FH AACHEN - UNIVERSITY OF APPLIED SCIENCES

Fachbereich 09 - Medizintechnik und Technomathematik

Seminararbeit

Untersuchung und Konfiguration von Traefik als Application Proxy

1. Betreuer: Prof. Dr. Alexander Voß

2. Betreuer: Laurenz Gohr

Semester: WiSe 2023/2024

Student: Michael Hamm

Matr.-Nr.: 3521690

E-Mail: michael.hamm@alumni.fh-aachen.de

Fachsemester: 05

Studiengang: Angewandte Mathematik und Informatik [AMI/Bachelor]

Datum: 15. Dezember 2023

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Untersuchung und Konfiguration von Traefik als Application Proxy

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Michael Hamm

Aachen, den 15. Dezember 2023

Unterschrift: 

Abstract

Diese Seminararbeit analysiert die Eignung von *Traefik* als Ersatz für die bestehende, von der Firma Continue Software GmbH (Continue) genutzten, High Availability-Proxy (HA-Proxy)-Konfiguration in einer Docker-Umgebung. Sie untersucht die Vor- und Nachteile beider Lösungen und stellt fest, dass *Traefik* eine vielversprechende Alternative bietet. Die Arbeit beginnt mit einer eingehenden Analyse der aktuellen Docker-Umgebung und identifiziert deren Herausforderungen, darunter komplexe Konfigurationen und hohe Dienstanforderungen. Anschließend erfolgt eine umfassende Untersuchung von *Traefik*, einschließlich seiner Funktionen als Reverse Proxy, Routing, Load-Balancing und HTTPS-Implementierung. Die Umstellung von HA-Proxy auf *Traefik* wird behandelt, inklusive der Konfiguration und Herausforderungen während des Übergangs. Die Arbeit schließt mit einem Vergleich beider Konfigurationen und einem Ausblick auf mögliche zukünftige Verbesserungen ab, insbesondere in Bezug auf die Wartbarkeit von *Traefik*-Konfigurationen und den zusätzlichen Wert, den *Traefik* bieten kann.

Inhaltsverzeichnis

1 Abkürzungsverzeichnis

HA-Proxy High Availability-Proxy

Continue Continue Software GmbH

RP Reverse-Proxy

HTTPS Hypertext Transfer Protocol Secure

SSL Secure Sockets Layer

TLS Transport Layer Security

DDoS Distributed Denial of Service

ELB Elastic Load Balancing

Git-Repo Git-Repository

2 Einleitung

2.1 Motivation

Bei Continue werden seit 2017 Docker-Umgebungen verwendet. Zu der damaligen Zeit waren spezifische Tools für Docker-Container noch nicht verbreitet, weshalb an einer firmeninternen Lösung gearbeitet worden ist. In den darauffolgenden Jahren wuchs die Relevanz von effizientem Ressourcenmanagement. Auch die Komplexität der von Continue entwickelten Lösung stieg über die Jahre an, weshalb bei Continue die Entscheidung getroffen wurde nach besseren Lösungen zu suchen. A priori hatte sich Continue für *Traefik* als mögliche Lösung entschieden. Diese Seminararbeit beschäfigt sich daher mit der Untersuchung und Konfiguration von Traefik als mögliche Alternative zu der bisherigen Docker-Infrastruktur im besonderen Hinblick auf die Probleme, die es bei der derzeitigen Docker-Umgebung gibt (siehe Kap. 4.2 ff).

2.2 Vorgehensweise

Zu Beginn dieser Arbeit werden zunächst die Grundlagen für die darauffolgenden Kapitel geschaffen. Danach werden die allgemeinen Anforderungen für eine docker-orientierte Umgebung aufgezählt. Anschließend werden die Stärken und Schwächen der aktuellen Docker-Umgebung evaluiert, um herausfinden zu können, in welchen Anforderungen eine Nachbesserung stattfinden sollte. Es stellt sich heraus, dass einige Anforderungen, die eine docker-orientierte Umgebung erfüllen sollte, nicht vollständig erfüllt sind, weshalb sich diese Arbeit mit möglichen Alternativen der bisherigen Umgebung auseinandersetzt. Daher beschäftigt sich diese Arbeit mit dem Tool *Traefik* als mögliche Alternative und erläutert dessen Funktionen. Anschließend findet ein direkter Vergleich zwischen *Traefik* und der aktuellen Umgebung statt, um klarzustellen, welche konkreten Vorteile *Traefik* gegenüber der derzeitigen Umgebung hat. Zum Abschluss dieser Arbeit wird ein kleiner Ausblick über Projekte gegeben, die außerhalb dieser Seminararbeit ausgeführt werden müssen (Kap. 3 ff.).

3 Grundlagen

Das folgende Kapitel widmet sich den Grundlagen, die zum Verständnis dieser Seminararbeit essentiell sind. Dabei werden die elementaren Grundlagen vorausgesetzt und nur noch auf die wichtigsten Begriffe eingegangen.

3.1 Docker-Umgebung

Docker ist ein Open-Source-Tool für die Containerisierung von Softwareanwendungen, das auf verschiedenen Betriebssystemen ausgeführt werden kann. Eine Docker-Umgebung besteht aus Containern, die Anwendungen isoliert ausführen. Ein wesentlicher Bestandteil ist das Docker Image, ein Softwarepaket, das alles enthält, was für die Ausführung einer Anwendung erforderlich ist. Das Dockerfile spezifiziert die Einstellungen eines Docker-Services.

Docker Compose ist ein Tool, das die Ausführung von Docker-Anwendungen vereinfacht. Docker Volumes ermöglichen das Speichern von Daten außerhalb von Containern. Letztlich bietet Docker Networking Netzwerkfunktionalitäten für die Kommunikation zwischen Containern.

3.2 Domain-Service-Matching

Im Kontext des Docker Service taucht häufig der Begriff „Domain-Service-Matching" auf, der einen Prozess beschreibt, bei dem Anfragen von Domains an die entsprechenden Services gerichtet werden. Dieser Vorgang ermöglicht beispielsweise die Weiterleitung von Anfragen an die Domain „example.com" an den Service mit dem Namen „service1" und Anfragen an die Domain „example2.com" an den Service mit dem Namen „service2". Eine grafische Darstellung eines Domain-Service-Matchings mit der Nutzung eines Reverse Proxys ist in Abbildung 4.1 zu finden. Folglich ermöglicht das Domain-Service-Matching eine flexible Strukturierung von Diensten und eine deutliche Abgrenzung der Verantwortlichkeiten.

3.3 Load Balancer

Ein Load Balancer ist eine Softwareanwendung, die den Datenverkehr gleichmäßig auf mehreren Servern oder Ressourcen verteilt, um die Auslastung zu optimieren, die Leistung zu verbessern oder die Verfügbarkeit von Anwendungen zu erhöhen. Die Anwendungsbereiche eines Load Balancers sind Netzwerk- und Cloudumgebungen. Außerdem ist der Load Balancer unter anderem auch für das Domain-Service-Matching zuständig.

3.4 Reverse Proxy

Der Reverse-Proxy (RP) ist ein Server, der vor einen Webserver geschaltet ist. Dieser Server analysiert die Anfrage des Clients, was unter anderem auch eine Prüfung von Sicherheitsregelungen beinhalten kann. Des Weiteren leitet er innerhalb der Analyse die Anfrage an den richtigen Webserver weiter, der die Website hostet. Die Vorteile eines RPs bestehen daher aus Sicherheit, Load-Balancing und der Möglichkeit, mehrere Webserver hinter einer einzigen öffentlichen IP-Adresse zu betreiben [3, 22].

