HAW HAMBURG

BACHELOR THESIS
Maryam Abdul-Noor

Entwicklung eines technischen Logging-Konzepts für eine Microservice-basierte Plattform

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Maryam Abdul-Noor

Entwicklung eines technischen Logging-Konzepts für eine Microservice-basierte Plattform

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

Betreuende Prüferin: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 30. Mai 2024

Maryam Abdul-Noor

Thema der Arbeit

Entwicklung eines technischen Logging-Konzepts für eine Microservice-basierte Plattform

Stichworte

Microservices Architektur, Log, Logging, Logging-Architektur, Log-Management

Kurzzusammenfassung

Diese Arbeit untersucht die Notwendigkeit eines Logging-Systems für Microservice-Architekturen zur Optimierung von Leistung und Fehlerbehebung. Im Rahmen des DaFne-Projekts wird ein skalierbares und sicheres Logging-Konzept ausgewählt, entwickelt und implementiert. Verschiedene Technologien werden bewertet und der EFK-Stack (Elasticsearch, Fluentd, Kibana) als Lösung ausgewählt. Der EFK-Stack besteht aus Elasticsearch, einer Such- und Analyse-Engine zur Speicherung und Abfrage von Log-Daten; Fluentd, einem flexiblen und erweiterbaren Datenprozessor zur Sammlung und Weiterleitung von Log-Daten; und Kibana, einem Visualisierungstool zur Darstellung der Daten aus Elasticsearch in interaktiven Dashboards. Diese Arbeit behandelt die Umsetzung eines Logging-Systems für Microservices. Die Analyse der Anforderungen und die exemplarische Implementierung zeigen, dass der EFK-Stack eine leistungsfähige und flexible Lösung darstellt, jedoch sind weitere Optimierungen erforderlich. Das ausgewählte Konzept ermöglicht die zentrale Erfassung und Analyse von Log-Daten und visualisiert diese zur Überwachung und Fehlerbehebung in Microservice-Architekturen.

Maryam Abdul-Noor

Title of Thesis

Development of a Technical Logging Concept for a Microservice-Based Platform

Keywords

Microservices Architecture, Log, Logging, Logging Architecture, Log Management

Abstract

This thesis investigates the need for a logging system for microservice architectures to optimise performance and troubleshooting. As part of the DaFne project, a scalable and secure logging concept is selected, developed and implemented. Various technologies are evaluated and the EFK stack (Elasticsearch, Fluentd, Kibana) is selected as the solution. The EFK stack consists of Elasticsearch, a search and analysis engine for storing and retrieving log data; Fluentd, a flexible and extensible data processor for collecting and forwarding log data; and Kibana, a visualisation tool for displaying data from Elasticsearch in interactive dashboards. This thesis deals with the implementation of a logging system for microservices. The analysis of the requirements and the exemplary implementation show that the EFK stack is a powerful and flexible solution, but further optimisation is required. The selected concept enables the central collection and analysis of log data and visualises it for monitoring and troubleshooting in microservice architectures.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Der Hintergrund und die Motivation hinter diesem Vorhaben werden im folgenden Abschnitt näher erläutert. Anschließend wird die Forschungsfrage definiert und die Ziele dieser Arbeit werden festgelegt. Dabei werden die Grenzen der Untersuchung festgelegt, um den Fokus der Arbeit klar zu definieren. Im Anschluss wird die Struktur der Arbeit erläutert.

1.1 Hintergrund und Motivation

Die vorliegende Arbeit entsteht im Rahmen "Data-Fusion-Generator-Platform" für Künstliche Intelligenz (DaFne). Diese Plattform steht vor der Herausforderung, verschiedene Arten von Daten zu verarbeiten, die unterschiedliche Untersuchungsansätze erfordern. Insbesondere auf microservice-basierten Plattformen werden sowohl fachliche als auch technische Log-Daten generiert [16].

Technische Log-Daten eines Services spielen eine entscheidende Rolle für die Fehlerdiagnose, Leistungsoptimierung und Sicherheitsüberwachung. Die technischen Log-Daten umfassen detaillierte Informationen über den internen Zustand und die Abläufe des Services, die für die Identifikation und Behebung von Fehlern, die Optimierung der Leistung und die Sicherstellung der Systemsicherheit unverzichtbar ist.

Eine der größten Herausforderungen besteht darin, ein effizientes und skalierbares Logging-Konzept auszuwählen, das die Erfassung, Speicherung und Analyse dieser technischen Log-Daten ermöglicht.

Dabei müssen folgende Herausforderungen adressiert werden:

- **Datenvolumen und -vielfalt:** Die große Menge und Vielfalt der generierten Daten erfordert eine Lösung, die sowohl skalierbar als auch flexibel genug ist, um verschiedene Arten von technischen Daten zu verarbeiten.

- **Integration und Kompatibilität:** Das Logging-Konzept muss nahtlos in die bestehende Microservices integriert werden können, ohne die Systemleistung zu beeinträchtigen.
- **Datensicherheit und -integrität:** Die Sicherstellung der Integrität und Vertraulichkeit der gesammelten Daten ist von zentraler Bedeutung, um die Zuverlässigkeit und Sicherheit des Systems zu gewährleisten.

1.2 Forschungsfrage und Zielsetzung

Das Hauptziel dieser Bachelorarbeit ist die Auswahl eines geeigneten technischen Logging-Konzepts für eine microservice-basierte Plattform. Es werden die Anforderungen an das Logging technischer Log-Daten untersucht und wie diese in einem entsprechenden Konzept umgesetzt werden können.

Zentral ist die Frage, wie ein effizientes und skalierbares Logging-Konzept ausgewählt und implementiert werden kann, um die Anforderungen an das Logging technischer Daten zu erfüllen. Es werden insbesondere die verfügbaren Methoden und Technologien analysiert, um ein effizientes Logging-Konzept zu implementieren und wie diese in einem Microservices-Umfeld eingesetzt werden.

Diese Forschungsfrage bildet den Kern der Arbeit und dient als Leitfaden für die Untersuchung und Auswahl des technischen Logging-Konzepts.

1.3 Abgrenzung

Der Fokus dieser Arbeit liegt ausschließlich auf dem technischen Aspekt des Loggings. Es wird nicht die Implementierung von anderen Systemfunktionen oder nichttechnischen Aspekten, wie etwa rechtliche oder organisatorische Aspekte des Datenschutzes, berücksichtigt. Die Evaluierung des implementierten Konzepts in einer produktiven Umgebung oder die Untersuchung etwaiger Auswirkungen auf die Systemleistung sind nicht Teil des Forschungsumfangs dieser Arbeit. Es wird sich auf die Auswahl von Logging-Plattformen für das Logging-Konzept beschränkt und schließt andere Arten von Logging-Technologien aus.

1.4 Struktur der Arbeit

Die Struktur dieser Bachelorarbeit ist darauf ausgelegt, eine klare und nachvollziehbare Darstellung der Forschungsergebnisse zu gewährleisten.

Die Einleitung bietet einen Überblick über den Hintergrund und die Motivation, die Forschungsfrage und Zielsetzung sowie die Abgrenzung der Arbeit.

In Kapitel Zwei werden zunächst die Merkmale und die Vorteile der Microservice erläutert. Hierbei wird auf die Definition und Merkmale von Microservices eingegangen, gefolgt von einer Diskussion der Vorteile und Herausforderungen, die mit dieser Architektur der Microservices einhergehen. Anschließend wird die Bedeutung des Loggings in verteilten Systemen analysiert. Es werden die Grundlagen und Arten des Loggings, der Unterschied zwischen dem technischem und fachlichem Logging sowie die spezifischen Herausforderungen beim Logging in Microservices behandelt.

Kapitel Drei widmet sich der Entwicklung und der Auswahl eines Logging-Konzepts. Dabei wird die angewandte Methodik beschrieben und eine Problemanalyse eines fehlenden Logging-Konzepts durchgeführt. Die Anforderungen an das Logging-Konzept werden ermittelt, und es erfolgt eine Auswahl der Technologien. Danach wird die Architektur des Logging-Systems vorgestellt, gefolgt von einer Diskussion der Prozesse für die Datenerfassung und das Datenmanagement.

In Kapitel Vier wird vorgestellt, wie das Logging-Konzepts implementiert wird. Anhand eines Beispiels wird die Implementierung an einem Microservice demonstriert. Die Planung der Implementierung umfasst die Entwurfsphase für die Konfiguration von Grok-Filtern, die Konfiguration der Schnittstellen der Technologien, Sicherheitsaspekte bei der Implementierung sowie die Integration eines Kibana Dashboards. Schließlich wird die Durchführung der Implementierung erläutert.

Kapitel Fünf umfasst die Diskussion der erreichten Ergebnisse, durch die Beantwortung der Forschungsfragen. Hier wird überprüft, inwiefern die Forschungsziele erreicht werden, welche Grenzen das ausgewählte Konzept aufweist, und es werden Empfehlungen für weiterführende Forschung und Entwicklung gegeben.

