Duale Hochschule Baden-Württemberg

Stuttgart Campus Horb

Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur

T3300 - Bachelorarbeit

---

eingereicht von:	Moris Kotsch
Matrikelnummer:	1317681
Kurs:	TINF2018
Studiengang:	Informatik
Hochschule:	DHBW Stuttgart Campus Horb
Ausbildungsfirma:	ENISCO by FORCAM GmbH
Ausbildungsleiterin:	Dipl.-Betriebsw. Angela Rasch
Unternehmen der Bachelorarbeit:	ENISCO by FORCAM GmbH
Betrieblicher Betreuer:	Dipl.-Ing. (FH) Franziska Simmank
Gutachter der DHBW:	Prof. Dr. phil. Antonius van Hoof
Bearbeitungszeitraum:	07.06.2021 - 31.08.2021

Freudenstadt, 27. August 2021

Sperrvermerk

Die vorliegende Bachelorarbeit zum Thema „Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“ beinhaltet interne vertrauliche Informationen der Firma ENISCO by FORCAM GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma ENISCO by FORCAM GmbH.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Bachelorarbeit mit dem Thema **„Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“** ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Bachelorarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Freudenstadt, 27. August 2021

Moris Kotsch

Zusammenfassung

Die vorliegende Bachelorarbeit mit dem Titel „Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“ erläutert die verschiedenen Möglichkeiten der Umsetzung einer Suchfunktionalität.

Die Suchfunktionalität soll in naher Zukunft in der MES-Software „MCC“ eingesetzt werden. Hierbei ist „MCC“ die Neugestaltung des aktuellen Produktionsleitsystems „E-MES“der Firma Enisco by Forcam GmbH. Bei der Neugestaltung wird von einer monolithisch betriebenen, modularen 3-Schichten-Architektur auf eine verteilte Microservice-Architektur gewechselt.

Im Rahmen dieser Arbeit wird zunächst auf die verschiedenen Arten von Suchfunktionalitäten in modernen Informationssystemen eingegangen. Dabei werden die Volltextsuche, die facettierte Suche und die semantische Suche näher betrachtet. Der spätere Kontext der Sucharten wird anhand des Funktionsumfangs des Produktionsleitsystems „MCC“ definiert. Hierfür erfolgt eine Erläuterung des Funktionsumfangs.

Bei der Erstellung eines geeigneten Konzeptes werden gültige Architektur-Prinzipien beachtet, um monolithische Seiteneffekte bei der Einführung einer Suchfunktionalität zu vermeiden. Innerhalb des Konzeptes wird eine Auswahl einer Search Engine und einer geeigneten Datenpipeline getroffen. Hierbei werden die Search Engines „Apache Solr“ und „Elasticsearch“ gegenübergestellt. Bezüglich einer Datenpipeline werden die Umsetzungsmöglichkeiten „Dual Write“, „Polling“ und „Change-Data-Capture“ verglichen.

Im Anschluss erfolgt eine prototypische Umsetzung der Change-Data-Capture - Datenpipeline in Verbindung mit der Search Engine „Elasticsearch“.

Abstract

This bachelor thesis with the title „Konzeption und prototypische Implementierung einer Search Engine in einer Microservice-Architektur“ explains the different possibilities for the implementation of a search functionality.

The search functionality is to be implemented in the MES software „MCC“ in the near future. „MCC“ is the redesign of the current manufacturing execution system „E-MES“ of the company Enisco by Forcam GmbH. In the redesign, a change is made from a monolithically operated, modular 3-layer architecture to a distributed microservice architecture.

In the context of this work, the different types of search functions in modern information systems will be discussed first. Full-text search, faceted search, and semantic search will be examined in more detail. The respective scope of the search types is defined on the basis of the functional scope of the production control system „MCC“. For this purpose, an explanation of the functional scope is given.

When creating a suitable concept, valid architectural principles are taken into account in order to avoid monolithic side effects when introducing a search functionality. Within the concept, a selection is made regarding a search engine and a suitable data pipeline. The search engines „Apache Solr“ and „Elasticsearch“ are compared, and the implementation options „Dual Write“, „Polling“ and „Change-Data-Capture“ are compared with regard to a data pipeline.

This is followed by a prototypical implementation of the change data capture data pipeline in conjunction with the search engine „Elasticsearch“.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CDC	Change Data Capture
DBMS	Datenbankmanagementsystem
E-MES	Enisco Manufacturing Execution System
HMI	Human-Machine-Interface
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency
MCC	Manufacturing Control Cloud
MES	Manufacturing Execution System
NLP	Natural Language Processing
PCS	Production Control System
QoS	Quality of Service
REST	Representational State Transfer
SCADA	Supervisory Control and Data Acquisition
SPS	speicherprogrammierbare Steuerung
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency

Kapitel 1

1 Einleitung

Im Rahmen der vorliegenden Arbeit werden verschiedene Umsetzungsmöglichkeiten für eine Suchfunktionalität in einem MES-Produkt der Firma Enisco betrachtet. Der Fokus liegt hierbei auf den diversen Arten für die Umsetzung einer Suche. Durch die architekturelle Neugestaltung des bisherigen MES-Produktes wird auch auf die technischen Besonderheiten bezüglich der Suchfunktionalität in einer Microservice-Architektur eingegangen.

Im folgenden, einleitenden Kapitel wird die betreuende Firma mit dem dazugehörigem Kernprodukt E-MES, die Aufgabenstellung und das geplante Vorgehen erläutert. Um das Lesen der Arbeit zu erleichtern, wird ein Überblick über den Aufbau der Arbeit gegeben.

1.1 Enisco und E-MES

Die vorliegende Arbeit wurde im Rahmen einer Bachelorarbeit bei der Firma Enisco verfasst. Enisco GmbH & Co. KG wurde 2015 als Tochtergesellschaft der Eisenmann SE gegründet und ist mittlerweile ein eigenständiges Unternehmen, welches unter dem Namen „Enisco by Forcam GmbH[1]“ agiert.

Als Kernprodukt vertreibt die Firma Enisco das Produktionsleitsystem „Enisco Manufacturing Execution System (E-MES)“. Dieses wird für die Überwachung und Steuerung von Produktionsanlagen eingesetzt. E-MES vernetzt dabei die Anlage sowohl horizontal, über den gesamten Fertigungsprozess, als auch vertikal, über alle Prozessebenen hinweg und bildet so ein System, welches zwischen Unternehmensebene (ERP, engl. für Enterprise Ressource Planning) und Steuerungsebene (PLC, engl. für Programmable Logic Controller) agiert. [Eni21a]

E-MES ist dafür modular aufgebaut und besitzt als Basis ein Platform-Modul. Dieses beinhaltet alle Grundlagen und Schnittstellen für die Installation weiterer Module. Die zusätzlichen Module (engl. Add-Ons) ergänzen E-MES um bestimmte Funktionen und können kundenspezifisch installiert und konfiguriert werden.

[1] Im Folgenden wird aus Gründen der Lesbarkeit auf die Rechtsform der Enisco by Forcam GmbH verzichtet

1.2 Aufgabenstellung

Derzeit erfährt das aktuelle Produktionsleitsystem E-MES eine Neugestaltung. Dabei wird von einer monolithisch betriebenen, modularen 3-Schichten-Architektur auf eine verteilte Microservice-Architektur gewechselt. Ein Wechsel der Architektur beruht auf dem Eintritt der Muttergesellschaft Forcam GmbH in die „Open Industry 4.0 Alliance“. Durch den Zusammenschluss von mehreren Unternehmen aus dem Bereich „Industrie 4.0“ können einheitliche Schnittstellen definiert werden, um so die Interoperabilität zwischen den Softwarelösungen der beteiligten Firmen zu stärken [Ope21]. Um die benötigte Interoperabilität zu ermöglichen, setzen die beteiligten Unternehmen vermehrt auf Technologien wie Docker und Kubernetes. Um nun auch die Neugestaltung von E-MES in diesem Umfeld anzubieten, wurde sich für eine verteilte Microservice-Architektur entschieden. Neben der Architektur wird auch der Produktname von „E-MES“ in den vorläufigen Produktnamen „Manufacturing Control Cloud (MCC)“ abgeändert.

Im Zuge der Neugestaltung von E-MES werden neue Funktionalitäten, wie eine Suchfunktion integriert. Eine Suchfunktion in modernen Informationssystemen wird von den Benutzern als gewohnter Komfort wahrgenommen. Für die Umsetzung einer Suchfunktion in einem Manufacturing Execution System (MES) muss festgelegt werden, welche Funktionen und Inhalte des MES von der Suchfunktion abgedeckt werden sollen. Neben der Suche nach Funktionalitäten des Systems, kann es auch hilfreich sein, nach bestimmten „Objekten“ innerhalb des Systems zu suchen. Solche Objekte können zum Beispiel in Form von eindeutigen Aufträgen oder Maschinen in einem MES vorkommen. Gibt der Benutzer die Kennung eines Objektes in das Suchfeld ein, sollen ihm alle Funktionen und Informationen bezüglich dieses Objektes angezeigt werden. So soll der Benutzer bei der Navigation durch das MES unterstützt werden.

Neben der Anforderungsklärung bezüglich der Suchoptionen und der Granularität der Suchanfragen, gilt es auch eine Konzeption für die Integration einer Search Engine in die MCC-Gesamtarchitektur zu entwerfen. Hierbei sind die besonderen Anforderungen zu beachten, welche durch die Einführung der verteilten Microservice-Architektur entstanden sind. So ist zu klären, welche Strategie für die Datenaktualisierung zwischen einer Search Engine und den Datenhaltungsschichten der einzelnen Services den Anforderungen am besten entspricht. Anhand von selbstgewählten technischen und lizenzbezogenen Kriterien sollen diesbezüglich Strategien und auch potentiell geeignete Search Engines verglichen werden. Bei der Konzeption für die Integration einer Search Engine sind monolithische Seiteneffekte, die durch die Missachtung von Prinzipien der Microservice-Architektur entstehen könnten, zu vermeiden.

Das erstellte Konzept gilt es anschließend mithilfe einer prototypischen Implementierung einer Suchanwendung als „Proof of Concept“ umzusetzen.

1.3 Vorgehensweise und Aufbau der Arbeit

Die Vorgehensweise und die schriftliche Ausarbeitung der vorliegenden Arbeit gliedert sich in drei Hauptteile. Als Vorarbeit für die eigentliche Bearbeitung werden in **Kapitel 2** die theoretischen Grundlagen über die verschiedenen Umsetzungsmöglichkeiten einer Suchfunktion erläutert. Durch die Erläuterung von allgemeingültigen Architektur-Prinzipien und im spezielleren Anti-Pattern der Microservice-Architektur, werden die theoretischen Grundlagen für die Vermeidung von monolithischen Seiteneffekten geschaffen.

Im ersten Teil der Arbeit wird definiert, mit welchem Suchumfang die Search Engine innerhalb von MCC nach Funktionen und Objekten agieren soll. Da zum Zeitpunkt der Erstellung dieser Arbeit noch keine produktreife Version von MCC existiert, wird sich an dem Produktumfang und den Funktionalitäten des aktuellen Produktes E-MES orientiert. In **Kapitel 3** wird der Suchumfang für die Suchfunktionalität definiert. Hierbei wird bei MCC analysiert, welche Objekte innerhalb des Systems „suchbar“ gemacht werden sollen.

Ein weiterer Schritt ist die Konzeption für die Integration einer Search Engine in MCC. Es werden hierbei in **Kapitel 4** verschiedene Search Engines anhand von technischen und lizenzbezogenen Kriterien miteinander verglichen. Ebenso werden verschiedene Umsetzungsmöglichkeiten der Datenaktualisierung zwischen einer Search Engine und den Datenhaltungsschichten der einzelnen Services erläutert und anhand von Vergleichskriterien miteinander verglichen. Vorbereitend für die prototypische Umsetzung wird zusätzlich ein Gesamtkonzept erstellt.

Anschließend an die Konzeption folgt in **Kapitel 5** eine Beschreibung der prototypischen Umsetzung anhand einer Proof-of-Concept-Anwendung. Hierfür wird zunächst festgelegt, welchen Umfang jene prototypische Umsetzung besitzen soll und ob bereits Softwareteile aus MCC verwendet werden können.

Abgeschlossen wird die Arbeit in **Kapitel 6** mit einer Zusammenfassung der gewonnen Erkenntnisse und mit einem Ausblick auf die spätere Integrierung in das Produktionsleitsystem MCC.

Kapitel 2

2 Grundlagen

Grundlegend wird in folgendem Kapitel auf den Funktionsumfang von Suchfunktionalitäten in modernen Informationssystemen eingegangen. Differenziert wird dabei betrachtet, inwiefern sich die Anforderungen an eine Suchfunktionalität zwischen einer Websuche, einer Suche in einem E-Commerce - Shop und einer Suche in einem MES-System unterscheiden. Neben den Anforderungen und Charakteristiken der Suchfunktionen, werden auch die verschiedenen Umsetzungsmöglichkeiten „Volltextsuche“, „Facetten-Suche“ und „semantische Suche“ erläutert. Für eine nach Relevanz sortierte Auflistung der Suchergebnisse, wird eine Möglichkeit der Relevanzbestimmung vorgestellt.

Um eine Konzeption und prototypische Umsetzung einer Volltextsuche in MCC zu ermöglichen, wird zusätzlich auf die architekturellen Hintergründe von MCC eingegangen. Hierfür werden zu Beginn allgemein gültige Architektur-Prinzipien erläutert, welche auch bei der Integration einer Volltextsuche berücksichtigt werden müssen und somit bei der Auswahl eines geeigneten Konzeptes von Bedeutung sind.

Aufbauend auf den Architektur-Prinzipien wird die verwendete Microservice-Architektur erläutert, wobei neben einer allgemeinen Einführung in die Architektur auch auf die Kommunikation von Microservices untereinander eingegangen wird. Um mögliche Fehlkonzeptionen zu vermeiden, werden häufig auftretende Anti-Pattern aufgezeigt, welche unter Umständen zu monolithischen Seiteneffekten führen können.

2.1 Suchfunktionen in modernen Informationssystemen

Aufgrund der fortlaufenden Digitalisierung und Themen wie „Industrie 4.0“ und „Internet of Things“ steigt die jährlich anfallende Datenmenge im weltweiten Internet. Das aufkommende Datenvolumen wird bereits für das Jahr 2025 auf 175 Zettabyte prognostiziert [F. 18]. Dies wäre ein Wachstum, vom Jahr 2018 bis zum Jahr 2025, um mehr als den Faktor fünf.

Neben den reinen Daten wachsen zunehmend auch die Anzahl der Websites und Web-Inhalten. So gibt es Stand 2021 rund 1,83 Milliarden Websites [Guy21], welche durch Suchmaschinen, wie Google gefunden werden können. Ein Nutzer kann hierbei einen

Volltext als Sucheingabe eingeben und erhält eine Liste mit den entsprechenden Treffern sortiert nach der Relevanz für den Nutzer. Solche Suchfunktionalitäten werden auch direkt auf den Websites angeboten, um den Besuchern so das Navigieren durch die Website und das Auffinden von Produkten und Dienstleistungen zu erleichtern.