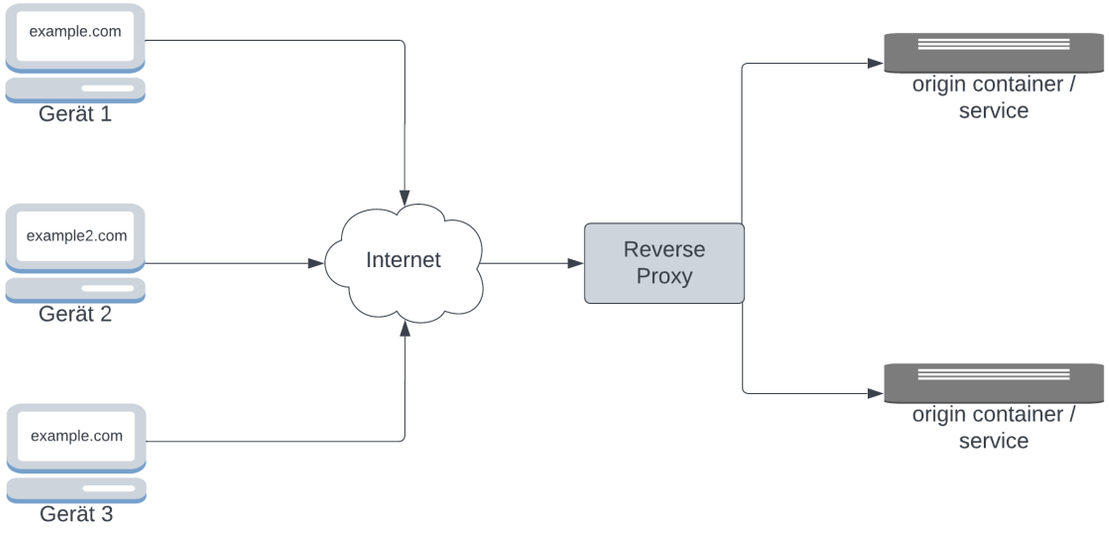

Abbildung 3.1: Grafische Darstellung eines Reverse Proxys [7].

3.5 HA-Proxy

Der High-Availability-Proxy (HA-Proxy) ist ein Open-Source-Load-Balancer und Proxy Server. Der HA-Proxy kann als Reverse-Proxy fungieren, indem er Anfragen von Clients entgegennimmt und sie an die entsprechenden Backend-Server weiterleitet. Er ist auch ein Load-Balancer, da er ein Domain-Service Matching durchführen kann. Ein HA-Proxy dient außerdem dazu sicherzustellen, dass Dienste immer erreichbar sind und Ausfallzeiten minimiert werden. Folglich kann ein HA-Proxy in Umgebungen eingesetzt werden, in denen die kontinuierliche Verfügbarkeit von Diensten von entscheidender Bedeutung ist. Zusätzlich ist er ressourcensparend, aufgrund des schlanken Aufbaus der Linux-Systeme beziehungsweise der Docker-Container.

4 Aktueller Stand

In diesem Kapitel wird die Verwendung von Docker als Container-Technologie und die Bedeutung des HA-Proxys in komplexen IT-Infrastrukturen eingehend behandelt. In diesem Zusammenhang werden zunächst die allgemeinen Anforderungen einer docker-orientierten Umgebung erörtert. Im Anschluss werden die bei Continue betroffenen Stakeholder erwähnt, bevor es um die Beschreibung und Erörterung der aktuellen Docker-Umgebung geht.

4.1 Anforderungen an die Umgebung

Das folgende Teilkapitel untersucht die präzisen Anforderungen, die erfüllt sein müssen, um einen **beliebigen** Dienst in einer docker-orientierten Umgebung effektiv zu implementieren. Diese Anforderungen sind von wesentlicher Bedeutung, um die Leistung, Sicherheit und Verfügbarkeit der gehosteten Anwendungen auf höchstem Niveau zu gewährleisten. Die folgenden Aspekte sind hierfür erforderlich:

- **Sicherheit:** Umfasst den Schutz vor unbefugtem Zugriff, die Verschlüsselung von Datenübertragungen (HTTPS) und die Implementierung von Sicherheitsmaßnahmen vor einem Distributed Denial of Service (DDoS)-Angriff[1]. Des Weiteren sollte die Anforderung *Sicherheit* auch die Möglichkeit regelmäßiger Updates ermöglichen.
- **Konfigurierbarkeit:** Die Docker-orientierte Umgebung sollte die Möglichkeit bieten verschiedene Parameter und Einstellungen zu konfigurieren, damit eine flexible Anpassung an die Befürdnisse der Anwendungen ermöglicht werden kann.
- **Monitoring**[2]: Die Docker-Umgebung sollte eine Monitoring-Möglichkeit anbieten können.
- **Wartbarkeit:** Die Docker-Infrastruktur sollte so gestaltet sein, dass Wartungsarbeiten effizient durchgeführt werden können, einschließlich Aktualisierungen, Patches und Erweiterungen.

[1] DDoS-Angriffe sind Cyberangriffe, bei denen versucht wird eine Website durch Überflutung von Anfragen zu einer Überlastung zu bringen.

[2] Ein Monitoring umfasst die Möglichkeit zu einer schnellen Fehlerbehebung, da dort Warnmeldungen, Fehler und Statusmeldungen eingesehen werden können.

- **Kosten:** Die Kostenoptimierung ist von großer Bedeutung, insbesondere in großen Umgebungen. Hierbei geht es darum, Ressourcen effizient zu nutzen um die Gesamtkosten zu minimieren, ohne die Leistung und Sicherheit zu beeinträchtigen.

- **Dokumentation:** Eine ausführliche Dokumentation ist unerlässlich, um Entwicklern und Administratoren die Arbeit in der Docker-Infrastruktur zu erleichtern. Sie sollte klare Anleitungen zu der Konfiguration, Nutzung und Wartung bieten.

- **Self-Hosting:** Die Umgebung sollte auch self-gehostet sein. Einerseits können so langfristig Kosten gespart werden, andererseits ist ein Unternehmen mit seiner self-gehosteten Umgebung nicht von einem Drittanbieter, wie z.B. Azure abhängig. Diese Abhängigkeit sorgt in der Regel dafür, dass die Daten auf den Servern der Drittanbietern, anstatt auf den Servern der Unternehmen gespeichert werden. Aus Datenschutzgründen ist es daher auch sinnvoll, die Umgebung self-gehostet zu betreiben.

4.1.1 Stakeholder

Momentan arbeitet die Firma Continue Software GmbH mit dem HA-Proxy und seinen definierten Services. Im Folgenden werden die Stakeholder des HA-Proxys identifiziert und ihre jeweiligen Interessen und Verantwortlichkeiten innerhalb des Unternehmens erörtert, die für die Erfüllung der Anforderungen notwendig sind:

1. **DevOps-Engineer:** Der DevOps-Engineer trägt die Verantwortung für die Planung und Implementierung der Servicekonfigurationen. Seine Hauptaufgabe besteht darin, zu gewährleisten, dass die Konfigurationen den geschäftlichen Anforderungen entsprechen. Er ist folglich für die Anforderungen *Konfigurationen* und *Wartbarkeit* zuständig.

