Mittel/Hoch
- B3: Aufaddierte Häufigkeit
- B4: Skala von 1 bis 10

3.2 Analysetechnik

Um ein Kriterium bewerten zu können stehen verschiedene Möglichkeiten zur Verfügung. Jedes Mesh verfügt über eine Dokumentation des Anbieters, anhand derer Features eingeschätzt werden können. Darüber hinaus kann auf veröffentlichte Artikel und Bücher zurückgegriffen werden. Neben der Literaturanalyse steht auch das Fallbeispiel im jeweils implementierten Service Mesh zur Verfügung. Hieraus können Daten gewonnen werden, die die Analyse der Kriterien stützen. Für weiche und nicht messbare Kriterien, die nicht anhand Literatur oder gewonnenen Daten bewertet werden können, werden aus der Implementierung gewonnene Erfahrungen zur Analyse herangezogen.

- A1: Literaturanalyse
- A2: Objektive Analyse anhand von Daten
- A3: Subjektive Analyse

3.3 Routing

Bewertet werden soll hier, inwieweit das Service Mesh das Lenken des Datenflusses im Netzwerk unterstützt und steuern kann. Sind bereits Features enthalten, die beispielsweise ein A/B Testing möglich machen, oder werden weitere Frameworks dafür benötigt? Inwieweit lässt sich das Load Balancing über das Service Mesh konfigurieren?

Die Unterkriterien für Routing definieren sich wie folgt:

- K1: Welche Load Balancing Techniken beziehungsweise Algorithmen sind verfügbar → B3, durch A1
- K2: Ist es möglich den Datenverkehr nach Prozent auf die Instanzen zu verteilen → B1, durch A1
- K3: Ist es möglich den Datenverkehr nach Header und Pfad auf die Instanzen zu verteilen → B1, durch A1

3.4 Observability

Mit dem Kriterium Observability soll bewertet werden, welche Möglichkeiten das Service Mesh für die Überwachung des Datenstroms und der Anwendung bietet. Welche Features unterstützen Monitoring und Tracing? Welche Metriken sind in der Service Mesh Implementierung enthalten?

Die Unterkriterien für Observability definieren sich wie folgt:

- K4: Sind die Log Einträge der Service Container über die Control Plane einsehbar? → B1, durch A1
- K5: Welche der "Golden Signal" Metriken[1] werden erzeugt? → B3, durch A1
  - a) Latenz: Zeit, die der Service für die Beantwortung eines Requests benötigt
  - b) Datenverkehr: Anzahl der Anfragen pro Zeiteinheit
  - c) Errors: Fehlermeldungen, die der Service zurückgibt
  - d) Auslastung des Systems: Angabe der prozentualen Auslastung der CPU und Speicherkapazität
- K6: Werden pro HTTP Endpunkt Metriken gesammelt? → B1, durch A1
- K7: Werden TCP Metriken gesammelt? → B1, durch A1
- K8: Ist Prometheus im Service Mesh vorhanden? → B1, durch A1
- K9: Ist Grafana im Service Mesh vorhanden? → B1, durch A1
- K10: Ist ein Dashboard für das Service Mesh verfügbar? → B1, durch A1
- K11: Wie viele Tracing Backends werden unterstützt? → B3, durch A1
- K12: Ist bereits ein Tracing Backend im Service Mesh vorhanden? → B1, durch A1
- K13: Werden Access Logs erstellt? → B1, durch A1

3.5 Resilience

Ein Service Mesh kann eine zentrale Konfiguration der Features ermöglichen, die die Robustheit der Anwendung fördern. Mögliche Fehler, die zum Ausfall der Anwendung führen, sollten frühzeitig erkannt und behoben werden können. Mit dem Resilience Kriterium sollen die Features bewertet werden, die eine Robustheit des Systems fördern.

Die Unterkriterien für Resilience definieren sich wie folgt:

- K14: Ist Circuit Breaking enthalten? → B1, durch A1
- K15: Ist Retry enthalten? → B1, durch A1
- K16: Ist Timeout enthalten? → B1, durch A1
- K17: Kann man Resilience Pattern für jeden Endpunkt einzeln konfigurieren? → B1, durch A1
- K18: Ist Fault Injection möglich? → B1, durch A1
- K19: Ist Delay Injection möglich? → B1, durch A1

3.6 Security

Für eine sichere Kommunikation innerhalb der Anwendung sind Autorisierung, Identität und Zertifikatsmanagement, sowie Authentifizierung wichtig. Mit dem Security Kriterium soll bewertet werden, welche Möglichkeiten das Service Mesh diesbezüglich bietet.

Die Unterkriterien für Security definieren sich wie folgt:

- K20: Ist mTLS verfügbar? → B1, durch A1
- K21: Kann man externe CA Zertifikate und Schlüssel einbinden? → B1, durch A1
- K22: Kann man Autorisierungsregeln erstellen? → B1, durch A1

3.7 Usability

Die Usability des Systems wird anhand der Implementierung des Service Meshs bewertet. Zur Usability gehören zum einen, wie einfach das Service Mesh in eine bestehende Anwendung integriert werden kann. Muss hierzu beispielsweise das Projekt angepasst werden? Wie verständlich ist die Dokumentation und wie sieht die Hilfe zur Fehlerbehebung aus? Zum anderen soll bewertet werden, wie intuitiv die GUI und die Handhabung der Control Plane ist.

Die Unterkriterien für Usability definieren sich wie folgt:

- K23: Inwieweit reicht die Dokumentation des Anbieters für eine problemlose Installation? → B4, durch A3
- K24: Wie viele Schritte werden zur Installation benötigt, bis die Anwendung und das Service Mesh laufen? → B3, durch A1
- K25: Muss für die Installation der Code beziehungsweise das Projekt angepasst werden? → B1, durch A1
- K26: Wie ausführlich ist eine mögliche Fehlerbehebung in der Dokumentation beschrieben? → B4, durch A3 und A1
- K27: Wird ein Graph für ein besseres Verständnis der Zusammenhänge erzeugt? → B1, durch A1
- K28: Wie intuitiv ist das Dashboard aufgebaut? → B2, durch A3
  - a) Metriken finden
  - b) Konfigurationen finden
  - c) Übersichtlichkeit der Oberfläche
- K29: Wie flexibel ist die Auswahl der Plattform, auf der das Service Mesh läuft? → B4, durch A1 und A3
- K30: Können die Sidecar Proxys automatisch injiziert werden? → B1, durch A1
- K31: Wie viele Protokolle werden unterstützt? → B3, durch A1
- K32: Wie hoch ist der Speicherverbrauch? → B3, durch A2

- K33: Kann das Service Mesh über das Dashboard konfiguriert werden? → B1, durch A1

3.8 Extensibility

Mit diesem Kriterium soll bewertet werden, inwieweit das Service Mesh erweitert werden kann. Hierfür soll bewertet werden, ob ein Service Mesh vielseitig konfigurierbar und durch eigene Implementierungen erweiterbar ist und wie das Service Mesh Drittanbieter unterstützt, beziehungsweise von diesen unterstützt wird. Zu diesen zählen zum einen das Service Mesh Interface (SMI) und zum anderen der Service Mesh Hub. Der Service Mesh Hub ist ein Service Mesh Management Tool, mit dem verschiedene Service Meshes über eine Oberfläche überwacht werden können [151]. Das Service Mesh Interface ist eine standardisierte Schnittstelle für auf Kubernetes laufende Service Meshes, auf der Erweiterungen aufsetzen können[52].

Die Unterkriterien für Extensibility definieren sich wie folgt:

- K34: Inwieweit lässt sich das Service Mesh über die Standard Installation hinaus anpassen und um Funktionalität erweitern? → B4, durch A1
- K35: Kann das Service Mesh mit Containern oder VMs außerhalb eines Kubernetes Clusters erweitert werden? → B1, durch A1
- K36: Werden Multicluster unterstützt? → B1, durch A1
- K37: Wird das SMI unterstützt? → B1, durch A1
- K38: Wird das Service Mesh vom Service Hub unterstützt? → B1 durch A1

4 Fallbeispiel

Die Basis der praktischen Analyse bildet eine auf Kubernetes laufende Microservice Anwendung. Es soll untersucht werden, wie ein Service Mesh in eine laufende Anwendung integriert werden kann. Im Folgenden werden der Aufbau des Fallbeispiels, die einzelnen Services und die Kommunikation innerhalb der Anwendung beschrieben.

4.1 Anwendungsumgebung

Um eine Vergleichbarkeit gewährleisten zu können, wurde jedes Service Mesh auf einer eigenen in Virtual Box laufenden Virtuellen Maschine implementiert. Als Betriebssystem wurde Ubuntu 18.04.4 gewählt, da die meisten Service Mesh Installationen für Linux-basierte Systeme ausgelegt sind. Die Konfiguration der VM kann in Abschnitt A.1 eingesehen werden.

Damit für jedes Service Mesh ein eigenes und unabhängiges Kubernetes Cluster existiert, wird dieses lokal in der VM betrieben. Hierfür wurde Minikube in der Verion v1.9.0 installiert. Für Minikube wird auch die Installation von Docker benötigt. Für das Fallbeispiel wurde die Version 19.03.8 installiert. Wichtig sind hierfür auch die 2 Prozessoren in der VM Konfiguration. Da eine verschachtelte Virtualisierung mit Virtual Box nicht möglich ist, muss beim Starten von Minikube über ein Parameter der VM Driver auf none gesetzt werden. Hiermit läuft Minikube direkt auf der VM[134]. Über die apiserver Parameter wird der API Server von außen erreichbar gesetzt. Auch wenn diese eigentlich mit dem None Driver überflüssig wären, ist andernfalls kein fehlerfreier Start möglich gewesen.

Minikube Start Befehl:

```
minikube start
        --vm-driver=none
        --apiserver-ips 127.0.0.1
        --apiserver-name localhost
```

Per Default wird hierbei die neuste Kubernetes Version genutzt. In diesem Anwendungsfall wurde Kubernetes v1.18.0 genutzt. Um mit Kubernetes arbeiten zu können, muss zudem das Commandline Tool "kubectl"installiert werden.

Weiterhin wurde für die Entwicklung der Microservice Anwendung Go installiert. Für die Nutzung von Modulen in Go muss mindestens Version 1.12 genutzt werden. Für das Fallbeispiel wurde Version 1.13.3 linux/amd64 installiert.

4.2 Architektur der Microservice Anwendung

Die Microservice Anwendung "coffee-house-fallbeispiel" basiert auf der Anwendung "coffee-house", die im Wahlpflichtfach "Microservices mit Java und Go" im Sommersemester 2019 bei Prof. Dr. Stefan Sarstedt an der HAW Hamburg genutzt wurde. Diese bestand aus zwei Services mit den dazugehörigen Backend Services und wurde im Zuge des Fallbeispiels auf fünf erweitert[7]. Ziel ist es möglichst viel Netzwerk Verkehr erzeugen zu können, um diesen dann mit dem entsprechenden Service Mesh überwachen zu können. Die Anwendung simuliert ein Café, in dem Kaffee bestellt und zubereitet wird.

Die Services gliedern sich in Receptionservice, Baristaservice, Stockservice, Managerservice und Supplierservice.

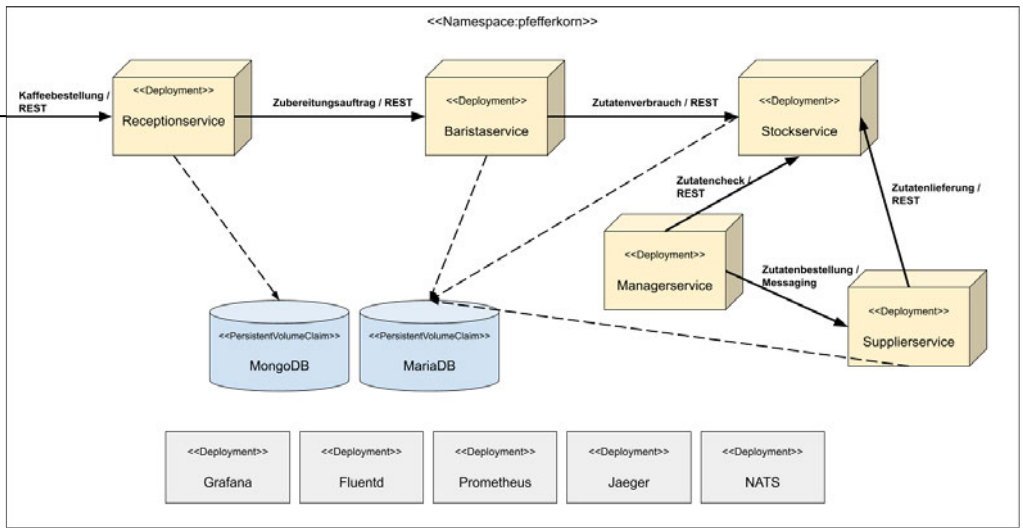

Abbildung 3: Aufbau des Fallbeispiels

[7] https://git.haw-hamburg.de/ace512/coffee-house-fallbeispiel inkl. Quellenangaben in der README

In Abbildung 3 ist die Übersicht der Services im Kontext des Fallbeispiels und ihre Interaktion miteinander zu sehen.

4.2.1 Receptionservice

Der Receptionservice ist für die Bestellungen zuständig. Über die REST Schnittstelle /orders kann per HTTP POST Request ein Kaffee bestellt werden.

Der Request Body muss ein JSON im folgenden Format besitzen:

```
{"customername":        "",
 "CreatedAt":           "YYYY-MM-DDTHH:MM:SS",
 "OrderItems":          [{
                        type": "",
                        "size": ""
                        }]
}
```

Der Name kann hierbei ein beliebiger String sein. Type des OrderItems muss "espresso", "cappuccino" oder "latte" sein. Size kann "single" oder "double" sein.

Um die Bestellungen zu speichern wird MongoDB genutzt. Diese werden dann der Collection orders hinzugefügt. Der Receptionservice leitet die Bestellung per HTTP POST Request an den Baristaservice weiter.

4.2.2 Baristaservice

Der Baristaservice ist für die Zubereitung des Kaffees zuständig. Über die REST Schnittstelle /orders nimmt er Bestellungen entgegen und bearbeitet diese der Reihe nach. Hierfür fragt er per HTTP POST Request Zutaten beim Stockservice an.

```
{"ReceptionOrderId:     "",
 "customername":        "",
 "CreatedAt":           "YYYY-MM-DDTHH:MM:SS",
 "OrderItems":          [{
                        type": "",
                        "size": ""
                        }]
}
```

Der Request Body enthält die Bestellung des Receptionservice und die Id, welche als Byte Array übergeben wird. Der Baristaservice speichert die Bestellungen in einer MariaDB. Hierfür wird die Datenbank orders angelegt.

4.2.3 Stockservice

Der Stockservice simuliert das Lager. Es werden Kaffeebohnen, Milch und Zucker gelagert. Diese werden in einer MariaDB Datenbank stock gespeichert. Über die Endpunkte /coffeebeans, /milk und /sugar kann der Bestand per POST Request geändert werden. Über einen GET Request kann der Bestand abgefragt werden. Ein entsprechender Wert wird über den URL-Parameter quantity übergeben. Ein Beispiel für einen POST Request, der dem Lager 50 Einheiten Kaffeebohnen hinzufügt, hätte also das folgende Format:

```
POST /coffeebeans?quantity=50
```

Um einen Kaffee zuzubereiten wird also pro Zutat ein Request an den Stockservice geschickt. Eine Visualisierung mittels Sequenzdiagramm befindet sich im Anhang unter Abschnitt A.3. Pro Bestell-Request werden also vier weitere Requests erzeugt.

4.2.4 Managerservice

Der Managerservice überwacht den Bestand des Stockservices. Hierfür wird in einem vorgegebenen Zeitintervall der Bestand von Kaffeebohnen, Milch und Zucker abgefragt. Wenn die Menge einen bestimmten Schwellenwert unterschreitet, wird eine Bestellung für den Supplierservice erzeugt. Diese wird als Supplies Order Event mit dem Betreff supplies veröffentlicht. Hierfür wird NATS Streaming genutzt (siehe Unterabschnitt 4.2.6). Eine Bestellung enthält den gesammelten Bedarf für Kaffeebohnen, Milch und Zucker.

```
type Order struct {
        gorm.Model [8]
        CoffeeBeans string
        Milk        string
        Sugar       string
}
```

[8] gorm.Model enthält die Felder ID, CreatedAt, UpdatedAt, DeletedAt. GORM ist die ORM Bibliothek für Go.

4.2.5 Supplierservice

Der Supplierservice ist dafür zuständig den Bestand an Kaffeebohnen, Milch und Zucker im Stockservice aufzufüllen. Hierfür abonniert er das Supplies Order Event mit dem Betreff supplies. Wenn er ein Event empfängt, schickt er entsprechende POST Requests an den Stockservice. Eingehende Bestellungen werden in der Datenbank supplier gespeichert. Hierfür wird MariaDB genutzt.