Im sechsten Kapitel werden die gesammelten Erkenntnisse zusammengefasst und es wird ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

2 Theoretische Grundlagen

Um die Arbeit dieser Bachelorarbeit zu beginnen, ist es unerlässlich, die theoretischen Grundlagen zu verstehen. Die theoretischen Grundlagen bilden die Basis für die Auswahl des technischen Logging-Konzepts und ermöglichen es, die Herangehensweise und die Entscheidungen im Verlauf der Arbeit nachvollziehen zu können.

2.0.1 Definition und Merkmale

In der Literatur gibt es verschiedene Definition für den Begriff Microservices. Im Rahmen des Buches "Das Microservices Praxisbuch" gilt folgende Definition nach Wolff: *„Microservices sind unabhängig deploybare Module"* [28]. Unabhängig deploybare (ausführbare) Module bedeutet, dass jedes Modul eigenständig implementiert, getestet und in Betrieb genommen werden kann, ohne dass andere Module davon direkt betroffen sind. Der Stil der Microservice unterteilt Systeme in kleinere, unabhängige Komponenten, die als Services oder Module bezeichnet werden.

Durch die Aufteilung in eigenständige Services kann das Gesamtsystem flexibler und anpassungsfähiger gestaltet werden, was die Wartbarkeit und Erweiterbarkeit verbessert. Darüber hinaus fördert dieser Ansatz eine klare Trennung der Verantwortlichkeiten. Jedes dieser Module ist autonom und wird separat getestet. Dieses unterscheidet sie von einem sogenannten Deployment-Monolithen. [23]

Ein Deployment-Monolith ist ein eigenständiges System, das im Gegensatz zu einer Microservice als eine Einheit betrieben wird. Das bedeutet, dass alle Komponenten und Funktionen des Monolithen gemeinsam bereitgestellt und ausgeführt werden. Ein Monolith kann in verschiedene Module unterteilt werden. Der Begriff sagt nichts über den internen Aufbau der Module aus. Die Definition von Microservices gibt keine Auskunft über deren Größe oder der Anzahl, wodurch Microservices unterschiedlich groß sein können, um als solches zu gelten. [28]

2.0.2 Vorteile und Herausforderung

Die Vorteile und Herausforderungen der Microservice werden basierend auf den Erkenntnissen aus der Studie von Bushong et al. (2021) im folgenden erläutern: [10].

Skalierbarkeit: Microservices bieten eine granulare Skalierung, bei der einzelne Dienste unabhängig voneinander skaliert werden können, was zu einer effizienteren Ressourcennutzung führt.

Technologische Vielfalt: Durch die Nutzung von Microservices ist es möglich, verschiedene Dienste mit unterschiedlichen TechStacks zu entwickeln. TechStacks bezeichnet die Kombination verschiedener Technologien, Frameworks, Programmiersprachen, Tools und Software. Diese werden zusammen verwendet, um eine Anwendung oder ein System zu entwickeln und zu betreiben. Dies erleichtert die Arbeitsteilung in kleinere Teams und beschleunigt die Einführung neuer Technologien.

Schnellere Entwicklungszyklen: Microservices fördern eine kontinuierliche Integration von Änderung und neuen Funktionen. Zudem führt eine kontinuierliche Bereitstellung neuer Funktionen zu kürzeren Entwicklungszyklen.

Einarbeitungszeit: Neue Entwickler benötigen mehr Zeit, um sich in große monolithische Systeme einzuarbeiten. Microservices können diese Einarbeitungszeit verringern.

Überwindung monolithischer Nachteile: Während die Microservice die Nachteile monolithischer Systeme überwindet, ist sie keine universelle Lösung und bringt eigene Herausforderungen mit sich.

Datenbankmuster: Microservice können unterschiedliche Datenbankmuster aufweisen. Eine Option ist die Bereitstellung einer individuellen Datenbank für jeden Dienst. Da lose Kopplung eine zentrale Eigenschaft von Microservices ist, wird dieses Muster oft bevorzugt.

Kommunikation zwischen Modulen: Eine Microservice-Anwendung besteht aus vielen kleineren Modulen, die miteinander kommunizieren. Eine REST-API (Representational State Transfer Application Programming Interface) ist eine Schnittstelle, die es verschiedenen Anwendungen ermöglicht, über das HTTP-Protokoll miteinander zu kommunizieren. In einer Microservice-Anwendung bestehen viele kleinere Module, die über die REST-APIs miteinander interagieren.

Komplexität des Testprozesses: Der Testprozess in einer Microservice ist komplex. Aufgrund der unabhängigen, kleineren Dienste ist das Testen der gesamten Anwendung komplex. Zusätzlich zu den üblichen Unit-, Integrations- und End-to-End-Tests erfordern Microservices auch Vertragstests und Leistungstests für einzelne Dienste.

2.1 Logging in verteilten Systemen

Bei Microservices ist es üblich, dass Instanzen regelmäßig gestartet und beendet werden. Daher reicht es nicht aus, Log-Daten nur innerhalb einer einzelnen Instanz zu speichern, da diese verloren gehen können, wenn die Instanz verschwindet. Zudem bestehen Microservices oft aus zahlreichen Modulen. Sich in jedes einzelne Modul einzuloggen, um die Log-Dateien zu analysieren, ist zeitaufwändig und unpraktisch. [19]

Im laufenden Betrieb können auch neue Microservices hinzugefügt werden, deren Logs ebenfalls analysiert werden müssen. Diese machen eine zentrale Speicherung und Verwaltung der Logs unverzichtbar. Alle Log-Daten müssen zentral gesammelt werden, um eine effiziente Analyse zu ermöglichen. Aufgrund der hohen Anzahl an Logs, die bei Microservices anfallen, sind herkömmliche Werkzeuge nicht ausreichend. Es werden daher leistungsfähigere Tools benötigt, die große Datenmengen effizient verarbeiten können. [18]

2.1.1 Grundlagen und Arten des Loggings

Logging bezeichnet den Prozess der Erstellung und Speicherung von Log-Einträgen in einer Datei. So werden Ereignisse systematisch erfasst und dokumentiert [11].

Logs sind textbasierte Dateien, die Informationen über aufgetretene Ereignisse in einem System enthalten. Diese Dateien sind mit nahezu jeder Programmiersprache und Infrastruktur kompatibel und werden langfristig gespeichert, sodass sie zu einem späteren Zeitpunkt analysiert werden können, um Ereignisse nachzuvollziehen und zu untersuchen. Das sequentielle Schreiben in eine Datei ist effizient und schnell, was die zeitnahe Dokumentation von Systemereignissen ermöglicht. [28]

Logs spielen somit eine wesentliche Rolle bei der Überwachung und Analyse von IT-Systemen, indem sie eine Grundlage für die Fehlerbehebung, Leistungsüberwachung und Sicherheitsüberwachung bieten [17].

In den Logging-Dateien werden Informationen zu jedem Ereignis erfasst. Dazu gehören unter anderem der genaue Zeitpunkt, die Verzögerungszeit, der Auslöser und der Status des Ereignisses oder Aufrufs. [26]

Es gibt verschiedene Arten von Logs, die zur Überwachung und Analyse von Systemen verwendet werden. Unter den Arten von Logs zählen System-Logs, Anwendungs-Logs, Sicherheits-Logs, Audit-Logs, Transaktions-Logs, Netzwerk-Logs, Ereignis-Logs, Fehler-Logs, Leistungs-Logs und Geschäfts-/Funktions-Logs ausgezählt [11]. Um die Analyse der Logs zu vereinfachen, ist es notwendig, ein einheitliches Format für die Logs zu definieren. Ein Format ist das Log-Level.

Folgende Log-Level können definiert werden:

Log-Level	Beschreibung
Error	Dieser Log-Level steht für Ereignisse, die eine negative Auswirkung auf Benutzer haben. Ein solcher Log-Eintrag kann beispielsweise das Ergebnis eines Abbruchs einer Operation aufgrund eines Fehlers sein.
Warnings	Hierbei handelt es sich um Warnmeldungen, bei denen die Auswirkungen auf den Benutzer noch verhindert werden konnten. Ein typisches Szenario wäre, dass ein anderes System nicht verfügbar war, der Ausfall jedoch durch einen Default-Wert kompensiert werden konnte.
Info	Diese Log-Einträge enthalten Informationen mit einer geschäftlichen Bedeutung, wie zum Beispiel eine neue Registrierung oder andere Ereignisse, die dokumentiert werden sollen.
Debug	Diese Logs enthalten Details, die hauptsächlich für Entwickler relevant sind, um den Code und das Verhalten der Anwendung im Detail zu verstehen und zu überprüfen.

Tabelle 2.1: Log-Level und Beschreibung

Neben dem Log-Level können in Log-Nachrichten weitere standardisierte Informationen aufgenommen werden. Eine Technik ist die Verwendung einer eindeutigen Anfrage-ID, die in jedem Log-Nachrichten ausgegeben wird. Dies ermöglicht es, die Zusammenhänge zwischen den Log-Einträgen besser zu verstehen.

Durch einheitliche und strukturierte Log-Einträge können IT-Systeme effektiver überwacht, analysiert und gewartet werden. Dies trägt maßgeblich zur Verbesserung der Systemleistung und -sicherheit bei.