2. **Administratoren:** Die Administratoren sind für die Sicherheit, Überwachung und Netzwerk verantwortlich. Sie treffen Vorkehrungen zum Schutz vor DDoS-Angriffen, setzen Firewall-Einstellungen fest und sorgen für die Einhaltung von Sicherheitsrichtlinien. Sie sind dafür verantwortlich, dass die Anforderung *Sicherheit* erfüllt ist.

3. **Geschäftsführung:** Die Geschäftsführung hat ein Interesse daran, dass die Implementierung kosteneffizient ist und einen positiven Beitrag zu der Unternehmensleistung hat oder die Geschäftsziele voran bringt. Sie sind entsprechend daran interessiert die Anforderung der *Kosten* möglichst gering zu halten.

4. **Endbenutzer:** Die Endbenutzer sind die Nutzer der Anwendungen, in diesem Fall des HA-Proxys. Sie erwarten einen reibungslosen und benutzerfreundlichen Dienst. Sie erwarten entsprechend, dass die Anforderung *Sicherheit* erfüllt ist.

Bei Continue sind beispielsweise die Benutzer des vom Unternehmen eigens entwickelten Shops die Endbenutzer eines Services des HA-Proxys und seiner konfigurierten Dienste. Sie haben keinen Einfluss auf den Prozess, könnten allerdings ohne den HA-Proxy auch nicht auf den Shop reibungslos zugreifen.

4.2 Aufbau der Infrastrukur

In der gegenwärtigen Konfiguration ist ein GitLab-Projekt eingerichtet, das sämtliche Dienste von der Continue Software GmbH enthält. Ein HA-Proxy fungiert als der primäre Zugangspunkt zu diesem GitLab-Projekt. Der Zugriff auf dieses Projekt erfolgt über ein spezielles Access-Token. Dieses Token dient der Authentifizierung und Autorisierung, um sicherzustellen, dass nur berechtige Benutzer und Systeme auf das Git-Repository (Git-Repo) zugreifen können. Darüber hinaus ist es ausschließlich innerhalb des firmeninternen Netzwerks erreichbar, um vor unbefugten Zugriffen von außen zu schützen.

Alle notwendingen Informationen, die der HA-Proxy für seine Funktionialität benötigt, sind in einer JSON-Datei mit dem Namen „cloudfront.json" enthalten. Diese Datei enthält eine Vielzahl von Konfigurationsparametern, darunter das Matching von Domains und Diensten, wodurch der HA-Proxy Anfragen an die entsprechenden Dienste oder Server weiterleiten kann, die mit diesen Domänen verknüpft sind. Damit kann sichergestellt werden, dass das System reibungslos läuft und ordnungsgemäß funktioniert.

Über die „cloudfront.json" ist es für den HA-Proxy möglich zu wissen an welchen Dienst er die Anfrage für eine definierte Domain weiterleiten muss. Des Weiteren wird für die bisherige Struktur eine weitere Datei benötigt, die eine .yml - Dateiendung hat. Diese Datei ist eine Docker-Stack Datei, die von einem weiteren Tool verwaltet wird, damit die Dienste hochgefahren werden können (siehe Kap. 8.1.1).

4.3 Vor- und Nachteile der bisherigen Nutzung

In dem folgenden Kapitel wird sich einer detaillierten Analyse der bisherigen Struktur gewidmet, welche die Nutzung eines HA-Proxys einschließt. Es ist von Bedeutung, die Vor- und Nachteile der bisherigen Struktur zu erörtern, um die Notwendigkeit einer neuen Struktur zu beurteilen.

4.3.1 Vorteile

Ein großer und zudem auch wichtiger Vorteil ist die mögliche Verwendung eines Access Tokens, welches ein hohes Maß an Sicherheit bietet, da es den Zugriff auf das Git-Repo beschränkt.
Die Verwendung einer zentralen Konfigurationsdatei ermöglicht eine effiziente Verwaltung aller erforderlichen Informationen, welche für die Funktionalität des HA-Proxys notwendig sind.

Ein weiterer Vorteil der derzeitigen Verwendung ist, dass die Umgebung zur Zeit self-gehostet ist. So können einerseits langfristig Kosten gespart werden, andererseits liegt der gesamte Datenverkehr auf den Servern von Continue und nicht auf denen einer externen Firma.

4.3.2 Nachteile

Ein großer Nachteil der derzeitigen Struktur ist die Komplexität. Sie entsteht, wenn viele Dienste, Domains und Konfigurationselemente in der „cloudfront.json"-Datei enthalten sind. Folglich ist die allgemeine Wartbarkeit und Fehlerbehebung erschwert. Durch die Komplexität ist auch ein Einstieg in das Gebiet des IT-Infrastrukturmanagements eine anspruchsvolle Aufgabe.

Ein weiterer Nachteil der derzeitigen HA-Proxy-Container Struktur ist, dass momentan Ubuntu 18.04 als Basis für den HA-Proxy-Container benutzt wird. In dieser Version gibt es das offizielle Certbot-Repository nicht mehr, weshalb eine ältere Version in den Container installiert wird [2]. Eine alternative Herangehensweise wäre die Nutzung der Snap-Installation, wobei dazu eine Aktualisierung auf Ubuntu 20.04 erforderlich wäre. Es besteht die Möglichkeit, dass während des Versionswechsels Komplikationen im Zusammenhang mit dem HA-Proxy auftreten könnten. Daher wurde eine solche Umstellung bislang nicht in Betracht gezogen. Außerdem verhinderten zusätzlich ein anderes Projekt und ein firmeninterner Umzug den frühzeitigen Versionswechsel.

Darüber hinaus ist es mithilfe einer alternativen Lösung eines HA-Proxys möglich, Dienste zu dekommissionieren[3]. Durch eine vereinfachte Konfiguration könnten sowohl der Proxy in seiner gegenwärtigen Form als auch der Zertifikatsspeicher von Continue ersetzt werden. Die bestehen-

[3]Dekommissionierung beschreibt in diesem Kontext nicht mehr benötigte IT-Systeme, die stillgelegt werden können.

de Struktur weist entsprechenden Overload auf, der vermieden werden sollte.

Ein weiterer Nachteil ist der bisherige Aufbau der Dienste von Continue. Momentan ist ein Dienst für die Erstellung der HTTPS-Zertifikate zuständig, indem er auf die CertStore-API zugreift. Alle anderen Dienste, die Zertifikate benötigen, sind von diesem Dienst abhängig. Bei einem Ausfall des für die Zertfikate zuständigen Dienstes können entsprechend alle anderen Dienste kein Zertifikat erstellen oder ausgeben, was ein großes Sicherheitsrisiko darstellt, da in diesem Fall keine HTTPS-Verbindung aufgebaut werden kann.

4.3.3 Zusammenfassung

Zusammenfassend lässt sich zunächst sagen, dass es mehr Nachteile als Vorteile gibt. Der größte Vorteil ist, dass die derzeitige Umgebung sowohl self-gehostet ist als auch ein weiteres hohes Maß an Sicherheit bietet. Der größte Nachteil ist die Komplexität bei vielen Diensten, Domains und Konfigurationselementen, wodurch die Einarbeitung für Mitarbeitende deutlich erschwert wird. Auch der Nachteil mit der Abhängigkeit von einem Dienst ist gravierend, da der Ausfall dieses Dienstes dafür sorgen kann, dass alle anderen Dienste kein HTTPS-Zertifikat erstellen können. Insgesamt lässt sich daher auch sagen, dass die Nachteile mehr Gewicht als die Vorteile haben, was die Notwendigkeit einer Alternative hervorhebt.