4.2.6 Backendservices

Die Backendservices der Anwendung bestehen aus Fluentd, Grafana, Jaeger, NATS und Prometheus. Fluentd ist dafür zuständig Logging Daten der Services zu sammeln. Grafana bietet eine Visualisierung für Monitoring und Analyse von Metriken. Diese werden mit Prometheus über den Endpunkt /metrics in den entsprechenden Services abgefragt und gesammelt. Jaeger ist eine Software, mit der sich Tracing realisieren und visualisieren lässt. Mit NATS wird Messaging mit dem Publish-Subscribe Model ermöglicht.

4.3 Anwendung im Kubernetescluster deployen

Im Rootverzeichnis der Anwendung befindet sich ein Makefile, in dem die make Befehle für den Umgang mit der Anwendung gesammelt sind. Wenn Kubernetes läuft, muss ein Namespace mit dem Namen "pfefferkorn" erzeugt werden. Um die Anwendung in dem Cluster zu deployen, wird der deploy-Befehl *make deploy* ausgeführt. Dieser Befehl führt die nötigen kubectl Befehle für das Deployment aus (siehe Abschnitt A.2).

5 Istio

Istio wurde als Projekt 2017 bekannt gegeben [9, S. 20] und ist seit dem 1.0er Release im Juli 2018 produktionsreif[149]. Es wurde von Google, IBM und Lyft basierend auf der Infrastruktur von Google entwickelt und gilt als das bekannteste Service Mesh[146, S. 14]. Istios Komponenten sind in Go geschrieben, der genutzte Proxy Envoy in C++[4].

5.1 Architektur

Am 5. März 2020 wurde Istio 1.5 veröffentlicht [53]. Die neue Version bringt im Vergleich zu den vorangegangenen Versionen auch eine neue Architektur mit sich, indem sie mit istiod die Komponenten der Control Plane zu einer Binary zusammenfasst, anstatt hierfür einzelne Services zu nutzen. Hiermit soll Istio einfacher zu installieren und zu betreiben sein[55]. In der Vergangenheit wurde Istio oft als komplex und anspruchsvoll zu konfigurieren bezeichnet[146, S. 14].

Istios Data Plane besteht aus Envoy Proxys[54]. Die Control Plane enthält die Komponenten Pilot, Citadel und Galley, wie in Abbildung 4 zu sehen ist.

Die Architekturbeschreibung bezieht sich auf die für das Fallbeispiel installierte Version 1.5.1. Die nach der Analyse veröffentlichte aktuelle Version 1.7 kann hiervon abweichen.

Die Komponente Pilot bietet einen Service-Discovery Dienst für die Sidecar Proxys an. Sie ist zudem für das Routing, Traffic Management und die Resilience Features zuständig. Pilot übersetzt die Routing Regeln für den Datenverkehr in Envoy-spezifische Konfigurationen und leitet diese zur Laufzeit an die Sidecar Proxys[54].

Die Komponente Citadel ist für die Sicherheitsfeatures im Service Mesh zuständig. Sie managt Identität und Berechtigungsnachweise und ermöglicht somit eine Service-to-Service und End-User Authentifizierung. Mit Citadel ist es möglich seinen Datenverkehr im

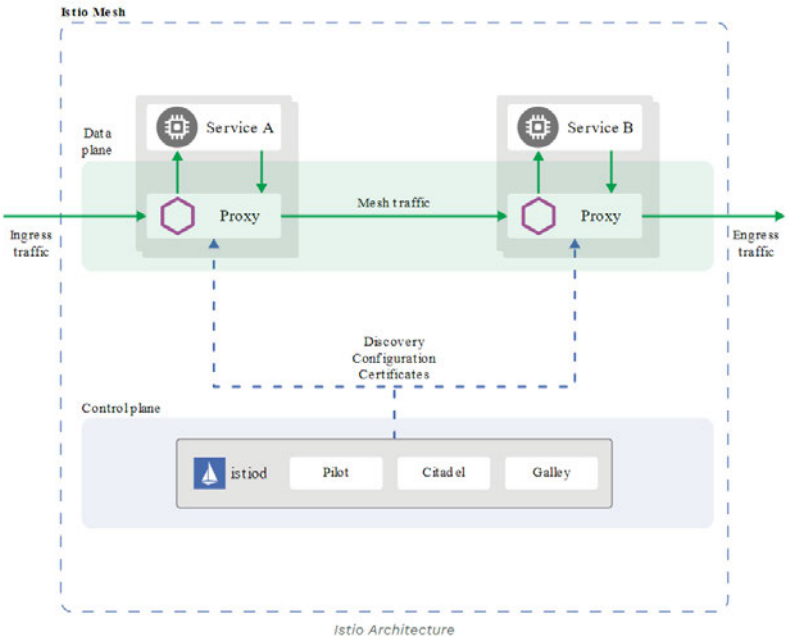

Abbildung 4: Istio Architektur [54]

Service Mesh zu verschlüsseln und Autorisierungs-Policies auf Anwendungsebene aufzustellen[54].

Galley ist dafür zuständig Konfigurationen zu sammeln, zu validieren und im Service Mesh zu verbreiten[54].

5.2 Installation

Für die Analyse wurde Istio 1.5.1 installiert. Nach Download der Installationsdatei muss istioctl der Umgebungsvariable PATH hinzugefügt werden. Hierüber werden alle weiteren Istio betreffende Schritte ausgeführt[57].

Mit *istioctl install* wird Istio installiert. Hierfür kann auch ein individuelles Profil angelegt werden. Es wird empfohlen, dem Namespace, im Fall des Fallbeispiels pfefferkorn, das Label istio-injection=enabled hinzuzufügen, das eine automatische Injektion des Sidecar Proxys zulässt[57].

Als nächstes wird die Anwendung deployed. Neben den Containern der Anwendung laufen nun auch die Sidecar-Proxys im Pod. Damit die Anwendung von außen erreichbar ist, muss ein Istio Ingress Gateway[56] erstellt werden. Die Konfiguration des Namespaces wird hiernach mit istioctl analyze überprüft und die Variablen INGRESS_HOST und INGRESS_PORT gesetzt (siehe auch Abschnitt A.4). Diese ermöglichen den Zugriff auf das Gateway[57].

Mit dem Befehl *minikube tunnel* wird ein Minikube Tunnel erzeugt, der den Traffic zum Ingress Gateway leitet. Zum Schluss wird die Variable GATEWAY_URL gesetzt und die Installation ist abgeschlossen[57].

5.3 Bewertung

Im Folgenden werden die Kriterien, die in Kapitel 3 genannt werden auf Istio angewandt. Hierfür wird für jeden Bereich eine Tabelle erstellt, in der neben der Analyse das Bewertungskriterium, Bewertungsmaßstab sowie das Ergebnis dargestellt werden. Die Bewertung bezieht sich auf die installierte Version 1.5.1 und Features, die im Istio Anwendungskatalog enthalten sind. Features, die nur durch externe Erweiterungen möglich sind, werden nicht einbezogen.

5.3.1 Routing

In Tabelle 1 sind die Ergebnisse der Analyse der Routing Features und deren Nachweis aufgelistet. Istios Konfiguration wird über Virtual Services in YAML Dateien definiert. Über diese lässt sich der Datenverkehr vielseitig konfigurieren und steuern. Datenverkehr, der von außerhalb des Meshs kommt, läuft über das Ingress Gateway. In der Beschreibung der Installation wurde bereits die Konfiguration des Ingress Gateways beschrieben. Für Istios Lauffähigkeit ist das notwendig, jedoch übernimmt Istio hiermit die Kontrolle über sämtlichen Ingress Datenverkehr[88]. Somit müssen alle Services im Kubernetes Cluster Istio bekannt sein, damit diese Daten empfangen können.

Tabelle 1: Istio Bewertung Routing

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K1	Round-Robin, Random, Weighted und Least Requests[58]	B3	4
K2	Über einen Virtual Service werden Traffic Regeln definiert[59]. Hierüber kann dann auch der Datenverkehr nach Prozent aufgeteilt werden.	B1	Ja
K3	Über einen Virtual Service lassen sich auch anhand von Headern Regeln aufstellen[59].	B1	Ja

5.3.2 Observability

Wie sich in Tabelle 2 zeigt, bietet Istio viele Obervability Features an. Besonders umfangreich ist das integrierte Kiali Dashboard, das unter anderem einen detaillierten Graph der Anwendung erstellt. Es wird zwar kein Tracing Backend mit installiert, jedoch unterstützt Istio eine Reihe von Anbietern und kann entsprechend erweitert werden. Prometheus und Grafana können bei der Installation mit installiert werden und erleichtern es somit Metriken einfach zu sammeln und konfigurierbar darzustellen. Diese integrierten Metriken können jedoch nur pro Service und nicht pro HTTP-Endpunkt gesammelt werden.

5.3.3 Resilience

Über die Virtual Services lassen sich alle analysierten Resilience Pattern einstellen (vgl. Tabelle 3). Diese sind jedoch nur pro Service und nicht pro Endpunkt konfigurierbar. Über die analysierten Features hinaus bietet Istio auch Traffic Mirroring an[89]. Hiermit lässt sich eingehender Datenverkehr kopieren und an noch nicht veröffentlichten Services testen, um einen Übergang zu einer neuen Version zu vereinfachen.

Tabelle 2: Istio Bewertung Observability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K4	Über das Kiali Dashboard lassen sich unter Workloads > Namespace:pfefferkorn > service die entsprechenden Logeinträge der gewünschten Container einsehen.	B1	Ja
K5	Pro Service werden per default die Anzahl an eingehenden Anfragen, Dauer der Anfragen, Anfragengröße und Antwortgröße gemessen[60].	B3	2
K6	Es werden bisher keine Metriken pro HTTP Endpunkt angeboten[61].	B1	Nein
K7	Für TCP Verbindungen werden die Metriken: gesendete Bytes, empfangene Bytes, geöffnete Verbindungen und geschlossene Verbindungen gemessen[60].	B1	Ja
K8	Es gibt die Möglichkeit ein Prometheus Beispiel bei der Installation mit zu installieren. Bestehende Prometheus Deployments können einfach über eine Configmap Konfigurationsdatei integriert werden. Die Service Mesh spezifischen Endpunkte beinhalten die Möglichkeit Metriken zu scrapen[62].	B1	Ja
K9	Es gibt die Möglichkeit Grafana mit voreingestellen Dashboards bei der Installation mit zu installieren[63].	B1	Ja
K10	Für Istio wird ein Kiali Dashboard angeboten, worüber man das Service Mesh auch konfigurieren kann[64].	B1	Ja
K11	Es werden Jaeger, Zipkin und Lightstep, sowie alle weiteren Zipkin kompatiblen Backends (Apache SkyWalking, Pitchfork) unterstützt[65].	B3	5
K12	Es wird nicht automatisch ein Tracing Backend mit installiert[66].	B1	Nein
K13	Die Envoy Proxys generieren mit entsprechender Konfiguration Access Logs [85].	B1	Ja

Tabelle 3: Istio Bewertung Resilience

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K14	In Istio lässt sich ein Circuit Breaker einrichten[67].	B1	Ja
K15	Über einen Virtual Service lassen sich Retry Konfigurationen anlegen[68].	B1	Ja
K16	Über einen Virtual Service lassen sich Timeout Konfigurationen anlegen[69].	B1	Ja
K17	Über einen Virtual Service ist es nur möglich Regeln pro Service zu definieren[70].	B1	Nein
K18	Fault Injection, in Istio Aborts genannt, ist per Konfiguration über einen Virtual Service möglich[71].	B1	Ja
K19	Über einen Virtual Service lässt sich eine Delay Injection einrichten[71].	B1	Ja

5.3.4 Security

Istio bietet für die im Security Kriterium beschriebenen gewünschten Eigenschaften zu Identität, Autorisierung und Authentifikation konfigurierbare Features an. Es lassen sich Policies sowohl zur Autorisierung als auch Authentifikation[87] einstellen, die das Mesh und den Datenverkehr absichern.

Tabelle 4: Istio Bewertung Security

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K20	Die mTLS Verbindung wird über Envoy Proxies realisiert. Istio tunnelt hierfür lediglich den Datenverkehr[72].	B1	Ja
K22	Externe CA Zertifikate und Schlüssel können eingebunden werden[73].	B1	Ja
K23	Anhand von Selektoren, Regeln, Bedingungen und Operationen können Autorisierungsregeln erstellt werden [74].	B1	Ja

5.3.5 Usability

Wer Istio mit den grundlegenden Funktionalitäten installieren möchte, bekommt mit der ausführlichen Dokumentation eine problemlos umsetzbare Schritt-für-Schritt Anleitung, mit der das Mesh schnell deploybar ist. Komplexer wird es, wenn man die vielen Features von Istio nutzen möchte. Diese werden fast ausschließlich über YAML Dateien deployed. Hierfür ist ein entsprechendes Verständnis der Struktur notwendig. Die in Tabelle 5 und Tabelle 6 analysierten Kriterien zeigen, dass Istio Wert auf Nutzerfreundlichkeit legt. Mit einem entsprechenden Wissen, lässt sich das Service Mesh in seinen, über Virtual Services einstellbaren, Features über das Dashboard anpassen. Hierdurch ergibt sich die Möglichkeit die Anpassung des laufenden Service Meshs unabhängig von dessen Installation zu betreiben.

Tabelle 5: Istio Bewertung Usability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K23	Das Service Mesh war problemlos installierbar. Die Dokumentation umfasst alle Handlungsschritte für eine Grundfunktionalität[57]. Es mussten keine Istio betreffenden zusätzlichen Schritte recherchiert werden.	B4	9
K24	Download Istio, istioctl der Umgebungsvariablen Path hinzufügen, Istio installieren[57], Autoinject Label im Namespace setzen, Anwendung deployen, Ingess Gateway definieren, Ingesshost und Ingessport als Umgebungsvariable setzen um die Anwendung von Außen erreichbar zu machen[75].	B3	7
K25	Für die Installation muss keine Datei der Anwendung angepasst werden[57]. Jedoch sind für eine aussagekräftige Abbildung im Kiali Dashboard die Labels app und Service unter Metadata in der Deployment YAML entscheidend und müssen hierfür ggf. hinzugefügt werden.	B1	Nein
K26	Die Dokumentation umfasst Debugging Hilfen für das Service Mesh und die Konfiguration[76] sowie Hilfestellungen für die am häufigsten auftretenden Fehler[77].	B4	10
K27	Im Kiali Dashboard wird ein detaillierter Graph angezeigt, dessen Ansicht konfiguriert werden kann[78].	B1	Ja

Tabelle 6: Istio Bewertung Usability 2

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K28	Istios Dashboard Kiali besteht aus einer Übersicht von Graph, Applications, Workloads, Services und Istio Konfigurationen, die man pro Namespace abrufen kann. Metriken lassen sich über Workloads und Services finden. Wobei über Services nur Inbound Metriken zu sehen sind und über Workload auch Outbound Metriken. Eine derartige Trennung ist nicht intuitiv. Konfigurationen lassen sich schnell über den Menü Tab Istio Config finden. Hier werden auch selbst angelegte Virtual-Services angezeigt. Die Oberfläche ist bis auf die des Graphen einfach gehalten. Dieser ist ohne die Legende schwer lesbar.	B2	Hoch
K29	Istio läuft auf Kubernetes, mit Consul und auf individuellen Virtuellen Maschinen[79].	B4	8
K30	Man kann als Attribut im Namespace einstellen, dass Proxies automatisch injiziert werden[57].	B1	Ja
K31	Es werden die Protokolle grpc, grpc-web, http, http2, https, mongo, mysql, redis, tcp, tls, udp unterstützt [80].	B3	11
K32	Mit dem Linux top Befehl berechnet, abzüglich des Verbrauchs vor der Installation (siehe Abschnitt A.10 und Abschnitt A.11) in KiB Mem used.	B3	2.334.268
K33	Unter dem Menüpunkt Istio Config lassen sich bisher erstellte Konfigurationen über die YAML Datei ändern und neue anlegen[86]. Weiterhin lässt sich das Routing über den Service im Kiali Dashboard ändern.	B2	Ja

5.3.6 Extensibility

Istio bringt bereits ohne Drittanbieter eine große Anzahl Konfigurationsmöglichkeiten mit. Besonders hervorzuheben ist hier die Erweiterbarkeit über WebAssembly. Im März 2020 wurde erstmals ein low-level Application Binary Interface (ABI) für Proxys zur Verfügung gestellt[5]. Eigene Erweiterungen können zudem im Solo.io WebAssembly Hub angeboten und andere abgerufen und genutzt werden. Neben WebAssembly unterstützt Istio auch das Service Mesh Interface und kann über den Service Mesh Hub angebunden werden. Die Bewertung der Kriterien befinden sich in Tabelle 7.