2.1.2 Technische und Fachliche Logging-Daten

Folgende Aufzählung bietet eine Übersicht und hilft, die unterschiedlichen Zwecke und Inhalte der beiden Arten von Logging-Daten zu verstehen. Diese Übersicht basiert auf "Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management"von Anton Chuvakin, Kevin Schmidt und Chris Phillips [11].

Technische Logging-Daten:

- Fehlermeldungen: Beschreibungen von Fehlern oder Ausnahmen, einschließlich Stack-Traces, die den Fehlerort aufzeigen.
- Ereignis-Logs: Aufzeichnungen spezifischer Ereignisse wie Systemstarts, -stopps und andere Vorkommnisse.
- Transaktions-Logs: Details zu den vom System verarbeiteten Transaktionen, einschließlich Zeitstempeln und Statuscodes.

Folgende fachliche Logs basieren auf Mathias Weske, "Business Process Management: Concepts, Languages, Architectures", das Einblicke in die Erhebung und Nutzung fachlicher Logs zur Verbesserung und Überwachung von Geschäftsprozessen bietet [27]:

Fachliche Logging-Daten

- Aufzeichnungen von Benutzeraktionen innerhalb der Anwendung. Benutzeraktionen sind Anmeldungen, Klicks, Formularübermittlungen und andere Interaktionen, die der Benutzer mit der Anwendung durchführt.
- Aufzeichnungen über Änderungen an den Anwendungsdaten, einschließlich der Person, die die Änderungen vorgenommen hat, und der Art der Änderungen.
- Informationen, die für Debugging-Zwecke verwendet werden, oft einschließlich Nachrichten über den Zustand und den Ablauf der Anwendung.

Technische Logging-Daten und fachliche Logging-Daten spielen eine entscheidende Rolle bei der Überwachung und Wartung von Systemen. Technische Logging-Daten beinhalten Informationen wie Fehlermeldungen, Ereignis-Logs und Transaktions-Logs. Diese Daten helfen dabei, technische Probleme zu identifizieren und zu beheben.

Fachliche Logging-Daten hingegen konzentrieren sich auf die Geschäftsprozesse und Benutzerinteraktionen innerhalb der Anwendung. Sie erfassen Benutzeraktionen, Änderungen an Anwendungsdaten und Informationen für Debugging-Zwecke. Beide Arten von Logs sind essenziell für die Sicherstellung der Systemintegrität und -leistung.

2.1.3 Herausforderungen beim Logging in Microservices

In einer Microservice-Architektur ergeben sich spezifische Herausforderungen beim Logging, die durch die verteilte Natur und die Autonomie der einzelnen Dienste verursacht werden. Diese Herausforderungen umfassen die unterschiedliche Struktur und den variierenden Inhalt der Logs, die hohe Anzahl von Logs, kurzlebige Logs, die Notwendigkeit der Log-Korrelation sowie die verteilte Architektur. Im Folgenden werden diese Herausforderungen erläutert und mögliche Lösungsansätze diskutiert. [11]

Hohe Anzahl von Logs: Da jeder Dienst in einer Microservice-Architektur eigene Logdateien erstellt, entsteht eine große Menge an Logs. Dies erfordert effektive Methoden zur Sammlung und Verwaltung dieser Datenmengen. [28]

Lösungsansatz: Zentralisiertes Logging. Durch das Sammeln und Speichern von Logs in einem zentralisierten System wird eine einheitliche Sicht auf alle Logs ermöglicht.

Kurzliebige Logs: Microservices die in Containern betrieben werden und keine persistente Speicherung von Log-Daten ermöglichen. [12]

Lösungsansatz: Nutzung von zentralen Log-Management-System mit einem persistenten Speichermedium, zum Sammeln der Logs während der Laufzeit oder vor dem Herunterfahren des Containers.

Log-Korrelation: Aufgrund der verteilten Natur von Microservices müssen Log-Daten zwischen verschiedenen Diensten miteinander in Wechselbeziehung stehen, um Zusammenhänge und Ursachen von Problemen zu identifizieren. Dies kann komplex und zeitaufwendig sein. [24]

Lösungsansatz: Einführung von eindeutigen IDs, die durch alle Microservices weitergegeben werden, um zusammenhängende Transaktionen zu identifizieren.

Verteilte Architektur: Logs müssen von mehreren Knoten gesammelt werden, da Microservices typischerweise verteilt sind. Dies erfordert robuste Lösungen zur Sammlung und Aggregation von Logs. [24]

Lösungsansatz: Implementierung einer Clusterweiten Log-Sammlung. Die implementierung eines Log-Agents auf jedem Knoten, die Logs sammeln und an ein zentrales System weiterleiten. [28]

3 Konzeptentwicklung

Dieser Abschnitt widmet sich der Entwicklung und der Auswahl des technischen Logging-Konzepts.

3.1 Methodik

Die Methodik, die in dieser Bachelorarbeit verfolgt wird, orientiert sich am Ansatz des Design Science Research (DSR). Dieser Ansatz zeichnet sich dadurch aus, dass er sowohl wissenschaftliche Strenge als auch praktische Relevanz in den Mittelpunkt stellt, um Probleme zu adressieren und zu lösen. Ziel des Design-Science-Research Ansatzes ist es, durch eine Auswahl von Lösungsansätzen praktisch anwendbare und theoretisch fundierte Beiträge zu liefern.[6]

Hierbei wird zunächst das Problemfeld analysiert und definiert, um daraufhin einen Lösungsansatz auszuwählen und zu entwickeln. Der ausgewählte Lösungsansatz wird nicht nur in der Theorie ausgearbeitet, sondern auch im Rahmen des Projekts praktisch umgesetzt. Dies dient dazu, seine Anwendbarkeit und Wirksamkeit unter realen Bedingungen zu überprüfen.

3.2 Problemanalyse

Insbesondere Systeme, die auf Microservices und containerisierten Lösungen basieren, stellen neue Herausforderungen dar, vor allem in den Bereichen Sicherheit, Skalierbarkeit und Ressourcenmanagement [28].

Microservices sind verteilt und kommunizieren über Netzwerke, was die Überwachung und Fehlersuche komplexer macht. Microservices skalieren dynamisch, um Lastspitzen

zu bewältigen. Ein Logging-System muss in der Lage sein, diese elastische Infrastruktur zu unterstützen und Logs von temporären Instanzen zu sammeln und zu verwalten.

Darüber hinaus sind Container oft kurzlebig und werden schnell gestartet und gestoppt. Dies stellt eine Herausforderung für die konsistente und umfassende Erfassung von Logs dar, bevor die Container entsorgt werden.

Die Dringlichkeit eines zuverlässigen, effizienten und sicherheitsorientierten Logging-Systems ist daher von Bedeutung, um den reibungslosen Betrieb und die Nutzbarkeit dieser Plattformen zu gewährleisten [13]. Ein robustes Logging-System stellt sicher, dass alle relevanten Daten erfasst und gespeichert werden, damit die Plattform trotz der Komplexität von Microservices und Containerisierung sicher und effizient betrieben werden kann.

Das Forschungsvorhaben „Data Fusion Generator Platform“ für künstliche Intelligenz (DaFne) ist ein passendes Beispiel in diesem Kontext. Es repräsentiert eine interdisziplinäre Zusammenarbeit zwischen akademischen Institutionen und Industriepartnern. Das Hauptziel von DaFne besteht darin, eine innovative Plattform zu entwickeln, die sich auf die Synthese von Daten spezialisiert und maßgeschneiderte Datensätze für maschinelles Lernen (ML) bereitstellt. Diese Plattform soll Forschern und Entwicklern im Bereich der Künstlichen Intelligenz (KI) als zentrale Anlaufstelle dienen, indem sie einfachen Zugang zu generierten Daten ermöglicht und die nahtlose Integration neuer Daten und Algorithmen unterstützt. [15]

Angesichts der Vielfalt der Datenquellen und der Komplexität der Plattform liegt es auf der Hand, dass ein effizientes Logging-System einen reibungslosen Betrieb und eine Benutzerfreundlichkeit gewährleistet. Zu den Datenquellen gehören Fehlermeldungen, Warnungen, Benutzeraktivitäten und Systemereignisse. [13]

3.3 Anforderungen an das Logging-Konzept

Um ein umfassendes Verständnis für den Zustand, die Leistung und das Verhalten der Anwendung sicherzustellen, müssen die Anforderungen an das Logging-Konzept sowohl technisch robust als auch praktisch umsetzbar sein. Diese Anforderungen umfassen:

- **Automatische Erfassung aller relevanten technischen Daten:** Dies beinhaltet Fehlermeldungen (Fehler beim Zugriff auf Dienste oder Datenbankfehler),

Warnungen vor potenziellen Problemen, Benutzeraktivitäten (Anmeldungen und durchgeführte Aktionen).

- **Langfristige Speicherung der Log-Daten:** Das System sollte historische Analysen ermöglichen. Die Konfigurationsoptionen müssen festlegen, wie lange die Log-Daten gespeichert werden, um vergangene Ereignisse nachzuvollziehen, Trends zu identifizieren und Probleme proaktiv angehen zu können.

- **Sicherheitsmaßnahmen:** Diese umfassen die Authentifizierung und Autorisierung von Benutzern und die Verschlüsselung der Log-Daten während der Übertragung und Speicherung, um die Integrität und Vertraulichkeit der Log-Daten zu gewährleisten.