4.4 Alternativen

In den vorausgegangenen Abschnitten dieser Seminararbeit wurden bereits die gegenwärtige Docker-Umgebung sowie die spezifischen Anforderungen an die erforderliche Infrastruktur eingehend untersucht und analysiert. Des Weiteren erfolgte eine kritische Betrachtung der Vor- und Nachteile der bislang verwendeten Herangehensweise. In diesem Kontext konnte identifiziert werden, dass es einige nachteilige Aspekte gibt, deren Vermeidung als Möglichkeit in Betracht gezogen werden kann. Das vorliegende Kapitel widmet sich demnach der kurzen Evaluierung verschiedener Alternativen.

4.4.1 Traefik

Eine Alternative, die a priori von Continue in Betracht gezogen wurde, ist *Traefik Community*. *Traefik Community* ist ein Open-Source Tool, das alle der bereits erwähnten Anforderungen erfüllt. Das Tool ist in der Lage bisherige Dienste abzulösen und die Arbeit zu vereinfachen. Neben *Traefik Community* gibt es auch noch *Traefik Enterprise*, eine kostenpflichtige Version von der Community-Version. Da Continue auf eine kostenlose Open-Source-Software setzt, wird in dieser Arbeit lediglich auf *Traefik-Community*, kurz *Traefik* eingegangen (siehe Kap. 5)[21].

Alternativ zu *Traefik* gibt es noch weitere Tools, welche die meisten Anforderungen einer Docker-orientierten Umgebung erfüllen. Dazu zählen beispielsweise F5 NGINX, Azure, AWS Elastic Load Balancing (ELB) und Caddy.

Azure und AWS ELB sind kostenintensive Alternativen. F5 NGINX ist zwar wie *Traefik* als kostenlose Open-Source Variante möglich, allerdings ist hier eine deutlich komplexere Konfiguration nötig, um Dienste laufen lassen zu können. F5 NGINX bietet des Weiteren keine automatische Zertifikatsverwaltung wie *Traefik* an. Dazu kommt auch noch, dass die gerade genannten drei Dienste den Datenverkehr auf ihren eigenen Servern speichern anstatt auf den Servern von Continue. Daher wurden diese drei Dienste nicht weiter in Betracht gezogen.

Caddy hat große Ähnlichkeiten mit *Traefik*. Außerdem zeichnet sich Caddy durch seine Benutzerfreundlichkeit und einfache Konfiguration aus. Es bietet auch standardmäßig HTTPS mit automatischer Zertifikatserstellung über „Let's Encrypt". Des Weiteren ist Caddy genauso wie *Traefik* auch ein kostenloses Open-Source Tool. Allerdings eignet sich Caddy nicht für größere Projekte, daher ist es nachvollziehbar, dass Continue *Traefik* als Alternative zu der derzeitigen HA-Proxy-Struktur bevorzugt.

5 Traefik

In Anlehnung an die vorherige Angabe, konzentriert sich diese Arbeit nur auf das Open-Source-Reverse-Proxy-Tool namens *Traefik*, das für die Verwaltung des Datenverkehrs in Anwendungen entwickelt wurde. *Traefik* ist ein unter DevOps-, Web- und Cloud-Engineering-Teams beliebtes Open-Source-Tool. Daher umfasst *Traefik* auch einige Funktionen, die für die Untersuchung und Konfiguration nicht relevant sind. Diese Funktionen werden im Rahmen dieser Seminararbeit nicht näher erläutert [6, 8, 21].

5.1 Funktionsweise Traefik

Traefiks grundlegende Funktionsweise umfasst folgende Schritte [21, 17, 9, 13]:

Schritt 1: Verbindungseingang

Jede ankommende Anfrage, die von einem Client kommt, wird an *Traefik* gerichtet.

Schritt 2: Routen- und Regelabgleich

Traefik analysiert die eingehende Anfrage und wendet Routing-Regeln und teilweise selbst einstellbare (einfache) Konfigurationen auf sie an.

Schritt 3: Service Discovery

Traefik kann sowohl in Dockern als auch in Kubernetes integriert werden. Ersteres ist für die Ersetzung des bisherigen HA-Proxys relevant. Durch das Service Discovery ist auch das Dienst-Domain-Matching möglich.

Schritt 4: Routing und Proxying

Traefik ist durch die Routing-Regeln in der Lage die aus Schritt 1 eingehende Anfrage an den richtigen Dienst weiterzuleiten.

Schritt 5: SSL/TLS Unterstützung

Traefik unterstützt SSL/TLS-Verschlüsselung (siehe Kap. 5.3.1), um verschlüsselte Verbindungen zwischen dem Client und *Traefik*, sowie zwischen *Traefik* und internen Diensten herzustellen.

Schritt 6: Monitoring

Traefik bietet ein webbasiertes Monitoring-Dashboard an, auf dem Status und Konfiguration der aktuell laufenden Dienste überwacht und verwaltet werden können.

5.1.1 Statische Konfiguration

Im zweiten Schritt des Kapitels 5.1 wurde bereits auf die Möglichkeit hingewiesen, Konfigurationen in *Traefik* anzuwenden, wobei zwei verschiedene Arten existieren: die statische und die dynamische Konfiguration. Bei der statischen Konfiguration von *Traefik* können spezifische Regeln und Routen für gewünschte Anwendungszwecke festgelegt werden. Neben der Definition und Anpassung von Routing-Regeln können auch über „Labels", die ein Konfigurationsparameter bei *Traefik* sind, Einstellungen einer dynamischen Konfiguration in eine statische Konfiguration geschrieben werden. Unter anderem ist es dadurch möglich SSL/TLS Zertifikate und Middlewares zu erstellen. Die statische Konfiguration ist für spezifische Anforderungen gedacht, die eine individuelle Konfiguration erfordern und nicht durch die automatische Dienstentdeckung (Kubernetes, Docker) oder Routing-Regeln abgedeckt werden können [10].

5.1.2 Dynamische Konfiguration

Die dynamische Konfiguration von *Traefik* enthält alles, was benötigt wird, um den Proxy Server in Echtzeit anzupassen. Die Echtzeit Anpassung läuft ohne Verzögerung des Dienstes, da ein sogenannter „Hot-reload"[1] durchgeführt wird. Die dynamische Konfiguration beinhaltet alle Informationen, die von den Service-Providern bereitgestellt werden (Routers, Services, Middlewares, Zertifikate) [10].

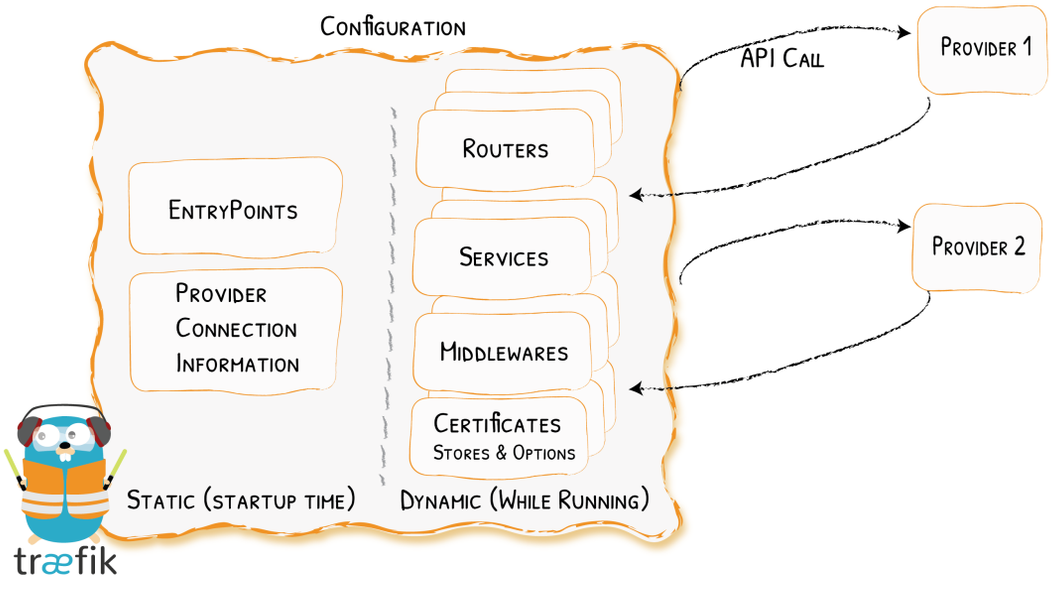

Abbildung 5.1: Dynamische vs. Statische Konfiguration [20].