Je nach Art der Benutzer und Umfang der Produkte und Informationen stehen bei der Umsetzung einer Suchfunktion mehrere Möglichkeiten zur Verfügung. Eine Möglichkeit ist die Suche mit Hilfe einer Volltextsuche. Für einen Benutzer stellt solch eine Art der Suchfunktion einen bekannten Komfort dar, da sie bereits aus Web-Suchmaschinen, wie zum Beispiel Google bekannt ist. Eine weitere Möglichkeit bietet die facettierte Suche. Hierbei werden die Details der Suchtreffer über verschiedene Facetten definiert. Durch Filterung der Facetten können somit die Suchtreffer eingegrenzt werden. Umgesetzt wird die facettierte Suche hauptsächlich in Online-Shops. Nach der Auflistung der Suchtreffer kann der Benutzer durch passende Filter, wie zum Beispiel die maximale Datenmenge bei USB-Sticks, die Menge an Suchtreffern eingrenzen.

Als Ergänzung zur Volltextsuche und facettierten Suche wird bei der semantischen Suche zusätzlich noch der semantische Wert einer Suchanfrage berücksichtigt. Dadurch ist es möglich, dem Benutzer der Suchfunktionalität relevante Suchergebnisse zu liefern.

2.1.1 Anwendungsgebiete für Suchfunktionen

Je nach Anwendungsgebiet einer Suchfunktionalität gibt es Unterschiede bezüglich dem Umfang des suchbaren Datenkontextes und der Größe des Benutzerkreises. Aufgrund dieser Unterschiede ist eine passende Auswahl beziehungsweise Kombination der Umsetzungsmöglichkeiten für die Suchfunktionalitäten von Nöten. Anschließend an die Betrachtung der drei Anwendungsgebiete „Websuche“, „E-Commerce - Suche“ und „MES-Suche“, werden die Umsetzungsmöglichkeiten näher erläutert.

Ein Anwendungsgebiet ist die Websuche, bei welcher explizit nach Websites und den Web-Inhalten gesucht wird. Hierbei stellt das World Wide Web den Datenkontext dar und ist aufgrund der Größe von rund 1,83 Milliarden Websites (Stand 2021) [Guy21] nicht mehr ohne Suchmaschinen, wie zum Beispiel Google, zu durchsuchen. Umgesetzt wird die Suchfunktionalität bei der Websuche durch die Verwendung einer Volltextsuche. Um den Benutzern effektivere Suchtreffer zu gewährleisten, wird bei Suchmaschinen, wie Google, zusätzlich noch eine semantische Suche verwendet.

Auch in Online-Shops beziehungsweise E-Commerce - Shops ist eine Suchfunktion für die Navigation innerhalb der Websites von Nöten. Der durchsuchbare Datenkontext ist abhängig vom angebotenen Artikelumfang der Website. Eine Suchfunktion ist im Gegensatz zur Websuche nicht unbedingt notwendig, da Produkte und Dienstleistungen auch über eine Navigation der Website erreichbar sind. Jedoch ist der Benutzerkreis oftmals nicht mit dem Aufbau der Website vertraut, so dass eine Suchfunktionalität

bei der Navigation innerhalb der Website unterstützen kann. Oft zu finden ist eine Kombination aus einer Volltextsuche und einer facettierten Suche. Hierbei wird dem Benutzer primär eine Volltextsuche in Form eines Eingabefeldes angeboten und anschließend kann über eine „Erweiterte Suche“ die Facettierung vorgenommen werden. Ein bekanntes Beispiel für die Verwendung einer Kombination beider Möglichkeiten ist der Onlineversandhändler „Amazon“. Durch die Eingabe eines Suchbegriffes wird dem Benutzer eine Auswahl geeigneter Suchtreffer angezeigt. Für eine weitere Eingrenzung der Suchergebnisse kann der Benutzer die Ergebnisse zum Beispiel anhand der Preise, Bewertungen oder Hersteller filtern.

Im Rahmen dieser Arbeit wird explizit die Einführung einer Suchfunktionalität in ein MES-System betrachtet. Der Datenkontext eines MES-Systems kann hierbei sowohl ein großes Volumen als auch eine hohe Komplexität aufweisen. Je nach Funktionsumfang und Struktur des MES-Systems kann auch die Navigation innerhalb des Systems erschwert sein. Der überwiegend explizite Benutzerkreis muss daher im Umgang mit dem MES-System geschult werden. Damit sowohl neuen Benutzern der Umgang mit dem System ermöglicht, und erfahrenen Benutzern der Umgang mit dem System erleichtert wird, kann eine Kombination aus Volltextsuche und facettierter Suche verwendet werden.

2.1.2 Volltextsuche

Eine Möglichkeit nach Produkten oder Informationen zu suchen, ist die Volltextsuche. Hierbei kann der Benutzer einen beliebigen Freitext in ein Suchfeld eingeben und erhält eine Auflistung der Suchtreffer, welche den eingegebenen Freitext enthalten.

Da auch Suchmaschinen wie Google auf einer Volltextsuche basieren, ist diese Art der Suchfunktion für die Benutzer ein gewohnter Komfort. Eine Volltextsuche wird dabei nicht nur in den Suchmaschinen oder direkt in auf den Websites angeboten, sondern auch für die Navigation und Suche in Software-Produkten. Ein Beispiel hierfür ist die Wiki-Software „Atlassian Confluence“, welche standardmäßig eine Suchfunktion zur Verfügung stellt. Für die Wissensaufnahme und Weitergabe der Firma Enisco wird diese Wiki-Software verwendet.

In Abbildung 2.1 ist beispielhaft die Trefferauflistung des Suchbegriffes „Studenten“ abgebildet. Hierbei dursucht die Suchfunktion den Datenbestand der Confluence-Anwendung der Firma Enisco und gibt als Ergebnis alle Seiten zurück, welche mit dem Suchbegriff übereinstimmen. In der gezeigten Ansicht erscheinen dabei alle Seiten, welche einen Treffer im Titel aufweisen. Durch die Schaltfläche „Suche nach **Studenten**“, welche in der Ansicht unten dargestellt wird, werden auch Confluence-Seiten angezeigt, in welchen der Suchbegriff auch im eigentlichen Inhalt der Seite gefunden wird.

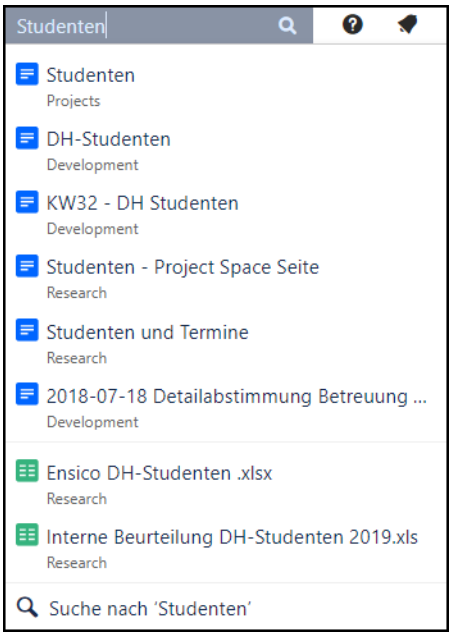

Abb. 2.1: Ergebnisse der Suchanfrage „Studenten“ - entnommen aus [Eni21b][1]

Grundlage für die technische Umsetzung einer Volltextsuche ist die Erstellung eines Suchindexes. Ein Suchindex für die Volltextsuche wird hierbei als invertierter Index umgesetzt [Seb17]. Die Funktionsweise von einem invertierten Index ist vergleichbar mit dem Stichwortverzeichnis in Büchern. Bei der Suche nach einem bestimmten Wort oder Begriff in Büchern, ist eine Suche durch das gesamte Buch umständlich und mit einem hohen Zeitaufwand verbunden. Durch die Verwendung eines Stichwortverzeichnis, wie in Abbildung 2.2 abgebildet, werden die Begriffe aus dem Buch den einzelnen Seiten zugewiesen. Eine Suche im Stichwortverzeichnis ist aufgrund der Vorsortierung schneller und man kann direkt an die entsprechende Stelle im Buch springen.

Abb. 2.2: Beispiel für ein Stichwortverzeichnis [wek14]

Bei der Eingabe von Freitexten durch menschliche Benutzer ist das Auftreten von

[1]Quelle aus dem Intranet (nicht öffentlich zugänglich) der Enisco by Forcam GmbH

Fehleingaben nicht auszuschließen. Aus diesem Grund müssen sowohl beim Aufbau des invertierten Indexes, als auch beim Bearbeiten der Suchanfrage gewisse Transformationsschritte berücksichtigt werden [Seb17]. Die Transformationsschritte sorgen für eine Fehlertoleranz innerhalb der Suchfunktion und verbessern die User Experience bezüglich der Benutzung der Suchfunktion.

Die Transformationsschritte werden benötigt, um die Vielzahl der unterschiedlichen Suchanfragen korrekt zu beantworten [Seb17]. Hierbei werden je nach Anforderungen verschiedene Tokenizer und Analyzer verwendet. In Abbildung 2.3 ist eine beispielhafte Abfolge von Transformationsschritten aufgezeigt. Ziel ist es, die variantenreiche Suchphrase durch eine Normalisierung in ein einheitliches Format zu überführen. Dadurch kann eine ausreichende Fehlertoleranz geschaffen werden. Neben der Suchphrase müssen die Transformationsschritte auch bei der Erstellung des Indexes berücksichtigt werden [Seb17].

Die in Abbildung 2.3 aufgezeigten Transformationsschritte werden nachfolgend näher erläutert.

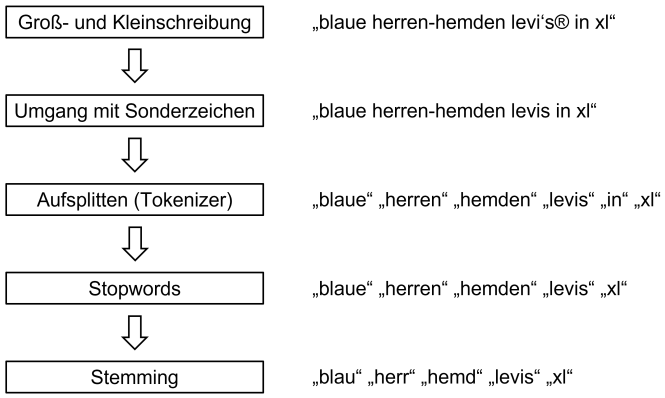

Abb. 2.3: Beispielhafter Ablauf der Transformationsschritte - in Anlehnung an [Seb17]

Tokenizer

Hierbei wird die komplette Suchphrase in verschiedene Suchbegriffe (auch Tokens genannt) aufgeteilt. So werden beispielsweise aus der Suchphrase „Hose Blau“ die Tokens „Hose“ und „Blau“. Neben der Trennung der Tokens anhand von Leerzeichen, ist auch eine Trennung anhand von bestimmten Zeichen möglich. Solche Zeichen können zum Beispiel das Subtraktionszeichen „-“ oder Semikolon „;“ sein. Die Aufteilung in Tokens ist hierbei die Vorbereitung für die Verwendung von einigen Analyzern.

Analyzer

Die Analyzer agieren bei der Transformation der Suchanfrage als Filter und ersetzen, löschen oder verändern einzelne Tokens. Darunter zählen unter anderem Filter bezüglich der Groß- und Kleinschreibung und eventuelle Sonderzeichen. Durch dieses Vorgehen wird eine gewisse Fehlertoleranz gegenüber den Eingaben der Benutzer gewährt. Zu den Sonderzeichen gehöhren zum Beispiel Symbole, wie die Kennzeichnung für eingetragene Handelsmarken „®“, oder Trennzeichen für verschiedene Begriffe. So werden die Begriffe „USA“ und „U.S.A“ auf einen gemeinsamen Begriff gemappt.

Eine weitere Möglichkeit die Suchphrase zu filtern, ist das Entfernen von Stopwords. Dies sind häufig auftretende Begriffe, welche eine geringe semantische Relevanz für die Suchphrase besitzen [Seb17]. Die Stopwords werden als Liste angelegt, welche überwiegend aus Füllwörtern wie zum Beispiel „mit“, „für“ und „von“ besteht. Lediglich bei der Verwendung einer semantischen Suche, werden die Stopwords benötigt, um die Bedeutung zu erkennen [Seb17]. Die Funktionsweise einer semantischen Suche wird in Unterabschnitt 2.1.4 erläutert.

Um die Fehlertoleranz zu erhöhen, muss auch die morphologische Varianz der einzelnen Begriffe berücksichtigt werden. So können Begriffe in unterschiedlichsten Zeitformen und Numeri vom Benutzer eingegeben werden. Um die Varianz der menschlichen Sprache zu beseitigen, müssen die Begriffe durch Normalisierung vereinheitlicht werden. Ein Verfahren für die Normalisierung ist das „Stemming“ (zu deutsch Stammformreduktion). Beim Stemming werden die unterschiedlichen morphologischen Varianten eines Begriffes auf einen gemeinsamen Wortstamm zurückgeführt. Als ein Arbeitsschritt aus dem Natural Language Processing (NLP) wird Stemming auch in Suchmaschinen verwendet [Ste20]. Hierbei muss dieses Verfahren sowohl bei der Behandlung der Suchphrase, als auch bei der Erstellung des Indexes berücksichtigt werden [Seb17]. Folgend sind Beispiele für Stemming aufgelistet [Seb17]:

- rot, rotes, roter, rote $\rightarrow$ rot
- schuh, schuhe $\rightarrow$ schuh
- töpfe, topf $\rightarrow$ topf

Ein beispielhafter Ablauf der Transformation sowohl beim Zeitpunkt der Abfrage, als auch zum Zeitpunkt der Erzeugung des Indexes ist in Abbildung 2.4 abgebildet. Hierbei wird deutlich, dass bei der technischen Umsetzung eines invertierten Indexes die Wahl der Analyzer sowohl für die Bearbeitung der Suchphrasen, als auch für die Erstellung des Indexes von Bedeutung ist.

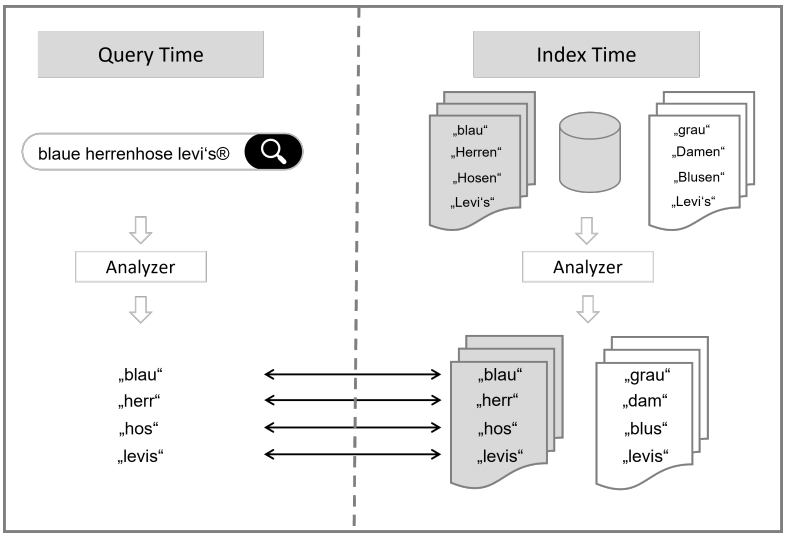

Abb. 2.4: Analyzer bei Indexierung und Abfrage - in Anlehnung an [Seb17]

2.1.3 Facettierte Suche

Bei der Facetten-Suche wird die Menge der Suchergebnisse durch das Setzen von Filtern immer weiter eingeschränkt. Die Objekte und Produkte eines Informationssystems werden dabei mit Metadaten angereichert, welche die jeweiligen Facetten definieren. Der Begriff Facette beschreibt die verschiedenen Teilaspekte eines Objekts oder Produkts. So kann beispielhaft ein „Hemd“ unter anderem mit den Facetten „Größe“, „Farbe“ oder „Muster“ versehen werden.