3.4 Auswahl der Technologien

Die Auswahl der untersuchten Technologien für das Logging-Konzept erfolgte nach einer Recherche und Bewertung verschiedener Lösungen. Untersucht werden die Technologien EFK-Stack (Elasticsearch [14], Fluentd [1], Kibana[2]), Graylog [4], Prometheus [20] und Splunk [5]. Die Auswahlkriterien umfassen die Aspekte Leistungsfähigkeit, Skalierbarkeit, Integrationsfähigkeit, Sicherheit und Visualisierungsmöglichkeiten. In Tabelle 3.1 wird eine zusammengefasste Übersicht der untersuchten Technologien dargestellt.

Technologie	Leistungsfähigkeit	Skalierbarkeit	Integration	Sicherheit	Visualisierung
EFK Stack	Hoch	Hoch	Nahtlos	Hoch	Umfangreich
Splunk	Hoch	Hoch	Nahtlos	Hoch	Umfangreich
Graylog	Hoch	Hoch	Nahtlos	Hoch	Umfangreich
Log4J/ Log4J2	Hoch	Mittel	Nahtlos	Mittel	Eingeschränkt
NLog	Mittel	Hoch	Nahtlos	Mittel	Eingeschränkt

Tabelle 3.1: Bewertung der Logging-Technologien

Prometheus wird frühzeitig aus der Betrachtung ausgeschlossen, da es primär für das Monitoring von Metriken konzipiert ist. Metriken sind numerische Messwerte, die die Leistung und den Zustand von Systemen über Zeiträume darstellt. Der Unterschied zwischen Metriken und technischen Logs liegt darin, das technische Logs Ereignisse und Fehlerzustände erfassen. [21]

Obwohl Splunk leistungsstark in der Analyse und beim maschinellen Lernen von Daten ist, erfüllt es ebenfalls nicht die spezifischen Anforderungen des Projekts, da es keine nahtlose Integration in Kubernetes bietet. Splunk ist als eigenständige Plattform zu implementieren. [8].

Log4J/Log4J2 und NLog wird aufgrund der eingeschränkten Skalierbarkeit und Visualisierungsmöglichkeiten nicht ausgewählt. Log4J/Log4J2 bietet zwar eine hohe Leistungsfähigkeit und nahtlose Integration, jedoch nur eine mittlere Skalierbarkeit und eingeschränkte Sicherheitsfunktionen. NLog bietet zwar Skalierbarkeit, im Gegenzug dafür aber eine mittlere Leistungsfähigkeit und ebenfalls eine eingeschränkte Visualisierungsmöglichkeit.

Die Technologie Graylog ist eine flexible Log-Management- und Analyseplattform, die hauptsächlich in IT-Infrastrukturen verwendet wird. Sie erfüllt jedoch nicht die Anforderung der nahtlosen Integration in Kubernetes und wurde daher nicht ausgewählt.

Der EFK-Stack besteht aus **Elasticsearch**, einer Such- und Analyse-Engine, die Log-Daten speichert und abfragt; **Fluentd**, einem flexiblen und erweiterbaren Datenprozessor, der Log-Daten sammelt und weiterleitet; und **Kibana**, einem Visualisierungstool, das die Daten aus Elasticsearch in interaktiven Dashboards darstellt. [29]

Die Wahl ist letztendlich auf den EFK-Stack gefallen, aufgrund der breiteren Akzeptanz des EFK-Stacks und die umfangreiche Community-Unterstützung. Dadurch wird eine größere Anzahl verfügbarer Ressourcen sichergestellt und eine schnellere Problemlösung gewährleistet. [29] Darüber hinaus bietet Kibana flexible Authentifizierungsoptionen, die es ermöglicht, Benutzeridentitäten effektiv zu verifizieren und entsprechende Zugriffsrechte zuzuweisen [25].

3.5 Architektur des Logging-Systems

In diesem Abschnitt wird die Architektur des Logging-Systems, für einen umfassenden Überblick über die Logging-Landschaft, dargestellt. Die Architektur des Logging-Systems wird in Abbildung 3.1 dargestellt und zeigt das Zusammenspiel zwischen den verschiedenen Komponenten des EFK-Logging-Systems in einer Kubernetes-Umgebung [3].

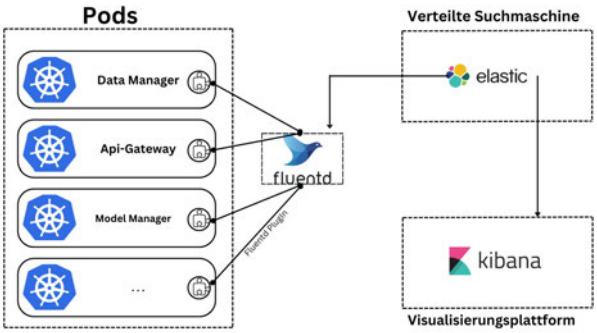

Abbildung 3.1: Architektur des EFK-Logging-Systems

Das EFK-Logging-System wird implementiert, um eine effiziente Erfassung, Verarbeitung und Visualisierung von Log-Daten aus mehreren Microservices zu ermöglichen. Die Log-Daten werden in Pods auf Kubernetes ausgeführt.

Pods sind die kleinste ausführbare Einheit in einem Kubernetes-Cluster. Das bedeutet, dass alle Container innerhalb eines Pods immer auf derselben Maschine ausgeführt werden. Jeder Container in einem Pod läuft auf derselben Netzwerkumgebung und teilt sich Speicherressourcen, wodurch eine enge Kommunikation und gemeinsame Nutzung von Log-Daten ermöglicht wird.[9]

Zentral in dieser Architektur des EFK-Logging-Systems ist der Einsatz von Fluentd, einem leistungsfähigen und flexiblen Logaggregator. Fluentd wird hierbei direkt in jedem Pod eingesetzt, sodass jeder Pod neben seiner Hauptanwendung einen Fluentd-Container hat. Diese Konfiguration ermöglicht es Fluentd, Logs nahtlos und in Echtzeit zu erfassen, ohne die Leistung der Hauptanwendung zu beeinträchtigen. [22]

Die Logging-Agenten sind nicht nur zur Erfassung der Logs zuständig, sondern auch für die Vorverarbeitung, um die Logs in eine einheitliche Form zu konfigurieren. Die verarbeiteten Logs werden von Fluentd direkt an Elasticsearch weitergeleitet. Elasticsearch speichert und indiziert die weitergeleiteten Logs und bietet eine Volltextsuche, eine hohe Skalierbarkeit und die Möglichkeit, große Mengen an Daten effizient zu verarbeiten.

Zur Analyse und Visualisierung wird Kibana als Frontend-Tool genutzt. In Kibana werden Dashboards erstellt und konfiguriert. Dashboards werden genutzt, um das Verhalten der Anwendung im Überblick zu behalten.

3.6 Prozesse für Datenerfassung und -management

Der Prozess der effizienten Datenerfassung und -verarbeitung in einem Kubernetes-Cluster beginnt mit der Verwendung der Python Logging-Bibliothek. Diese Bibliothek erfasst die Log-Daten der verschiedenen Pods, die als Datenquelle dienen. Nach der Erfassung werden die Logs in ein JSON-Format konvertiert, um eine einheitliche und strukturierte Analyse zu ermöglichen. Die Wahl der Python-Logging-Bibliothek erleichtert den gesamten Prozess, da sie leicht zu integrieren ist und keine zusätzlichen Installationen erfordert. Dadurch wird eine effiziente und konsistente Datenerfassung und -verarbeitung sichergestellt.

Fluentd sammelt die Log-Daten aus den verschiedenen Pods im Kubernetes-Cluster. Die Datenverarbeitung erfolgt in mehreren Schritten, um die erfassten Log-Daten zu filtern, zu aggregieren und zu transformieren, bevor sie analysiert oder gespeichert werden. Zunächst werden redundante Daten vermieden, gefolgt von einer Filterung basierend auf dem Inhalt. Anschließend erfolgt die Aggregation der Daten, um Trends zu erkennen. Die Aggregation erfolgt, indem ähnliche Ereignisse oder Meldungen nach den Kriterien Zeitstempel, Quelle oder Art des Ereignisses zusammengefasst werden.

Nach der Verarbeitung durch Fluentd werden die Logs an Elasticsearch weitergeleitet. In Elasticsearch werden die Log-Daten indexiert, wodurch sie effizient durchsucht und analysiert werden können. Als Lösung für die Datenspeicherung wird Elasticsearch auf dem Kubernetes-Cluster bereitgestellt. Dies gewährleistet eine skalierbare und robuste Speicherinfrastruktur für die gesammelten Log-Daten, die eine effiziente Analyse und Verwaltung ermöglicht.

4 Implementierung des Logging-Konzepts

In diesem Kapitel wird das Logging-Konzept für das DaFne-Projekt betrachtet und dessen Umsetzung beschrieben. Dabei liegt der Fokus auf der Entwicklung und Implementierung eines effektiven Logging-Systems, das die im Kapitel Drei beschriebenen technischen Anforderungen des Projekts erfüllt. Es wird aufgezeigt, wie das Logging-Konzept entworfen, implementiert und in die bestehende Infrastruktur integriert wird.