[1] „Hot-reload" ist ein Prozess bei dem Änderungen in Echtzeit angewendet werden, ohne die Anwendung neu starten zu müssen

5.1.3 Middlewares

Middlewares[2] sind in der Lage Anforderungen zu optimieren, bevor sie an einen Service gesendet werden. Es gibt verschiedene verfügbare Middlewares in *Traefik*, einige davon können eine Anfrage ändern. Mögliche Änderungen sind Umleitungen und das Hinzufügen von Authentifizierungen. Über die „Labels" können entsprechende Middlewares in einer .yml-Datei definiert werden (siehe Kap. 8.1.3)[15].

5.2 Routing- und Loadbalancing

In *Traefik* ist das Routing- und Loadbalancing von entscheidender Bedeutung, um den eingehenden Datenverkehr effizient und zuverlässig an verschiedene Dienste und Anwendungen weiterzuleiten. Dieser Abschnitt führt in die Konzepte von EntryPoints, Routers und Services ein, die eine zentrale Rolle in der Konfiguration und Steuerung des Datenverkehrs spielen.

5.2.1 EntryPoints

In *Traefik* sind EntryPoints Konfigurationselemente, die verschiedene Eingangstore für unterschiedliche Protokolle wie HTTP, HTTPS und mehr definieren. Diese EntryPoints ermöglichen es, den Datenverkehr basierend auf einer Vielzahl von Kriterien wie Hostnamen, Ports und Protokollen effizient an verschiedene Dienste und Anwendungen zu lenken [12].

5.2.2 Routers

Routers sind bei *Traefik* Konfigurationselemente, die den Datenverkehr basierend auf bestimmten Regeln und Kriterien an verschiedene Dienste weiterleiten. Routers definieren Router-Regeln, welche basierend auf Daten wie Hostname, Pfad oder Header entscheiden, wie der Datenverkehr behandelt werden soll. Routers können auch EntryPoints zugeordnet werden, um festzulegen, über welche Ports und Protokolle der Datenverkehr an diesen Router gelangen kann [16].

5.2.3 Services

Die Konfiguration der Services bei *Traefik* ist von entscheidender Bedeutung, um sicherzustellen, dass der Datenverkehr präzise an die entsprechenden Backend-Dienste weitergeleitet wird. Services bilden die Verbindung zwischen *Traefik* und den tatsächlichen Anwendungen, welche die Anfragen bearbeiten [18].

[2]Middlewares werden meist in der Mitte einer Softwarearchitektur platziert, um bestimmte Aufgaben oder Funktionen zu steuern.

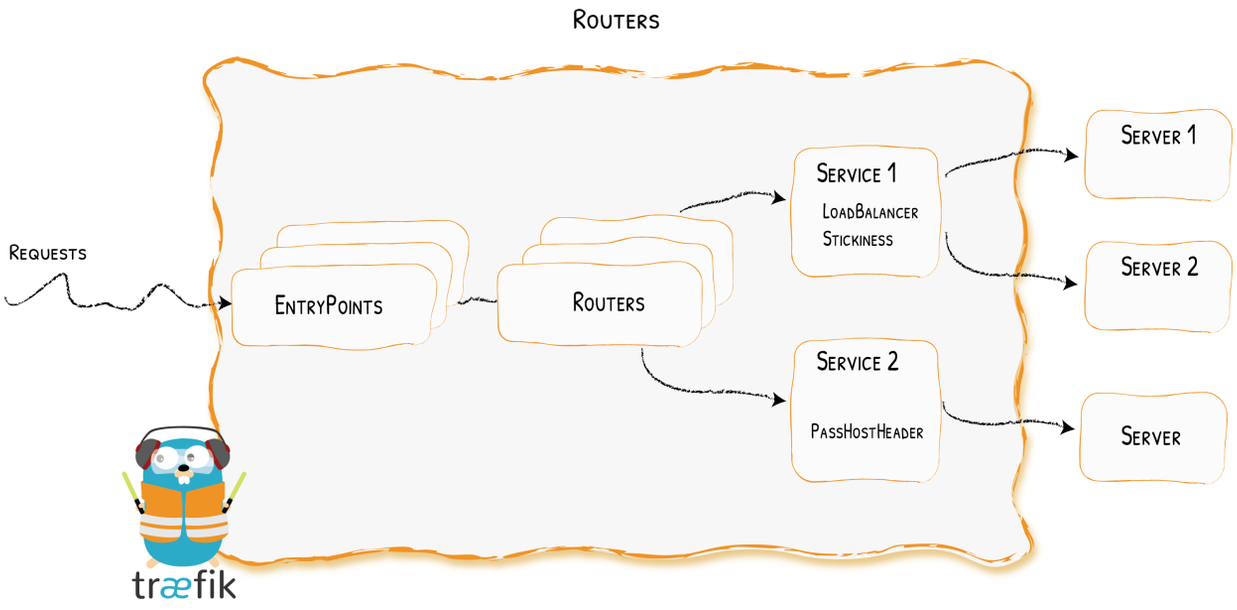

Abbildung 5.2: Grafische Darstellung des Ablaufes von Request, Entrypoint, Routers und Services [19].

5.3 HTTPS

Traefik unterstützt die Verwendung des Hypertext Transfer Protocol Secure (HTTPS), wodurch Daten zwischen Browser und Website verschlüsselt übertragen werden können. Zur Einrichtung von HTTPS gemäß dem Hypertext Transfer Protocol ist die Zertifizierung bei vertrauenswürdigen Zertifizierungsstellen erforderlich, um die Authentizität einer Identität nachzuweisen. Die Verwendung von SSL/TLS ist bei der Implementierung von HTTPS in Traefik unerlässlich.

5.3.1 SSL/TLS-Unterstützung

Secure Sockets Layer (SSL) werden unter anderem auch als digitale Zertifikate bezeichnet und werden häufig zu der verschlüsselten Verbindung zwischen einem Browser und einem Computer verwendet. Transport Layer Security (TLS) ist eine aktualisierte und sichere Version von SSL und wird heutzutage standardmäßig an Stelle von SSL benutzt [1].