Eine Mindestanforderung nach Sperling [Mar18] ist, dass die Filter sich dynamisch aus den bereits gefundenen Suchergebnistreffern bilden. So wechseln dynamisch die Filtermöglichkeiten je nach Suchergebnis und sinnfreie Filtermöglichkeiten beziehungsweise Filtermöglichkeiten ohne Suchergebnisse werden vermieden. In der Praxis ist ein solch dynamischer Wechsel der Filtermöglichkeiten in Online-Shops zu finden. Ein Benutzer ist zum Beispiel bei einem USB-Stick daran interessiert mithilfe eines Filters die maximale Datenmenge zu begrenzen und bei der Suche nach Waschmaschinen ist die Angabe des maximalen Fassungsvermögen eine aussagekräftige Filterung.

Wie in Abbildung 2.5 zu erkennen ist, kann man diese Filter mit unterschiedlichsten Bedienelementen visualisieren. Dabei ist zu unterscheiden, ob die Werte der Facetten explizit genannt werden können oder sich in einem Bereich befinden. Wie von Schubert Motors [Sch21] in Abbildung 2.5 umgesetzt, wurde zum Beispiel für die Preis-Facette ein Slider und für die Auswahl der Marke ein Drop-Down-Bedienelement ausgewählt.

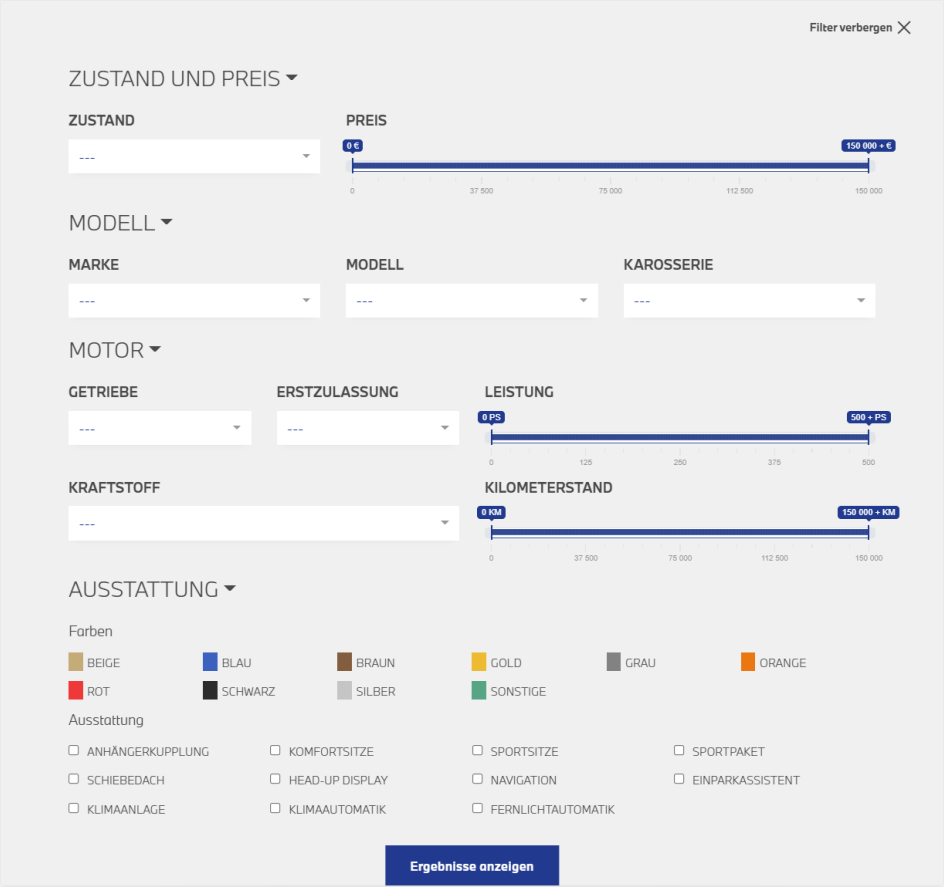

Abb. 2.5: Facettierte Suche bei Schubert Motors [Sch21]

Bei der praktischen Umsetzung einer facettierten Suche müssen die logischen Verknüpfungen zwischen den einzelnen Filter konfiguriert werden [Mar18]. Aus den Ergebnissen der unterschiedlichen Filter müssen Vereinigungs- oder Schnittmengen gebildet werden, welche wiederum sortiert, durch eine Relevanzbestimmung, dem Benutzer präsentiert werden. Zum Einsatz kommen hierfür neben „UND“- auch „ODER“- Verknüpfungen. Die Wahl der richtigen Verknüpfung ist vom jeweiligen Kontext abhängig. So ist die Intention eines Benutzers bei der Benutzung der Facette „Farbe“, dass die einzelnen Ergebnisse mit einer ODER-Verknüpfungen miteinander kombiniert werden. Ein Benutzer erhält somit beispielsweise als Ergebnis alle Produkte der Farbe rot **UND** blau. Verwendet man jedoch mehrere Facetten bei der Suche, wie zum Beispiel die „Farbe“ und „Größe“, ist die Schnittmenge aus den Ergebnissen von Vorteil. Hier ist eine UND-Verknüpfungen notwendig.

2.1.4 Semantische Suche

Während die Varianten „Volltextsuche“ und „Facettierte Suche“ lediglich auf einer syntaktischen und statistischen Auswertung von einzelnen Suchbegriffen basieren, greift eine semantische Suche zusätzlich noch auf ein semantisches Modell zurück [Hop20, S.

4f]. Solch ein, auch als „Wissensmodell“ bezeichnetes, semantisches Modell, beschreibt die begrifflichen Zusammenhänge und Beziehungen. Dadurch ist es möglich neben syntaktisch ähnlichen, auch inhaltlich verwandte Treffer zu finden [Hop20, S. 3]. Nach Hoppe [Hop20, S. 3] kann demnach eine semantische Suche, im Sprachgebrauch der Künstlichen Intelligenz, als wissensbasiert bezeichnet werden.

Ziel ist es, die Suchfunktion weiter der menschlichen Sprache anzunähern. So liegt es, laut Tamblé [Mel12], in der Natur der menschlichen Sprache, dass mit verschiedenen Wörtern inhaltlich gleiche oder ähnliche Bedeutungen gemeint sind. So kann eine Landfläche in der deutschen Sprache auch mit den Begriffen „Bereich“, „Gebiet“ oder „Areal“ beschrieben werden. Neben Synonymen werden Begriffe auch durch Assoziationen miteinander verbunden. So wird der Begriff „Rot“ mit dem Begriff „Liebe“ assoziiert.

Für die Umsetzung einer semantischen Suche bedarf es einer Wissensdatenbank, in welcher Entitäten und deren Beziehungen untereinander abgespeichert werden. Mit dem Knowledge Graphen betreibt Google seit 2012 solch eine Wissensdatenbank, um Funktionen einer semantischen Suche bereitzustellen [Ola21]. Der Knowledge Graph wird dabei in drei Ebenen unterteilt: [Ola21]

Entitäten-Katalog:
: Die unterste Ebene ist eine reine Auflistung der verschiedenen Entitäten, welche über die Zeit hinweg identifiziert wurden. Die Art der Entitäten ist abhängig vom Kontext der Anwendung, in welcher eine Suchfunktion integriert ist. Allgemein handelt es sich um Objekte und Konzepte, welche eindeutig identifiziert werden können. Darunter zählen zum Beispiel Persönlichkeiten, Orte, Organisationen oder auch Farben und Gefühle.

Knowledge Repository:
: Die unterschiedlichen Entitäten aus dem Entitäten-Katalog werden im Knowledge Repository mit Informationen beziehungsweise Attributen erweitert. Neben dem Ergänzen der Entitäten um Beschreibungen, werden diese auch in semantische Klassen und Gruppen eingeteilt. Die Gruppierungen werden auch Entitätstypen genannt.

Knowledge Graph:
: Im eigentlichen Graphen werden die Beziehungen zwischen den unterschiedlichen Entitäten hergestellt.

In Abbildung 2.6 ist das Ergebnis einer semantische Suchanfrage von Google abgebildet. Gesucht wurde mit der Suchphrase „**small green guy with lightsaber as child**“. Als Ergebnis erhält der Benutzer eine sofortige Übersicht über das wahrscheinlichste Ergebnis. In diesem Beispiel wurden die einzelnen Begriffe aus der Suchphrase mit dem Knowledge-Graphen von Google verglichen. Dabei erwiesen die Beziehungen der Begriffe untereinander, dass der fiktive Charakter „Grogu“ am wahrscheinlichsten gesucht wird.

Während bei einer reinen syntaktischen Suchfunktion die einzelnen Begriffe durch Transformationsschritte vereinheitlicht und anschließend mit einem invertierten Index verglichen werden, kann die semantische Suche nun auch mit Synonymen und Assoziationen umgehen. So würde die Suchmaschine Google das gleiche Ergebnis, wie in Abbildung 2.6, liefern, wenn statt dem Begriff „small“ das Synonym „little“ verwendet wird.

Abb. 2.6: Ergebnis einer semantischen Suche in Google

Um genauere und personalisierte Suchergebnisse zu erhalten, verwendet Google bei der semantischen Suche zusätzlich noch die persönlichen Daten und den Suchverlauf der Benutzer [Mic20]. So ist ein Programmierer mit der Suchphrase „Python“ eher daran interessiert Suchergebnisse über die selbige Programmiersprache zu erhalten, anstatt über das Tier.

2.1.5 Relevanzbestimmung

Unabhängig, ob eine Volltextsuche, facettierte Suche, semantische Suche oder eine Kombination zum Einsatz kommt, müssen die Ergebnisse der Suchanfrage in einer geeigneten Reihenfolge angezeigt werden. Es gilt die einzelnen Treffer der Suche mit einer Relevanz zu gewichten und dem Benutzer die Treffer mit der höchsten Relevanz als erstes anzuzeigen.

Eine Möglichkeit der praktischen Umsetzung einer Relevanzbestimmung ist die Durchführung einer Termgewichtung mit dem „TF-IDF“-Modell. „Term Frequency - Inverse Document Frequency (TF-IDF)“ ist dabei ein statistisches Maß, welches angibt, wie relevant ein Begriff für ein Dokument in einer Sammlung von Dokumenten ist [Bru19]. Durch die Multiplikation der beiden Metriken „Term Frequency (TF)“ und „Inverse

Document Frequency (IDF)“ kann eine Sortierung der Suchergebnisse nach Relevanz erfolgen.

Term Frequency - TF

Eine der Metrik bezieht sich auf die Häufigkeit eines Begriffes in einem Dokument. Um diese Häufigkeit zu bestimmen, gibt es verschiedene Möglichkeiten. Das einfachste Vorgehen ist die reine Zählung des Vorkommens des Begriffes in einem Dokument [Bru19]. Weitergehend kann man die Häufigkeit auch von der Länge des Dokumentes oder von der Häufigkeit des häufigsten Wortes abhängig machen [Bru19].

Unter der Einbeziehung der Gesamtzahl aller Terme eines Dokumentes wird mit folgender Formel [Jen20] die Häufigkeit eines Begriffes berechnet:

$$TF_{(\mathrm{i})} = \frac{\log_2(Freq(i,j)+1)}{\log_2(L)}$$

- i = Term
- j = Dokument
- L = Gesamtzahl aller Terme im Dokument j
- Freq(i,j) = Häufigkeit des Terms i im Dokument j

Inverse Document Frequency - IDF

Multipliziert wird die erste Metrik mit der inversen Dokumentenhäufigkeit. Darunter versteht man die Häufigkeit des Vorkommens eines Begriffes über die gesamte Dokumentenmenge hinweg. Einem Begriff wird hierbei ein größerer IDF-Wert zugeordnet, wenn der Begriff seltener vorkommt. Stopwords würden hierbei einen niedrigen IDF-Wert erhalten, da diese Begriffe in vielen Dokumenten vorkommen.

Für die Berechnung der invertierten Dokumentenhäufigkeit wird folgende Formel [Jen20] verwendet:

$$IDF_{\mathrm{t}} = \log(\frac{N_{\mathrm{D}}}{f_{\mathrm{t}}})$$

- t = Term
- N_D = Gesamtzahl der Dokumente in der Dokumentensammlung D
- f_t = Anzahl aller Dokumente, die t enthalten

Für die Bestimmung der Relevanz eines Begriffes in einem Dokument wird die Multiplikation beider Metriken benötigt. Bei lediglich der Bestimmung der Begriffshäufigkeit,

würden Stopwords eine große Relevanz bekommen. Erst durch die Einbeziehung der inversen Dokumentenhäufigkeit werden solche Begriffe, wie Stopwords mit einer niedrigeren Relevanz versehen.

Beispiel

Bei der Anwendung des TF-IDF - Wertes für die Relevanzbestimmung der Suchtreffer werden die IDF-Werte der einzelnen Begriffe aus der Suchphrase miteinander verglichen. Sucht der Benutzer zum Beispiel nach der Suchphrase „Website Analytics“, wird vermutlich dem Begriff „Website“ ein niedriger IDF-Wert zugeordnet, da dieser Begriff in vielen Dokumenten vorkommt.

Weitergehend werden dann die jeweiligen TF-Werte bestimmt. Hierbei wird jeder Begriff aus der Suchphrase mit jedem Dokument aus der Trefferliste in die TF-Formel eingegeben. So wird ein Dokument, welches den Begriff „Analytics“ häufig enthält, einen hohen TF-Wert erhalten.

Durch die Multiplikation beider Metriken pro Dokument kann eine Sortierung nach Relevanz durchgeführt werden.

2.2 Software-Architekturen

> „Die Software-Architektur eines Systems ist die Menge von Strukturen, die benötigt werden, um Entscheidungen über das System zu treffen, welche die Software-Elemente, die Relationen zwischen ihnen und die Eigenschaften von beiden betreffen.“ ~ Len Bass [BCK13, S. 4]

Wie aus der Definition von Len Bass zu entnehmen ist, beschreibt eine Software-Architektur die Eigenschaften und Beziehungen von Software-Bausteinen zueinander [BCK13, S. 4]. Ein Software-Baustein wird hierbei als eine Teil-Komponente der gesamten Software betrachtet und wird bei der Erstellung einer Architektur als elementarer Bestandteil angesehen. Dabei wird ein Software-Baustein nicht näher spezifiziert, sondern als Komponente betrachtet, dessen konkrete Implementierung für die Architektur nicht von Bedeutung ist. Der Fokus einer Software-Architektur liegt auf den Schnittstellen der Software-Bausteine, über welche die Bausteine miteinander kommunizieren können.