4.1 Beispielhafte Implementierung an einem Microservice

Für die Implementierung des Logging-Konzepts wird exemplarisch der Pod, namens Data-Manager, ausgewählt, der ein zentraler Bestandteil der Gesamtarchitektur von DaFne ist. Forscher und Entwickler nutzen den Data-Manager im Bereich der Künstlichen Intelligenz (KI), um den einfachen Zugriff auf generierte Daten, sowie die Integration neuer Daten und Algorithmen zu ermöglichen.

Der Data-Manager bietet eine Vielzahl von Endpunkten an, über die Benutzer Funktionen, wie das Hochladen, Herunterladen, Listen und Löschen von Dateien ausführen können.

Die Logging-Praktiken werden durch die Verwendung eines Loggers mit dem Namen “datamanagerLog“ durchgeführt. Dieser Logger ermöglicht es, Log-Einträge in verschiedenen Teilen des Codes zu erzeugen und auszugeben. Die Konfiguration des Loggings wird in einer separaten Datei (logging.conf) definiert. Das ermöglicht eine Anpassung verschiedener Aspekte des Loggings, wie Log-Level, Dateiformat und Ausgabekanäle. Diese Konfigurationsdatei wird beim Start des Programms geladen, um die Logging-Parameter zu initialisieren.

In der Code-Implementierung werden verschiedene Log-Einträge mit unterschiedlichen Log-Leveln erzeugt, um Ereignisse, Informationen und Fehler während der Programmausführung zu erfassen. Je nach Bedeutung und Schweregrads des Ereignisses werden entsprechende Log-Level ausgewählt.

Die folgenden Log-Level werden in der emoji_log-Datei definiert und mit entsprechenden Emoji-Symbolen verknüpft:

- „INFO“ (Information-Emoji) wird verwendet, um allgemeine Informationen zu protokollieren
- „WARNING“ (Warnungs-Emoji) um Warnungen vor potenziellen Problemen zu protokollieren
- „ERROR“ (Fehler-Emoji) zum Protokollieren von Fehlern
- „DEBUG“ (Käfer-Emoji) wird verwendet, um Debugging-Informationen zu protokollieren
- „CRITICAL“ (Kritisch-Emoji): um kritische Fehler zu protokollieren.

Zusätzlich enthalten die Log-Einträge kontextbezogene Informationen wie Nachrichten und Werte, um eine aussagekräftige Logging zu gewährleisten. Die hinzugefügten Log-Nachrichten dokumentieren, wenn eine Anfrage empfangen wird und falls ein Fehler auftritt, wird dieser ebenfalls protokolliert. Dies verbessert die Transparenz und Nachvollziehbarkeit der Code-Ausführung.

4.2 Planung der Implementation

Die Planung der Implementierung umfasst die Auswahl geeigneter Tools gemäß den Anforderungen des Systems, die bereits in Kapitel Drei evaluiert wurden. Dabei werden die notwendigen Schritte definiert, um die ausgewählten Tools in dem entsprechenden Container zu installieren und bereitzustellen.

4.2.1 Entwurfsphase für die Konfiguration von Grok-Filtern

Ein wesentlicher Aspekt der Planung ist die Entwurfsphase für die Konfiguration von Grok-Filtern, die dazu dienen, die erfassten Logs zu analysieren und zu verarbeiten. Die Verwendung von Grok-Filtern werden in Fluentd über ein Plugin genutzt. Dieses Plugin ermöglicht die Extraktion und Analyse von Daten aus den Logmeldungen des Data-Manager. Aus der folgenden Tabelle kann entnommen werden, welche Filterung für die „Fluentd.conf“-Datei geplant ist.

Filterung	Beschreibung
IP-Adressen	Verfolgung des Benutzerverhaltens, Überwachung von Zugriffen, Analyse von Angriffen und Untersuchung von Anomalien.
Zeitstempel	Gruppierung von Logmeldungen nach Zeitintervallen, Analyse von Trends und Fehlerbehebung.
HTTP-Statuscode	Überwachung des Anfragestatus, Identifizierung von Fehlern und Verbesserung der Anwendungsleistung.

Tabelle 4.1: Grok-Filterung

Diese Filter geben spezifische Informationen aus den Logs und extrahieren gemäß den Anforderungen.

4.2.2 Konfiguration der Verbindung der Technologien

Ein weiterer Schritt besteht darin, festzulegen, wie die Logs an die Visualisierungsplattform Kibana weitergeleitet werden. Hierzu wird Kibana mit Elasticsearch verbunden, indem die URL des Elasticsearch-Servers in der Konfigurationsdatei von Kibana eingetragen wird. Sobald die Technologien miteinander verbunden sind, kann Kibana Anfragen an Elasticsearch senden.

4.2.3 Sicherheitsaspekte bei der Implementierung

Bei der Planung der Implementierung der Sicherheitskonfiguration müssen Zugriffsrechte auf die Log-Daten definiert und sichergestellt werden, sodass nur autorisierte Benutzer auf sensible Informationen zugreifen kann.

Die Implementierung von Zugriffskontrollen und Berechtigungsmechanismen bieten solche Sicherheitskonfigurationen. Zusätzlich spielt die Verschlüsselung der Log-Daten während der Übertragung und Speicherung eine Rolle für die Sicherheit, um die Vertraulichkeit der Daten zu gewährleisten und den Schutz vor unbefugtem Zugriff zu gewährleisten.

Durch die Verwendung von sicheren Übertragungsprotokollen wie HTTPS wird dies gewährleistet und geplant.

4.2.4 Kibana Dashboard Integration

Nachdem alle geplanten Punkte der Datensammlung und Indexierung integriert sind, wird die Visualisierung durchgeführt. Die Dashboards bieten den Entwicklern einen übersichtlichen und intuitiven Einblick.

Um dies zu erreichen, werden verschiedene Arten von Visualisierungen erstellt, wie z.B. Liniendiagramme, Balkendiagramme und Tortendiagramme. Diese Visualisierungen ermöglichen es den Entwicklern, Trends zu erkennen, Anomalien zu identifizieren und Analysen durchzuführen [7].

Die Erstellung der Dashboards erfolgt in mehreren Schritten:

1. **Datenquellen auswählen:** Zuerst werden die relevanten Datenquellen in Kibana konfiguriert. Dies stellt sicher, dass die benötigten Daten für die Visualisierungen zur Verfügung stehen.

2. **Visualisierungen erstellen:** Anschließend werden die verschiedenen Visualisierungstypen erstellt. Dabei wird darauf geachtet, dass die Daten klar und verständlich dargestellt werden.

3. **Dashboards zusammenstellen:** Die erstellten Visualisierungen werden dann zu Dashboards zusammengefügt. Diese Dashboards werden so gestaltet, dass sie die Metriken und KPIs übersichtlich darstellen.

4. **Filtern und Interaktivität:** Um den Entwicklern die Arbeit zu erleichtern, werden interaktive Filter und Suchmöglichkeiten integriert. Dies erlaubt es den Nutzern, die Daten nach bestimmten Kriterien zu filtern und Analysen durchzuführen.

5. **Testen und Feedback:** Bevor die Dashboards finalisiert werden, erfolgt eine Testphase, in der Entwickler Feedback geben können. Basierend auf diesem Feedback werden gegebenenfalls Anpassungen vorgenommen, um die Benutzerfreundlichkeit und Nützlichkeit der Dashboards zu optimieren.

Mit diesen Schritten wird sichergestellt, dass die Kibana Dashboards nicht nur benutzerfreundlichen, sondern auch funktional und nützlich für die Nutzung der Entwickler sind.

4.3 Durchführung der Implementierung

Durch die Planung erfolgt die Implementierung des Logging-Konzepts effizient und zielgerichtet. Die Implementierung des Logging-Systems und der dazu passenden Tools wie Elasticsearch, Fluentd und Kibana erfolgt in mehreren Schritten, die im Folgenden näher erläutert werden.