Tabelle 7: Istio Bewertung Extensibility

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K34	Istio und die Envoy Proxys lassen sich mit WebAssembly erweitern [81]. Für diverse Erweiterungen kann der WebAssembly Hub von solo.io genutzt werden oder eigene Erweiterungen geschrieben werden[5]. Hiermit werden in erster Linie die Proxys und nicht die Control Plane erweitert. Über die IstioOperator API kann das Service Mesh bei der Installation nach Bedarf konfiguriert werden[82].	B4	8
K35	Virtuelle Maschinen können angebunden werden[83]. Hierfür muss die Istio Installation konfiguriert werden, sodass die VM sich mit dem Mesh verbinden kann.	B1	Ja
K36	Es können Multicluster mit replizierten Control Planes oder einer geteilten Control Plane erstellt werden[84].	B1	Ja
K37	Istio wird im Ecosystem der Service Mesh Interface Website gelistet[52].	B1	Ja
K38	Istio in der Version 1.5 wird vom Service Mesh Hub unterstützt[151].	B1	Ja

6 Linkerd 2

Linkerd wurde ursprünglich in Scala JVM-basiert für verschiedene Plattformen entwickelt[137]. Nachdem das Ergebnis jedoch als Latenz-lastig und komplex galt, entschied sich Buoyant Linkerd nochmal von Grund auf neu zu entwickeln und im Juli 2018 wurde dann Linkerd 2 veröffentlicht[3]. Die neue Version ist in Go für die Control Plane und in Rust für die Data Plane geschrieben[137].

6.1 Architektur

6.1.1 Control Plane

Die Linkerd Control Plane (vgl. Abbildung 5) besteht aus einer Reihe von Services, die auf Kubernetes im linkerd Namespace laufen[113]. Das controller Deployment stellt mit dem public-api Container eine Schnittstelle für die CLI und das Dashboard zur Verfügung. Der destination Container fungiert als Service Discovery Service. Der identity Container stellt eine Certificate Authority für die Kommunikation mit mTLS zur Verfügung. Der Proxy Injector wird über jeden neu erstellten Pod benachrichtigt. Wenn eingestellt ist, dass Proxies automatisch injiziert werden sollen, fügt er dem Pod einen initContainer und den Sidecar Proxy hinzu. Mit dem Service Profile Validator werden neu angelegte Service Profile validiert, bevor sie gespeichert werden. Tap ist ein Deployment, das auf Anfrage aus dem CLI und Dashboard Requests und Responses in Echtzeit mittels eines Streams überwacht. Das Dashboard wird über das web Deployment zur Verfügung gestellt[113].

6.1.2 Data Plane

Linkerds Data Plane besteht aus leichtgewichtigen Proxies, die in Rust geschrieben sind. Beim Injizieren durch die Control Plane werden ein Init Container und der Sidecar Container erstellt und dem entsprechenden Pod hinzugefügt. Der Init Container konfiguriert

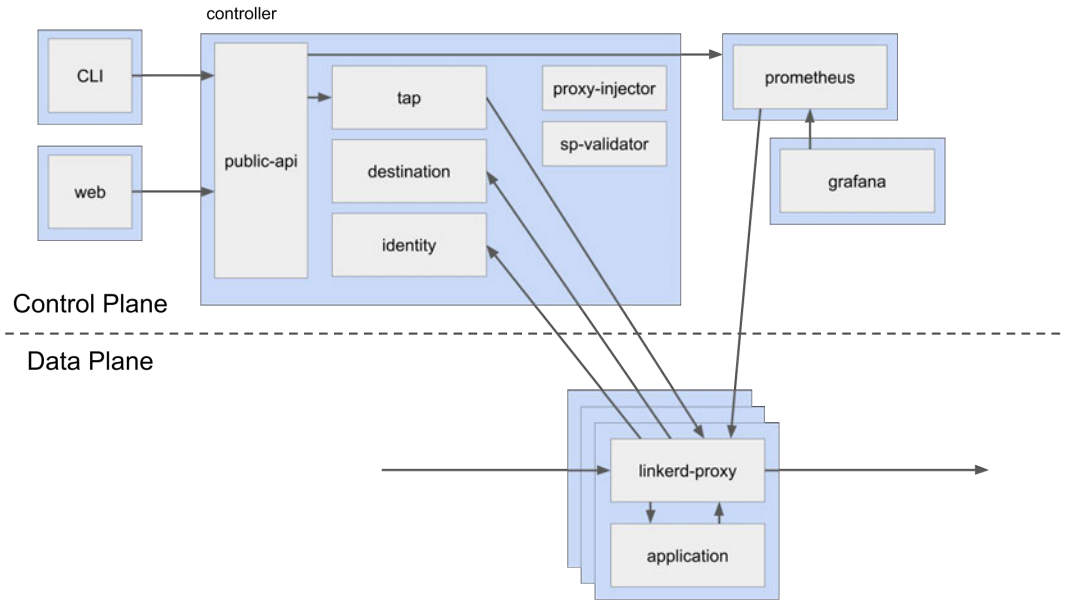

Abbildung 5: Linkerd Architektur [113]

die IP Tables, sodass der ein- und ausgehende Traffic korrekt über den Sidecar Proxy geleitet wird[113].

6.2 Installation

Für den Vergleich der Service Meshes wurde Linkerd in der Version 2.7.1 installiert. Für die Installation wird Kubernetes in der Version 1.13 und aufwärts benötigt. Als erster Schritt wird das Command Line Interface linkerd installiert. Das ist manuell oder über Homebrew möglich. Als nächstes wird überprüft, ob das Kubernetes Cluster kompatibel konfiguriert ist. Nun wird Linkerd in das Cluster unter dem Namespace linkerd deployed und die Installation erneut geprüft[123] (siehe hierzu Abschnitt A.5).

Nachdem die eigene Anwendung deployed ist, kann man die Sidecar Proxys injizieren. Für das Fallbeispiel sieht der Befehl wie folgt aus:

```
kubectl get -n pfefferkorn deploy -o yaml  | linkerd inject
        --skip-outbound-ports=4222,8222
        --skip-inbound-ports=4222,8222 - | kubectl apply -f -
```

Konkret werden in die Deployment YAML Dateien der laufenden Anwendung mit linkerd inject die Sidecar Proxy injiziert. Diese können als Datei oder als Stream übergeben werden[124]. Es werden daraufhin neue Pods mit dem Service und dem injizierten Proxy gestartet. Sobald dieser läuft, wird der ursprüngliche Pod heruntergefahren.

Dem inject Befehl können verschiedene Parameter übergeben werden. Im Fall des Fallbeispiels müssen die Parameter –skip-inbound-ports und –skip-outbound-ports mit den Werten 4222 und 8222 übergeben werden. Diese sind die Ports des NATS-Streaming Services. Da dieser ein "Server-speaks-first" Protokoll nutzt, kann Linkerd dieses nicht erkennen[130] und das führt zu Fehlermeldungen beim Versuch der anderen Services sich mit dem NATS-Streaming Service zu verbinden. Für eine MySQL oder MongoDB Datenbank trifft zwar auch das "Server-speaks-first" Protokoll zu, jedoch ließ sich durch das Fallbeispiel feststellen, dass die Ports von Linkerd erkannt und automatisch ignoriert werden.

6.3 Bewertung

Im Folgenden werden die Kriterien, die in Kapitel 3 genannt werden auf Linkerd angewandt. Hierfür wird für jeden Bereich eine Tabelle erstellt, in der neben der Analyse das Bewertungskriterium, Bewertungsmaßstab sowie das Ergebnis dargestellt werden. Da Linkerds Features zum Teil auf dem Service Mesh Interface beruhen, werden diese als Teil von Linkerds Anwendungskatalog mit bewertet.

6.3.1 Routing

Linkerd bietet über den Exponentially Weighted Moving Average Algorithmus (EWMA) einen Load Balancer für den Datenverkehr an (vgl. Tabelle 8). Dieser sendet die Anfragen an den schnellsten Endpunkt. Darüber hinaus bietet Linkerd selbst jedoch keine Routing Features an. Um für beispielsweise ein Canary Deployment den Datenverkehr zu teilen, muss das Service Mesh Interface genutzt werden. Flagger setzt auf Linkerd auf und über Custom Ressources kann dann die Verteilung des Datenverkehrs eingestellt werden[114].

Tabelle 8: Linkerd Bewertung Routing

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K1	Linkerd nutzt den Algorithmus EWMA[126].	B3	1
K2	Über das Service Mesh Interface und Flagger kann der Datenverkehr aufgeteilt werden [136].	B1	Ja
K3	Flagger bietet für Linkerd keine Aufteilung des Traffics basierend auf HTTP Übereinstimmungen, wie beispielsweise dem Header [13].	B1	Nein

6.3.2 Observability

Linkerd ist darauf ausgerichtet schnell und einfach die grundlegenden Metriken erfassen und veranschaulichen zu können. Zu diesen gehören das Anfragevolumen, die Erfolgsquote der Anfragen und die Latenz der Aufrufe. Die Metriken können für HTTP, HTTP2 und gRCP gesammelt werden. Zudem werden die ein- und ausgehenden Bytes als TCP Metriken aufgezeichnet. Prometheus und Grafana sind bei der Installation dabei, um die Daten zu sammeln und zu veranschaulichen, wobei diese nur für 6 Stunden vorgehalten werden und exportiert werden sollten[131]. Hervorzuheben bei Linkerd ist, dass die Metriken nicht nur pro Service, sondern auch pro Endpunkt gesammelt werden können.

Es ist zwar kein Tracing Backend in der Installation enthalten, jedoch können alle Backends, die von Open Census unterstützt werden, integriert werden (siehe Tabelle 9). Es werden keine Access Logs generiert, jedoch lassen sich eingehende Anfragen über den linkerd tap Befehl überwachen, der auf den eingehenden Datenstream einer angegebenen Ressource hört[129].

Tabelle 9: Linkerd Bewertung Observability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K4	Für die Log Einträge muss entweder ein Backendservice zum Sammeln der Logeinträge oder der direkte Zugriff in den Container genutzt werden. Es ließ sich keine Möglichkeit finden die Log Einträge der Services über das Dashboad oder linkerd Befehle einzusehen.	B1	Nein
K5	Es lassen sich Latenz, Datenverkehr und Errors überwachen[113].	B3	3
K6	Mit Linkerd lassen sich Metriken pro Service, Aufruf und pro Route sammeln[131].	B1	Ja
K7	Es können auf TCP-Level ein- und ausgehende Bytes gemessen werden[131].	B1	Ja
K8	Linkerd nutzt Prometheus zum Sammeln der Metriken[113].	B1	Ja
K9	Linkerd nutzt Grafana für die Darstellung der Metriken[113].	B1	Ja
K10	Die Komponente Web bietet ein Linkerd Dashboard an[113].	B1	Ja
K11	Es können alle Tracing Backends genutzt werden, die von OpenCensus unterstützt werden[120]: Honeycomb, Wavefront, AWS X-Ray, Datadog, Apache Kafka, OpenCensus, Jaeger, Zipkin und Stackdriver[139].	B3	9
K12	In der Architekturbeschreibung wurde kein Tracing Backend aufgeführt[113].	B1	Nein
K13	Linkerd generiert keine Access Logs. Eingehende Requests können über linkerd tap erfasst werden[129].	B1	Nein

6.3.3 Resilience

Wie Tabelle 10 zu entnehmen ist, bietet Linkerd an Resilience Pattern nur Retry und Timeout an. Diese lassen sich pro Route konfigurieren. Auch wenn Linkerd kein Fault Injection beinhaltet, so kann dies über die Verteilung des Datenverkehrs und einen gewollt fehlerhaften Service erreicht werden. Hierfür befindet sich auch eine detaillierte Beschreibung in der Dokumentation[121].

Tabelle 10: Linkerd Bewertung Resilience

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K14	Bisher ist kein Circuit Breaking Feature enthalten[122].	B1	Nein
K15	In der Dokumentation ist die Konfiguration von Retries enthalten[117].	B1	Ja
K16	Linkerd bietet die Konfiguration von Timeouts an[118].	B1	Ja
K17	Retry und Timeout lassen sich für jede Route konfigurieren[117][118].	B1	Ja
K18	Fault Injection ist nur möglich, indem man einen weiteren Service anlegt, der eine gewünschte Fehlermeldung zurückgibt und mit Traffic Split einen Teil der Anfragen an diesen leitet[121].	B1	Nein
K19	In der Dokumentation von Linkerd gibt es keinen Unterpunkt zu Delay Injection oder etwas Vergleichbarem[122].	B1	Nein

6.3.4 Security

An Security Features bietet Linkerd eine Kommunikation über mTLS als default der HTTP Kommunikation zwischen den Pods an. Linkerds Control Plane stellt hierfür Zertifikate an Proxies aus, die dann an den Kubernetes Service Account gehen und im 24 Stunden Takt rotieren[115]. Die Credentials können sowohl von Linkerd selbst, als auch von einem externen Anbieter erzeugt werden (siehe Tabelle 11).

Tabelle 11: Linkerd Bewertung Security

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K20	Per Default nutzt Linkerd automatisch mTLS für die Kommunikation zwischen den Proxies[115].	B1	Ja
K22	Seit Linkerd 2.7 gibt es Unterstützung für externe PKI-Anbieter[138].	B1	Ja
K23	Bisher sind noch keine Netzwerk Policies in Linkerd möglich, die als Autorisierungsregeln genutzt werden können. Hierzu befindet sich ein offenes Ticket auf GitHub [10].	B1	Nein

6.3.5 Usability

Was Linkerd auszeichnet, ist seine Einfachheit und schnelle Installation, wie in Tabelle 12 und Tabelle 13 analysiert. Bekannte Probleme bei der Installation lassen sich in der Dokumentation finden und somit schnell beheben. Diese können zum Beispiel mit einem falsch konfigurierten Kubernetes Cluster zusammenhängen. Linkerd läuft nur auf Kubernetes und das Dashboard gleicht dem integrierten Dashboard bei Kubernetes stark. Es ist sichtbar auf die Metriken ausgerichtet, welche den Großteil der Oberfläche füllen. Weitere Konfigurationen lassen sich nur bedingt über das Dashboard nachvollziehen und sich nicht ändern. Linkerd funktioniert ohne das Projekt anpassen zu müssen und ist in seiner Handhabung nutzerfreundlich.

Tabelle 12: Linkerd Bewertung Usability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K23	Das Service Mesh war fast ohne Probleme installierbar. Die Dokumentation umfasst alle Handlungsschritte und bietet für häufig auftretende Fehler Abhilfe. Es mussten keine zusätzlichen Schritte recherchiert werden.	B4	9
K24	CLI installieren, Linkerd der Umgebungsvariable Path hinzufügen, Kubernetes Cluster validieren, Linkerd deployen, Anwendung deployen, Proxies injizieren[123].	B3	6
K25	Für die Installation muss keine Datei der Anwendung angepasst werden[123].	B1	Nein
K26	Die Dokumentation bezüglich der Fehlerbehebung bezieht sich in erster Linie auf die Probleme, die bei den Checks während der Installation auftreten[132]. Darüber hinaus gibt es jedoch keine Hilfe zum De-buggen von Linkerd oder eine Sammlung häufig auftretender Fehler.	B4	7
K27	Es wird im Namespace der Anwendung ein einfacher Graph der Services ohne Kantenbeschreibung erzeugt. In der Übersicht des jeweiligen Services wird ein Graph mit den Kanten, die zu dem Service und von ihm weg führen erzeugt. Die Knoten haben als Attribute die Golden Signal Metriken.	B1	Ja

Tabelle 13: Linkerd Bewertung Usability 2

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K28	Das Dashboard ist in erster Linie auf das Monitoring der Metriken, die Linkerd mitliefert, ausgerichtet. Diesbezüglich ist das Dash-board intuitiv aufgebaut, da diese sofort sichtbar sind, sobald man den Namespace, in dem die Anwendung deployed ist, auswählt. Weitere Details erhält man, wenn man auf einzelne Deployments klickt. Hierzu gehört zum Beispiel auch der Echtzeit Datenverkehr. Auch die Übersicht der Traffic Splits ist mit Angabe von der Route und den Gewichten intuitiv aufgebaut. Jedoch werden Konfigurationen darüber hinaus nicht im Dashboard angezeigt. Konfiguration von Retry oder Timeout sind über die Service Profile, die als Custom Kubernetes Resource (CRD) angelegt werden[117][118], möglich und können nur über die Console überwacht werden. Jedoch ist das Dash-board aufgrund der Einfachheit übersichtlich; es lässt sich leicht navigieren.	B2	Hoch
K29	Linkerd läuft nur auf Kubernetes. Es ist also verhältnismäßig unflexibel[146, S. 33].	B4	2
K30	Man kann als Attribut im Namespace einstellen, dass Proxies automatisch injiziert werden[116].	B1	Ja
K31	Es werden die Protokolle gRPC, HTTP, HTTP/2, TCP (inklusive TLS, WebSockets und HTTP Tunneling) unterstützt. [130].	B3	4
K32	Mit dem Linux top Befehl berechnet, abzüglich des Verbrauchs vor der Installation (siehe Abschnitt A.10 und Abschnitt A.12) in KiB Mem used.	B3	2.143.168
K33	Über das Dashboard lässt sich keine der Konfigurationen anlegen oder ändern.	B2	Nein

6.3.6 Extensibility

Linkerd ist darauf ausgerichtet schnell und sofort zu funktionieren. Abgesehen von den Anpassungsmöglichkeiten, die Linkerd selbst bietet, ist das Service Mesh nicht durch eigene Implementierungen erweiterbar (siehe Tabelle 14). Jedoch wird das Service Mesh Interface unterstützt und der Service Mesh Hub unterstützt die Anbindung Linkerds. Multicluster sind möglich, jedoch erfordert dies eine manuelle Einstellung der Umgebung[125].