2.2.1 Architektur-Prinzipien

Für das Erstellen einer guten Software-Architektur wurden von Vogel [Vog09, S. 128-147] einige Grundprinzipien definiert. Diese Prinzipien sollten bei der Erstellung einer Software-Architektur beachtet werden: [Vog09, S. 128-147]

Lose Kopplung:

Der Kern einer Software-Architektur besteht aus der Beschreibung der Software-Bausteine eines Software-Systems und deren Interaktionen zueinander. Unter dem Begriff Kopplung versteht man hierbei die Beziehung unter den Software-Bausteinen einer Software-Architektur. Eine Kopplung charakterisiert demnach die Interaktionen der Software-Bausteine.

Eine starke Kopplung von Software-Bausteinen hat zur Folge, dass beim Verstehen und Ändern eines Software-Bausteines auch zwingend weitere Software-Bausteine verstanden und geändert werden müssen. Um dieses Problem zu umgehen, besagt das Prinzip der losen Kopplung, dass die Kopplung zwischen Software-Bausteinen möglichst niedrig gehalten werden sollen.

Um eine lose Kopplung in einer Architektur zu erreichen, ist die Einführung von Schnittstellenabstraktionen ein wichtiger Aspekt. Dabei werden die Implementierungsinformationen hinter den Schnittstellen verborgen. Durch die Begrenzung von Schnittstellenelementen und der Häufigkeit des Austauschs der Schnittstellenelemente kann eine Kopplung von Software-Bausteinen kontrollierbar gemacht werden.

Hohe Kohäsion:

Im Gegensatz zur Kopplung, in welcher die Beziehungen zwischen Software-Bausteinen gemeint ist, versteht man unter dem Begriff Kohäsion die Abhängigkeiten innerhalb eines Software-Bausteins.

Beim Prinzip der hohen Kohäsion ist das Ziel die Abhängigkeiten innerhalb eines Software-Bausteins möglichst hoch zu gestalten. Wie bei der losen Kopplung geht es auch hier um die lokale Änderbarkeit und Verständlichkeit von Software-Bausteinen.

Wie in Abbildung 2.7 zu erkennen, stehen Kopplung und Kohäsion normalerweise miteinander in einer Wechselbeziehung. Hierbei gilt, dass je höher die Kohäsion individueller Software-Bausteine einer Architektur ist, desto geringer ist die Kopplung zwischen den Software-Bausteinen. Schematisch ist dieser Zusammenhang in Abbildung 2.7 abgebildet, worin zu erkennen ist, das eine Gesamtstruktur mit einer hohen Kohäsion und einer losen Kopplung (rechte Seite) eine höhere Übersichtlichkeit besitzt.

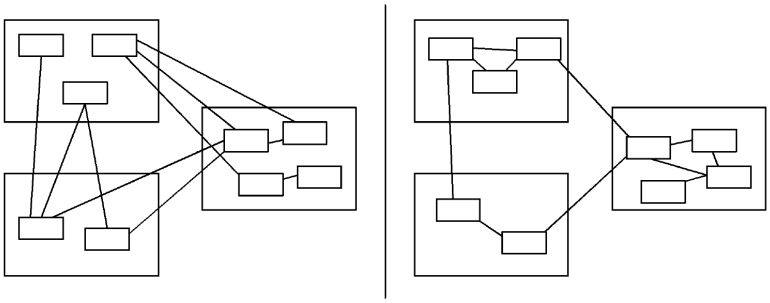

Abb. 2.7: Zusammenspiel von loser Kopplung und hoher Kohäsion - [Vog09]

Entwurf für Veränderung:

Durch den stetigen Wandel von Software-Systemen in Form von Anforderungen und Technologien ist es von Vorteil, solche Änderungen bereits in der Phase der architekturellen Konzeption zu berücksichtigen. Das Prinzip des Entwurfs für Veränderung (englisch: Design for Change) sieht nun vor, dass man vorhersehbare Änderungen architektonisch vorausplant. Dabei sollte man versuchen, die Architektur so zu entwerfen, dass man leicht mit den wahrscheinlichen Änderungen eines Software-Systems umgehen kann.

Separation of Concerns:

Abgeleitet von dem römischen Prinzip „Teile und herrsche“ wird beim Prinzip Separation of Concerns ausgesagt, dass ein Software-System in individuelle Software-Bausteine zerlegt werden soll.

Separation of Concerns unterstützt hierbei die Modularisierung eines Software-Systems. Es geht darum Teile eines Software-Systems zu identifizieren, welche für bestimmte Angelegenheiten, Aspekte und Aufgaben verantwortlich sind. Diese Teile werden dann als eigene Software-Bausteine gekapselt. Eine Zerteilung des Gesamtsystems in relativ unabhängige Einzelteile ermöglicht zudem noch die Verteilung von Verantwortlichkeiten für verschiedene Software-Bausteine. Auch das parallele Arbeiten an dem Software-System durch mehrere Entwickler wird dadurch ermöglicht.

Durch das Aufteilen des Software-Systems in relativ unabhängige Software-Bausteine werden auch die Prinzipien lose Kopplung und hohe Kohäsion begünstigt.

Information Hiding:

Das Prinzip Information Hiding sagt aus, dass man einem Klienten nur die für die Bearbeitung eines Problems notwendigen Informationen zeigen soll. Dies erleichtert die Gliederung und das Verständnis von komplexen Software-Systemen. Die restlichen Informationen sollen nach außen hin verborgen bleiben. Ermöglicht wird solch ein „geheim halten“ von Informationen durch die Bereitstellung von

definierten Schnittstellen, über welche nur bestimmte Informationen zu erreichen sind.

Abstraktion:

Als übergeordnetes Prinzip dient eine Abstraktion dazu, ein komplexes System verständlicher zu machen. Dazu werden wichtige Aspekte identifiziert und unwichtige Details vernachlässigt. Im Bereich der Software-Architektur gilt die Schnittstellenabstraktion als Teilprinzip der Abstraktion. Hierbei liegen die Schnittstellen im Fokus, welche für das Zustandekommen und die Qualität von Beziehungen verantwortlich sind.

Solch eine Schnittstellenabstraktion in einem Software-System ist eng verbunden mit dem Prinzip der losen Kopplung und dem Information Hiding. Ein Aspekt für den starken Zusammenhang zwischen der Abstraktion und dem Information Hiding ist die Portabilität von Software-Systemen. So sollte eine Architektur oder ihre Software-Bausteine auch in anderen Umgebungen verwendbar sein. Um solch eine Plattformunabhängigkeit sicherzustellen, werden Abstraktionen verwendet, die ein Information Hiding der Plattformdetails leisten.

Modularität:

Das Modularitätsprinzip, welches bereits auch in den Beschreibungen der anderen Prinzipien vorkam, definiert die Aufteilung eines Systems in klar definierte Software-Bausteine mit abgegrenzten funktionalen Verantwortlichkeiten. Die Modularität ist dabei eine Kombination aus den Prinzipien Abstraktion, Separation of Concerns und Information Hiding, welche bei der Umsetzung der Prinzipien der losen Kopplung und der hohen Kohäsion kombiniert werden.

Auch für die spätere Konzeption einer Volltextsuche in einer Microservice-Architektur werden die eingeführten Prinzipien als Grundlage dienen.

2.2.2 Monolithische und verteilte Architekturen

Bei der Neugestaltung von E-MES wird von einer monolithischen 3-Schichten-Architektur auf eine verteilte Microservice-Architektur gewechselt.

In einer monolithischen Architektur wird die gesamte Architektur in nur einem Software-Baustein zusammengefasst. Dadurch erfolgt keine explizite Gliederung in Teilsysteme und Architektur-Prinzipien, wie lose Kopplung und Separation of Concerns sind nur schwer umsetzbar [Vog09, S. 216]. Zu finden sind monolithische Architekturen oftmals in Altsystemen, welche oft über Jahrzehnte gewachsen sind. Aufgrund der mangelnden Modularisierung steigt die Komplexität des Systems. Die Wartung und Anpassung des Quellcodes wird erschwert [Pro12a]. Ein weiterer Nachteil der mangelnden Modularisierung ist die kaum mögliche nebenläufige Ausführung von Teilen des Systems

auf verschiedenen Rechnern [Pro12a]. Somit kann eine horizontale Skalierung nicht ermöglicht werden, und eine effiziente, lastverteilende Programmausführung ist nicht gegeben.

Die Architektur von MCC wird eine verteilte Struktur aufweisen. Hierbei werden Teile des Gesamtsystems in unterschiedliche Software-Bausteine aufgeteilt. Eine Modularisierung der Software ist dadurch möglich, und Architekturen-Prinzipien, wie lose Kopplung und Separation of Concerns sind umsetzbar [Vog09]. Durch die strikte Aufteilung der Geschäftslogik kann auch die Komplexität aufgeteilt werden. Somit können die einzelnen Software-Bausteine mit wenig Aufwand angepasst oder erweitert werden. Durch die Modularisierung von verteilten Architekturen kann die Ausführung bestimmter Aufgaben auf redundanter Hardware nebenläufig erfolgen [Pro12b]. Durch horizontale Skalierung kann eine effiziente und lastverteilende Programmausführung erfolgen, welche auch zur Ausfallsicherheit des Gesamtsystems beiträgt [Pro12b].

2.3 Microservice-Architektur

Bei der Neugestaltung des MES der Firma Enisco wird auf eine Microservice-Architektur aufgebaut, welche eine verteilte Struktur aufweist.

Die Kernelemente dieser Architektur sind die Microservices, welche der Modularisierung der Software dienen. Der Begriff Microservice wird hierbei für kleine unabhängige Services verwendet, welche über klar definierte APIs miteinander kommunizieren [Ama21]. Durch die lose Kopplung ist eine einfache Skalierbarkeit und unabhängige Entwicklung durch verschiedene Teams möglich. Dies wiederum verringert die Entwicklungszeit und Komplexität bei Wartung.

Durch die Verwendung von Microservices ist eine Aufteilung des Gesamtsystems in verschiedene Software-Bausteine möglich. Ein Software-Baustein stellt dabei jeweils eine Funktionalität des Gesamtsystems dar. Im Gegensatz zu einer monolithischen Architektur läuft das Gesamtsystem nicht innerhalb eines Prozesses, sondern auf verschiedenen Prozessen. Dabei wird jedem Software-Baustein ein eigener Prozess zugeordnet. Jene Prozesse können nun nahezu beliebig auf verschiedene Rechner verteilt und durch Replizierung ausfallsicher gemacht werden. [Gar18]

Neben den Vorteilen der horizontalen Skalierung ergeben sich aus der Aufteilung des Gesamtsystems in unterschiedliche Software-Bausteine auch Auswirkungen auf die Entwicklungsorganisation. So wird beim Umgang mit Microservices nach der Unix-Philosophie von Ken Thompson „Do one thing and do it well“ [ION21] gearbeitet. Durch die Modularität von Microservices können diese von unterschiedlichen Entwicklerteams unabhängig entwickelt werden. Durch die Abstraktion der Microservices können diese mit unterschiedlichen Technologien und Programmiersprachen implementiert werden. Auch

der Datenhaushalt kann von jedem Microservice separat verwaltet werden. Zudem wird die Einarbeitung eines Entwicklers in die Codebasis reduziert, da durch die Aufteilung weniger Code verstanden werden muss.

Die Microservice-Architektur berücksichtigt die Architektur-Prinzipien Separation of Concerns, Information Hiding und Modularität und gewährleistet somit eine lose Kopplung zwischen den Microservices. Innerhalb der Microservices entsteht dadurch eine hohe Kohäsion.

Da die jeweiligen Microservices repliziert auf verschiedenen Rechnern laufen können, ist die Kommunikation zwischen den Microservices schwieriger als bei einem monolithischen System. Auf die Kommunikation zwischen Microservices wird in Unterabschnitt 2.3.1 näher eingegangen.

Eine Herausforderung bei der Konzeption einer Microservice-Architektur ist die Vermeidung von Abhängigkeiten, welche eine lose Kopplung der Microservices verhindern würden. Um solche monolithischen Seiteneffekte zu vermeiden, werden in Unterabschnitt 2.3.2 die häufigsten Microservice-Anti-Pattern aufgezeigt.

2.3.1 Kommunikation zwischen Microservices

Auch wenn das Ziel einer Microservice-Architektur ist, dass einzelne Funktionalitäten des Gesamtsystems in getrennte Microservices gekapselt werden, müssen diese miteinander kommunizieren. Aufgrund der Modularität können die Microservices horizontal skaliert und auf verschiedenen Rechnern betrieben werden. Dies erhöht die Komplexität bei der Kommunikation der Microservices untereinander. [Mic19]

Bei der Wahl der Kommunikation zwischen Microservices kann zwischen einer synchronen und asynchronen Kommunikation entschieden werden.

Synchrone Kommunikation

Bei der synchronen Kommunikation handelt es sich um eine eins-zu-eins Kommunikation, bei der eine Anfrage geschickt und auf eine Antwort gewartet wird. Klassischerweise erfolgt die Kommunikation über HTTP mit einer REST-Schnittstelle. Representational State Transfer (REST) ist hierbei eine Spezifikation, wie eine über HTTP kommunizierende API konzipiert werden soll [Mic19]. Eine solche API sollte demnach vordefinierte HTTP-Methoden implementiert haben. Unter anderem sind das Methoden wie GET, POST, PUT und DELETE. Bei einer GET-Anfrage werden Ressourcen angefragt. Soll ein neuer Datensatz übermittelt werden, wird die POST-Methode verwendet. Zur Änderung eines bestehenden Datensatzes gibt es die PUT-Methode. Mit der DELETE-Methode kann ein Datensatz entfernt werden. Jeder Datensatz bekommt hierbei einen

eigenen Endpunkt, und die jeweiligen Anfragen können mit URL-Parametern und Query-Parametern spezifiziert werden. Das Standard-Datenformat bei REST ist JSON.

Bei der Kommunikation von zwei Microservices über die jeweiligen REST-Schnittstellen muss die Adresse des anderen Microservice bekannt sein. Durch die Verteilung der Microservices auf unterschiedliche Rechner in Folge einer horizontalen Skalierung kann es während des Betriebes vorkommen, dass einzelne Microservices auf zum Beispiel Rechner A gestoppt und auf Rechner B wieder gestartet werden. Dadurch ändern sich auch die Adressen der Microservices. Da die Verwaltung der Adressen ab einer Vielzahl an Microservices nicht mehr trivial ist, wird in einer Microservice-Architektur eine sogenannte Service-Discovery eingesetzt. Die Service-Discovery ist eine Software, bei der sich alle neuen Microservices registrieren. Bei einem REST-Aufruf wird dann zuerst eine Liste mit allen verfügbaren Adressen abgerufen.

Wie in Abbildung 2.8 dargestellt ist, wird bei einer Anfrage eines Services, diese zunächst an einen Router-Service geleitet. Nach dem Abfragen der Adresse mithilfe der Service-Discovery wird gezielt der entsprechende Service beziehungsweise dessen Instanz mit einer REST-Anfrage angesprochen.