Zunächst werden die Systemereignisse in die Hauptdatei des Data-Managers hinzugefügt. Diese Systemereignisse umfassen:

- **Neustart des Data-Managers** Diese Ereignisse werden protokolliert, um sicherzustellen, dass jeder Neustart des Data-Manager erfasst und überwacht wird. Dies ist für die Fehlersuche und die Sicherstellung der Systemstabilität relevant.

```
@prefix_router.post("/pod-restart")
async def pod_restart():
    logger.info('Pod restart detected!')
    return {"message": "Pod restart detected!"}
```

- **Skalierungsvorgänge:** Ereignisse, die mit der Skalierung der Anwendung verbunden sind, werden protokolliert. Dies umfasst sowohl das Hochskalieren (Hinzufügen neuer Instanzen) als auch das Herunterskalieren (Entfernen von Instanzen).

```
@prefix_router.post("/scaling-operation")
async def scaling_operation():
    logger.info('Scaling operation detected!')
    return {"message": "Scaling operation detected!"}
```

Listing 4.1: Implementierung der Skalierungsvorgänge

- **Aktualisierungen des Data-Managers:** Alle Updates oder Deployments neuer Versionen des Data-Managers werden protokolliert. Dies ermöglicht die Nachverfolgung von Änderungen und hilft bei der Identifizierung von Problemen, die durch neue Versionen eingeführt werden könnten.

```
@prefix_router.post("/application-update")
async def application_update():
    logger.info('Application update detected!')
    return {"message": "Application update detected!"}
```

Listing 4.2: Implementatierung der Aktualisierungsvorgängen

- **Konfigurationsänderungen:** Jede Änderung an den Konfigurationseinstellungen des Data-Manager wird protokolliert. Dies umfasst Änderungen an den Umgebungsvariablen, Konfigurationsdateien und anderen Einstellungen, die die Funktionsweise der Anwendung beeinflussen.

```
@prefix_router.post("/configuration-change")
async def configuration_change():
    logger.info('Configuration change detected!')
    return {"message": "Configuration change detected!"}
```

Listing 4.3: Implementation der Konfigurationsänderung

Zusätzlich wird der JSON-Logger aus der Python-Bibliothek implementiert, um sicherzustellen, dass die Log-Ausgaben einheitlich und strukturiert im JSON-Format vorliegen. Nachfolgend ist ein Beispiel zur Implementierung des JSON-Loggers dargestellt:

```
from pythonjsonlogger import jsonlogger

app = FastAPI()
BASEPATH = "/data"
prefix_router = APIRouter(prefix=BASEPATH)

```

```
7
8  logger = logging.getLogger('datamanagerLog')
9  logger.setLevel(logging.DEBUG)
10
11 json_handler = logging.StreamHandler()
12 json_handler.setFormatter(jsonlogger.JsonFormatter())
13 logger.addHandler(json_handler)
```

Listing 4.4: Implementierung des JSON-Logger

Diese Vorgehensweise stellt sicher, dass alle gesammelten Ereignisse im Data-Manager protokolliert und leicht analysierbar sind.

Installation des Log-Agents von Fluentd:

Der Fluentd Log-Agent wird lokal im Data Manager installiert.

Konfiguration von Fluentd:

Die Konfigurationsdatei "Fluentd.conf" legt fest, wie die Log-Daten erfasst, verarbeitet und an das gewünschte Ziel weitergeleitet wird. Diese Datei definiert unter anderem die Quellen, Filter und Ziele für die Weiterleitung der Log-Daten. Die in Kapitel 4.1.1 benannten Grok-Filterungen werden ebenfalls in die Konfigurationsdatei implementiert, um IP-Adressen, HTTP-Statuscodes und Zeitstempel zu extrahieren.

```
1  <filter data_manager_logs>
2  @type parser
3  key_name log
4  <parse>
5  @type grok
6  <grok>
7  pattern %{IP:client_ip}
8  tag ip_adresse
9  </grok>
10 <grok>
11 pattern %{NUMBER:http_status_code}
12 tag http_status_code
13 </grok>
14 <grok>
15 pattern %{TIMESTAMP_ISO8601:zeitstempel}
16 tag zeitstempel
```

```
17 </grok>
18 </parse>
19 </filter>
```

Listing 4.5: Implementation der Grok-Filterung

In Listing 4.5 wird in den Zeilen 1-19 der Konfigurationsdatei „fluentd.conf“ eine Filterkonfiguration für die Log-Daten des Data-Managers definiert. Dabei werden die drei verschiedenen Grok-Patterns verwendet, um spezifische Informationen aus den Logs zu extrahieren und entsprechende Tags zuzuweisen.

Integration im Data-Manager:

Die Konfiguration von Fluentd wird im Python-Code des Services hinzugefügt, um Logmeldungen an Fluentd zu senden. Dadurch werden die Log-Daten aus dem Data-Manager erfasst und an Fluentd weitergeleitet, um sie dort weiter zu verarbeiten und an Elasticsearch zu senden.

```
1  import logging
2  import fluent.handler
3
4  # Konfiguration des Fluentd-Loggers
5  fluent_handler =
6  fluent.handler.FluentHandler('datamanager', host='localhost', port
       =24224)
7  logging.getLogger().addHandler(fluent_handler)
8
9  # Setze das Log-Level auf INFO
10 logging.basicConfig(level=logging.INFO)
```

Listing 4.6: Fluentd-Implementierung im Python-Code

In Listing 4.6. wird in Zeile 6 der Fluentd-Handler konfiguriert, um Log-Nachrichten an Fluentd zu senden. Der Host und der Port werden festgelegt, um die Kommunikation mit dem Fluentd-Server zu ermöglichen.

Installation und Konfiguration von Elasticsearch:

Elasticsearch wird über den Paketmanager installiert. Nach der Installation wird die Konfigurationsdatei so eingestellt, dass externe Verbindungen zugelassen werden. Das

geschieht über „elasticsearch.yml“-Datei in der die Adresse des Hosts eingetragen wird, die von Kibana und Fluentd erreichbar ist. Die Kommunikation erfolgt über die Netzwerkeinstellungen.

Installation und Konfiguration von Kibana:

Wie auch zuvor bei Elasticsearch und Fluentd wird Kibana ebenfalls über den Paketmanager installiert. In die Kibana Konfigurationsdatei „kibana.yml“ werden die Elasticsearch-Instanzen integriert, um die Kommunikation zwischen den Technologien zu befähigen.

Sicherheit in Kibana:

Um den Zugang zu Kibana zu steuern und zu sichern, werden Maßnahmen ergriffen, die sowohl die Konfiguration der Software als auch die Implementierung von Netzwerkrichtlinien umfassen. Die Einstellungen „server.port“ und „server.host“ in der Konfigurationsdatei von Kibana („kibana.yml“) werden für den Zugriff und die Sicherheit der Kibana-Instanz angepasst.

Erstellung und Visualisierung des Dashboards:

Kibana wird über die URL http://<Kibana-Host>:<Port> in einem Webbrowser geöffnet. Der *<Kibana-Host>* entspricht der IP-Adresse des Kibana-Servers und der *<Port>* der entsprechende Port. Zunächst wird eine Visualisierung für die Daten ausgewählt. Folgend wird die Datenquelle ausgewählt und anschließend wird die Visualisierung konfiguriert, indem Felder, Filter und Optionen angegeben werden.

Skalierbarkeit und Performance:

Das Logging-System ist so konzipiert, dass es mit wachsendem Datenvolumen skalieren kann. Dies wird durch die verteilte Architektur von Elasticsearch und die Flexibilität von Fluentd ermöglicht.

5 Diskussion

In diesem Kapitel werden die Ergebnisse der vorliegenden Arbeit diskutiert und im Kontext der zu Beginn formulierten Forschungsfragen analysiert. Die Diskussion konzentriert sich darauf, wie das entwickelte Konzept zur Erfüllung der Anforderungen an das technische Logging-Konzept in einer microservice-basierten-Plattform beiträgt. Außerdem wird die Empfehlung für zukünftige Forschung und Entwicklung thematisiert.

5.1 Erreichung der Forschungsziele

Das Hauptziel dieser Arbeit ist die Entwicklung eines technischen Logging-Konzept für eine Microservice-basierten-Plattform. Dabei werden die Anforderungen identifiziert und in einem passenden Konzept umgesetzt. Um dieses Ziel zu erreichen, wird zunächst die Anforderungen an das Logging von technischen Log-Daten im Rahmen des DaFne-Projekts analysiert. Die Fragen, die im Rahmen dieser Arbeit untersucht wird, sind folgende:

Forschungsfrage 1: Welche Anforderungen bestehen an das Logging von technischen Log-Daten? Eine der ersten Aufgaben besteht darin, die Anforderungen an das technische Logging zu identifizieren. Hierzu führe ich eine Analyse des ausgewählten Services innerhalb des DaFne-Projekts durch. Dabei untersuche ich, welche Arten von Daten dieser Service generiert und welche Informationen für die Leistungsoptimierung relevant sind.

Die Anforderungen an das Logging von technischen Log-Daten umfassen die automatische Erfassung von Fehlermeldungen, Warnungen und Benutzeraktivitäten. Diese Erfassung ist notwendig, um eine umfassende Übersicht über den Systemzustand und das Benutzerverhalten zu erhalten, was wiederum zur Optimierung der Leistung des Services beiträgt.

Fehlermeldungen werden erfasst, um auftretende Probleme identifizieren und beheben zu können. Diese Informationen sind für die Diagnose und die Verbesserung der Systemstabilität. Warnungen werden aufgezeichnet, um potenzielle Probleme frühzeitig zu erkennen und proaktive Maßnahmen zur Vermeidung von Ausfällen zu ermöglichen. Das Logging von Benutzeraktivitäten dient der Nachvollziehbarkeit von Interaktionen und der Verbesserung der Benutzerfreundlichkeit.

Die aufgezeichneten Daten werden über einen Zeitraum gespeichert, um historische Analysen und die Identifikation von Trends zu ermöglichen. Diese langfristige Speicherung unterstützt die kontinuierliche Überwachung und Optimierung des Systems, indem sie Einblicke in wiederkehrende Muster und Anomalien bietet.