5.3.2 Traefik-Proxy mit HTTPS

Je nach Version von *Traefik* gibt es verschiedene Möglichkeiten eine HTTPS-Verbindung aufzubauen. Konkret geht es hier um die Verbindungsmöglichkeiten für *Traefik 2.10*. Die Version 3.0 ist seit Oktober 2023 ein Release Candidate, daher ist davon auszugehen, dass in absehbarer Zeit Version 2.10 durch die neuste Version 3.0 vollständig ersetzt wird. Die Version 3.0 hat

momentan noch Schwierigkeiten eine HTTPS-Verbindung mit *Traefik* aufzubauen, daher wird im Folgenden etwas über die Version 2.10 beschrieben. [13, 4, 14].

Folgende Schritte sind bei der Version 2.10 notwendig:

- Nutzung des „certificatesresolvers" in dem command Bereich einer noch zu erstellenden .yml-Datei:
  - certificatesresolvers.myresolver.acme.email=test@email.de
  - certificatesresolvers.myresolver.acme.storage=/acme.json
  - certificatesresolvers.myresolver.acme.httpchallenge=true
  - certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web
- Speicherung der Zertifikate in einem Volume, damit nicht bei jedem Ausführen ein neuer Account bei „Let's Encrypt" erstellt wird.
- Nutzung von TLS und Middlewares in dem Labels-Bereich
  - traefik.http.routers.api.tls.certresolver=myresolver
  - traefik.http.middlewares.auth.basicauth.users=username:password
- Ausführung der Datei mit:

```
#Falls der Dateiname nicht docker-compose.yml oder einen ähnlichen standardmäßig
#unterstützten Dateinamen hat:
docker-compose -f <Dateiname>.yml up -d

#sonst:
docker-compose docker-compose.yml -d
```

Hierbei sind „test@email.de" durch die eigene Email-Adresse, „username" durch einen eigenen Benutzernamen und „password" durch ein eigenes Passwort zu ersetzen. Ein vollständiges Beispiel für eine einfache HTTPS-Konfiguration befindet sich in Kapitel 9.1.4. Bei erfolgreicher Konfiguration in der Version 2.10 wird automatisch in dem Dashboard die Aktivierung von TLS unter dem Reiter „Routers" mit einem grünen Symbol kenntlich gemacht.

5.4 Vor- und Nachteile der Nutzung von Traefik

Die Nutzung von *Traefik* bietet eine Vielzahl von Möglichkeiten für die Verwaltung von Anwendungen. In diesem Kapitel werden nun die Vor- und Nachteile von *Traefik* beleuchtet.

5.4.1 Vorteile

Traefik ist ein beliebtes Tool, welches viele Vorteile hat. Durch die in *Traefik* eingebaute automatische Diensterkennung ist das Tool auch in mehreren Umgebungen (Docker, Kubernetes) integrationstauglich.

Ein weiterer positiver Aspekt von *Traefik* ist, dass jede individuelle Anforderung durch die statische oder dynamische Konfiguration eingestellt werden kann. Außerdem ist bei *Traefik* ein automatisches Monitoring eingebaut, sofern es in der Konfiguration aktiviert wurde. Dadurch ist es möglich den Status der Routers, Services und Middlewares einzusehen. Dazu muss nur das Dashboard aufgerufen werden.

Ein weiterer Vorteil von *Traefik* ist, dass es ein kostenloses Open-Source-Tool ist. Die Verwendung von Open-Source Software hat bei *Traefik* folgende positive Aspekte:

- **Community-Unterstützung:** Die Community von *Traefik* trägt dazu bei, die langfristige Wartung und Weiterentwicklung voran zu bringen. Auch potentielle Sicherheitslücken können von der Community entdeckt und behoben werden.
- **Transparenz:** Es ist möglich den Quellcode von *Traefik* einzusehen, was insbesondere bei Sicherheitsbedenken hilfreich sein kann.

Ein zusätzlicher Vorteil ist, dass die offizielle Dokumentationsseite von *Traefik* eine detaillierte Einführung bietet. Es ist jedoch wichtig zu beachten, dass einige Beispiele in der Dokumentation nur in Bruchstücken vorliegen, was es gelegentlich erschwert das gesamte Beispiel vollständig nachzuvollziehen und selbst zu erstellen. Da *Traefik* ein Reverse-Proxy benutzt, ist auch zusätzlich Sicherheit geboten.

Des Weiteren hat die Nutzung von *Traefik* den Vorteil, dass sowohl der HA-Proxy als auch der bisherige CertStore-Service nicht mehr benötigt werden. Durch *Traefik* ist es daher möglich zwei Dienste zu dekommissionieren. Aufgrund der Abschaffung des Certstore-Services wird auch die Abhängigkeit des Services mit allen anderen Diensten aufgelöst.

5.4.2 Nachteile

Neben den erwähnten Vorteilen hat *Traefik* auch einige Nachteile. Ein Nachteil ist, dass das Tool in gewissen Aspekten abhängig von Drittanbietern ist. Bei der Zertifikatsverwaltung ist dies „Let‘s Encrypt”. Neben „Let‘s Encrypt” ist es auch noch möglich andere Zertifikatsherstel-ler zu benutzen. Continue hat sich für „Let‘s Encrypt” entschieden, da es ein kostenloses und bekanntes Tool ist.

„Let‘s Encrypt” hat den Nachteil, dass nur zehn Accounts pro IP-Adresse alle drei Stunden registriert werden können [5]. Wenn kein Volume zum Speichern der Zertifikate angelegt wurde, dann sollte zumindest eine Testumgebung aktiviert werden, um mehrere Accounts innerhalb von drei Stunden registrieren lassen zu können. Anderenfalls kann es zu einer IP-Sperrung kommen. Drei weitere Nachteile können jedoch bei der Konfiguration entstehen:

1. Komplexität bei erweiterten Konfigurationen (z.B. bei Header-Manipulationen)
2. Sicherheitsrisiko bei falscher Konfiguration (z.B. Zugang zu Ports versehentlich freigege-ben, HTTPS fehlerhaft konfiguriert). Dieses Sicherheitsrisiko besteht auch bei unserem derzeitigen HA-Proxy und seinen Konfigurationen.
3. Geringe Performance bei falscher und komplexer Konfiguration (z.B: Router-Regeln bein-halten unnötig viele Überprüfungen)

Ein letzter Nachteil ist, wie im vorherigen Teilkapitel bereits angedeutet, die Dokumentation. Die Dokumentation hat teilweise .yml Code-Beispiele, bei denen nicht klar wird, zu welcher .yml-Datei sie gehören (siehe Kapitel 8.1.5). Genau wie in dem Beispiel sah die Dokumentation auch in der *Traefik* Dokumentation aus. Hierbei wird nicht deutlich, dass der untere Ausschnitt zu der Docker-Stack und der obere Abschnitt zu der *Traefik* Konfigurationsdatei gehört [11].

5.4.3 Zusammenfassung

Zusammenfassend lässt sich feststellen, dass *Traefik* als Open-Source Reverse-Proxy-Tool mit kostenloser Verfügbarkeit, automatischer Diensterkennung, flexibler Konfiguration und integrier-tem Monitoring zahlreiche Vorteile bietet. Trotz Abhängigkeiten von Drittanbietern überwiegen diese Vorzüge und machen *Traefik* im Vergleich zum aktuellen HA-Proxy zu einer bevorzugten Lösung, obwohl die Konfiguration aufgrund der Dokumentation komplex sein kann.

6 Das Ersetzen des HA-Proxys durch Traefik

Im Rahmen der praktischen Arbeit musste überlegt werden, wie der bisherige HA-Proxy mit *Traefik* ersetzt werden kann. Dazu war es zunächst notwendig sich in die Dokumentation von *Traefik* einzulesen. Danach konnte mit der praktischen Arbeit begonnen werden, wie in den nächsten Kapiteln näher erläutert wird.