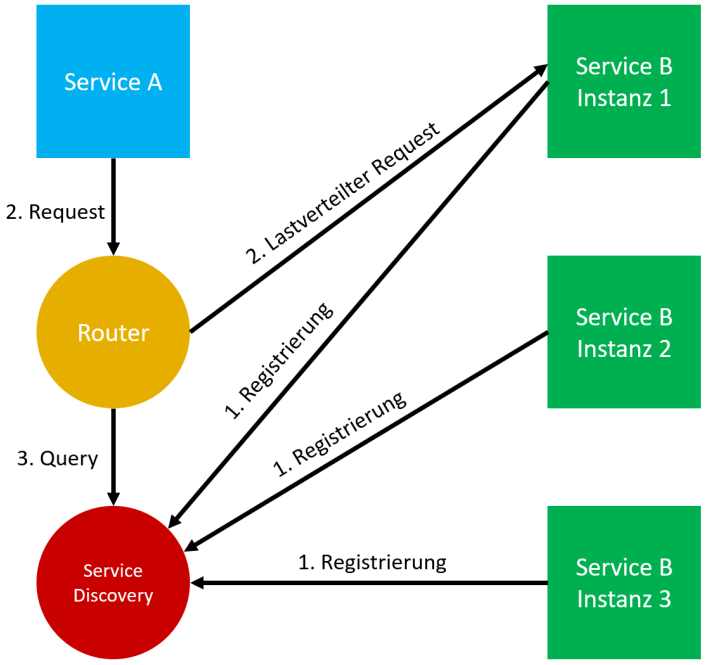

Abb. 2.8: Ablauf einer Anfrage mit der Service-Discovery - in Anlehnung an [Mic19]

Asynchrone Kommunikation

Anders als bei der synchronen Kommunikation wird bei der asynchronen Kommunikation nicht auf jede Anfrage eine Antwort erwartet. Kern einer asynchronen Kommunikation ist ein zentraler Nachrichtenkanal, auf dem Nachrichten ausgetauscht werden. Ein Service schickt Nachrichten an solch einen Nachrichtenkanal, auch Message Broker

genannt, und wenn ein anderer Service an der Nachricht interessiert ist, kann er diese konsumieren. Eine Nachricht kann dabei auch von mehreren Services konsumiert werden.

Die Verantwortlichkeiten innerhalb des Systems werden bei der Verwendung eines Message Brokers umgedreht. Ein Service wird nun nicht mehr explizit von einem anderen Service aufgerufen, sondern schickt seine Nachrichten an den Message Broker, ohne zu wissen, wer diese Nachrichten konsumiert. Sender und Empfänger sind somit lose gekoppelt.

Zur Einteilung der Nachrichten werden in Message Brokern sogenannte Message Queues verwendet. Über eine Message Queue, welche einen bestimmten Topic besitzt, können Services dann entweder Nachrichten senden oder sich als Konsumenten an diesem Topic registrieren. In Abbildung 2.9 ist die Funktionsweise beispielhaft an dem Message Broker „Apache Kafka“ abgebildet. Durch den Publish/Subscribe-Mechanismus kann Service A in diesem Beispiel sowohl Publisher von zwei Topics sein, als auch Konsument von einem Topic.

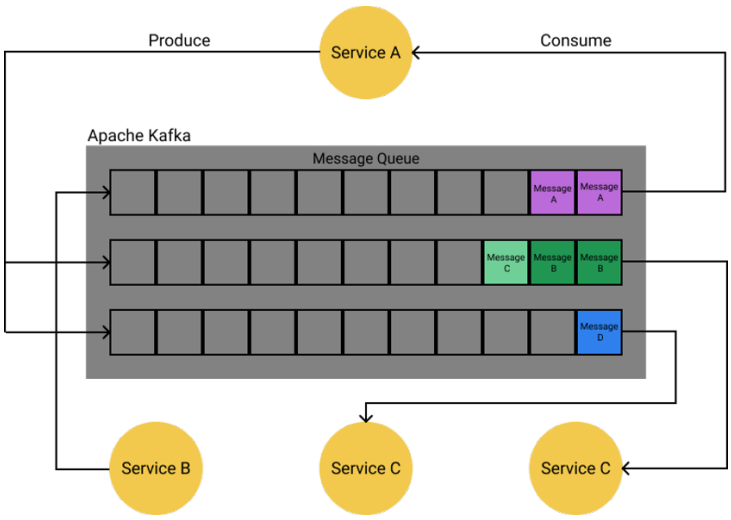

Abb. 2.9: Funktion eines Message Brokers - [Mic19]

Die Nachrichten werden in der Message Queue zwischengespeichert, bis die registrierten Konsumenten diese Nachrichten gelesen haben. Wenn ein Service sich an einer Message Queue registriert, können verschiedene Servicequalitäten für die Übertragung der Nachrichten angegeben werden. Folgende drei „Quality of Service (QoS)“ können angegeben werden: [Apa21b]

QoS 0 „At most once“:

Eine Nachricht wird genau einmal versendet. Nach dem Prinzip „fire and forget“ wird keine Rückbestätigung erwartet. Verwendet wird diese Servicequalität meist für Nachrichten, welche mit einer hohen Frequenz versendet werden. So ist es für viele Sensoren in einer IoT-Umgebung nicht erforderlich, dass jeder gemessene Wert erfolgreich übermittelt wird.

QoS 1 „At least once“:

Es wird sichergestellt, dass eine Nachricht minimal einmal beim Empfänger ankommt. Hierzu werden die Nachrichten beim Sender solange zwischengespeichert, bis vom Empfänger eine Bestätigung (PUBACK) empfangen wird. Wird über eine gewisse Zeitspanne keine Bestätigung empfangen, wird die Nachricht erneut versendet. Hierbei kann es vorkommen, dass ein Empfänger eine Nachricht mehrfach erhält. [Flo17]

QoS 2 „Exactly once“:

Eine Nachricht wird genau einmal zugesendet. Hierbei wird ein 4-Wege-Handshake zwischen Sender und Empfänger versendet. Durch diesen Ablauf ist eine sichere Übertragung der Nachricht gewährleistet. Durch den großen Overhead ist jedoch diese Servicequalität die langsamste Übertragungsmöglichkeit.

2.3.2 Microservice - Anti-Pattern

Um die Vorteile einer Microservice-Architektur im Bezug auf lose Kopplung und der damit eingehenden Unabhängigkeit der Microservices bezüglich Wartbarkeit und Erweiterbarkeit zu gewährleisten, ist es von Nöten die möglichen Anti-Pattern einer Microservice-Architektur zu kennen. Mit Hilfe von Anti-Pattern können nach Roth und Hafner [SM19] wiederkehrende Fehler bei der Softwareentwicklung identifiziert und in generalisierter Form dokumentiert werden. So können Anti-Pattern dabei helfen häufig auftretende Fehler von vornherein zu vermeiden.

In einer Studie von Tighilt et al. [TAM+20] wurden 27 Paper und 67 Open-Source Anwendungen, welche alle Bezug zur Microservice-Architektur haben, untersucht. Die Ergebnisse der Studie [TAM+20] ergeben eine Sammlung von häufig aufgetretenen Anti-Pattern, welche durch Tighilt et al. in vier Kategorien unterteilt wurden. Folgend werden die Kategorien mit Beispielen erläutert:

Design Anti-Pattern

In diese Kategorie gehören Anti-Pattern, welche bereits in der architektonischen Entwurfsphase einer Microservice-Architektur zu Fehlern führen. Beispiele sind hierfür das „Falscher Schnitt“ - Anti-Pattern und das „Zyklische Abhängigkeiten“ - Anti-Pattern.

Falscher Schnitt:

Beim „Falscher Schnitt“ - Anti-Pattern wird die Aufteilung des Gesamtsystems in unterschiedliche Microservices nicht nach Geschäftsfunktionen, sondern nach technischen Aspekten durchgeführt. Ähnlich wie bei einer monolithischen 3-Schichten-Architektur übernehmen dann ein Teil der Microservices die Präsentationsschicht, andere Microservices wiederum bedienen die Geschäftsschicht und

die anderen bedienen die Datenzugriffsschicht. Durch dieses Vorgehen verliert die Architektur die lose Kopplung zwischen den Microservices, und die Modularität geht verloren. Auch müssen bei einer Änderung einer Geschäftslogik mehrere Microservice abgeändert werden, was zu einem Mehraufwand führt.

Verhindert werden kann dieses Anti-Pattern, indem man die Trennung des Gesamtsystems anhand der Geschäftsfunktionen durchführt. Jeder Microservice sollte von einem Team implementiert und gewartet werden können. Die Abhängigkeiten zu anderen Microservices sollten so gering wie möglich ausfallen, um eine lose Kopplung und die Modularität der Architektur zu gewährleisten.

Zyklische Abhängigkeiten:

Ein weiterer Fehler ist die Missachtung der losen Kopplung, indem man mit zyklischen Abhängigkeiten eine starke Gebundenheit an andere Microservices verursacht. Dadurch müssen die abhängigen Microservices sowohl bei der Implementierung als auch bei der Wartung berücksichtigt werden. Dies erhöht die Komplexität und die Wahrscheinlichkeit, dass Fehler auftreten.

Eine zyklische Abhängigkeit zwischen solchen Microservices kann verhindert werden, indem man stark voneinander abhängige Microservices in einen gemeinsamen Microservice zusammenfasst.

Implementation Anti-Pattern

Implementation Anti-Pattern entstehen aus der Art und Weise wie Microservices implementiert werden. Beispiele sind hierfür das „Festcodierte Endpunkte“ - Anti-Pattern.

Festcodierte Endpunkte:

Bei der synchronen Kommunikation zwischen Microservices werden in der Regel Anfragen über REST APIs gestellt. Hierfür müssen die IP-Adresse und der Port des anderen Microservices bekannt sein. Ein Ansatz für schnelle Laufzeiten und eine einfache Implementierung ist es, die Endpunkte fest im Quellcode zu hinterlegen. Dieser Ansatz ist jedoch bei einer größeren Anzahl von Microservices nicht mehr wartbar. Sobald sich ein Endpunkt eines Microservice ändert, müssen alle abhängigen Microservices geändert und neu deployed werden.

Abhilfe bringt die Einführung einer Service-Discovery (siehe auch Abschnitt 2.3.1), bei welcher sich die einzelnen Services registrieren. Ein anfragender Service kann nun über diese Service-Discovery die aktuelle Adresse des anderen Services erhalten.

Deployment Anti-Pattern

Auch in der Bereitstellungsphase einer Microservice-Architektur können Fehler auftreten. Diese Anti-Pattern werden Deployment Anti-Pattern genannt. Beispiele sind hierfür das „Manuelle Konfiguration“ - Anti-Pattern und das „Zeitüberschreitungen“ - Anti-Pattern.

Manuelle Konfiguration:

In einer Microservice-Architektur müssen aufgrund der Verteilung der Microservices auf unterschiedliche Rechner, verschiedene Konfigurationen durchgeführt werden. Eine manuelle Konfiguration von jedem Microservice führt zu unzähligen Konfigurationsdateien und Abhängigkeiten zu Umgebungsvariablen. Mit wachsender Anzahl von Microservices führt die manuelle Konfiguration zu einer ansteigenden Komplexität bei der Wartung der Konfigurationen.

Durch die Einführung eines Konfigurations-Servers kann dieses Anti-Pattern vermieden werden. Durch geeignete Konfigurationsmanagement-Tools können sämtliche Konfigurationen zentral verwaltet werden.

Zeitüberschreitungen:

In einer verteilten Microservice-Architektur kann es vorkommen, dass einzelne Services ausfallen und nicht mehr zur Verfügung stehen. Hat nun ein anderer Microservice eine Anfrage über eine REST API gestellt, kann es vorkommen, dass unter Umständen gar keine Antwort kommt. Hierfür kann der Entwickler Timeouts festlegen, nach welchen einem anfragenden Service mitgeteilt wird, dass ein anderer Service nicht erreichbar ist. Den richtigen Wert für solch einen Timeout zu wählen, hängt von der Art des Service ab. Ein zu kurzer Timeout kann dazu führen, dass langsame Service frühzeitig beendet werden. Bei einer zu langen Wahl des Timeouts kann es vorkommen, dass der Endnutzer zu lange warten muss, bis ihm signalisiert wird, dass der gewünschte Service nicht verfügbar ist.

Um dem entgegen zu wirken, verwendet man in einer Microservice-Architektur üblicherweise einen Circuit Breaker. Ein Circuit Breaker überwacht hierbei alle Instanzen einzelner Microservices und stellt fest, ob einzelne Timeouts bei der Kommunikation mit der jeweiligen Instanz überschritten wurden. Da solch ein Circuit Breaker als Proxy für alle Kommunikationskanäle verwendet wird, werden alle Anfragen an einen ausgefallenen Service mit einer Fehlermeldung beantwortet. Die Anfragen werden solange durch den Circuit Breaker blockiert, bis der ausgefallene Service wieder zur Verfügung steht. Auch der Timeout des Circuit Breaker muss vom Entwickler festgelegt werden.

Monitoring Anti-Pattern

Zu dem Betrieb einer Microservice-Architektur gehört auch die Überwachung der Änderungen im System. Hierbei kann es auch zu häufig auftretenden Anti-Pattern

kommen. Beispiele sind hierfür das „Kein Gesundheitscheck“ - Anti-Pattern und das „Lokale Protokollierung“ - Anti-Pattern.

Kein Gesundheitscheck:

In der Natur der Microservices kann es vorkommen, dass einige Services über gewisse Zeitspannen nicht erreichbar sind. Um ein kompletten Systemausfall zu vermeiden, dienen Mechanismen, wie der Circuit Breaker, dazu die anfragenden Services über das Überschreiten von Timeouts zu benachrichtigen.

Mit der Integration von Gesundheitschecks können solche Ausfälle von einzelnen Services im Voraus erkannt und abhängige Microservices darüber informiert werden, dass das Senden von Anfragen zu unterlassen ist. Für solche Gesundheitschecks werden API-Endpoints angeboten, welche periodisch abgefragt werden können.

Lokale Protokollierung:

Um eine spätere Fehlerdiagnose zu ermöglichen, speichern Microservices einige Informationen über Änderungen in Log-Dateien. Ein häufiger Fehler ist, dass solch eine Protokollierung lokal in den Datenbanken der einzelnen Microservices gespeichert werden. Die Herausforderung liegt nun bei der Analyse und Abfrage von Logs aus verschiedenen Microservices.

Mit einem verteilten Logging-Mechanismus können die Logs der unterschiedlichen Microservices in einem zentralen Speicherort gespeichert werden. Durch eine einheitliche Formatierung der Log-Einträge kann auch deren Abfrage und Analyse vereinfacht werden.

Auch bei der Integration einer Suchfunktion in eine Microservice-Architektur gilt es die vorgestellten Anti-Pattern zu beachten, um so monolithische Seiteneffekte zu vermeiden.

Kapitel 3

3 Suchumfang in MCC

Aufbauend auf die Vorstellung verschiedener Möglichkeiten für die Umsetzung einer Suchfunktionalität aus Abschnitt 2.1, wird in folgendem Kapitel der Suchumfang für die unterschiedlichen Möglichkeiten aufgezeigt.

Als Basis wird sich am Funktionsumfang der zukünftigen MES-Lösung „MCC“ orientiert. Der Funktionsumfang wird der bestehenden MES-Lösung „E-MES“ gleichen und ist derzeit noch in der Entwicklungsphase.