Zusätzlich sind Sicherheitsmaßnahmen erforderlich, um die Integrität und Vertraulichkeit der Log-Daten während der Übertragung und Speicherung zu gewährleisten. Dies umfasst die Verschlüsselung der Daten, den kontrollierten Zugriff und regelmäßige Sicherheitsüberprüfungen, um sicherzustellen, dass die Daten vor unbefugtem Zugriff und Manipulation geschützt sind.

Forschungsfrage 2: Wie kann durch die Auswahl und Implementierung einer geeigneten Technologie ein effizientes und skalierbares Logging-Konzept realisiert werden, um die festgelegten Anforderungen zu erfüllen?

Um ein effizientes und skalierbares Logging-Konzept auszuwählen und zu implementieren, das die Anforderungen erfüllt, wird der EFK-Stack (Elasticsearch, Fluentd, Kibana) verwendet. Auf Grundlage der gewonnenen Erkenntnisse entsteht ein umfassendes Logging-Konzept, das den Anforderungen an Effizienz und Skalierbarkeit gerecht wird. Dieses Konzept umfasst die Architektur des Logging-Systems, die Implementierung des Log-Collector-Tools Fluentd, die Strukturierung der Log-Daten mithilfe von Elasticsearch und die Integration in die bestehende Microservices-Plattform. Zur Visualisierung der Daten wird Kibana eingesetzt, um eine benutzerfreundliche und leistungsstarke Darstellung der Log-Daten zu ermöglichen.

Der erste Bestandteil, Fluentd aggregiert und transformiert Log-Daten aus verschiedenen Quellen vor der Weiterleitung und sorgt so für eine standardisierte und strukturierte Datenerfassung, wodurch die Effizienz der Datenerfassung und -verarbeitung gesteigert wird. Der zweite Bestandteil, Elasticsearch Technologie ermöglicht die schnelle Suche und Analyse von Log-Daten, was die Effizienz der Datenverarbeitung verbessert und somit eine effektive Datenauswertung ermöglicht. Der dritte Bestandteil, Kibana, erlaubt

die Erstellung von Dashboards, die Echtzeitüberwachung und Informationen zur Systemleistung und Benutzeraktivität bieten. Durch die Visualisierung und Analyse wird die Effizienz der Dateninterpretation erhöht.

Zur Erfüllung der Anforderung der Skalierbarkeit orchestriert Kubernetes mehrere Instanzen des EFK-Stacks (Elasticsearch, Fluentd, Kibana). Die Skalierung der Technologien stellt sicher, dass der EFK-Stack auch bei wachsendem Datenvolumen und steigender Anzahl von Quellen effizient Log-Daten sammelt, ohne Einbußen in der Leistungsfähigkeit. Durch die Verteilung von Fluentd-Agenten auf verschiedene Microservices wird eine gleichmäßige Lastverteilung bei der Sammlung von Log-Daten erreicht, was zur Skalierbarkeit beiträgt.

5.2 Grenzen des ausgewählten Konzepts

Die ausgewählte Logging-Architektur wird unter Berücksichtigung der identifizierten Anforderungen konzipiert und in einer Testumgebung implementiert. Die Evaluierung des implementierten Konzepts in einer produktiven Umgebung wird nicht durchgeführt. Dies kann in zukünftigen Arbeiten adressiert werden, um die Praxistauglichkeit und langfristige Stabilität des Konzepts zu validieren.

Die Auswirkungen des Loggings auf die Systemleistung wird nicht umfassend untersucht. In einer produktiven Umgebung könnten die Performance-Einbußen durch das Logging anders ausfallen als in der Testumgebung. Rechtliche und organisatorische Aspekte des Datenschutzes wurden nicht behandelt. Diese Aspekte sind jedoch für die Implementierung eines ganzheitlichen Logging-Systems in einer realen Anwendung unerlässlich und sollten in weiteren Untersuchungen berücksichtigt werden.

5.3 Empfehlung für weiterführende Forschung und Entwicklung

Für die weiterführende Forschung und Entwicklung empfiehlt es sich, das Logging-Konzept im Bereich der Anomalieerkennung und Benachrichtigung über auffällige Aktivitäten in den Log-Daten auszubauen. Dies würde die Fähigkeit des Systems zur frühzeitigen Erkennung und Reaktion auf ungewöhnliche Muster oder potenzielle Sicherheitsbedrohungen verbessern.

Darüber hinaus ist die Integration mit anderen Logging-Technologien, wie beispielsweise Prometheus, von Vorteil. Diese Integration ermöglicht es, sowohl Metriken als auch Log-Daten in einem einheitlichen System zu überwachen und zu analysieren, was die Gesamtüberwachung und Fehlersuche weiter optimiert.

Insgesamt bieten die Erweiterung und Integration des Logging-Systems zahlreiche Möglichkeiten, die Effizienz und Sicherheit von microservices-basierten Plattformen weiter zu steigern. Diese Aspekte kann in zukünftigen Forschungs- und Entwicklungsprojekten vertieft untersucht werden.

6 Fazit

Im Fazit werden die gesammelten Erkenntnisse dieser Bachelorarbeit zusammengefasst.

6.1 Zusammenfassung

Die vorliegende Bachelorarbeit konzentriert sich auf die Entwicklung eines geeigneten technischen Logging-Konzepts für eine microservices-basierte Plattform. Im Mittelpunkt stehen die Fragestellungen, welche Anforderungen an das Logging von technischen Log-Daten bestehen und wie ein effizientes und skalierbares Logging-Konzept entworfen und umgesetzt werden kann, um diese Anforderungen zu erfüllen.

Zu Beginn dieser Arbeit werden die spezifischen Anforderungen an das Logging analysiert und identifiziert. Anhand eines Service des DaFne-Projekts wird untersucht, welche Arten von Daten generiert werden und welche davon für die Leistungsoptimierung, Fehlerdiagnose und Sicherheitsüberwachung relevant sind.

Nach der Identifizierung der Anforderungen werden verschiedene Logging-Methoden und -Technologien bewertet. Das Log-Collector-Tool Fluentd ist als geeignete Lösung ausgewählt und implementiert. Fluentd ermöglicht die effiziente Erfassung und Vorverarbeitung der Log-Daten. Diese Daten werden mithilfe von Elasticsearch, das für seine leistungsfähige Volltextsuche und hohe Skalierbarkeit bekannt ist, strukturiert. Zur Visualisierung der Log-Daten wird Kibana verwendet, das eine benutzerfreundliche und leistungsstarke Darstellung bietet.

Das entwickelte Logging-Konzept umfasst die Architektur des Logging-Systems, die Implementierung der notwendigen Tools und die Integration in die bestehende Microservices-Plattform. Durch den Einsatz moderner Technologien wurde ein leistungsfähiges und skalierbares Logging-System realisiert. Die Arbeit zeigt, dass durch eine systematische Herangehensweise ein robustes Logging-Konzept entwickelt werden kann.

Zusammengefasst werden folgende Hauptziele erreicht:

- Identifikation der Anforderungen: Durch die Analyse der technischen Log-Daten eines exemplarischen Service konnten relevante Anforderungen definiert werden.
- Auswahl und Implementierung geeigneter Technologien: Fluentd, Elasticsearch und Kibana wurden als die geeignetsten Tools identifiziert und erfolgreich implementiert.
- Auswahl eines Logging-Konzepts: Die Architektur des Logging-Systems wurde entworfen und in die bestehende Plattform integriert, was eine effektive Erfassung, Analyse und Visualisierung der Log-Daten ermöglicht.

Die Ergebnisse dieser Arbeit bieten eine Grundlage für zukünftige Erweiterungen und Optimierungen. Das entwickelte Konzept zeigt, dass durch den Einsatz moderner Technologien ein effizientes und skalierbares Logging-System realisiert werden kann, das den Anforderungen einer microservices-basierten Plattform gerecht wird.

6.2 Ausblick

Die Arbeit zeigt, dass der Einsatz von Logging-Technologien das Potenzial besitzt, um die Leistung, Sicherheit und Fehlerdiagnose in microservices-basierten Plattformen zu verbessern. Dabei ist jedoch eine intensive Analyse der Anforderungen von Bedeutung. Ohne eine sorgfältige Anforderungserhebung kann die Visualisierung der Log-Daten eher hinderlich als hilfreich sein. Es besteht das Risiko, dass unzureichende Daten gesammelt werden, was zu einem Verbrauch von Datenspeicher führen kann.

Für zukünftige Schritte sollten für ein ganzheitliches Logging-Konzept weitere Services in Betracht gezogen werden. Dabei ist es entscheidend, für jeden Service individuell zu entscheiden, welche Anforderungen zur Überwachung und Analyse benötigt werden. Da jeder Service sein eigenes System ist und unterschiedliche Funktionen bietet, ist die Herangehensweise passend auszuwählen.

Dies führt zu mehreren konkreten Maßnahmen:

1. **Erweiterung des Anwendungsbereichs:** Implementierung des Logging-Konzepts für weiteren Microservices, wobei individuelle Anpassungen vorgenommen werden, um die spezifischen Anforderungen jedes Services zu erfüllen.

2. **Technologische Weiterentwicklung:** Untersuchung und Integration neuer Technologien und Trends im Bereich Logging.

3. **Erweiterte Analysefähigkeiten:** Entwicklung und Implementierung von Machine-Learning-Algorithmen zur Anomalieerkennung und Prognose von Systemausfällen basierend auf den gesammelten Log-Daten.