Tabelle 14: Linkerd Bewertung Extensibility

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K34	Die Linkerd Installation lässt sich mit Kustomize ändern und anpassen.[119]. Der genutzte Proxy ist eigens für Linkerd geschrieben und bietet eine Reihe von vorgegebenen Konfigurationsmöglichkeiten. Darüber hinaus ist dieser jedoch nicht änderbar[128].	B4	6
K35	Da Linkerd nur in Kombination mit einem Kubernetes Cluster läuft[146, S. 33], sind VMs oder Container, die sich nicht in einem Kubernetes Cluster befinden, nicht über das Service Mesh anzubinden.	B1	Nein
K36	Linkerd unterstützt Service Discovery über Clustergrenzen hinweg. Somit sind auch automatisches mTLS, Traffic Splitting und Observability über die Clustergrenzen hinweg möglich. Der Support ist jedoch noch in der Testphase[127].	B1	Ja
K37	Linkerd unterstützt das SMI seit Version 2.4[136].	B1	Ja
K38	Linkerd wird vom Service Mesh Hub unterstützt[151].	B1	Ja

7 Consul Connect

Consul wurde ursprünglich als Service Discovery Lösung veröffentlicht[146, S. 34] und ist mittlerweile ein Service Management Framework. Es wurde erstmals 2014 durch Hashicorp released[27]. Connect ist die Komponente, die die Service Mesh Funktionen bereitstellt. Sie ist seit der Version 1.2, die Mitte 2018 released wurde, ein fester Bestandteil von Consul[51]. Consul kann, muss jedoch nichts zwangsweise als Service Mesh betrieben werden[25]. Für die Analyse wird Consuls Service Mesh Funktionalität betrachtet und im weiteren Verlauf als Consul statt Consul Connect bezeichnet.

7.1 Architektur

Consul ist ein verteiltes System, das aus Consul Agents besteht. Ein Agent kann im Fall des Service Meshs ein Server oder ein Client sein[21]. Auf jedem Knoten, der Services für Consul bereitstellt, werden Client Agents ausgeführt[26]. Ein Client ist ein Agent, der mit einem oder mehreren Servern spricht und alle RPCs an diese weiterleitet[26]. Er ist zustandslos und Teil des LAN Gossip Pools, um mit anderen Clients kommunizieren zu können[22]. Ein Client ist für den Knoten und dessen Gesundheit zuständig[26].

Ein Server ist ein Agent, der für den Zustand des Clusters oder Datacenters verantwortlich ist. Er antwortet auf RPC Queries und tauscht Informationen mit anderen Datacenters aus[23]. Auf den Consul-Servern werden die Daten gespeichert und repliziert. Unter den Servern wird ein Leader ausgewählt. Es werden drei bis fünf Server pro Datacenter empfohlen[26].

Ein Datacenter ist eine private Netzwerkumgebung mit niedriger Latenz und hoher Bandbreite[24], in der ein Cluster aus Consul-Servern ausgeführt wird[26]. Die Control Plane von Consul besteht aus den Clients und den Servern. Die Data Plane besteht aus eigenen Sidecar Proxys, die jedoch durch Drittanbieter Proxys ausgetauscht werden können[25].

In Abbildung 6 ist Consuls Architektur zu sehen. Für das Fallbeispiel wurde Consul mit einem Server und einem Client pro Service Pod konfiguriert. Der Server läuft im gleichen Kubernetes Cluster in einem eigenen Pod.

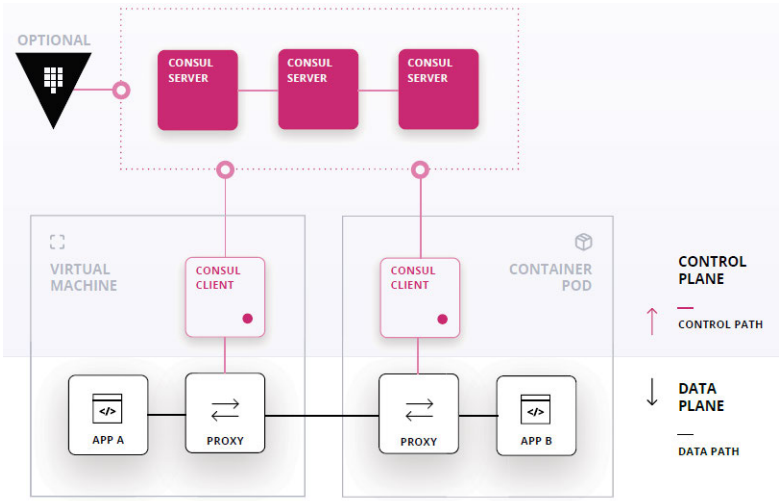

Abbildung 6: Consuls Architektur [18, S. 7]

7.2 Installation

Consul wurde auf Minikube mittels eines Helm Charts deployed[20]. Die im folgenden beschriebenen Befehle sind im Anhang zu finden (siehe Abschnitt A.6). Nachdem man Hashicorp dem Helm Repo hinzugefügt hat, werden mittels der neu angelegten Datei helm-consul-values.yaml (siehe Abschnitt A.7) die default Werte im helm Chart für die lokale Entwicklungsumgebung überschrieben. Consul wird nun mittels *helm install* auf Minikube installiert. Hiermit werden die Helm-Consul-Values auf das Chart angewendet und der Installation einen Namen gegeben. Es wurde für das Fallbeispiel Consul in der Version 1.7.2 installiert. Um das Consul CLI nutzen zu können, wird die Konsole mit dem Server über dessen Pod verknüpft.

Um nun die die Anwendung im Service Mesh zu deployen müssen sämtliche Deployment Dateien angepasst werden. Über die Values Datei wurde bereits der Connect Injektor aktiviert, der dafür sorgt, dass alle Services automatisch bei Consul registriert werden. In den Deployment YAML Dateien müssen nun im Deployment auf gleicher Höhe der Labels die Connect-Inject Annotationen eingefügt werden. Wenn nun die Services mit kubectl deployed werden, wird automatisch der Proxy injiziert.

7.3 Bewertung

Im Folgenden werden die Kriterien, die in Kapitel 3 genannt werden auf Consul angewandt. Hierfür wird für jeden Bereich eine Tabelle erstellt, in der neben der Analyse das Bewertungskriterium, Bewertungsmaßstab sowie das Ergebnis dargestellt werden. Consul hat ohne zusätzliche Konfiguration einen eingebetteten Layer 4 Proxy[38]. Dieser ist jedoch nur für Entwicklung und Testen von Connect sinnvoll, da er viele Features nicht unterstützt. Consul bietet eine sehr gute Unterstützung von Envoy. Hierfür sind zahlreiche Konfigurationsmöglichkeiten in Consul enthalten[37]. Die Analyse wird daher basierend auf Envoy als Proxy erstellt.

7.3.1 Routing

Consuls enthaltene Features werden mittels JSON Dateien konfiguriert. Hiermit kann der Datenverkehr sowohl nach Prozent, als auch Pfad oder nach Header geteilt werden (siehe Tabelle 15). Der integrierte Load Balancer bietet eine Basisfunktionalität mittels Round-Robin Algorithmus an. Dieser kann jedoch auch durch Load Balancer von Drittanbietern ausgetauscht werden. In der Dokumentation lassen sich hier Beispiele für Fabio, Nginx oder HAProxy finden[152].

Tabelle 15: Consul Bewertung Routing

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K1	Der eingebaute Load Balancer nutzt nur Round-Robin. Dieser kann aber zum Beispiel durch Fabio ersetzt werden[152].	B3	1
K2	Über JSON Konfigurationsdateien kann der Datenverkehr nach Prozent geteilt werden[28].	B1	Ja
K3	Über JSON Konfigurationsdateien kann der Datenverkehr basierend auf Pfad oder Header geteilt werden[29].	B1	Ja

7.3.2 Observability

Wie schon eingangs erwähnt, besitzt Consul einen integrierten Proxy, der jedoch kaum Features unterstützt. Mit Envoy als Proxy lassen sich diverse Metriken, wie alle vier Golden Signal Metriken oder TCP Verbindungsdaten sammeln (siehe Tabelle 16). Diese sind jedoch nur pro Service und nicht pro Endpunkt möglich. Prometheus und Grafana werden unterstützt, müssen jedoch separat installiert werden. Gleiches gilt auch für Tracing im Service Mesh. Ein gewünschtes Backend muss separat installiert werden. Mit Envoy als Proxy werden Access Logs erstellt, mit denen eingehende Anfragen überwacht werden können.

Tabelle 16: Consul Bewertung Observability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K4	Nur die Log Einträge der Agents lassen sich abrufen[30].	B1	Nein
K5	Ohne eine Proxy Erweiterung werden nur Latenz[32] und Auslastung[31] gemessen. Mit Envoy können auch Datenverkehr und Errors gemessen werden[37].	B3	4
K6	Es werden bisher keine Metriken pro HTTP Endpunkt angeboten.	B1	Nein
K7	Es werden nur für interne Consul Prozesse TCP Metriken gesammelt[33]. Mit Envoy können auch offene Verbindungen und der Durchsatz gemessen werden[19].	B1	Ja
K8	Prometheus muss extra installiert werden[34].	B1	Nein
K9	Grafana muss extra installiert werden[34].	B1	Nein
K10	Für Consul wird ein Dashboad zur Verfügung gestellt[35].	B1	Ja
K11	Datadog, Jaeger, Open Tracing, Honeycomb, Zipkin[133].	B3	5
K12	Es wird nicht automatisch ein Tracing Backend mitinstalliert[133].	B1	Nein
K13	Mit Envoy als Proxy werden Access Logs erstellt[36].	B1	Ja

7.3.3 Resilience

Mittels einer Service-Router Konfiguration werden alle Einstellungen, die Daten-Routing oder -Manipulation beinhalten, definiert[29]. Hierzu zählen zum Beispiel Circuit Breaking, Retry oder Timeout (siehe Tabelle 17). Diese können auch pro Pfad eingerichtet werden. Fault oder Delay Injection sind jedoch bisher nicht enthalten.

Tabelle 17: Consul Bewertung Resilience

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K14	Über die Konfiguration eines Envoy Clusters lässt sich Circuit Breaking einrichten[37].	B1	Ja
K15	Über die Service Router Konfiguration lässt sich Retry einrichten[29].	B1	Ja
K16	Über die Service Router Konfiguration lassen sich Timeouts einrichten[29].	B1	Ja
K17	Über die Service Router Konfiguration lassen sich Pfad basierende Resilience Patterns einrichten[29].	B1	Ja
K18	Fault Injection ist nicht Teil der Features.	B1	Nein
K19	Delay Injection ist nicht Teil der Features.	B1	Nein

7.3.4 Security

Der Security Aspekt ist in Consuls Dokumentation sehr ausführlich beschrieben. Durch die verschiedenen Kommunikationswege über RPCs oder das Gossip Protokoll, versucht Consul auch in einer verteilten Umgebung Vertraulichkeit, Integrität und Authentifizierung sicherzustellen. Hashicorps Vault ist ein Tool um Secrets sicher managen zu können und kann in das Service Mesh integriert werden. Consul besitzt jedoch auch eine integrierte Certificate Authority. Drittanbieter werden zum jetzigen Zeitpunkt noch nicht unterstützt (siehe Tabelle 18). Intentions sind Regeln, die per Befehl oder auch das Dashboard angelegt werden können und die Zugriffseinschränkungen zwischen den Services definieren.

Tabelle 18: Consul Bewertung Security

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K20	Die Verbindungen über Connect basieren alle auf mTLS[39].	B1	Ja
K22	Bisher ist es möglich die integrierte CA oder Vault zu nutzen. Es ist jedoch geplant auch externe CAs einbinden zu können[40].	B1	Ja
K23	Über das Anlegen von sogenannten Intentions lassen sich Autorisierungsregeln erstellen[41].	B1	Ja

7.3.5 Usability

Consuls Dokumentation umfasst neben einer detaillierten Beschreibung der Anwendung auch diverse Tutorials und Beispiele. Bekannte Fehler und häufige Fragen sind in der Dokumentation enthalten und erleichtern einem den Einstieg. Damit ein Proxy injiziert werden kann, müssen jedoch die Deployment Dateien des Projekts angepasst werden. Consuls Dashboard ist in erster Linie auf ein Monitoring des Gesundheitszustandes der Services und deren Proxys ausgelegt. Konfigurationen, abgesehen von Intentions und Splitter, werden weder angezeigt, noch sind die über das Dashboard änderbar. Der Aufbau des Dashboards ist auch nicht sofort ersichtlich und verliert hiermit an Nutzerfreundlichkeit. Es wird auch kein Graph der Services erzeugt. Eine hervorzuhebende Stärke von Consul ist die Flexibilität der Plattform, auf der es läuft (siehe Tabelle 19 und Tabelle 20).

Tabelle 19: Consul Bewertung Usability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K23	Über die HashiCorp Learn Website sind diverse Tutorials verfügbar. Hiermit ist eine Installation problemlos durchführbar, wenn man ein entsprechendes Beispiel hat[20].	B4	7
K24	Hashicorp dem Helm Repo hinzufügen, Helm Consul Values anlegen, helm install, Service Deployment Files anpassen, Services deployen[20].	B3	5
K25	Damit die Proxys injiziert werden können, muss eine entsprechende Anpassung in den Deployment Dateien ergänzt werden[20].	B1	Ja
K26	Die Dokumentation enthält die häufigsten Fehlermeldungen[42] und Fragen[43].	B4	5
K27	Es wird kein Graph erzeugt.	B1	Nein

Tabelle 20: Consul Bewertung Usability 2

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K28	Consuls Dashboard ist sehr einfach gehalten und konzentriert sich auf den Status der Services und der erstellten Intentions. Es werden keine Metriken angezeigt. Zur Konfiguration lassen sich die Intentions schnell finden, die Splitter über den Service weniger schnell. Durch die Einfachheit des Dashboards ist es jedoch sehr übersichtlich.	B2	Mittel
K29	Istio läuft auf Nomad[45], Kubernetes, VMs und Bare Metal[44].	B4	10
K30	Über die Helm Consul Values und Deployment Dateien kann das automatische Injizieren konfiguriert werden[20].	B1	Ja
K31	Es werden die Protokolle grpc, http, https, tcp und udp unterstützt[46].	B3	5
K32	Mit dem Linux top Befehl berechnet, abzüglich des Verbrauchs vor der Installation (siehe Abschnitt A.10 und Abschnitt A.13) in KiB Mem used	B3	1.854.832
K33	Über den Menüpunkt Intentions lassen sich diese erstellen und löschen. Weitere Konfigurationen lassen sich nicht ändern.	B2	Ja

7.3.6 Extensibility

Consul ist vielseitig erweiterbar und modular austauschbar. Bereits erwähnt wurden der austauschbare Proxy und der Load Balancer. Consul bietet auch das Consul Integration Program, in dem Integrationen von externen Anbietern verifiziert werden können (siehe Tabelle 21). Consul wird zwar nicht vom Service Mesh Hub unterstützt, ist aber mit dem Service Mesh Interface kompatibel. Über mehrere Datacenter können Multicluster erstellt werden, die auch externe Service Meshes anbinden können und auf diversen unterschiedlichen Plattformen laufen können[50].

Tabelle 21: Consul Bewertung Extensibility

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K34	Consul bietet diverse Integrationen. Über ein Partnerprogramm können auch eigene Integrationen in den Pool mitaufgenommen werden. Diese sind jedoch nicht auf Control Plane Ebene verfügbar[47].	B4	8
K35	Virtuelle Maschinen können angebunden werden, indem der Proxy sich bei Consul registriert[48].	B1	Ja
K36	Über mehrere Datacenters, die miteinander kommunizieren, kann ein Multicluster erstellt werden[49].	B1	Ja
K37	Consul wird im Ecosystem der Service Mesh Interface Website gelistet[52].	B1	Ja
K38	Consul wird noch nicht vom Service Mesh Hub unterstützt[151], bietet jedoch ein eigenes Multi-Plattform Management an[50].	B1	Nein

8 Kuma

Die Entwickler von Kong waren mit dem Problem konfrontiert, dass sich die damals bestehenden Service Meshes auf Kubernetes spezialisiert hatten und in großen Teilen schwer zu installieren und zu konfigurieren waren[141]. Mit dem Anspruch schnell und einfach ein sofort funktionierendes Service Mesh, das auf allen Plattformen läuft, bereit zu stellen, wurde Kuma im September 2019 released. Kong, welches urspünglich für ihr API Gateway bekannt ist, bezeichnet Kuma als das universelle Service Mesh[140]. Mit dem Release 0.6.0 im Juni 2020 wurde Kuma als Sandbox Projekt an die Cloud Native Computing Foundation (CNCF) übergeben[143].