Folgend wird somit nach einer Einführung in den zukünftigen Funktionsumfang von „MCC“ auch definiert, welche Informationen über die Suchfunktionalität gesucht werden können. Betrachtet werden die Suchfunktionalität „Volltextsuche“, „facettierte Suche“ und „semantische Suche“.

3.1 Funktionsumfang von MCC

Bei der Neugestaltung der bestehenden MES-Lösung „E-MES“ wird neben dem Produktnamen hauptsächlich der architekturelle Aufbau der Software angepasst. Der Funktionsumfang von „MCC“ wird sich dem jetzigen Funktionsumfang von „E-MES“ angleichen und ihn in Zukunft zusätzlich erweitern.

Die ersten Planungen für die Aufteilung der Funktionen in „MCC“ sehen eine Aufteilung in vier unterschiedliche Schichten vor. Niedrigere Schichten sollen dabei nicht von höheren Schichten abhängig sein und können somit ohne höhere Schichten existieren. Folgend werden die vier Funktions-Schichten „MCC Platform“, „Core Services - Production“, „PCS“ und „SCADA“ erläutert, und es werden die darin enthaltenen Funktionen betrachtet.

In Abbildung 3.1 sind die verschiedenen Schichten abgebildet. Zu erkennen ist, dass die Funktionalitäten in der „MCC Platform“ - Schicht noch keine Anwendungsfunktionalitäten beinhalten. Enthalten sind in der „MCC Platform“ Funktionalitäten, welche die Grundfunktionen für den reibungslosen Betrieb einer auf Microservice basierenden Architektur ermöglichen. „MCC Platform“ bildet damit die niedrigste Schicht und kann unabhängig von den darüber liegenden Schichten betrieben werden. Aufbauend auf den Grundfunktionalitäten können verschiedene Anwendungen betrieben werden. Im Umfeld der Firma Enisco sind dies Anwendungen für den Betrieb von Produktionsanlagen. Aber

auch branchenferne Anwendungen sind durch die Grundfunktionalitäten der „MCC Platform“ - Schicht umsetzbar.

Im Umfeld von Produktionsanlagen sind die Funktionalitäten von MCC zusätzlich in die Schichten „Core Services - Production“, „PCS“ und „SCADA“ unterteilt. In Abbildung 3.1 ist dabei zu erkennen, dass die „Core Services - Production“ - Schicht eine Basisschicht für die „PCS“ und „SCADA“ - Schichten darstellt. Enthalten sind Funktionalitäten, welche für den Betrieb von übergeordneten Schichten benötigt werden.

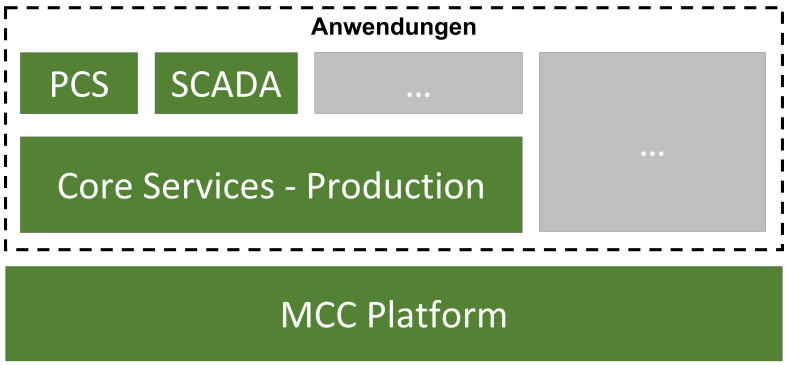

Abb. 3.1: Übersicht der verschiedenen Schichten von MCC

Im Folgenden werden die vier verschiedenen Schichten mit den darin enthaltenen Funktionalitäten abstrahiert erläutert.

3.1.1 MCC Platform

Die Schicht „MCC Platform“ fungiert als Grundschicht für die Anwendungsentwicklung und stellt die Grundfunktionalitäten für die Erstellung und das Betreiben der Anwendungsschichten zur Verfügung.

Zu den Grundfunktionalitäten zählen unter anderem die Themen:

Internationalisierung:

Um mit einer Software auch eine internationale Kundschaft zu bedienen, ist es erforderlich die Themen Lokalisierung und Internationalisierung zu berücksichtigen. Hierbei geht es darum, mit den länderspezifischen Formaten für Daten und Zahlen und mit unterschiedlichen Zeitzonen umgehen zu können. Auch die Übersetzung der Texte in die jeweilige Kundensprache wird unter dem Thema Internationalisierung berücksichtigt.

Migrations-API:

Aufgrund der späteren Weiterentwicklung der Module kann es bei neuen Versionen dazu kommen, dass sich die Strukturen der Daten in den Datenbanken geändert haben. Bei der Aktualisierung der Module müssen deswegen die bestehenden Daten in den Datenbanken in die neue Struktur migriert werden.

Durch die Umsetzung einer Migrations-API wird nun für jedes Modul eine versionierte Liste mit allen inkrementellen Migrationen abgespeichert. In Kombination mit einem Protokoll über bereits durchgeführte Migrationen kann so eine automatische Ausführung der Migrationen gewährleistet werden.

Security:

Produktionsleitsysteme, wie „E-MES“, haben vermehrt einen komplexen Funktionsumfang, welcher bei falscher Bedienung zu hohen Kosten oder Beschädigungen der Anlage führen kann. Aus diesem Grund ist es erforderlich, den Zugriff auf bestimmte Aktionen nur für fachkundiges Personal zugänglich zu machen. So können ganze Funktionen für bestimmte Benutzergruppen entweder gesperrt oder durch Weglassen von Bedienelementen angepasst werden.

Beim Thema „Security“ geht es demnach um die Authentifizierung und Autorisierung der Benutzer. Neben verschiedenen Authentifizierungsmöglichkeiten müssen Autorisierungsmechanismen wie Rechtepaketen, Benutzerrollen sowie deren Verwaltung zur Verfügung gestellt werden.

Streaming Base:

Nachrichten, welche aufgrund der Kommunikation mit mehreren Topics und Datenhaltungschichten nicht mehr als atomare Nachrichtenübermittlungen angesehen werden können, sollten als eine Transaktion angesehen werden. Die Streaming Base stellt hierbei sicher, dass alle Nachrichtenübermittlungen innerhalb einer Transaktion rückgängig gemacht werden können, sobald eine der Nachrichten fehlschlägt. So wird innerhalb des gesamten Systems eine Datenkonsistenz sichergestellt.

Der Austausch von Nachrichten zwischen Microservices wird über Kafka-Topics des Message Brokers „Apache Kafka“ koordiniert. Der funktionale Ablauf der Nachrichtenweitergabe besteht dabei grundsätzlich aus drei Schritten:

- **Konsumieren** der Nachrichten aus teils mehreren Kafka-Topics
- **Verarbeiten** der Nachrichten (Interaktionen mit den persistenten Speichern oder mit anderen Microservices)
- **Produzieren** der Nachricht an teils mehrere Kafka-Topics

Unabhängig von der späteren Geschäftslogik wird in der „Streaming Base“ die Kernimplementierung für den Nachrichtentransport umgesetzt.

Entity Model:

In einem Produktionsleitsystem wie „E-MES“, existieren verschiedene Geschäftsentitäten. Die Entitäten mit ihren Attributen unterscheiden sich dabei von Anlage zu Anlage. Deswegen ist es notwendig, eine Möglichkeit der Konfiguration bereitzustellen. Durch Import- und Export-Funktionalitäten soll das Austauschen von Entitätensammlungen vereinheitlicht werden.

Zusätzlich soll das MCC Entity Model eine Abstraktion der Datenhaltungsschicht bereitstellen, sodass verschiedene Datenbanken zum Persistieren verwendet werden können. Über standardisierte API's sollen die jeweiligen Entitäten anschließend abgefragt bzw. bearbeitet werden.

Für den Kontext einer Suchfunktionalität sind die meisten Funktionalitäten aus der „MCC Platform“ - Schicht nicht relevant, da ein Benutzer mit diesen Funktionen nicht direkt interagiert. Zum Beispiel soll die Funktions- und Arbeitsweise der „Streaming Base“ dem Benutzer verborgen bleiben.

Geeignet für den Kontext einer Suchfunktionalität sind die Objekte und Informationen aus dem „Security“ - Modul. Dabei kann zum Beispiel nach den unterschiedlichen Benutzerrollen, den damit verbundenen Rechtepaketen oder diversen Benutzerrechten gesucht werden. So kann durch die Suche nach einem Benutzernamen auch die jeweiligen Rechtepakete gefunden werden. Durch die Verwendung eines Sicherheitsaudits, können die sicherheitsrelevanten Aktivitäten nachverfolgt werden.

3.1.2 Core Services - Production

Aufbauend auf den Grundfunktionalitäten der „MCC Platform“, welche für den Betrieb einer auf Microservices basierenden Architektur benötigt werden, stellt die „Core Services“ - Schicht die Grundfunktionen für die eigentliche Anwendung zur Verfügung. Im Umfeld von Produktionsanlagen werden Grundfunktionen für die anwendungsspezifischen Schichten „PCS“ und „SCADA“ definiert. Oftmals handelt es sich dabei um Funktionen, welche von mehreren darüber liegenden Schichten verwendet werden.

Im Falle des Funktionsumfangs des Produktionsleitsystems „E-MES“ handelt es sich in dieser Schicht um Funktionen, welche für die übergeordneten Schichten „PCS“ und „SCADA“ von Nutzen sind.

Zum Funktionsumfang zählen zum Beispiel die folgenden Funktionalitäten:

Shift Model:

Dem Schichtenmodel sind Funktionalitäten für das Anlegen und Verwalten von Schichten zugeordnet. Hierbei sind die Schichten Bestandteil von vielen weiteren Entitäten. So wird zum Beispiel im SCADA-Bereich angegeben, in welcher Schicht gewisse Sensorwerte aufgezeichnet wurden und zu welcher Schicht Fehler- oder Warnmeldungen erschienen sind. Auch im Bereich der Qualitätskontrolle ist die Angabe der Schicht beim Entdecken von Mängeln für eine spätere Nachverfolgung notwendig.

Zu den Schichten gehört auch ein Schichtlogbuch, in welchem die Benutzer die Möglichkeit haben beliebige Freitexte zu den einzelnen Schichten einzugeben. Durch die Verwendung von Vorlagen haben die Benutzer zusätzlich die Möglichkeit

die Kategorie der Freitexte festzulegen. So können auch Schadensberichte oder Qualitätsberichte erstellt und den spezifischen Schichten zugeordnet werden.

Asset Model:
Um auch große Anlagen übersichtlich zu strukturieren, werden Anlagen üblicherweise in sogenannte „Assets“ unterteilt. Hierbei kann der Begriff „Asset“ von den einzelnen Maschinen und Sensoren bis hin zur kompletten Anlage reichen. Eine tiefergehende Erläuterung der „Assets“ bezüglich dem Einsatz im Umfeld eines Produktionsleitsystems ist in Unterabschnitt 3.1.3 **(SCADA)** zu aufgeführt.

Maschinenkonnektivität:
Die späteren Anlagen, auf welchen MCC betrieben wird, sind sehr Vielfältig. Neben dem Alter der Maschinen und Sensoren gibt es auch eine breite Masse an verschiedenen Herstellern, die überwiegend eigene Protokolle und Schnittstellen anbieten. Damit solche Anlagen mit MCC verbunden und betrieben werden können, bedarf es einer Funktionalität, welche für die Abstaktion der einzelnen Maschinen- und Sensor-Schnittstellen zuständig ist.

Da die Funktionalitäten des Core-Services Grundlagen für die übergeordneten Schichten „SCADA“ und „PCS“ sind, besteht eine hohe Relevanz bezüglich dem Suchkontext einer Suchfunktionalität. Durch Eingabe, zum Beispiel einer Asset-Bezeichnung, sollen dem Benutzer alle mit dem Asset verknüpften Objekte und Ereignisse aufgelistet werden.

In Bezug auf Schichten eignen sich vor allem die Freitexte der Schichtlogbücher für den Suchkontext einer Volltextsuche. Hierbei sollen dem Benutzer nach Eingabe einer Suchphrase die relevantesten Einträge aller Schichtlogbücher ausgegeben werden.

3.1.3 SCADA

Damit MCC im Umfeld von Produktionsleitsystemen vertrieben werden kann, sind die Funktionalitäten aus der „SCADA“-Schicht notwendig. Die „SCADA“-Schicht zählt somit zu den anwendungsspezifischen Schichten.

Bei Supervisory Control and Data Acquisition (SCADA) geht es um die Überwachung und Steuerung von technischen Prozessen in automatisierten Fertigungen [Col20]. Dabei werden Daten von Maschinen und Sensoren empfangen und somit deren Zustand überwacht. Durch das Sammeln und Speichern solcher Daten über einen längeren Zeitraum hinweg kann eine statistische Auswertung der verschiedenen Maschinen und Sensoren angefertigt werden. Mithilfe solcher Auswertungen können, durch die frühzeitige Erkennung von Unregelmäßigkeiten, Stillstände in der Anlage vermieden werden.

Ein SCADA-System besteht dabei häufig aus einer Kombination von Software- und Hardware-Elementen [cop21]. Die unterste Ebene der Datenerfassung ist die speicherprogrammierbare Steuerung (SPS). Eine SPS interagiert hierbei direkt mit den Maschinen

und Sensoren und leitet die gesammelten Daten an die nächsthöheren Ebenen weiter. In industriellen Anlagen sind dies oftmals Leitstände, bei welchen die Benutzer die Möglichkeit haben, mithilfe von einem Human-Machine-Interface (HMI), die Steuerung zu überwachen. Eine HMI-Schnittstelle ist ein Bestandteil eines SCADA-Systems und dient zu Visualisierung und Bereitstellung einer Interaktionsmöglichkeit zwischen den Benutzern und dem System. Ein Beispiel für eine Visualisierung ist in Abbildung 3.2 abgebildet. Sobald eine Verbindung mit den einzelnen SPSen hergestellt wurde, werden in dieser Ansicht die Echtzeitdaten aus dem jeweiligen Bereich der Anlage angezeigt.

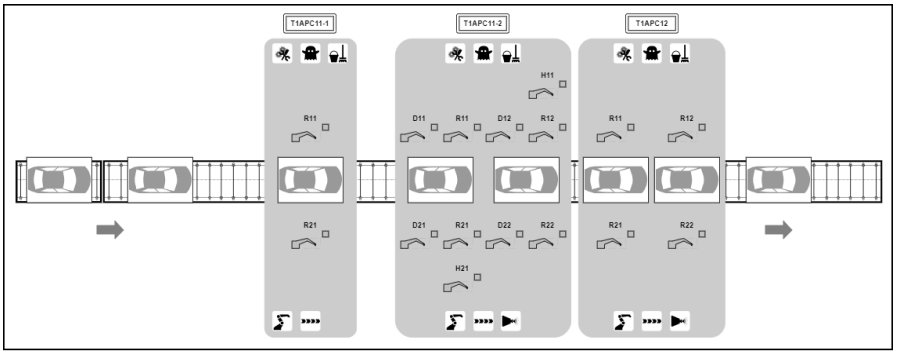

Abb. 3.2: Beispiel für eine HMI in einer SCADA-Umgebung

Für den Suchkontext einer Suchfunktionalität im SCADA-Umfeld, gibt es zwei mögliche Sichtweisen auf die Fertigungsanlage:

Technische Sicht:

Industrielle Fertigungsanlagen durchlaufen während der Lebenszeit mehrere Stadien. Darunter zählt Stadien wie Planung, Entwurf, Realisation, Betrieb, Instandhaltung und Demontage [Ste19]. Um physische Objekte innerhalb des Systems und über alle Lebensstadien hinweg eindeutig identifizieren zu können, wird ein einheitliches Kennzeichnungssystem benötigt.