4. **Automatisierung:** Einführung von Automatisierungstools zur kontinuierlichen Überwachung und Wartung des Logging-Systems, um manuelle Eingriffe zu minimieren und die Effizienz zu erhöhen.

Insgesamt bietet die Weiterentwicklung des Logging-Konzepts im Rahmen des DaFne-Projekts die Chance, die Plattform kontinuierlich zu verbessern und an die sich ändernden Anforderungen anzupassen. Dies wird nicht nur die Effizienz und Leistungsfähigkeit der Plattform steigern, sondern auch ihre Relevanz und Nutzerfreundlichkeit erhöhen. Die kontinuierliche Verbesserung und Anpassung des Logging-Systems wird dazu beitragen, dass die Plattform flexibel und zukunftssicher bleibt.

Glossar

Elasticsearch Elasticsearch ermöglicht schnelles und skalierbares Durchsuchen und Analysieren von großen Mengen an Daten in nahezu Echtzeit.

Fluentd Fluentd ist ein Open-Source-Datenkollektor, der zum Sammeln, Verarbeiten und Weiterleiten von Logdaten verwendet wird. Es unterstützt verschiedene Input- und Output-Plugins, die eine einfache Integration mit anderen Systemen und Datenquellen ermöglichen.

Graylog Graylog ist eine leistungsstarke Open-Source-Lösung für das Management und die Analyse von Log-Daten. Es bietet eine web-basierte Benutzeroberfläche und unterstützt die zentrale Erfassung, Speicherung und Analyse von Log-Daten.

Kibana Kibana ist ein Open-Source-Datenvisualisierungs- und Explorationstool, das auf Elasticsearch aufbaut. Es ermöglicht Benutzern, auf einfache Weise Dashboards zu erstellen, Daten zu visualisieren und zu durchsuchen sowie Echtzeitanalysen durchzuführen.

Kubernetes Kubernetes ist ein Open-Source-System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen.

Prometheus Prometheus ist ein Open-Source-Monitoring- und Alarmsystem, das speziell für zeitlich sequenzierte Daten entwickelt wurde. Es sammelt und speichert Metriken in einer zeitlichen Datenbank und ermöglicht leistungsstarke Abfragen und Visualisierungen.

Splunk Splunk ist eine Plattform zur Überwachung, Suche, Analyse und Visualisierung von maschinell generierten Daten in Echtzeit. Sie unterstützt das Sammeln, Indizieren und Korrelieren von Daten in einem durchsuchbaren Repository.

Literaturverzeichnis

[1] : *Fluentd | Open Source Data Collector | Unified Logging Layer.* – URL https://www.fluentd.org/. – Zugriffsdatum: 2024-05-28

[2] : *Kibana: Visualisieren, Analysieren und Erkunden von Daten | Elastic.* – URL https://www.elastic.co/de/kibana. – Zugriffsdatum: 2024-05-28

[3] : *Kubernetes.* – URL https://kubernetes.io/. – Zugriffsdatum: 2024-05-28

[4] : *SIEM, Log Management & API Protection.* – URL https://graylog.org/. – Zugriffsdatum: 2024-05-28

[5] : *Splunk | The Key to Enterprise Resilience.* – URL https://www.splunk.com/. – Zugriffsdatum: 2024-05-28

[6] Baskerville, Richard ; Baiyere, Abayomi ; Gregor, Shirley ; Hevner, Alan ; Rossi, Matti: Design Science Research Contributions: Finding a Balance between Artifact and Theory. In: *Journal of the Association for Information Systems* 19 (2018), Nr. 5, S. 358–376. – Publisher: Association for Information Systems. – ISSN 1558-3457

[7] Bhatnagar, Divyesh ; SubaLakshmi, R J. ; Vanmathi, C.: Twitter Sentiment Analysis Using Elasticsearch, LOGSTASH And KIBANA. In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Februar 2020, S. 1–5

[8] Bruzzese, Roberto: *An Analisys of Application Logs with Splunk : developing an App for the synthetic analysis of data and security incidents.* 2019. – _eprint: 1912.11283

[9] Burns, B. ; Beda, J. ; Hightower, K. ; Demmig, T.: *Kubernetes: Eine kompakte Einführung.* dpunkt.verlag, 2020. – URL https://books.google.de/books?id=FCIPEAAAQBAJ. – ISBN 978-3-96910-048-6

[10] Bushong, Vincent ; Abdelfattah, Amr S. ; Maruf, Abdullah A. ; Das, Dipta ; Lehman, Austin ; Jaroszewski, Eric ; Coffey, Michael ; Cerny, Tomas ; Frajtak, Karel ; Tisnovsky, Pavel ; Bures, Miroslav: On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. In: *Applied Sciences* 11 (2021), Nr. 17. – URL https://www.mdpi.com/2076-3417/11/17/7856. – ISSN 2076-3417

[11] Chuvakin, Anton ; Schmidt, Kevin ; Phillips, Chris: *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management.* Syngress Publishing, 2013

[12] Desina, Ganesh C.: Evaluating The Impact Of Cloud-Based Microservices Architecture On Application Performance. In: *arXiv preprint arXiv:2305.15438* (2023). – URL https://doi.org/10.48550/arXiv.2305.15438

[13] Di Nocera, Francesco ; Tempestini, Giorgia ; Orsini, Matteo: Usable Security: A Systematic Literature Review. In: *Information* 14 (2023), Nr. 12. – URL https://www.mdpi.com/2078-2489/14/12/641. – ISSN 2078-2489

[14] elasticsearch: *elasticsearch/elasticsearch.* 2015. – URL https://github.com/elasticsearch/elasticsearch

[15] Gedigk, Sebastian ; Menk, Alexander: *plattform erstellung einer prototypischen plattform zur datensynthetisierung*

[16] Glass, Ayse ; Tokuc, Kübra ; Noennig, Jorg R. ; Steffens, Ulrike ; Bek, Burak: *DaFne: Data Fusion Generator and Synthetic Data Generation for Cities*

[17] Koryugin, Danila: Analysing and alerting on application logs within Kubernetes infrastructure. (2023)

[18] Li, Zheng ; Seco, Diego ; Sánchez Rodríguez, Alexis: Microservice-Oriented Platform for Internet of Big Data Analytics: A Proof of Concept. In: *Sensors* 19 (2019), März, Nr. 5, S. 1134. – URL https://www.mdpi.com/1424-8220/19/5/1134. – Zugriffsdatum: 2024-04-30. – ISSN 1424-8220

[19] Newman, Sam: *Building microservices: designing fine-grained systems.* First Edition. Beijing Sebastopol, CA : O'Reilly Media, 2015. – OCLC: ocn881657228. – ISBN 978-1-4919-5035-7

[20] Prometheus: *Prometheus - Monitoring system & time series database.* – URL https://prometheus.io/. – Zugriffsdatum: 2024-05-28

[21] Schünke, M. ; Schulte, E. ; Schumacher, U. ; Voll, M. ; Wesker, K.: *Prometheus, LernAtlas der Anatomie, Allgemeine Anatomie und Bewegungssystem*. Stuttgart : Thieme, 2005

[22] Späth, Peter: Logging Pipeline with Fluentd. In: *Pro Jakarta EE 10: Open Source Enterprise Java-based Cloud-native Applications Development*. Berkeley, CA : Apress, 2023, S. 427–436. – URL https://doi.org/10.1007/978-1-4842-8214-4_35. – ISBN 978-1-4842-8214-4

[23] Starke, Gernot: *Effektive Softwarearchitekturen : ein praktischer Leitfaden*. München : Hanser, 2014. – URL http://www.worldcat.org/search?qt=worldcat_org_all&q=9783446436145. – ISBN 978-3-446-43614-5 3-446-43614-6 978-3-446-43653-4 3-446-43653-7

[24] Söylemez, Mehmet ; Tekinerdogan, Bedir ; Tarhan, Ayça K.: Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. In: *Applied Sciences* 12 (2022), Nr. 11, S. 5507. – Publisher: MDPI

[25] Takase, Wataru ; Nakamura, Tomoaki ; Watase, Yoshiyuki ; Sasaki, Takashi: *A solution for secure use of Kibana and Elasticsearch in multi-user environment*. 2017. – _eprint: 1706.10040

[26] Tzanettis, Ioannis ; Androna, Christina-Maria ; Zafeiropoulos, Anastasios ; Fotopoulou, Eleni ; Papavassiliou, Symeon: Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications. In: *Sensors* 22 (2022), Nr. 5. – URL https://www.mdpi.com/1424-8220/22/5/2061. – ISSN 1424-8220

[27] Weske, Mathias: *Business Process Management: Concepts, Languages, Architectures*. Springer Science & Business Media, 2012

[28] Wolff, Eberhard: *Das Microservices-Praxisbuch*. Heidelberg : dpunkt, 2018. – ISBN 978-3-86490-526-1

[29] Zhang, Haiyang ; Zeng, Hao: Design and implementation of blockchain platform operation and maintenance support system based on Kubernetes+EFK framework. In: *ISCTT 2021; 6th International Conference on Information Science, Computer Technology and Transportation*, 2021, S. 1–8

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen 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