8.1 Architektur

Kuma besteht aus der Control Plane Kuma-CP (siehe Abbildung 7), die verschiedene Entities umfasst. Es gibt die Möglichkeit mehrere Mesh Entitäten zu erstellen und somit zentralisiert mehrere Meshes zu managen. Damit die Dataplanes (Kuma-DP) sich mit der Control Plane verbinden können, muss mindestens ein Mesh auf der Control Plane laufen. Per default wird ein Mesh mit dem Namen default angelegt, wenn die Control Plane das erste Mal läuft. Des Weiteren muss eine Data Plane Entität im Mesh vorhanden sein, mit der sich dann eine Kuma-DP verbinden kann. Jede Kuma-DP bündelt einen Envoy Proxy. Über die Dataplane Entities wird kommuniziert, welche Kuma-DP für welchen Service zuständig ist. Auf Kubernetes erfolgt das Injizieren der Dataplanes automatisch. Bei einer abweichenden Umgebung muss der Aufbau manuell konfiguriert werden[95]. Die Konfiguration des Service Meshs wird zur Laufzeit von der Control Plane erzeugt und den Proxies mitgeteilt[111].

Abbildung 7: Kuma Architektur [95]

8.2 Installation

Kuma ist nicht kompatibel mit Kubernetes Versionen höher als v1.17.4[9]. Zudem speichert Kuma den Zustand auf dem darunter liegenden Kubernetes API Server[11]. Hierfür muss genug Speicher konfiguriert werden. Damit Kuma läuft, muss Minikube mit weiteren entsprechenden Parametern gestartet werden (siehe Abschnitt A.8).

Um Kuma zu installieren, muss die entsprechende Version heruntergeladen werden[102]. Im bin Verzeichnis befindet sich kumactl. Für das Fallbeispiel wurde Kuma in der Version 0.4.0 geladen. Mit kumactl install (siehe Abschnitt A.8) wird die Control Plane deployed. Zeitgleich wird auch der Namespace kuma-system angelegt. Nun können Services deployed werden.

Um die Proxies automatisch zu injizieren muss der Namespace des Fallbeispiels entsprechend gelabelt werden. Zusätzlich muss in jeder Deployment Datei des Fallbeispiels angegeben werden zu welchem Mesh der Service gehört (siehe Abschnitt A.8).

Wenn mTLS aktiviert ist, wird per default der Datenverkehr unterbunden. Um diesen zu erlauben muss eine Traffic Permission Policy angelegt werden (siehe Abschnitt A.9).

[9] Inkompatibel zum Zeitpunkt der Installation. Dies wurde zu einem späteren Zeitpunkt behoben[93]

8.3 Bewertung

Im Folgenden werden die Kriterien, die in Kapitel 3 genannt werden auf Kuma angewandt. Hierfür wird für jeden Bereich eine Tabelle erstellt, in der neben der Analyse das Bewertungskriterium, Bewertungsmaßstab sowie das Ergebnis dargestellt werden. Die Analyse bezieht sich auf Kuma in der Version 0.4.0. Diese kann deutlich von der letzten veröffentlichten Version 0.7.2 abweichen, die am 6. Oktober veröffentlicht wurde[144].

8.3.1 Routing

Zum Zeitpunkt der Installation umfasste die Dokumentation Kumas noch nicht alle Informationen, die für die Analyse relevant gewesen wären. So steht dort zum Zeitpunkt der Version 0.4.0 noch keine Angabe zum Load Balancer. Zu einem späteren Zeitpunkt wurde dann ergänzt, dass der genutzte Algorithmus Round-Robin ist. Dies fließt jedoch aufgrund der abweichenden Version nicht in die Analyse ein. Über TrafficRoute Policies kann der Datenverkehr basierend auf getaggten Dataplane Ressourcen nach Prozent geteilt werden (siehe Tabelle 22). Um den Datenverkehr basierend auf Header oder Pfad zu teilen, sind die Policies noch nicht geeignet.

Tabelle 22: Kuma Bewertung Routing

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K1	Keine Angabe in der Dokumentation.	B3	-
K2	Über TrafficRoute Policies kann der Datenverkehr nach Prozent aufgeteilt werden[109].	B1	Ja
K3	Die TrafficRoute Policies basieren auf Tags der Dataplane, somit ist kein Datenverkehr nach Header oder Pfad möglich[109].	B1	Nein

8.3.2 Observability

Bei den Observability Features zeigt sich, dass die installierte Kuma Version noch einiges an Zuarbeit benötigt. Metriken können zwar durch die Envoy Proxies erzeugt werden, jedoch werden diese weder gesammelt noch visuell dargestellt. Gleiches gilt für die Access Logs, die ohne ein Logging Backend, das zustätzlich installiert werden muss, nicht gesammelt werden können (siehe Tabelle 23). Prometheus und Grafana werden unterstützt, müssen jedoch auch separat installiert werden. Als Tracing Backends werden Zipkin und Jaeger unterstützt.

8.3.3 Resilience

Die Analyse der Resilience Features in Tabelle 24 zeigt, dass in der Version 0.4.0 noch keine nutzerfreundlichen Möglichkeiten enthalten sind. Retry und Timeout sind nur möglich, wenn ein eigenes Proxy Template für Envoy geschrieben wird und der Proxy so eigenhändig konfiguriert wird[105]. Circuit Breaking, Fault Injection und Delay Injection sind in dieser Version noch nicht enthalten. Jedoch zeigt der Changelog von Kuma, dass in den letzten Versionen einige der fehlenden Punkte implementiert wurden. Seit Version 0.5.1 sind Circuit Breaking, Fault Injection und Delay Injection über Policies oder die HTTP API von Kuma einstellbar[94][96].

8.3.4 Security

Über die Mesh Policies kann mTLS aktiviert werden. Wie im Abschnitt der Installation erwähnt, müssen dann einzelne oder alle Routen über Traffic Permission Policies freigegeben werden. Hiermit können zudem Autorisierungsregeln aufgestellt werden. Kuma bietet eine eingebaute CA. Diese kann über die Mesh Ressource geändert werden[91].

Tabelle 23: Kuma Bewertung Observability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K4	Für die Log Einträge muss entweder ein Backendservice zum Sammeln der Logeinträge oder der direkte Zugriff in den Container genutzt werden. Es ließ sich keine Möglichkeit finden die Log Einträge der Services über das Dashboard oder kumactl Befehle einzusehen.	B1	Nein
K5	Es lassen sich Datenverkehr, Errors und die Auslastung des Systems überwachen. Nach Bedarf können weitere Metriken konfiguriert werden[107].	B3	3
K6	Mit Kuma lassen sich Metriken auf Dataplane-, Mesh- und Service-Ebene sammeln[107].	B1	Nein
K7	Es können TCP Metriken gesammelt werden[101].	B1	Ja
K8	Um die Metriken abrufen zu können muss Prometheus installiert werden. Es ist nicht von Anfang an enthalten[107].	B1	Nein
K9	Kuma hat voreingestellte Grafana Dashboards. Grafana ist nicht von Anfang an enthalten[107].	B1	Nein
K10	Es ist eine GUI zur grundlegenden Übersicht enthalten[98].	B1	Ja
K11	Es werden Zipkin und Jaeger unterstützt[142].	B3	2
K12	Tracing Backends müssen integriert werden[142].	B1	Nein
K13	Kumas Dataplanes generieren Access Logs. Hierfür muss jedoch ein Loggingbackend integriert werden[106].	B1	Nein

Tabelle 24: Kuma Bewertung Resilience

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K14	In Version 0.4.0 ist noch kein Circuit breaking enthalten[92, 0.5.1], dafür ein passiver Health Check[99].	B1	Nein
K15	Retry ist nicht ohne ein Überschreiben der Standard Einstellungen des Proxys möglich[105].	B1	Nein
K16	Timeout ist nicht ohne ein Überschreiben der Standard Einstellungen des Proxys möglich[105].	B1	Nein
K17	Resilience Pattern sind ohne ein Überschreiben der Standard Einstellungen nicht pro Endpunkt möglich[105].	B1	Nein
K18	Fault Injection ist in der Version noch nicht enthalten.[92, 0.5.0].	B1	Nein
K19	Delay Injection ist in der Version noch nicht enthalten.[92, 0.5.0][97].	B1	Nein

Tabelle 25: Kuma Bewertung Security

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K20	mTLS kann über die Mesh Policies aktiviert werden[104].	B1	Ja
K22	Es können externe CAs konfiguriert werden[91].	B1	Ja
K23	Über Traffic Permissions können Autorisierungsregeln erstellt werden[108].	B1	Ja

8.3.5 Usability

Kuma ist eines der jüngsten Service Meshes und zum Zeitpunkt der Installation war das deutlich spürbar. Die Dokumentation umfasst nur die wesentlichen Punkte. Troubleshooting ist nicht enthalten und die Bereiche, die bisher noch nicht weiterentwickelt wurden, zeigen deutliche Defizite. Nach der Installation wurden fast monatlich neue Versionen veröffentlicht, die viele kritisierte Punkte verbessern. In der installierten Version ist das Dashboard in erster Linie Monitoring des Gesundheitszustandes der Services und der Proxys und eine Liste erzeugter Policies. Positiv aufgefallen ist die einfach gehaltene Installation und die Flexibilität der Plattform (siehe Tabelle 26 und Tabelle 27).

Tabelle 26: Kuma Bewertung Usability

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K23	Zum Zeitpunkt der Installation war das erste Release von Kuma ein halbes Jahr her. Die Dokumentation war noch sehr grundlegend und Hilfestellungen nicht vorhanden. Inkompatibilität mit der damals aktuellen Kubernetesversion und eine geeignete Memory Konfiguration waren nicht ersichtlich, was den Installationsprozess erschwert hat.	B4	4
K24	Kuma herunterladen, Kuma installieren, Deployment Dateien anpassen, Namespace labeln, wenn mTLS aktiviert ist Traffic Policies anlegen, Services deployen[102].	B3	5
K25	Für die Installation müssen die Deployment Dateien angepasst werden[102].	B1	Ja
K26	Es gibt keine Dokumentation zur Fehlerbehebung.	B4	0
K27	Es wird kein Graph erzeugt.	B1	Nein

Tabelle 27: Kuma Bewertung Usability 2

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K28	Es werden in Version 0.4.0 noch keine Metriken im Dashboard angezeigt[98]. Angelegte Konfigurationen können schnell über die Policy Menüpunkte eingesehen werden. Durch die Einfachheit ist die Oberfläche sehr übersichtlich.	B2	Hoch
K29	Kuma läuft auf Kubernetes, VMs und Bare Metal [110].	B4	9
K30	Man kann als Attribut im Namespace einstellen, dass Proxies automatisch injiziert werden[102].	B1	Ja
K31	Es werden die Protokolle HTTP und TCP unterstützt [101].	B3	4
K32	Mit dem Linux top Befehl berechnet, abzüglich des Verbrauchs vor der Installation (siehe Abschnitt A.10 und Abschnitt A.14) in KiB Mem used.	B3	1.593.704
K33	Über das Dashboard lässt sich keine der Konfigurationen anlegen oder ändern.	B2	Nein

8.3.6 Extensibility

Kuma stellt eine HTTP API bereit, über die die Entities angepasst werden können. Externe Erweiterungsmöglichkeiten sind bisher noch nicht verfügbar (siehe Tabelle 28). Seit den ersten Versionen ist jedoch die Möglichkeit mehrere Meshes einfach zu erstellen und zu managen in Kuma enthalten.

Tabelle 28: Kuma Bewertung Extensibility

Bewertungs-kriterium	Analyse	Bewertungs-maßstab	Ergebnis
K34	Die verschiedenen Entities können über die HTTP API angepasst werden. Wenn Kuma auf Kubernetes läuft, kann der Zustand über CRDs angepasst werden[100].	B4	3
K35	Eine Erweiterung durch VMs außerhalb eines Kuberntes Clusters ist aus der Dokumentation nicht ersichtlich.	B1	Nein
K36	Es können mehrere Meshes pro Kuma Cluster erstellt werden[103].	B1	Ja
K37	Kuma unterstützt das SMI nicht[52].	B1	Nein
K38	Kuma wird nicht vom Service Mesh Hub unterstützt[151].	B1	Nein

9 Vergleich

Im Folgenden werden die verschiedenen Service Meshes anhand ihrer Bewertung miteinander verglichen. Hierfür werden die Ergebnisse in einer Tabelle pro Feature Bereich nebeneinander aufgestellt.

9.1 Routing

Wenn man die Ergebnisse in Tabelle 29 miteinander vergleicht, sticht klar Istio hervor. Es sind verschiedene Load Balancing Algorithmen einstellbar und der Datenverkehr sowohl nach Prozent, als auch nach Header und Pfad teilbar. Kuma schneidet anhand der fehlenden Dokumentation am schlechtesten ab. Wenn man einen austauschbaren Load Balancer bevorzugt, bietet jedoch nur Consul diese Möglichkeit an. Linkerd bietet einfache, aber begrenzte Routing Features.

Tabelle 29: Vergleich Routing

Bewertungskriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K1 (LB)	B3	4	1	1	-
K2 (nach Prozent)	B1	Ja	Ja	Ja	Ja
K3 (nach Header und Pfad)	B1	Ja	Nein	Ja	Nein

9.2 Observability

Observability Features, basierend auf Logs, Metriken und Tracing, sind in jedem Service Mesh vorhanden. In der Gesamtheit der Kriterien in Tabelle 30 betrachtet schneiden Istio und Linkerd gleichauf am besten ab. Wenn man die Service Meshes basierend auf Logs, Metriken und Tracing miteinander vergleicht, bietet jedoch Istio bezüglich der Logs eine bessere Unterstützung, da sowohl die Service Logs einsehbar sind, als auch Access Logs generiert werden.

Wenn man die Kriterien zu Metriken betrachtet, bietet allein Linkerd die Möglichkeit diese auch pro Endpunkt zu sammeln. Linkerd ist auch führend, was die möglichen Tracing Backends angeht. Von den Golden Metrics enthält jedoch allein Consul alle vier. Prometheus und Grafana werden zwar von allen Service Meshes unterstützt, jedoch ist es nur in Istio und Linkerd direkt mitinstallierbar. Bei Kuma zeigt sich wiederum der junge Entwicklungsstand. Keine Möglichkeit zu haben direkt aus Kuma heraus die Metriken abrufen zu können, führt zu einem höheren Aufwand der Implementierung.

9.3 Resilience

An Resilience Features deckt Istio fast alle Kriterien ab. Jedoch sind diese nicht pro Endpunkt konfigurierbar. Dies ist nur bei Linkerd und Consul möglich. Eine aus dem Service Mesh automatisch konfigurierbare Fault oder Delay Injection ist zu diesem Zeitpunkt nur bei Istio möglich. In einer neueren Version von Kuma wurden die Resilience Features weiterentwickelt und dort sind auch Fault und Delay Injection möglich. Für Linkerd und Consul müssen diese manuell eingerichtet werden. Trotz Linkerds langem Bestehen fällt auf, dass nach wie vor kein Circuit Breaking enthalten ist (siehe Tabelle 31).

9.4 Security

Wenn man die Ergebnisse in Tabelle 32 der Security Kriterien vergleicht, fällt sofort auf, dass bis auf Linkerd alle Service Meshes die Kriterien erfüllen. Nur in Linkerd ist es nicht möglich Autorisierungsregeln zu konfigurieren. Abgesehen davon ist in allen Meshes mTLS vorhanden und auch eine externe CA ergänzbar.

Tabelle 30: Vergleich Observability

Bewertungs-kriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K4 (Service Log)	B1	Ja	Nein	Nein	Nein
K5 (Metriken)	B3	2	3	4	3
K6 (Metriken pro Endpunkt)	B1	Nein	Ja	Nein	Nein
K7 (TCP Metriken)	B1	Ja	Ja	Ja	Ja
K8 (Prometheus)	B1	Ja	Ja	Nein	Nein
K9 (Grafana)	B1	Ja	Ja	Nein	Nein
K10 (Dashboard)	B1	Ja	Ja	Ja	Ja
K11 (Tracing)	B3	5	9	5	2
K12 (inkl. Tracing Backend)	B1	Nein	Nein	Nein	Nein
K13 (Access Log)	B1	Ja	Nein	Ja	Nein

Tabelle 31: Vergleich Resilience

Bewertungs-kriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K14 (Circuit Breaking)	B1	Ja	Nein	Ja	Nein
K15 (Retry)	B1	Ja	Ja	Ja	Nein
K16 (Timeout)	B1	Ja	Ja	Ja	Nein
K17 (pro Endpunkt)	B1	Nein	Ja	Ja	Nein
K18 (Fault Injection)	B1	Ja	Nein	Nein	Nein
K19 (Delay Injection)	B1	Ja	Nein	Nein	Nein

Tabelle 32: Vergleich Security

Bewertungs-kriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K20 (mTLS)	B1	Ja	Ja	Ja	Ja
K22 (Externe CA)	B1	Ja	Ja	Ja	Ja
K23 (Autorisierungs-regeln)	B1	Ja	Nein	Ja	Ja

9.5 Usability

Die Usability Analyse umfasst die meisten Kriterien. Basierend auf den Kriterien in Tabelle 33 liegt Istio vor den anderen drei Service Meshes. Sowohl Istio, als auch Linkerd bieten in ihrer Dokumentation einen sehr guten Einstieg in die Implementierung. Kuma schneidet aufgrund der teilweise fehlenden Informationen hier weniger gut ab. Besonders hilfreich bei der Implementierung waren bekannte Fehler, die in der Dokumentation aufgeführt wurden. Hier ist die Dokumentation von Istio sehr umfassend, wo hingegen Kuma keine besitzt.