Als Referenz für solch ein einheitliches Kennzeichnungssystem dient die Norm „EN 81346“ [Ste19]. Seit Mai 2005 dient die Norm als einheitliches Kennzeichnungssystem für physikalische Objekte in den Bereichen Elektrotechnik, Maschinenbau und Verfahrenstechnik. Berücksichtigt werden auch Objekte ohne elektrische Relevanz. So werden Objekte, wie Ventile genauso berücksichtigt, wie Schalter und Sensoren. Ziel ist es, dadurch das technische System als Gesamtheit zu betrachten [Ste19].

Ein beispielhafter Auszug aus der Norm „EN 81346“ ist in Abbildung 3.3 abgebildet. Gezeigt werden die drei Hauptaspekte „Produktaspekt“, „Funktionsaspekt“ und „Ortsaspekt“, welche der Strukturierung der physikalischen Objekte dienen [Ste19]. Den einzelnen Aspekten werden unterschiedliche Vorzeichen zugeordnet. So können Objekte auf zum Beispiel Schaltplänen eindeutig bezeichnet werden.

Nr.	Vorzeichen	Bezug	Beispiel	Benennung
Funktionsaspekt	=	Erfüllung einer Funktion	Ausschaltverzögerung eines Lüfters	=GQ2
Produktaspekt	-	Benennung eines Objektes	Abfallverzögertes Relais	-KF4
Ortsaspekt	+	Ort des Objektes	Nebengebäude, Herrentoilette	+Z2W3

Abb. 3.3: Gliederung der Vorzeichen in der EN 81346 [Ste19]

Asset Sicht:

Es gibt weiterhin die Möglichkeit, die Anlage als eine Menge von „Assets“ zu interpretieren. Im Gegensatz zur technischen Sichtweise auf die Anlage können Assets auch gruppiert werden. Es ist zum Beispiel nicht immer notwendig, auf der Ebene der einzelnen Maschinen und Sensoren zu interagieren. Die Notwendigkeit den Zustand von zum Beispiel einem hydraulischem Ventil zu erfassen, ist im Falle der Wartungsdokumentation gerechtfertigt. Jedoch aus Sicht eines Mitarbeiters, der einen großen Teil der Anlage überwachen und steuern muss, ungeeignet.

Es ist hilfreich die Assets nach physikalischen Eigenschaften zu gruppieren. So können zum Beispiel verschiedene Antriebsmotoren und Sensoren als „Rollenbahn“ zusammengefasst werden und dem Mitarbeiter als ein einzelnes Asset dargestellt werden. Die Gruppierung wird dabei solange weitergeführt, bis das Assest „Produktionsanlage“ erreicht wird. So können mehrere „Rollenbahnen“ Bestandteile einer „Bearbeitungsstation“ sein. Und mehrere „Bearbeitungsstationen“ können zu „Anlagen-Bereichen“ gruppiert werden. Die dadurch entstehende Hierarchie von Assets kann den kompletten Aufbau der Produktionsanlage widerspiegeln.

Neben streng hierarchischen Beziehungen zwischen Assets können auch logische Beziehungen von Relevanz sein. So ist es möglich, dass mehrere Geräte am selben Bedienfeld angeschlossen sind und darüber eine Beziehung zueinander haben. Auch eine Gruppierung über die Zugehörigkeit zu Not-Aus-Schaltern oder Schaltschränken für die Stromversorgung kann von Relevanz sein. Eine weitere Beziehung zwischen den Assets sind die Materialflüsse, welche beschreiben, wie Produktionseinheiten durch die Produktionsanlage bewegt werden.

Für den Kontext einer Suchfunktionalität finden sich im SCADA-Umfeld von MCC mehrere Objekte und Informationen, welche relevant für eine Suche sind.

Durch die hierarchische und logische Aufteilung der Produktionsanlage in verschiedene Assets ist eine Suche nach expliziten Assets ein relevanter Anwendungsfall und soll durch die Suchfunktionalität in MCC abgedeckt werden. So kann ein Mitarbeiter zum Beispiel nach den Begriffen „Lackierbereich“, „Trockenofen“ oder „Rollenbahn“ suchen und bekommt eine Auswahl an Funktionen angeboten, welche mit Bezug zu den Assets

ausgeführt werden können. Sollte demnach ein Asset mit der Bezeichnung „Rollenbahn“ im System zu finden sein, sollen dem Mitarbeiter Funktionalitäten, wie die Anzeige von Fehlermeldungen (Alarming) und Auswertung von Prozesswerten (Trending) angeboten werden. Auch die explizite Suche nach bestimmten Schaltschränken oder Pultbereichen kann durch die logische Hierarchie der Assets realisiert werden.

Neben der Verwendung der Assets kann zusätzlich nach dem technischen Kennzeichnungssystem der Maschinen und Sensoren gesucht werden. So kann nach der Eingabe einer Kodierung durch den Benutzer eine Auflistung mit allen relevanten Attributen aufgelistet werden. Darunter zählt zum Beispiel die Zugehörigkeit der Maschine oder des Sensors zu Anlagenbereichen, Pultbereichen oder Schaltschränken.

3.1.4 PCS

Neben der „SCADA“-Schicht ist die „PCS“-Schicht eine weitere anwendungsspezifische Schicht im Funktionsumfang von MCC.

Bei einer Production Control System (PCS) - Software geht es um die Aufbereitung der Produktionsaufträge für die jeweilige Anlage. Die eigentlichen Produktionsaufträge, welche in der Produktionsanlage ausgeführt werden sollen, werden von den kundenspezifischen ERP-Systemen eingespeist. Durch die Verwendung von vordefinierten Templates besteht dann die Möglichkeit, die Produktionsaufträge automatisiert in verschiedene Jobs zu unterteilen. Für die direkte Anlagensteuerung ist eine weitere Aufteilung in nahezu atomare Arbeitsschritte notwendig. Durch die Einbeziehung von Anlagenressourcen und einer intelligenten Planungssoftware werden die einzelnen Arbeitsschritte schließlich koordiniert. Exemplarisch ist solch eine Ressourcenplanung in Abbildung 3.4 abgebildet. Bei der Ressourceplanung wird festgelegt, zu welchem Zeitpunkt ein Bauteil mit seinem passenden Warenträger bei einem bestimmten Arbeitsschritt vorliegt. Mithilfe solch einer Planung wird versucht eine optimale Auslastung der Anlage zu erreichen.

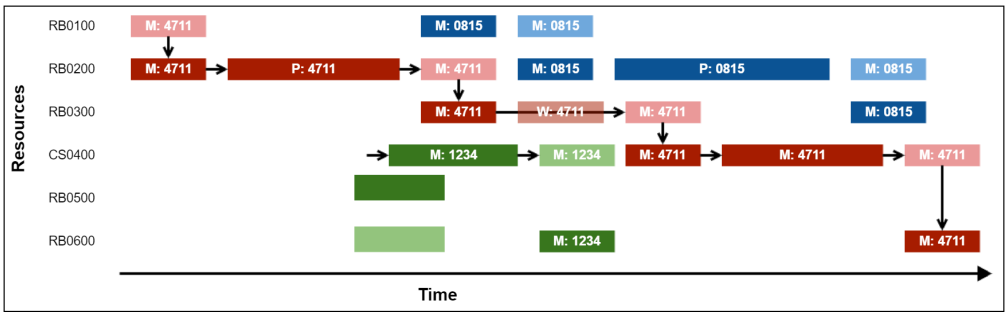

Abb. 3.4: Exemplarische Ressourcenplanung [Eni21b][1]

Eine weitere Funktionalität von PCS ist das Erstellen einer digitalen Lebenslaufakte für die jeweiligen Produktionsaufträge. Eine Lebenslaufakte ist im Umfeld der digitalen

[1] Quelle aus dem Intranet (nicht öffentlich zugänglich) der Enisco by Forcam GmbH

Vernetzung ein Werkzeug zur lückenlosen Dokumentation des Produktlebenszyklus eines industriellen Produktes [Max21]. Innerhalb einer solchen Akte werden alle anlagenbezogenen Informationen gesammelt und aktuell gehalten. Sie dient als Basis für industrielle Kollaborationen zwischen Anlagenherstellern und IT- und Servicedienstleistern und ist durch die Vielseitigkeit und Anpassungsfähigkeit in den unterschiedlichsten Produktlebenszyklen einsetzbar [Max21]. Für die Standardisierung einer solchen Lebenslaufakte ist im September 2018 die Norm „EN 77005-1“ in Kraft getreten. Innerhalb von MCC wird die Lebenslaufakte sowohl durch die Funktionalitäten der PCS-Schicht, als auch durch die Funktionalitäten der SCADA-Schicht aktuell gehalten. So können neben allgemeinen Informationen zum Produkt auch Werte gespeichert werden, welche bei den einzelnen Arbeitsschritten entstanden sind. So ist eine nachträgliche Auswertung der einzelnen Arbeitsschritte des jeweiligen Produktes möglich.

Für den Kontext einer Suchfunktionalität in MCC sind die Informationen aus den einzelnen Lebenslaufakten von hoher Relevanz. Abhängig von der späteren, tatsächlichen Umsetzung wird jedem Produktionsauftrag und somit jeder Lebenslaufakte eine eindeutige Identifikationsnummer zugeordnet. Mithilfe einer solchen Identifikationsnummer soll es für den Benutzer von MCC möglich sein, unter anderem den Status des Produktionsauftrags und eine Auflistung von noch ausstehenden Arbeitsschritten zu erhalten.

Neben gezielten Informationen zu einzelnen Produkten und den dazugehörigen digitalen Lebenslaufakten sind unter anderem auch Informationen zu den einzelnen Jobs und deren Arbeitsschritte von Relevanz. Hierfür besitzen die einzelnen Prozesse wiederum eindeutige Identifikationsnummern. Relevant ist zudem eine gezielte Suche nach den Ressourcen der Produktionsanlage. So kann ein Benutzer eine Identifikationsnummer eines Warenträgers eingeben und erhält alle relevanten Informationen bezüglich dem Warenträger.

3.2 Umfang einer Volltextsuche

Die Möglichkeit durch die Eingabe einer Suchphrase geeignete Suchtreffer zu erhalten, resultiert überwiegend aus der Verwendung einer Volltextsuche. Die Umsetzungsmöglichkeiten der facettierten Suche und der semantischen Suche bauen auf der Volltextsuche auf. So ergänzt die facettierte Suche die reine Volltextsuche um die Möglichkeit der Filterung nach Facetten. Bei der semantischen Suche wird die Volltextsuche durch eine semantische Analyse der Suchphrase ergänzt.

Eine reine Volltextsuche ist vor allem in den Bereichen und Funktionalitäten von Vorteil, wo durch die Eingabe von Freitexten große Textmengen entstehen. Solche Freitexte werden innerhalb von MCC vor allem in den unterschiedlichen Logbüchern angelegt. Benutzer haben zum Beispiel im Funktionsumfang der Schichtlogbücher die Möglichkeit

einen Freitext einzugeben, welcher die Vorkommnisse der Schicht enthält. Zu einem späteren Zeitpunkt können dann Suchphrasen, wie „Beschädigung an Anlagenteil XY“, gestellt werden. Eine Eingrenzung der Suchtreffer durch das Setzen von Filtern, ist hierbei nicht zielführend. Nach der Eingabe der Suchphrase können dem Benutzer dann die jeweiligen Logbücher angezeigt werden. Neben den Schichtlogbüchern können Freitexte auch in Qualitätsberichten auftreten.

3.3 Umfang einer facettierten Suche

Während sich die reine Volltextsuche für große Textmengen eignet, wird die facettierte Suche üblicherweise im Umfeld von vordefinierten Entitäten verwendet.

In einem Produktionsleitsystem gibt es eine große Anzahl von vordefinierten Entitäten. So gibt es im PCS-Bereich unter anderem die unterschiedlichen Produktionsaufträge mit den dazugehörigen Templates und Jobs. Aus dem SCADA-Bereich sind unter anderem die verschiedenen Assets interessant für die Verwendung in einer facettierten Suche.

Anhand der Metadaten der einzelnen Entitäten ist schlussendlich eine Facettierung der Suchergebnisse möglich.

Kapitel 4

4 Konzeption

Um die prototypische Umsetzung einer Search Engine in MCC umzusetzen, werden im folgenden Kapitel verschiedene technische Umsetzungsmöglichkeiten vorgestellt und miteinander verglichen.

Im Rahmen dieser Arbeit wird innerhalb der prototypischen Umsetzung nur eine Volltextsuche umgesetzt. Die Umsetzung einer semantischen Suchfunktionalität bedarf der Verwendung eines semantischen Modells, in welchem die begrifflichen Zusammenhänge und Beziehungen definiert sind. Um solch ein wissenbasiertes Modell aufzubauen, wird eine gewisse Datengrundlage benötigt, welche im Umfang des Prototypen nicht gegeben ist.

Für das Erstellen eines Gesamtkonzeptes bedarf es einer Validierung der Komponenten „Datenpipeline“ und „Search Engine“. Hierbei beschreibt die Datenpipeline, in welcher Form eine Datenaktualisierung zwischen den Microservices und einer Search Engine umgesetzt werden kann.

Bezüglich der Datenpipeline für die Datenaktualisierung zwischen den Microservices und der Search Engine werden verschiedene Umsetzungsmöglichkeiten miteinander verglichen. Zu den Konzepten zählen die „Dual Write Aktualisierung“, die „Polling Aktualisierung“ und die „Change-Data-Capture Aktualisierung“.

Beim Vergleich und der Gegenüberstellung der unterschiedlichen Search Engines werden die Search Engines „Apache Solr“ und „Elasticsearch“ miteinander verglichen. Beide beruhen auf dem Funktionsumfang der Programmbibliothek „Apache Lucene“. Da es aus technischer Sicht auch die Möglichkeit gibt eine eigene Search Engine auf Basis von „Apache Lucene“ zu implementieren, wird die Programmbibliothek, vor der eigentlichen Auswahl der Search Engine für den Prototypen, erläutert.

Nach Auswahl aller benötigten Komponenten wird zum Schluss dieses Kapitels ein Gesamtkonzept erstellt. Das Gesamtkonzept ist zudem die Grundlage für die prototypische Umsetzung in Kapitel 5.

4.1 Auswahl einer Datenpipeline für die Datenaktualisierung

Für die Erstellung eines invertierten Indexes innerhalb der Search Engine muss diese mit Daten aus den verschiedenen Microservices befüllt werden. Durch die Verwendung einer geeigneten Datenpipeline kann der invertierte Index innerhalb der Search Engine erzeugt und aktuell gehalten werden. Anhand von Vergleichskriterien werden nachfolgend verschiedene Umsetzungsmöglichkeiten für eine Datenpipeline vorgestellt und anschließend miteinander verglichen.