6.1 Traefik Konfiguration

Zu Beginn der praktischen Arbeit wurde damit begonnen simple Konfigurationen von *Traefik* anzuwenden. Ein Beispiel dazu findet sich in Kapitel 8.1.2. Anschließend wurde die Konfiguration erweitert, um die Implementierung zusätzlicher Funktionalitäten zu ermöglichen. Unter anderem wurde auch *Traefik* mit HTTPS eingebunden, damit eine sichere Verbindung hergestellt werden konnte. Schließlich wurde die Konfiguration erweitert und umgeschrieben, sodass es möglich war ein Service des HA-Proxys auf *Traefik* umzuschreiben.

6.2 Schwierigkeiten

Bei dem Wechsel des HA-Proxys zu *Traefik* kam es zu einigen Schwierigkeiten. Eine Schwierigkeit war die Abhängigkeit *Traefiks* von Drittanbietern, in diesem Fall „Let‘s Encrypt”. Das Testen von *Traefik*-Konfigurationen bei einem fehlendem Volume und einer nicht existierenden Testumgebung sorgte für eine Überschreitung der maximal zulässigen Konto-Erstellungen in einer Zeitspanne von drei Stunden und der daraus resultierenden Sperrung der IP-Adresse für sieben Tage [5]. Nach diesem Vorfall wurde nach Möglichkeiten gesucht diese Problematik zu umgehen und es wurde ein Volume für die Zertifikate in die .yml-Datei eingefügt. Des Weiteren musste der Port 4711 in der Firewall freigegeben werden, damit dieser benutzt werden konnte.

7 Abschließende Betrachtungen zu der Traefik-Konfiguration

In diesem Kapitel wird sich einer Abschließenden Betrachtung zu der *Traefik*-Konfiguration beschäftigt. Dazu werden zunächst die Hauptergebnisse der Untersuchung von *Traefik* wiederholt dargestellt. Anschließend wird ein Ausblick gegeben.

7.1 Vergleich zwischen Traefik und dem bisherigen Zustand

Im Verlauf dieser Arbeit wurde sich mit der bisherigen HA-Proxy-Docker-Struktur beschäftigt. Dabei wurden die Vor- und Nachteile dieses Systems beleuchtet und festgestellt, dass eine alternative Lösung sinnvoll ist. Es wurde sich mit möglichen Alternativen beschäftigt und es hat sich herauskristallisiert, dass *Traefik* eine gute Alternative zu der bisher benutzten HA-Proxy Konfiguration ist.

Im direkten Vergleich zeichnet sich *Traefik* insbesondere in der Konfiguration aus. Sie ist deutlich simpler gehalten und benötigt nicht mehrere JSON Dateien, wie es bei der aktuellen HA-Proxy-Struktur der Fall ist. Des Weiteren ist es bei *Traefik* auch einfacher eine HTTPS-Verbindung aufzubauen, was ein weiterer Vorteil im Vergleich zum bisherigen Zustand ist.

Außerdem zeichnet sich bei der bisher laufenden HA-Proxy Umgebung eine starke Abhängigkeit ab, die vermieden werden sollte. In diesem Kontext liegt die Verantwortung bei einem bestimmten Dienst, auf die CertStore-API zuzugreifen, um eine sichere HTTPS-Verbindung herstellen zu können. Alle anderen Dienste initiieren den Zugriff auf diese API durch die Anforderung an den genannten Dienst. Bei *Traefik* könnte ein dezentraler CertStore Dienst konfiguriert werden, der dafür verantwortlich ist eine HTTPS-Verbindung aufzubauen. Dadurch könnte jeder einzelne Service eine HTTPS-Verbindung aufbauen, ohne von einem einzigen Dienst abhängig zu sein.

Die Konfiguration von *Traefik* gestaltet sich jedoch etwas schwieriger als bei der HA-Proxy-Struktur, da die Dokumentation von *Traefik* zwar groß und ausführlich ist, aber vereinzelt die Beispiele zur Konfiguration eines Dienstes nur in Bruchstücken vorliegen. Aus diesem Grund war es insbesondere zu Beginn schwierig ein gesamtes Beispiel nachzuvollziehen und eine Konfiguration anlegen zu können.

Sowohl *Traefik* als auch der bisherige HA-Proxy weisen im Allgemeinen eine hohe Sicherheit auf, da beide self-gehostet und Reverse Proxies sind. Durch die Nutzung des von außen nicht erreichbaren Git-Repos, ist außerdem die Sicherheit verstärkt.

Insgesamt lässt sich daher sagen, dass *Traefik* eine gute Alternative zu der bisherigen HA-Proxy-Struktur ist, obwohl die Konfiguration komplex sein kann. Das *Traefik* eine gute Alternative ist, liegt vor allem daran, dass durch *Traefik* zwei Dienste dekommissioniert werden können und die schwerwiegenden Abhängigkeiten aufgelöst werden können.

7.1.1 Wartbarkeit von Traefik-Konfigurationen

Die Wartbarkeit der bisherigen HA-Proxy Umgebung wird mit zunehmender Anzahl an Diensten unübersichtlicher und komplexer. Durch *Traefik* wird die Wartbarkeit vereinfacht, da bei *Traefik* nicht für die Nutzung eines Dienstes drei verschiedene Dateien benötigt werden. Stattdessen braucht *Traefik* im besten Fall nur eine .yml-Datei, damit der Dienst läuft. Auch für Mitarbeitende, die bei Continue in die Docker-Thematik einsteigen, ist es nachvollziehbarer mit so wenig Dateien wie möglich zu arbeiten.

7.2 Ausblick

Die Untersuchung und Konfiguration von *Traefik* hat ergeben, dass es möglich ist den bisherigen HA-Proxy mit *Traefik* auszutauschen. Zum Abschluss dieser Seminararbeit wird es daher um mögliche Ausblicke gehen, die im Bezug zu der Untersuchung und Konfiguration stehen.

7.2.1 Vollständiger Wechsel auf Traefik

Das Ziel der Firma Continue Software GmbH besteht darin, den bisherigen HA-Proxy mit *Traefik* zu ersetzen. Dazu wurde im Rahmen dieser Seminararbeit exemplarisch ein Dienst des HA-Proxy mit einer *Traefik*-Konfiguration ersetzt. Langfristig ist das Ziel alle Services des jetzigen HA-Proxy auf *Traefik* umzustellen. Dazu müssten noch in etwa 70 Dienste und somit Konfigurationen in *Traefik* geschrieben werden. Aufgrund der Anzahl der Services, für die *Traefik*-Konfigurationen geschrieben werden müssen, ist es möglich, dass sich ein vollständiger Wechsel auf *Traefik* noch um Monate in die Länge ziehen wird.

7.2.2 Erklärung von Traefik und Dockern

Ein weiterer Ausblick besteht darin, die Mitarbeitenden des Unternehmens über den bevorstehenden Wechsel auf *Traefik* zu informieren. Wie in Kap 4.3.1 bereits erwähnt, betrifft das die folgenden Stakeholder: DevOps-Engineers, Administratoren, die Geschäftsführung und die Endbenutzer. Zum Zweck der Weiterbildung ist auch eine eine Präsentation geplant, die den Stakeholdern einen Einblick in *Traefik* und Docker geben soll. Damit sollen langfristig verschiedene Beschäftigte des Unternehmens die Möglichkeit erlernen Konfigurationen selbst anlegen und bearbeiten zu können.