Für keines der Service Mesh musste der Programmcode geändert werden. Jedoch müssen für Consul und Kuma Annotationen in den Deploymentdateien ergänzt werden: für Consul, um die Proxies zu injizieren, für Kuma, um den Service einem Mesh zuzuweisen. Istio und Linkerd erstellen beide automatisch einen Graphen, wobei der von Linkerd sehr einfach gehalten und der von Istio sehr detailliert ist.

Die Intuitivität des Dashboards wurde anhand von drei Faktoren bewertet: Metriken zu finden, Konfigurationen zu finden und wie Übersichtlich die Oberfläche gestaltet ist. Die Intuitivität von Kumas Dashboard wurde mit Hoch bewertet, ebenso wie das von Istio und Linkerd. Im Gegensatz zu Linkerd und Kuma ist die Konfiguration des Service Mesh bei Istio jedoch auch über das Dashboard möglich, was deutlich mehr Funktionalität bietet. Consuls und Kumas Dashboard ist in erster Linie auf die Gesundheit des Meshs und der Konfigurationen ausgelegt. Diese sind in Consul auch über das Dashboard anpassbar. Linkerds Schwerpunkt liegt auf den Metriken, die sowohl im Dashboard als auch über eine Verlinkung in Grafana angezeigt werden.

Zur Bedienbarkeit gehört auch die Evaluierung der Implementierung der Grundfunktionalität. Hier schneiden Consul und Kuma am besten ab, die jeweils in 5 Schritten die Services im Service Mesh lauffähig haben. Weiterhin sind Consul und Kuma am flexibelsten was die Plattform, auf der sie laufen, angeht. Während Linkerd nur auf Kubernetes läuft und Istio zwar laut Dokumentation auch auf anderen Plattformen laufen kann, aber auf Kubernetes spezialisiert ist und hierfür den besten Support bietet, können Consul und Kuma auf so gut wie jeder Plattform laufen.

Ein weiterer Punkt, der untersucht wurde, ist der Verbrauch der Ressourcen, die das Service Mesh benötigt. Hierfür wurde in der Ubuntu Umgebung über den top Befehl der Speicherverbrauch und die Belastung der CPU abgefragt. Auszüge aus den Ergebnissen befinden sich im Anhang ab Abschnitt A.10. Der Speicherverbauch wurde mit in den

Vergleich aufgenommen. Hier zeigt sich, dass Kuma am wenigsten benötigt. Dies ist jedoch auch auf die fehlende Funktionalität zurückzuführen. Da es bei der CPU keine eindeutig vergleichbaren Ergebnisse gab, wurde dieser Punkt nicht mit in den Vergleich aufgenommen.

Tabelle 33: Vergleich Usability

Bewertungs-kriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K23 (Dokumen-tation)	B4	9	9	7	4
K24 (Installations-schritte)	B3	7	6	5	5
K25 (Projekt-anpassung)	B1	Nein	Nein	Ja	Ja
K26 (Trouble-shooting)	B4	10	7	5	0
K27 (Graph)	B1	Ja	Ja	Nein	Nein
K28 (Dashboard intuitiv)	B2	Hoch	Hoch	Mittel	Hoch
K29 (Plattform Flexibili-tät)	B4	8	2	10	9
K30 (Auto Injektion)	B1	Ja	Ja	Ja	Ja
K31 (Protokolle)	B3	11	4	5	4
K32 (Speicher-verbrauch)	B3	2.334.268	2.143.168	1.854.832	1.593.704
K33 (Konfigu-ration über Dashboard)	B2	Ja	Nein	Ja	Nein

9.6 Extensibility

Auch bei dem Vergleich der Kriterien zur Extensibility in Tabelle 34 liegt Istio vor den anderen Service Meshes. Jedoch ist Istio dicht gefolgt von Consul, das lediglich nicht vom Service Mesh Hub unterstützt wird. Da Consul die Funktionalität des Hubs selbst bereitstellt, ist dies jedoch auch nicht relevant. Sowohl Consul, als auch Istio bieten jeweils gute Unterstützung das Service Mesh selbst zu erweitern. Linkerd bietet hier zwar eine umfassende API für Konfigurationen ihrer Proxys beziehungsweise Konfigurierbarkeit über Kustomize und Kuma auch die Möglichkeit die Envoy Proxies zu konfigurieren, jedoch wird darüber hinaus keine Erweiterungsmöglichkeit angeboten.

Am Interessantesten gestaltet sich der Vergleich der Multicluster Funktionalität der Service Meshes. Diese ist bei allen vorhanden, jedoch sehr unterschiedlich implementiert. In Istio und Linkerd muss ein Multicluster aufwändig über diverse manuelle Einstellungen angelegt werden. In Consul kann dies über je eine Konfigurationsdatei pro Cluster ermöglicht werden. Hierbei ist die Plattform, auf der das Cluster läuft, frei auswählbar. Kuma ist seit Beginn darauf ausgelegt gewesen mehrere Meshes zentral managen zu können. Hierfür müssen lediglich die verschiedenen Endpunkte dem Cluster über die API mitgeteilt werden.

Tabelle 34: Vergleich Extensibility

Bewertungskriterium	Maßstab	Istio	Linkerd	Consul	Kuma
K34 (Anpassung und Erweiterung)	B4	8	6	8	3
K35 (VM außerhalb Cluster)	B1	Ja	Nein	Ja	Nein
K36 (Multicluster)	B1	Ja	Ja	Ja	Ja
K37 (SMI)	B1	Ja	Ja	Ja	Nein
K38 (Service Mesh Hub)	B1	Ja	Ja	Nein	Nein

10 Fazit

Das Ziel dieser Arbeit war es die verschiedenen Service Mesh Implementierung zu vergleichen und anhand eines Fallbeispiels zu evaluieren. Das Fallbeispiel diente in erster Linie für eine Analyse der Usability des jeweiligen Service Meshs. Die Verfügbarkeit verschiedener Features wurde primär über die Dokumentationen evaluiert.

Der im vorangegangenen Kapitel durchgeführte Vergleich der Bewertungen zeigt, dass basierend auf den Bewertungen Istio klar vor und Kuma durch seine noch nicht so weit vorangeschrittene Entwicklung weit hinter den anderen Service Meshes liegt. Jedoch wird im Laufe der Arbeit klar, dass nicht allein auf Grund einer Bewertung der Kriterien ein Service Mesh einem anderen gegenüber als besser angesehen werden kann.

Die Analyse der verschiedenen Kriterien veranschaulicht, dass jedes Service Mesh seine eigenen Vorzüge und Nachteile bietet und je nach Anwendungsfall Istio nicht immer die erste Wahl sein sollte. Istio bietet eine Fülle an Erweiterungsmöglichkeiten und beinhaltet fast alles, was anhand der Kriterien evaluiert wurde. Jedoch hat die große Anzahl an Konfigurationsmöglichkeiten auch eine hohe Komplexität zur Folge. Zumal die Art der Konfiguration häufig fast schon manuell geschieht. Bei der Installation fällt auf, dass Istio die meisten Installationsschritte beinhaltet. Istio zu betreiben setzt eine lange und intensive Auseinandersetzung mit der Dokumentation voraus. Auch wenn diese sehr umfassend ist, benötigt es viel Aufwand Istio in seiner Fülle an Möglichkeiten nutzen zu können.

Linkerd hat vor allem bei den Observability Kriterien seine Stärke gezeigt. Ohne weitere Konfigurationen sind Metriken, Prometheus und Grafana im Mesh enthalten. Wer schnell ein simples Service Mesh, bevorzugt muss sich nicht lange in der Dokumentation einlesen. Linkerd konzentriert sich auf einige wenige grundlegende Features. Jedoch ist es in seiner Einfachheit auch sehr begrenzt. Es ist kaum erweiterbar und benötigt Kubernetes als Plattform. Bei der Implementierung des Fallbeispiels war es jedoch das einzige Service

Mesh, das im Verhältnis zu den anderen Meshes sofort und ohne Probleme funktioniert hat.

Consuls Stärke ist seine Flexibilität. Hierzu zählt die Austauschbarkeit vieler Komponenten. Quasi jeder gewünschte Proxy kann genutzt werden. Weiterhin kann auch der Load Balancer ausgetauscht werden. Consul läuft auf allen Plattformen und kann diese einfach verbinden. Somit kann sich ein Mesh von Bare Metal bis in jede Cloud erstrecken. Jedoch müssen alle Ergänzungen extra installiert werden. Consul bietet viele der Features, die analysiert wurden, schneidet bei der Bewertung dieser jedoch vor allem in den Bereichen Usability und Observability weniger gut als die vorangegangenen ab. Auch wenn die Installation die wenigsten Schritte benötigt, ist das Dashboard und auch die Dokumentation nicht sehr intuitiv aufgebaut und wirkt eher wie ein System, dem immer weitere Ergänzungen angehängt wurden.

Kuma, als neustes der analysierten Service Meshes, hat in der installierten Version noch keine sinnvoll anwendbare Struktur. Jedoch muss hier beachtet werden, dass das Service Mesh zu diesem Zeitpunkt erst ein halbes Jahr seit dem ersten Release hinter sich hatte. Und es zeigt sich, dass Kuma viel der anderen Service Meshes aufgreifen und verbessern möchte. Es werden wie in der Analyse beschrieben momentan fast monatlich neue Versionen veröffentlicht, die das Service Mesh stetig verbessern. Somit darf man dieses Service Mesh bei der Wahl eines Service Meshs nicht aus den Augen verlieren. Kuma baut von Anfang an auf Nutzerfreundlichkeit und Flexibilität. Das zeigt sich schon in der frühen Phase, indem es die wenigsten Installationsschritte enthielt und von Anfang an auf das einfache Managen mehrerer Meshes über eine Control Plane setzt.

Es lässt sich also erkennen, dass jedes Service Mesh seine Vor- und Nachteile mit sich bringt, denen man sich bewusst sein sollte, wenn man sich für die Implementierung entscheidet. Hierbei spielt auch eine Rolle, wie und von wem das Service Mesh genutzt werden soll. Wie eingangs beschrieben, bietet sich über die Abstraktion des Service Managements die Möglichkeit dies außerhalb des Entwicklerteams anzusiedeln. Je komplexer die Konfiguration ist, desto enger ist diese an die Personen gekoppelt, die das Wissen tragen. Im Endeffekt entscheidet jedoch der Anwendungsfall welches Service Mesh geeignet ist. Benötigt man ein einfaches Service Mesh out-of-the-Box, eines, das flexibel erweiterbar und einsetzbar ist oder eines, das in einer vielseitigen Art und Weise konfiguriert werden kann und auf die meiste Erfahrung zurückblickt?

11 Ausblick

Die Landschaft der Service Meshes ist vielseitig und erweitert sich stetig. Das neuste Service Mesh ist Microsofts Open Service Mesh (OSM), dessen Alpha Version im August 2020 vorgestellt wurde[145]. Das OSM verfolgt den Ansatz das SMI vollständig zu unterstützen und Community-basiert durch die CNCF weiterentwickelt zu werden. Dieser Ansatz kann nicht nur beim OSM beobachtet werden. Generell geht die Entwicklung der Service Meshes in eine gemeinsame Zusammenarbeit. So sind Kuma und Linkerd bereits Teil der CNCF. Gemeinsame Erweiterungen über das Service Mesh Interface, klare Standards der Schnittstellen und mehr Entwicklung durch Community sind die Ziele der Beteiligten. Der einzige Ausreißer hier ist Google, die im Juli 2020 angekündigt haben mit ihrer neuen Open-Source Stiftung die eigenen Marken schützen zu wollen, worunter auch Istio fällt[17].

Die ersten Service Meshes sind aus Architekturen gewachsen, die diese entweder schon integriert hatten oder nicht mehr weit von einem Service Mesh entfernt waren. Durch die Anbieterspezifika brachte jedes Service Mesh seine Eigenheiten und Spezialisierungen mit sich. Von einer Entwicklung hin zu gemeinsamen Erweiterungen wird jedes Service Mesh sowie die Nutzer profitieren.

Es sind zudem noch nicht alle Herausforderungen und Probleme gemeistert. So fiel im Zuge der Analyse auf, dass sich bisher kein Service Mesh mit asynchroner Kommunikation beschäftigt hat, die für ein Microservice System viele Vorteile gegenüber einer rein synchronen Kommunikation mit sich bringt.

Die Entwicklung der Service Meshes bleibt also weiterhin spannend und ist noch lange nicht abgeschlossen.

Literaturverzeichnis

[1] Arilla, C.: *How to monitor Golden signals in Kubernetes.* Web, 2019. – URL https://sysdig.com/blog/golden-signals-kubernetes/. – (aufgerufen am 13.05.2020)

[2] AspenMesh: *Service Mesh: Adopting a Zero-Trust Approach to Security for Containerized Applications.* Web, 2020. – URL https://aspenmesh.io/wp-content/uploads/2020/01/Zero-Trust-Security-Aspen-Mesh.pdf. – (aufgerufen am 03.05.2020)

[3] Bornkessel, D. ; Prinz, H.: *Alle 11 Minuten verliebt sich ein Microservice in Linkerd.* Web, 2020. – URL https://www.innoq.com/de/articles/2020/01/linkerd2/#service-mesh:jaoderneinlinkerdoderistio. – (aufgerufen am 24.05.2020)

[4] Box, C.: *Introducing istiod simplifying the control plane.* Web, 2020. – URL https://istio.io/latest/blog/2020/istiod/. – (aufgerufen am 12.10.2020)

[5] Box, C. ; Jog, M. ; Plevyak, J. ; Ryan, L. ; Sikora, Y. ; Weiss, S.: *Redefining extensibility in proxies - introducing WebAssembly to Envoy and Istio.* Web, 2020. – URL https://istio.io/latest/blog/2020/wasm-announce/. – (aufgerufen am 25.09.2020)

[6] Bryant, D.: *Service Mesh Ultimate Guide: Managing Service-to-Service Communications in the Era of Microservices.* Web, 2020. – URL https://www.infoq.com/articles/service-mesh-ultimate-guide/. – (aufgerufen am 30.04.2020)

[7] Buehrle, A.: *Introduction to Service Meshes on Kubernetes and Progressive Delivery.* Web, 2019. – URL https://dzone.com/articles/introduction-to-service-meshes-on-kubernetes-and-p. – (aufgerufen am 03.05.2020)

[8] Burns, B.: *Designing Distributed Systems*. O'Reilly Media, 2018. – URL https://www.oreilly.com/library/view/designing-distributed-systems/9781491983638/. – ISBN 9781491983638

[9] Calcote, L.: *The Enterprise Path to Service Mesh Architectures*. O'Reilly Media, 2018. – URL https://layer5.io/books/the-enterprise-path-to-service-mesh-architectures. – ISBN 978-1-492-04176-4

[10] Campbell, C.: *Support service network policies*. Github, 2019. – URL https://github.com/linkerd/linkerd2/issues/2746. – (aufgerufen am 27.05.2020)

[11] Chen, K.: *Securing Kubernetes Applications in 5 Minutes with Service Mesh*. Web, 2019. – URL https://konghq.com/blog/securing-kubernetes-applications-5-minutes-service-mesh/. – (aufgerufen am 09.10.2020)

[12] Feinbube, L. ; Pirl, L. ; Polze, A.: *Software Fault Injection: A Practical Perspective*. Web, 2017. – URL https://www.intechopen.com/books/dependability-engineering/software-fault-injection-a-practical-perspective. – (aufgerufen am 03.05.2020)

[13] Flagger: *Flagger*. Web, o.J.. – URL https://flagger.app/. – (aufgerufen am 24.05.2020)

[14] Fowler, M.: *Microservice Trade-Offs*. Web, 2015. – URL https://martinfowler.com/articles/microservice-trade-offs.html. – (aufgerufen am 26.04.2020)

[15] Fowler, M. ; Lewis, J.: *Microservices - a definition of this new architectural term*. Web, 2014. – URL https://martinfowler.com/articles/microservices.html. – (aufgerufen am 26.04.2020)

[16] Grant, B. ; DuMars, J. S.: *Kubernetes, Cloud Native, and the Future of Software*. Web, 2019. – URL https://kubernetes.io/blog/2019/05/17/kubernetes-cloud-native-and-the-future-of-software/. – (aufgerufen am 30.04.2020)