Abbildungsverzeichnis

Quellenverzeichnis

[1] *AWS: What is a SSL/TLS Certificate*. https://aws.amazon.com/de/what-is/ssl-certificate/. Stand 19.10.2023.

[2] *Certbot: What is that?* https://certbot.eff.org/pages/about. Stand 23.10.2023.

[3] *Cloudflare: Reverse Proxy Erklärung*. https://www.cloudflare.com/de-de/learning/cdn/glossary/reverse-proxy/. Stand 19.10.2023.

[4] *Dockerswarm: Example of Traefik Proxy with HTTPS Version 2.0*. https://dockerswarm.rocks/traefik/. Stand 30.10.2023.

[5] *Let's Encrypt: Rate-Limits*. https://letsencrypt.org/docs/rate-limits/. Stand 19.10.2023.

[6] *Linux-Magazin: Traefik Report*. https://www.linux-magazin.de/ausgaben/2021/05/traefik/. Stand 03.11.2023.

[7] *Reverse Proxy: Picture*. https://lucid.co/. Erstellt am 13.11.2023 mit Hilfe von Lucidchart.

[8] *SWNetwork: Was ist Traefik?* https://www.swnetwork.de/de/software/traefik/. Stand 19.10.2023.

[9] *Traefik: Configuration Discovery*. https://doc.traefik.io/traefik/providers/overview/. Stand 19.10.2023.

[10] *Traefik: Configurations*. https://doc.traefik.io/traefik/getting-started/configuration-overview/. Stand 19.10.2023.

[11] *Traefik: Docker*. https://doc.traefik.io/traefik/providers/docker. Stand 06.11.2023.

[12] *Traefik: EntryPoints*. https://doc.traefik.io/traefik/routing/entrypoints/. Stand 19.10.2023.

[13] *Traefik: HTTPS und TLS*. https://doc.traefik.io/traefik/https/overview/. Stand 31.10.2023.

[14] *Traefik: Let's Encrypt*. https://doc.traefik.io/traefik/https/acme/. Stand 19.10.2023.

[15] *Traefik: Middlewares*. https://doc.traefik.io/traefik/middlewares/overview/. Stand 19.10.2023.

[16] *Traefik: Routers*. https://doc.traefik.io/traefik/routing/routers/. Stand 22.10.2023.

[17] *Traefik: Routing und Load Balancing*. https://doc.traefik.io/traefik/routing/overview/. Stand 19.10.2023.

[18] *Traefik: Services*. https://doc.traefik.io/traefik/routing/services/. Stand 16.11.2023.

[19] *Traefik: Services Picture*. https://doc.traefik.io/traefik/assets/img/services.png. Stand 19.10.2023.

[20] *Traefik: Static vs. Dynamic Configuration*. https://doc.traefik.io/traefik/assets/img/static-dynamic-configuration.png. Stand 17.11.2023.

[21] *Traefik: Was ist Traefik?* https://doc.traefik.io/traefik/. Stand 19.10.2023.

[22] *Wikipedia: Reverse Proxy Erklärung*. https://de.wikipedia.org/wiki/Reverse_Proxy. Stand 19.10.2023.

8 Anhang

8.1 Beispielkonfigurationen

In den folgenden Unterkapiteln befinden sich einige Beispiele für .yml-Dateien, die im Rahmen der praktischen Arbeit dieser Seminararbeit entstanden sind. Allerdings handelt sich aus Datenschutzgründen nicht überall um echte Daten.

8.1.1 HA-Proxy Beispielkonfiguration

```
{
  "cert-store-url": "http://certStore.firmenName.de",
  "hosts": [{
      "host": "exampleDomain",
      "service": "serviceName",
      "ssl_cert_type": "manual",
      "forceSsl": true
    }, {
      "host": "exampleDomain",
      "service": "kuerzel_frontend-latest",
      "ssl_cert_type": "manual",
      "forceSsl": true
    }
  ],
  "services": [
    { "service": "kuerzel_frontend-stable" },
    { "service": "kuerzel_frontend-latest" }
  ]
}

{
  "sos": {"kuerzel":  "sos-stack.yml"}
```

```
}
```

```
#sos-stack.yml
version: "3"

services:

  worker:
    image: registry.test.de/shop/kuerzel:stable
    command: "worker run"
    environment:
        # ... #
    deploy:
      replicas: 1
    stop_signal: SIGTERM
    stop_grace_period: 10s
    networks:
      - ha-net-internal

    # ... #

networks:
  ha-net-internal:
    external: true

```

8.1.2 Simple Traefik-Konfiguration

```
#--- whoami.yml
version: '3'

services:
  whoami:
    image: traefik/whoami
    labels:
      - "traefik.http.routers.whoami.rule=Host(`exampleDomain`)"

#--- traefik.yml
```

```yaml
version: '3'

services:
  reverse-proxy:
    image: traefik:v3.0
    command:
      - --api.insecure=true
      - --providers.docker
      - --entryPoints.web.address=:80
      - --entryPoints.websecure.address=:443
    ports:
      - "80:80"
      - "443:443"
      - "4711:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

8.1.3 Erstellung von Middlewares

```yaml
# ...
labels:
  # Erstellt eine Middleware namens "beispiel" und dem Prefix /foo
  - "traefik.http.middlewares.beispiel.addprefix.prefix=/foo"
  # Anwendung der Middleware "beispiel" auf den Router "router_beispiel"
  - "traefik.http.routers.router_beispiel.middlewares=beispiel@docker"
# ...

```

8.1.4 Traefik mit HTTPS

```yaml
version: '3.3'

services:
  reverse-proxy:
    image: traefik:v2.10
    command:
      - --api.insecure=true
```

```
8         - --providers.docker
9         - --entryPoints.web.address=:80
10        - --entryPoints.websecure.address=:443
11        - --api.dashboard=true
12        - --api.debug=true
13        - --log.level=INFO
14        - --certificatesresolvers.myresolver.acme.email=mail@test.de
15        - --certificatesresolvers.myresolver.acme.storage=/certificates/acme.json
16        - --certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web
17      ports:
18        - "80:80"
19        - "443:443"
20        - "4711:8080"
21      volumes:
22        - /var/run/docker.sock:/var/run/docker.sock
23        - traefik-public-certificates:/certificates
24      labels:
25        - "traefik.http.routers.http-catchall.rule=hostregexp(`{host:.+}`)"
26        - "traefik.http.routers.http-catchall.entrypoints=web"
27        - "traefik.http.routers.http-catchall.middlewares=redirect-to-https"
28        - "traefik.http.middlewares.redirect-to-https.redirectscheme.scheme=https"
29      networks:
30        - public-net
31
32    web:
33      image: traefik/whoami
34      container_name: web
35      labels:
36        - "traefik.http.routers.web.rule=Host(`exampleDomain`)"
37        - "traefik.http.routers.web.tls=true"
38        - "traefik.http.routers.web.tls.certresolver=myresolver"
39
40      networks:
41        - public-net
42
43  volumes:
```

```
44    traefik-public-certificates:
45
46  networks:
47    public-net:
48      external: true
```

8.1.5 Dokumentation

```
providers:
  docker:
    endpoint: "tcp://127.0.0.1:2377"
    swarmMode: true
```

```
version: "3"
services:
  my-container:
    deploy:
      labels:
        - traefik.http.routers.my-container.rule=Host(`example.com`)
        - traefik.http.services.my-container-service.loadbalancer.server.port=8080
```