[17] Hahn, S.: *Googles frisch gegründete Open-Source-Stiftung soll vor allem Marken schützen*. Web, 2020. – URL https://www.heise.de/news/Googles-frisch-gegruendete-Open-Source-Stiftung-soll-vor-allem-Marken-schuetzen-4790587.html. – (aufgerufen am 14.10.2020)

[18] HashiCorp: *Service Mesh and Microservices Networking*. Web, 2018. – URL https://www.datocms-assets.com/2885/1536681707-consulwhitepaperaug2018.pdf. – (aufgerufen am 13.10.2020)

[19] HashiCorp: *Layer 7 Observability with Consul Service Mesh*. Web, 2019. – URL https://www.hashicorp.com/blog/layer-7-observability-with-consul-service-mesh. – (aufgerufen am 03.10.2020)

[20] HashiCorp: *Consul Service Discovery and Mesh on Minikube*. Web, o.J.ab. – URL https://learn.hashicorp.com/tutorials/consul/kubernetes-minikube. – (aufgerufen am 29.09.2020)

[21] HashiCorp: *Consul Glossary - Agent*. Web, o.J.ac. – URL https://www.consul.io/docs/install/glossary#agent. – (aufgerufen am 30.09.2020)

[22] HashiCorp: *Consul Glossary - Client*. Web, o.J.ad. – URL https://www.consul.io/docs/install/glossary#client. – (aufgerufen am 30.09.2020)

[23] HashiCorp: *Consul Glossary - Server*. Web, o.J.ae. – URL https://www.consul.io/docs/install/glossary#server. – (aufgerufen am 30.09.2020)

[24] HashiCorp: *Consul Glossary - Datacenter*. Web, o.J.af. – URL https://www.consul.io/docs/install/glossary#datacenter. – (aufgerufen am 30.09.2020)

[25] HashiCorp: *Introduction to Consul - What is Consul*. Web, o.J.ag. – URL https://www.consul.io/docs/intro#what-is-consul. – (aufgerufen am 30.09.2020)

[26] HashiCorp: *Introduction to Consul - Basic Architecture of Consul*. Web, o.J.ah. – URL https://www.consul.io/docs/intro#basic-architecture-of-consul. – (aufgerufen am 30.09.2020)

[27] HashiCorp: *Changelog - 0.1.0 (April 17, 2014)*. Web, o.J.ai. – URL https://github.com/hashicorp/consul/blob/master/CHANGELOG.md#010-april-17-20141. – (aufgerufen am 30.09.2020)

[28] HashiCorp: *Service Splitter*. Web, o.J.aj. – URL https://www.consul.io/docs/agent/config-entries/service-splitter.html. – (aufgerufen am 03.10.2020)

[29] HashiCorp: *Service Router*. Web, o.J.ak. – URL https://www.consul.io/docs/agent/config-entries/service-router.html. – (aufgerufen am 03.10.2020)

[30] HashiCorp: *Configuration*. Web, o.J.al. – URL https://www.consul.io/docs/agent/options.html. – (aufgerufen am 03.10.2020)

[31] HashiCorp: *Telemetry - Memory usage*. Web, o.J.am. – URL https://www.consul.io/docs/agent/telemetry#memory-usage. – (aufgerufen am 03.10.2020)

[32] HashiCorp: *Telemetry - Transaction Timing*. Web, o.J.an. – URL https://www.consul.io/docs/agent/telemetry#transaction-timing. – (aufgerufen am 03.10.2020)

[33] HashiCorp: *Telemetry - Cluster Health*. Web, o.J.ao. – URL https://www.consul.io/docs/agent/telemetry#cluster-health. – (aufgerufen am 03.10.2020)

[34] HashiCorp: *Layer 7 Observability with Consul Service Mesh, Prometheus, Grafana, and Kubernetes*. Web, o.J.ap. – URL https://learn.hashicorp.com/tutorials/consul/kubernetes-layer7-observability. – (aufgerufen am 03.10.2020)

[35] HashiCorp: *Consul Service Discovery and Mesh on Minikube - Access the Consul UI*. Web, o.J.aq. – URL https://learn.hashicorp.com/tutorials/consul/kubernetes-minikube?in=consul/kubernetes#access-the-consul-ui. – (aufgerufen am 03.10.2020)

[36] HashiCorp: *Consul Connect Envoy*. Web, o.J.ar. – URL https://www.consul.io/commands/connect/envoy. – (aufgerufen am 03.10.2020)

[37] HashiCorp: *Envoy Integration*. Web, o.J.as. – URL https://www.consul.io/docs/connect/proxies/envoy. – (aufgerufen am 03.10.2020)

[38] HashiCorp: *Built-In Proxy Options*. Web, o.J.at. – URL https://www.consul.io/docs/connect/proxies/built-in. – (aufgerufen am 03.10.2020)

[39] HashiCorp: *How Connect Works*. Web, o.J.au. – URL https://www.consul.io/docs/connect/connect-internals.html. – (aufgerufen am 04.10.2020)

[40] HashiCorp: *Connect Certificate Management*. Web, o.J.av. – URL https://www.consul.io/docs/connect/ca. – (aufgerufen am 04.10.2020)

[41] HashiCorp: *Intentions*. Web, o.J.aw. – URL https://www.consul.io/docs/connect/intentions.html. – (aufgerufen am 04.10.2020)

[42] HashiCorp: *Common Error Messages*. Web, o.J.ax. – URL https://www.consul.io/docs/troubleshoot/common-errors. – (aufgerufen am 04.10.2020)

[43] HashiCorp: *Frequently Asked Questions*. Web, o.J.ay. – URL https://www.consul.io/docs/troubleshoot/faq. – (aufgerufen am 04.10.2020)

[44] HashiCorp: *Service Networking Across Any Cloud*. Web, o.J.az. – URL https://www.consul.io/. – (aufgerufen am 04.10.2020)

[45] HashiCorp: *Connect on Nomad*. Web, o.J.ba. – URL https://www.consul.io/docs/connect/nomad. – (aufgerufen am 04.10.2020)

[46] HashiCorp: *Required Ports*. Web, o.J.bb. – URL hhttps://www.consul.io/docs/install/ports. – (aufgerufen am 04.10.2020)

[47] HashiCorp: *Consul Integration Program*. Web, o.J.bc. – URL https://www.consul.io/docs/integrate/partnerships. – (aufgerufen am 04.10.2020)

[48] HashiCorp: *Proxy Registration*. Web, o.J.bd. – URL https://www.consul.io/docs/connect/registration. – (aufgerufen am 04.10.2020)

[49] HashiCorp: *Multi-Cluster Federation Overview*. Web, o.J.be. – URL https://www.consul.io/docs/k8s/installation/multi-cluster. – (aufgerufen am 04.10.2020)

[50] HashiCorp: *Multi-Platform Service Mesh*. Web, o.J.bf. – URL https://www.consul.io/use-cases/multi-platform-service-mesh. – (aufgerufen am 04.10.2020)

[51] Hashimoto, M.: *HashiCorp Consul 1.2: Service Mesh*. Web, 2018. – URL https://www.hashicorp.com/blog/consul-1-2-service-mesh. – (aufgerufen am 01.10.2020)

[52] Interface, Service M.: *Ecosystem*. Web, o.J.. – URL https://smi-spec.io/#ecosystem. – (aufgerufen am 25.09.2020)

[53] Istio: *1.5.x Releases*. Web, 2020aa. – URL https://istio.io/news/releases/1.5.x/. – (aufgerufen am 010.05.2020)

[54] Istio: *Architecture*. Web, 2020ab. – URL https://istio.io/v1.5/docs/ops/deployment/architecture/. – (aufgerufen am 27.10.2020)

[55] Istio: *Announcing Istio 1.5*. Web, 2020ac. – URL https://istio.io/news/releases/1.5.x/announcing-1.5/. – (aufgerufen am 10.05.2020)

[56] Istio: *Traffic-Management - Gateways*. Web, 2020ad. – URL https://istio.io/v1.5/docs/concepts/traffic-management/#gateways. – (aufgerufen am 27.09.2020)

[57] Istio: *Getting Started*. Web, 2020ae. – URL https://istio.io/v1.5/docs/setup/getting-started/. – (aufgerufen am 06.09.2020)

[58] Istio: *Traffic-Management - Load Balancing Options*. Web, 2020af. – URL https://istio.io/v1.5/docs/concepts/traffic-management/#load-balancing-options. – (aufgerufen am 27.09.2020)

[59] Istio: *Virtual Service*. Web, 2020ag. – URL https://istio.io/v1.5/docs/reference/config/networking/virtual-service/. – (aufgerufen am 27.09.2020)

[60] Istio: *Istio Standard Metrics*. Web, 2020ah. – URL https://istio.io/v1.5/docs/reference/config/telemetry/metrics/. – (aufgerufen am 27.09.2020)

[61] Istio: *Observability*. Web, 2020ai. – URL https://istio.io/v1.5/docs/concepts/observability/. – (aufgerufen am 27.09.2020)

[62] Istio: *Querying Metrics from Prometheus*. Web, 2020aj. – URL https://istio.io/v1.5/docs/tasks/observability/metrics/querying-metrics/. – (aufgerufen am 27.09.2020)

[63] Istio: *Visualizing Metrics with Grafana*. Web, 2020ak. – URL https://istio.io/v1.5/docs/tasks/observability/metrics/using-istio-dashboard/. – (aufgerufen am 27.09.2020)

[64] Istio: *Visualizing Your Mesh*. Web, 2020al. – URL https://istio.io/v1.5/docs/tasks/observability/kiali/. – (aufgerufen am 27.09.2020)

[65] Istio: *Distributed Tracing FAQ*. Web, 2020am. – URL https://istio.io/v1.5/faq/distributed-tracing/. – (aufgerufen am 27.09.2020)

[66] Istio: *Distributed Tracing - Overview*. Web, 2020an. – URL https://istio.io/v1.5/docs/tasks/observability/distributed-tracing/overview/. – (aufgerufen am 27.09.2020)

[67] Istio: *Circuit Breaking*. Web, 2020ao. – URL https://istio.io/v1.5/docs/tasks/traffic-management/circuit-breaking/. – (aufgerufen am 27.09.2020)

[68] Istio: *Traffic Management - Retries*. Web, 2020ap. – URL https://istio.io/v1.5/docs/concepts/traffic-management/#retries. – (aufgerufen am 27.09.2020)

[69] Istio: *Traffic Management - Timeouts*. Web, 2020aq. – URL https://istio.io/v1.5/docs/concepts/traffic-management/#timeouts. – (aufgerufen am 27.09.2020)

[70] Istio: *Virtual Service - Destination*. Web, 2020ar. – URL https://istio.io/v1.5/docs/reference/config/networking/virtual-service/#Destination. – (aufgerufen am 27.09.2020)

[71] Istio: *Traffic Management - Fault Injection*. Web, 2020as. – URL https://istio.io/v1.5/docs/concepts/traffic-management/#fault-injection. – (aufgerufen am 27.09.2020)

[72] Istio: *Security - Mutual TLS Authentication*. Web, 2020at. – URL https://istio.io/v1.5/docs/concepts/security/#mutual-tls-authentication. – (aufgerufen am 06.09.2020)

[73] Istio: *Plugging in External CA Key and Certificate*. Web, 2020au. – URL https://istio.io/v1.5/pt-br/docs/tasks/security/citadel-config/plugin-ca-cert/. – (aufgerufen am 27.09.2020)

[74] Istio: *Authorization Policy*. Web, 2020av. – URL https://istio.io/v1.5/docs/reference/config/security/authorization-policy/. – (aufgerufen am 06.09.2020)

[75] Istio: *Bookinfo Application*. Web, 2020aw. – URL https://istio.io/v1.5/docs/examples/bookinfo/. – (aufgerufen am 06.09.2020)

[76] Istio: *Diagnostic Tools*. Web, 2020ax. – URL https://istio.io/v1.5/docs/ops/diagnostic-tools/. – (aufgerufen am 23.09.2020)

[77] Istio: *Common Problems*. Web, 2020ay. – URL https://istio.io/v1.5/docs/ops/common-problems/. – (aufgerufen am 23.09.2020)

[78] Istio: *Visualizing Your Mesh - Generating a service graph*. Web, 2020az. – URL https://istio.io/v1.5/docs/tasks/observability/kiali/#generating-a-service-graph. – (aufgerufen am 23.09.2020)

[79] Istio: *What is Istio - Platform Support*. Web, 2020ba. – URL https://istio.io/v1.5/docs/concepts/what-is-istio/#platform-support. – (aufgerufen am 23.09.2020)

[80] Istio: *Protocol Selection*. Web, 2020bb. – URL https://istio.io/v1.5/docs/ops/configuration/traffic-management/protocol-selection/. – (aufgerufen am 23.09.2020)

[81] Istio: *Extensibility*. Web, 2020bc. – URL https://istio.io/v1.5/docs/concepts/wasm/. – (aufgerufen am 25.09.2020)

[82] Istio: *IstioOperator Options*. Web, 2020bd. – URL https://istio.io/v1.5/docs/reference/config/istio.operator.v1alpha1/. – (aufgerufen am 25.09.2020)

[83] Istio: *Virtual Machines in Single-Network Meshes*. Web, 2020be. – URL https://istio.io/v1.5/docs/examples/virtual-machines/single-network/. – (aufgerufen am 25.09.2020)

[84] Istio: *Multicluster Installation*. Web, 2020bf. – URL https://istio.io/v1.5/docs/setup/install/multicluster/. – (aufgerufen am 25.09.2020)

[85] Istio: *Getting Envoys Access Logs*. Web, 2020bg. – URL https://istio.io/v1.5/docs/tasks/observability/logs/access-log/. – (aufgerufen am 27.09.2020)

[86] Istio: *Visualizing Your Mesh - Viewing and editing Istio configuration YAML*. Web, 2020bh. – URL https://istio.io/v1.5/docs/tasks/observability/kiali/#viewing-and-editing-istio-configuration-yaml. – (aufgerufen am 27.09.2020)

[87] Istio: *Authentication Policy*. Web, 2020bi. – URL https://istio.io/v1.5/docs/tasks/security/authentication/authn-policy/. – (aufgerufen am 11.10.2020)

[88] Istio: *Ingress Gateways*. Web, 2020bi. – URL https://istio.io/v1.5/docs/tasks/traffic-management/ingress/ingress-control/. – (aufgerufen am 11.10.2020)

[89] Istio: *Mirroring*. Web, 2020bi. – URL https://istio.io/v1.5/docs/tasks/traffic-management/mirroring/. – (aufgerufen am 11.10.2020)

[90] Klein, M.: *Service mesh data plane vs. control plane*. Web, 2017. – URL https://blog.envoyproxy.io/service-mesh-data-plane-vs-control-plane-2774e720f7fc. – (aufgerufen am 03.05.2020)

[91] Kuma: *Certificates*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/documentation/security/#certificates. – (aufgerufen am 10.10.2020)

[92] Kuma: *Changelog*. Web, o.J.. – URL https://github.com/kumahq/kuma/blob/master/CHANGELOG.md. – (aufgerufen am 10.10.2020)

[93] Kuma: *chore update k8s to 1.18*. Web, o.J.. – URL https://github.com/kumahq/kuma/pull/720. – (aufgerufen am 09.10.2020)

[94] Kuma: *Circuit Breaker*. Web, o.J.. – URL https://kuma.io/docs/0.5.1/policies/circuit-breaker/#usage. – (aufgerufen am 14.10.2020)

[95] Kuma: *DPs and Data Model*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/documentation/dps-and-data-model/. – (aufgerufen am 10.10.2020)

[96] Kuma: *Fault Injection*. Web, o.J.. – URL https://kuma.io/docs/0.5.1/policies/fault-injection/. – (aufgerufen am 14.10.2020)

[97] Kuma: *Fault Injection*. Web, o.J.. – URL https://kuma.io/docs/0.5.0/policies/fault-injection/. – (aufgerufen am 10.10.2020)

[98] Kuma: *GUI*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/documentation/gui/. – (aufgerufen am 10.10.2020)

[99] Kuma: *Health Check*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/health-check/. – (aufgerufen am 10.10.2020)

[100] KUMA: *HTTP API*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/documentation/http-api/. – (aufgerufen am 10.10.2020)

[101] KUMA: *HTTP support in Kuma*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/http-support-in-kuma/. – (aufgerufen am 10.10.2020)

[102] KUMA: *Kubernetes*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/installation/kubernetes/. – (aufgerufen am 10.10.2020)

[103] KUMA: *Mesh*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/mesh/. – (aufgerufen am 10.10.2020)

[104] KUMA: *mTLS*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/documentation/security/#mtls. – (aufgerufen am 10.10.2020)

[105] KUMA: *Proxy Template*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/proxy-template/. – (aufgerufen am 10.10.2020)

[106] KUMA: *Traffic Log*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/traffic-log/. – (aufgerufen am 10.10.2020)

[107] KUMA: *Traffic Metrics*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/traffic-metrics/. – (aufgerufen am 10.10.2020)

[108] KUMA: *Traffic Permissions*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/traffic-permissions/. – (aufgerufen am 10.10.2020)

[109] KUMA: *Traffic Route*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/policies/traffic-route/. – (aufgerufen am 09.10.2020)

[110] KUMA: *What is Kuma*. Web, o.J.. – URL https://kuma.io/docs/0.4.0/overview/what-is-kuma/. – (aufgerufen am 10.10.2020)

[111] KUMA: *Why Kuma?* Web, o.J.. – URL https://kuma.io/docs/0.4.0/overview/why-kuma/. – (aufgerufen am 05.10.2020)

[112] KUMAR, A.: *Circuit Breaker Pattern*. Web, 2018. – URL https://dzone.com/articles/circuit-breaker-pattern. – (aufgerufen am 06.05.2020)

[113] LINKERD: *Architecture*. Web, o.J.. – URL https://linkerd.io/2/reference/architecture/. – (aufgerufen am 22.05.2020)

[114] LINKERD: *Automated Canary Releases*. Web, o.J.. – URL https://linkerd.io/2/tasks/canary-release/. – (aufgerufen am 12.10.2020)

[115] LINKERD: *Automatic mTLS*. Web, o.J.. – URL https://linkerd.io/2/features/automatic-mtls/. – (aufgerufen am 27.05.2020)

[116] LINKERD: *Automatic Proxy Injection*. Web, o.J.. – URL https://linkerd.io/2/features/proxy-injection/. – (aufgerufen am 28.05.2020)

[117] LINKERD: *Configuring Retries*. Web, o.J.. – URL https://linkerd.io/2/tasks/configuring-retries/. – (aufgerufen am 24.05.2020)

[118] LINKERD: *Configuring Timeouts*. Web, o.J.. – URL https://linkerd.io/2/tasks/configuring-timeouts/. – (aufgerufen am 24.05.2020)

[119] LINKERD: *Customizing Installation*. Web, o.J.. – URL https://linkerd.io/2/tasks/customize-install/. – (aufgerufen am 28.05.2020)

[120] LINKERD: *Distributed tracing with Linkerd*. Web, o.J.. – URL https://linkerd.io/2/tasks/distributed-tracing/. – (aufgerufen am 24.05.2020)

[121] LINKERD: *Fault Injection*. Web, o.J.. – URL https://linkerd.io/2/tasks/fault-injection/. – (aufgerufen am 27.05.2020)

[122] LINKERD: *Features*. Web, o.J.. – URL https://linkerd.io/2/features/. – (aufgerufen am 27.05.2020)

[123] LINKERD: *Getting Started*. Web, o.J.. – URL https://linkerd.io/2/getting-started/. – (aufgerufen am 24.05.2020)

[124] LINKERD: *inject*. Web, o.J.. – URL https://linkerd.io/2/reference/cli/inject/. – (aufgerufen am 24.05.2020)

[125] LINKERD: *Installing Multicluster*. Web, o.J.. – URL https://linkerd.io/2/tasks/installing-multicluster/. – (aufgerufen am 12.10.2020)

[126] LINKERD: *Load Balancing*. Web, o.J.. – URL https://linkerd.io/2/features/load-balancing/. – (aufgerufen am 24.05.2020)

[127] LINKERD: *Multicluster support*. Web, o.J.. – URL https://linkerd.io/2/features/multicluster_support/. – (aufgerufen am 31.05.2020)

[128] LINKERD: *Proxy Configuration*. Web, o.J.. – URL https://linkerd.io/2/reference/proxy-configuration/. – (aufgerufen am 28.05.2020)

[129] Linkerd: *tap*. Web, o.J.. – URL https://linkerd.io/2/reference/cli/tap/. – (aufgerufen am 27.09.2020)

[130] Linkerd: *TCP Proxying and Protocol Detection*. Web, o.J.. – URL https://linkerd.io/2/features/protocol-detection/. – (aufgerufen am 24.05.2020)

[131] Linkerd: *Telemetry and Monitoring*. Web, o.J.. – URL https://linkerd.io/2/features/telemetry/. – (aufgerufen am 24.05.2020)

[132] Linkerd: *Troubleshooting*. Web, o.J.. – URL https://linkerd.io/2/tasks/troubleshooting/. – (aufgerufen am 27.05.2020)

[133] McCarron, P.: *Load Balancing Strategies for Consul*. Web, 2019. – URL https://www.hashicorp.com/blog/enabling-distributed-tracing-with-hashicorp-consul. – (aufgerufen am 03.10.2020)

[134] Minikube: *none - Linux none (bare-metal) driver*. Web, o.J.. – URL https://minikube.sigs.k8s.io/docs/drivers/none/. – (aufgerufen am 14.05.2020)

[135] Morgan, W.: *What's a service mesh? And why do I need one?* Web, 2017. – URL https://buoyant.io/2017/04/25/whats-a-service-mesh-and-why-do-i-need-one/. – (aufgerufen am 03.05.2020)

[136] Morgan, W.: *Announcing Linkerd 2.4: Traffic Splitting and SMI*. Web, 2019a. – URL https://linkerd.io/2019/07/11/announcing-linkerd-2.4/. – (aufgerufen am 24.05.2020)

[137] Morgan, W.: *Linkerd v2 How Lessons from Production Adoption Resulted in a Rewrite of the Service Mesh*. Web, 2019b. – URL https://www.infoq.com/articles/linkerd-v2-production-adoption/. – (aufgerufen am 11.10.2020)

[138] Morgan, W.: *Announcing Linkerd 2.7: External PKI support, better gitops workflows, streamlined cert rotation, and more*. Web, 2020. – URL https://linkerd.io/2020/02/10/announcing-linkerd-2.7/. – (aufgerufen am 27.05.2020)

[139] OpenCensus: *Exporters*. Web, o.J.. – URL https://opencensus.io/service/exporters/. – (aufgerufen am 24.05.2020)

[140] Palladino, M.: *Introducing Kuma: The Universal Service Mesh.* Web, 2019a. – URL https://konghq.com/blog/introducing-kuma-universal-service-mesh/. – (aufgerufen am 05.10.2020)

[141] Palladino, M.: *Introduction to Kuma.* Web, 2019b. – URL https://jaxenter.com/kuma-service-mesh-164281.html. – (aufgerufen am 14.10.2020)

[142] Palladino, M.: *Kuma 0.4 Released With L7 Tracing + Grafana Dashboards!* Web, 2020. – URL https://konghq.com/blog/kuma-0-4-released-with-l7-tracing-grafana-dashboards/. – (aufgerufen am 10.10.2020)

[143] Palladino, M.: *Kuma 0.6.0 Released With Hybrid Universal Support for Service Mesh and CNCF Donation.* Web, 2020b. – URL https://konghq.com/blog/kuma-0-6-0-released-with-hybrid-universal-support-for-service-mesh/. – (aufgerufen am 14.10.2020)

[144] Palladino, M.: *Kuma 0.7.2 Released.* Web, 2020c. – URL https://konghq.com/blog/kuma-0-7-2-released/. – (aufgerufen am 14.10.2020)

[145] Parbel, M.: *Service Meshes Microsoft startet SMI-kompatibles Open Service Mesh.* Web, 2020. – URL https://www.heise.de/news/Service-Meshes-Microsoft-startet-SMI-kompatibles-Open-Service-Mesh-4864883.html. – (aufgerufen am 14.10.2020)

[146] Prinz, H. ; Wolff, E.: *Service Mesh - The New Infrastructure for Microservices.* innoQ Deutschland GmbH, 2019. – URL http://leanpub.com/service-mesh-primer. – ISBN 978-3-9821126-1-9

[147] RedHat: *DIE MIGRATION ZU CLOUDNATIVEN ANWENDUNGEN - Eine 8-Schritte-Anleitung für Ihre digitale Migration.* RedHat, o.J.. – URL https://www.redhat.com/cms/managed-files/mi-path-to-cloud-native-apps-ebook-f12255cs-201805-a4-de.pdf. – (aufgerufen am 30.04.2020)

[148] RedHat: *Was ist ein Service Mesh?* Web, o.J.. – URL https://www.redhat.com/de/topics/microservices/what-is-a-service-mesh. – (aufgerufen am 03.05.2020)

[149] Rupp, H.: *Istio: Das Service-Mesh für verteilte Systeme.* Web, 2018. – URL https://m.heise.de/developer/artikel/Istio-Das-Service-

Mesh-fuer-verteilte-Systeme-4153426.html?seite=all. – (aufgerufen am 10.05.2020)

[150] Smith, F.: *What Is a Service Mesh?* Web, 2018. – URL https://www.nginx.com/blog/what-is-a-service-mesh/. – (aufgerufen am 03.05.2020)

[151] solo.io: *A Multi Mesh Management Tool.* Web, o.J.. – URL https://docs.solo.io/service-mesh-hub/latest. – (aufgerufen am 31.05.2020)

[152] Vargo, S.: *Load Balancing Strategies for Consul.* Web, 2017. – URL https://www.hashicorp.com/blog/load-balancing-strategies-for-consul. – (aufgerufen am 03.10.2020)

[153] Wolff, E. ; Prinz, H.: *Das Service Mesh - Die Lösung aller Microservice-Probleme?* Web, 2019. – URL https://www.innoq.com/de/articles/2019/10/service-mesh/. – (aufgerufen am 03.05.2020)

A Anhang

A.1 Konfiguration der VM

```
Allgemein
Betriebssystem: Ubuntu(64-bit)

System
Hauptspeicher: 8129 MB
Prozessoren: 2
Bootreihenfolge: Diskette,DVD,Platte
Beschleunigung: VT-x/AMD-v, Nested Paging, KVM-Paravirtualisierung

Anzeige
Graphikspeicher: 16 MB
Fernsteuerung: deaktiviert
Videoaufzeichnung: deaktiviert

Massenspeicher
Controller IDE
        Sekundaerer Master: [DVD] leer
Controller SATA
        SATA-Port 0: name.vdi (normal, 25,00 GB)
```

A.2 Service Deployment

```
kubectl -n pfefferkorn apply -f deployments/
kubectl -n pfefferkorn apply -f receptionservice/deployments/
kubectl -n pfefferkorn apply -f baristaservice/deployments/
kubectl -n pfefferkorn apply -f stockservice/deployments/
kubectl -n pfefferkorn apply -f supplierservice/deployments/
kubectl -n pfefferkorn apply -f managerservice/deployments/
```

A.3 Sequenzdiagramm Kaffeebestellung

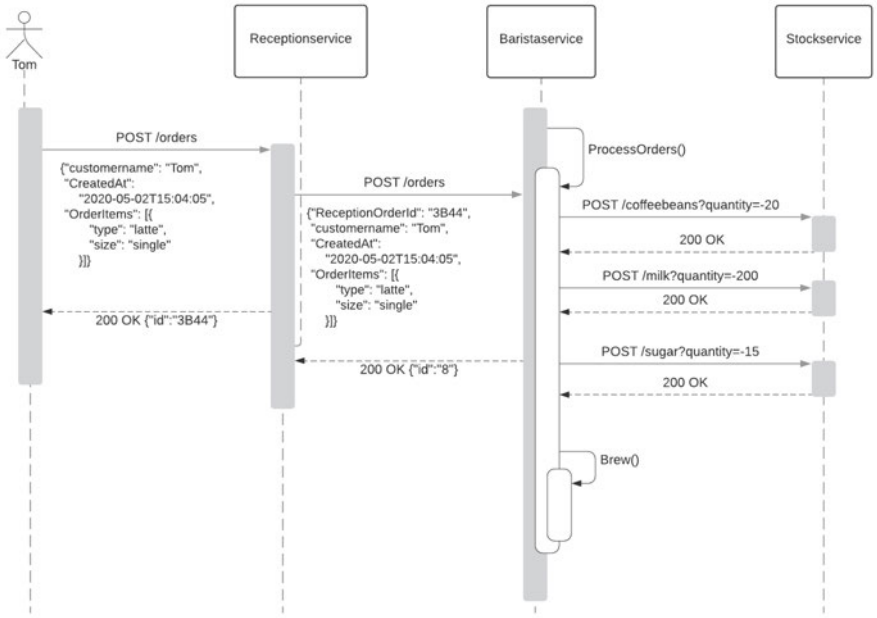

Abbildung 8: Kunde Tom bestellt einen Kaffee

A.4 Istio Ingress Konfiguration

```
$ export INGRESS_PORT=
 $(kubectl -n istio-system get service istio-ingressgateway -o
  jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')

$ export SECURE_INGRESS_PORT=
 $(kubectl -n istio-system get service istio-ingressgateway -o
  jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')

$ export INGRESS_HOST=$(minikube ip)

$ export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

A.5 Linkerd Precheck

```
brew install linkerd
linkerd check --pre
linkerd install | kubectl apply -f -
linkerd check
```

A.6 Consul Installationsbefehle

Hashicorp dem Helm Repo hinzufügen

```
helm repo add hashicorp https://helm.releases.hashicorp.com
```

Consul installieren

```
helm install -f helm-consul-values.yaml hashicorp hashicorp/consul
```

Verbindung zum Server

```
kubectl exec -it consul-server-pod-name -- /bin/sh
```

Injection Annotation

```
  annotations:
    'consul.hashicorp.com/connect-inject': 'true'
```

A.7 Helm-Consul-Values

```
# Choose an optional name for the datacenter
global:
  datacenter: minidc

# Enable the Consul Web UI via a NodePort
ui:
  service:
    type: 'NodePort'

# Enable Connect for secure communication between nodes
connectInject:
  enabled: true

client:
  enabled: true

# Use only one Consul server for local development
server:
  replicas: 1
  bootstrapExpect: 1
  disruptionBudget:
    enabled: true
    maxUnavailable: 0
```

A.8 Kuma Installationsbefehle

Minikube mit weiteren Parametern starten

```
sudo minikube start --vm-driver=none --apiserver-ips 127.0.0.1
--apiserver-name localhost  --memory 6144 --kubernetes-version v1.17.4
```

Kuma installieren

```
./kumactl install control-plane | kubectl apply -f -
```

Namespace labeln

```
kubectl label namespace pfefferkorn kuma.io/sidecar-injection=enabled
```

Mesh in Deployment Datei des Services definieren

```
kuma.io/mesh: default
```

A.9 Kuma TrafficPermission Policy

```
echo "apiVersion: kuma.io/v1alpha1
kind: TrafficPermission
mesh: default
metadata:
  namespace: pfefferkorn
  name: enable-all-traffic
spec:
  sources:
    - match:
        service: '*'
  destinations:
    - match:
        service: '*'" | kubectl apply -f -
```

A.10 Linux Top Befehl vor Installation

```
Tasks: 187 total,   1 running, 141 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18,8 us, 12,2 sy,  0,0 ni, 64,7 id,  1,9 wa,  0,0 hi,  2,4 si,  0,0 st
KiB Mem :  8153308 total,  6791388 free,   712224 used,   649696 buff/cache
KiB Swap:  1214880 total,  1214880 free,        0 used.  7176148 avail Mem
```

A.11 Linux Top Befehl Istio

```
Tasks: 363 total,   2 running, 304 sleeping,   0 stopped,   0 zombie
%Cpu(s): 27,0 us, 10,1 sy,  0,5 ni, 60,3 id,  1,0 wa,  0,0 hi,  1,2 si,  0,0 st
KiB Mem :  8153308 total,  1975328 free,  3046492 used,  3131488 buff/cache
KiB Swap:  1214880 total,  1214880 free,        0 used.  4891900 avail Mem
```

A.12 Linux Top Befehl Linkerd

```
Tasks: 378 total,   2 running, 309 sleeping,   0 stopped,   0 zombie
%Cpu(s): 28,0 us,  9,5 sy,  0,4 ni, 60,6 id,  0,5 wa,  0,0 hi,  1,0 si,  0,0 st
KiB Mem :  8153308 total,  2127104 free,  2855392 used,  3170812 buff/cache
KiB Swap:  1214880 total,   998048 free,   216832 used.  5116024 avail Mem
```

A.13 Linux Top Befehl Consul

```
Tasks: 383 total,   2 running, 316 sleeping,   0 stopped,   1 zombie
%Cpu(s): 23,4 us, 10,4 sy,  6,3 ni, 58,1 id,  0,7 wa,  0,0 hi,  1,2 si,  0,0 st
KiB Mem :  8153308 total,  3299744 free,  2567056 used,  2286508 buff/cache
KiB Swap:  1214880 total,   740512 free,   474368 used.  5400792 avail Mem
```

A.14 Linux Top Befehl Kuma

```
Tasks: 345 total,   1 running, 277 sleeping,   0 stopped,   0 zombie
%Cpu(s): 21,5 us,  9,4 sy,  0,0 ni, 67,8 id,  0,5 wa,  0,0 hi,  0,7 si,  0,0 st
KiB Mem :  8153308 total,  2872760 free,  2305928 used,  2974620 buff/cache
KiB Swap:  1214880 total,  1197216 free,    17664 used.  5649264 avail Mem
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Evaluierung verschiedener Service Mesh Implementierungen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original