4.1.1 Vergleichskriterien

Um die verschiedenen Umsetzungsmöglichkeiten für eine Datenpipeline zwischen den Microservices und der Search Engine untereinander zu vergleichen, werden die folgenden Vergleichskriterien verwendet:

Mehraufwand bei der Entwicklung:

Je nach Art und Umfang der eingesetzten Datenpipeline entsteht für die Entwickler ein Mehraufwand bei der Umsetzung neuer Funktionalitäten beziehungsweise Microservices.

- **In welchem Umfang ist die Programmierung neuer Funktionalitäten und somit Microservices von der Einführung einer Search Engine mit dazugehöriger Datenpipeline betroffen?**

Datenaktualität:

Mit dem Begriff „Datenaktualität“ wird die Aktualität der Search Engine hinsichtlich der indexierten Daten beschrieben. Dabei gibt die Datenpipeline vor, ob die Indexierung der neuen oder bearbeiteten Objekte in nahezu Echtzeit oder zu einem späteren Zeitpunkt erfolgt.

- **Wie aktuell sind die Daten innerhalb der Search Engine?**
- **In welcher Zeitspanne gelangen die neuen Daten und Information aus den Speicherschichten der Microservices in die Search Engine?**

Kompatibilität mit Datenbanksystemen:

Die eigentlichen Daten und Informationen, welche durch eine Search Engine „durchsuchbar“ gemacht werden sollen, sind zunächst in den Speicherschichten der jeweiligen Microservices abgelegt. Durch geeignete Schnittstellenabstraktionen können hierbei unterschiedliche Datenbankmanagementsystem (DBMS) für die dauerhafte Speicherung der Daten verwendet werden.

- **Mit welchen DBMS ist die jeweilige Datenpipeline kompatibel?**

- **Mit welchem Mehraufwand ist das Verwenden von „neuen“ DBMS verbunden?**

Stabilität bei Ausfall von Komponenten:

Komponentenausfälle sind auch in einer Produktionsumgebung nicht zu verhindern. Die Gründe für den Ausfall einzelner Komponenten können sowohl Hardware-seitig als auch Software-seitig entstehen. Auch Komponenten der Datenpipeline können demnach ausfallen und für einen begrenzten Zeitraum nicht zur Verfügung stehen.

- **Aus wie vielen Komponenten besteht die Datenpipeline und wie anfällig sind diese für einen Ausfall?**
- **Kann die Datenkonsistenz bei Ausfall einzelner Komponenten der Datenpipeline garantiert werden?**

Auslastung des Systems:

Neben der reinen Speicherung der Vorgänge in den Speicherschichten der Microservices muss nun auch die Search Engine mit Daten versorgt werden. Hierfür ist es notwendig, Nachrichten zwischen den einzelnen Komponenten auszutauschen. Dieser zusätzliche Nachrichtenaustausch stellt eine zusätzliche Belastung für das System dar und erhöht dessen Auslastung.

- **In welchem Umfang stellt die Integrierung der Datenpipeline eine zusätzliche Belastung für das System dar?**

Kompatibilität mit Softwarekomponenten von MCC:

Für die spätere Integrierung der Search Engine und der Datenpipeline in das Softwareprodukt MCC ist eine Kompatibilität mit bereits verwendeten Softwarekomponenten von Vorteil. Darunter zählt zum Beispiel der Message Broker „Apache Kafka“, welcher bei MCC unter anderem für den Nachrichtenaustausch zwischen den Microservices verwendet wird.

- **Mit welchen Softwarekomponenten ist die Datenpipeline kompatibel?**

4.1.2 Dual Write Aktualisierung

Eine Umsetzungsmöglichkeit für die Datenpipeline ist die „Dual Write Aktualisierung“. Führt der Benutzer eine Aktion aus, welche zu einer Änderung der Datenhaltung eines Microservices führt, wird parallel auch ein Search Service über die Aktion informiert. Aktionen, welche mit den Datenhaltungsschichten interagieren, werden auch CRUD-Befehle genannt. CRUD steht dabei für die Aktionen „Create“, „Read“, „Update“ und „Delete“.

Eine Darstellung der „Dual Write Aktualisierung“ ist in Abbildung 4.1 dargestellt. Über ein User Interface hat der Benutzer die Möglichkeit mit den Funktionalitäten der Software zu interagieren. Sobald der Benutzer eine Aktion ausführt, welche zu einem CRUD-Befehl führt, wird sowohl der jeweilige Service als auch der Search Service benachrichtigt [1]. Zu einem späteren Zeitpunkt kann der Benutzer dann über die Suchfunktionalität direkt mit dem Search Service interagieren[2].

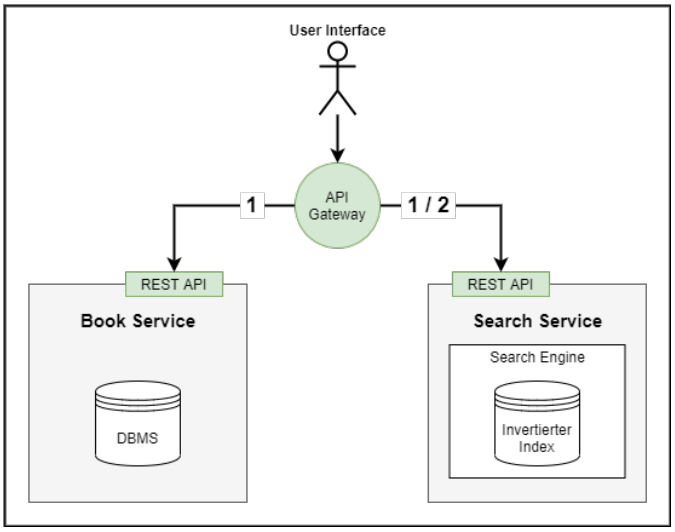

Abb. 4.1: Datenpipeline: Dual Write Aktualisierung

Vorteile:

Ein Vorteil bei dieser Art der Datenpipeline ist die **Datenaktualität** innerhalb der Search Engine. Da die CRUD-Befehle nicht nur in die Datenhaltung der Microservice, sondern auch zur Search Engine geschickt werden, findet eine ständige Synchronisation statt.

Für die Datenpipeline werden keine zusätzlichen Softwarekomponenten benötigt. Die Kommunikation mit den Datenbanksystemen findet über abstrahierte Schnittstellen der Services statt. Somit sind die Auswahlkriterien bezüglich der **Kompatibilität mit Datenbanksystemen und Softwarekomponenten von MCC** nicht von Bedeutung, da weder eine direkte Kommunikation mit einem Datenbanksystem, noch eine Verwendung von Softwarekomponenten vorhanden ist.

Nachteile:

Die Nachteile einer Dual Write Aktualisierung liegen beim **Mehraufwand für die Entwicklung** und der Wartung von bestehenden oder neuen Funktionalitäten. Die Entwickler müssen gezielt die einzelnen CRUD-Befehle nicht nur an den eigentlichen Service, sondern zusätzlich an den Search Service schicken.

[1] In der Abbildung 4.1 durch eine „1“ markiert
[2] In der Abbildung 4.1 durch eine „2“ markiert

Ein weiteres Problem ist die **Datenkonsistenz zwischen Datenhaltung der Microservices und der Search Engine**, sobald eine Komponente des Systems ausfällt. Durch das parallele Benachrichtigen der Microservices und der Search Engine ist die Atomarität der Transaktionen nicht mehr gegeben.

Bei der **Auslastung des Systems** besteht bei dieser Art der Datenpipeline das Problem, dass bei jeder Aktion, welche einen CRUD-Befehl beinhaltet, ein Nachrichtenaustausch mit dem Search Service stattfindet. Dadurch entsteht eine Engstelle in der Software, durch welche die Performance des gesamten Systems verlangsamt wird.

4.1.3 Polling Aktualisierung

Eine weitere Umsetzungsmöglichkeit für die Datenpipeline ist die „Polling Aktualisierung“. Hier wird vom Benutzer lediglich die Aktualisierung der eigentlichen Datenhaltung initiiert. Innerhalb einer vordefinierten Zeitspanne werden dann die bearbeiteten Daten entweder mit einem Pull- oder Push-Mechanismus an den Search Service geschickt.

Pull:

Der Search Service initiiert die Aktualisierung des invertierten Index nach einer vordefinierten Zeitspanne und holt sich dann die Datenänderungen aus den verschiedenen Microservices.

Push:

Nach einer vordefinierten Zeitspanne schicken die verschiedenen Microservices die Datenänderungen an den Search Engine, wo dann eine Aktualisierung des invertierten Index stattfinden kann.

In Abbildung 4.2 ist eine Darstellung der Polling Aktualisierung abgebildet. Über das User Interface kann der Benutzer Aktionen initiieren, welche dann von den jeweiligen Microservices bearbeitet werden. Im Falle von CRUD-Befehlen werden lediglich die Datenhaltungsschichten der Microservices benötigt [3]. Die Aktualisierung des invertierten Indexes der Search Engine wird zu einem späteren Zeitpunk durch einen Pull- oder Push-Mechanismus durchgeführt [4]. Für das Stellen einer Suchanfrage kann anschließend der Benutzer direkt mit dem Search Service interagieren [5].

[3] In der Abbildung 4.2 durch eine „1“ markiert
[4] In der Abbildung 4.2 durch eine „2“ markiert
[5] In der Abbildung 4.2 durch eine „3“ markiert

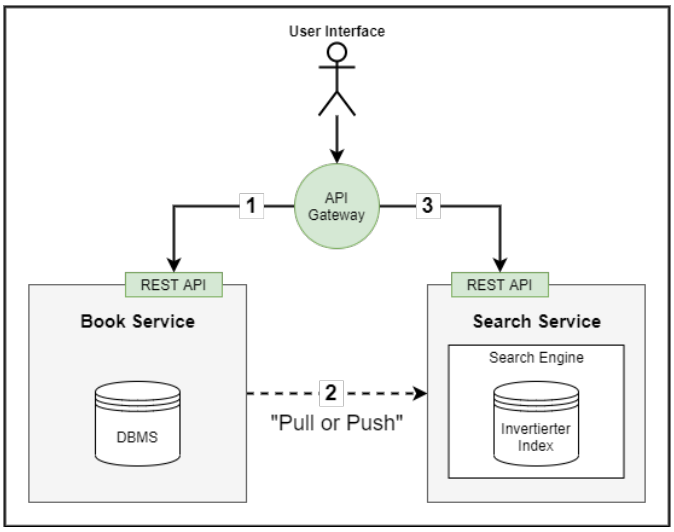

Abb. 4.2: Datenpipeline: Polling Aktualisierung

Vorteile:

Im Gegensatz zur Dual Write Aktualisierung gibt es bei der Polling Aktualisierung keine Engstelle in der Software, da bei jeder Benutzeraktion lediglich die Datenhaltungen der Microservices beeinträchtigt werden. So entsteht durch diese Art der Datenpipeline nur eine minimale **Mehrbelastung des Systems**.

Ein weiterer Vorteil ist die **Kompatibilität zu den Datenbanksystemen und den Softwarekomponenten von MCC**, da hier über vordefinierte Schnittstellen kommuniziert wird.

Nachteile:

Ein großer Nachteil bei dieser Datenpipeline ist die **Datenaktualität** innerhalb der Search Engine. Die Datenaktualität ist dabei von der Zeitspanne des Pull- oder Push-Mechanismen abhängig. So kann es bei der Bearbeitung einer Suchanfrage dazu kommen, dass die Search Engine zu dem Zeitpunkt noch nicht aktualisiert wurde und somit eine unvollständige Trefferliste erzeugt wird.

Bei der Entwicklung und Wartung von neuen Funktionalitäten müssen die Entwickler bei den Polling Mechanismen entscheiden, wann die Datenaktualisierung stattfinden soll. Dies resultiert in einen **Mehraufwand bei der Entwicklung**, da auch definiert werden muss, welche Teile des Datenkontextes „durchsuchbar“ gemacht werden sollen.

4.1.4 Change-Data-Capture Aktualisierung

Neben der Umsetzungsmöglichkeit der Dual Write Aktualisierung und der Polling Aktualisierung gibt es noch die Umsetzungsmöglichkeit mithilfe des Change Data Capture (CDC) - Pattern. Bei CDC handelt es sich um ein Software Design Pattern,

welches die Datenintegration zwischen einer Datenbank und deren Datenquellen definiert [Dv21].

Durch das Beobachten von Datenänderungen in den Quelldatenbanken ermöglicht CDC ein Bewegen und Verarbeiten der Daten in nahezu Echtzeit [Joh21]. Somit können Datenreplikationen mit geringer Latenz erzeugt werden. Für die Umsetzung einer CDC-Datenpipeline gibt es vier gängige Methoden [Mar21]:

Zeitpunkt der letzten Aktualisierung:

Um hierbei die Datenbankänderungen mit CDC zu beobachten, müssen die Datenbanktabellen eine „DATE_MODIFIED"-Spalte enthalten, welche den Zeitpunkt der letzten Änderung des Datensatzes protokolliert. Bei der Datensynchronisierung zwischen einer Zieldatenbank und den Quelldatenbanken filtert man nun mithilfe von CDC nach einer solchen „DATE_MODIFIED"-Spalte und entnimmt die Datenänderungen, welche seit der letzten Datenextraktion geändert wurden.

Ein Problem bei diesem Vorgehen ist das Beobachten von Löschoperationen. Hierbei werden die Datensätze häufig komplett entfernt und ein Beobachten der Spalte „DATE_MODIFIED" ist nicht mehr möglich.

Auch muss solch eine Spalte für jede Datenbanktabelle zur Verfügung stehen und aktuell gehalten werden. Durch Datenbank-Trigger können diese Spalten aktuell gehalten werden, was jedoch einen Overhead verursacht.

Tabellenunterschiede:

Eine weitere Möglichkeit für die Umsetzung einer CDC-Datenpipeline ist das regelmäßige Vergleichen der Tabellenunterschiede. Hierbei wird der aktuelle Zustand der Daten mit dem vorherigen Zustand verglichen.

Im Vergleich zur Methode, bei welcher man auf die letzte Aktualisierung einer Zeile achtet, können hierbei auch die Löschoperationen berücksichtigt werden. Ein Nachteil ist der hohe Ressourcenverbrauch, der bei der Berechnung der Unterschiede zwischen den Datenbeständen entsteht. Somit kann die Methode der Tabellenunterschiede nicht in Echtzeit durchgeführt werden, da für eine ständige Durchführung zu viele Ressourcen benötigt werden.

Trigger-basierte CDC:

Mit der Verwendung von Datenbank-Triggern können Schattentabellen erzeugt werden, auf welchen dann die CDC-Datenpipeline die Änderungen erfassen kann. Hierbei gibt es die Möglichkeit die kompletten Datensätze in eine Schattentabelle zu speichern oder die Möglichkeit lediglich die Primärschlüssel der Datensätze in Verbindung mit der vorgenommen Art der Datenänderung (Einfügen, Aktualisieren oder Löschen) abzuspeichern.

Bei der Variante der kompletten Speicherung der Datensätze in eine Schattentabelle entsteht bei der Erstellung und Aktualisierung der Schattentabelle ein Overhead.

Ein geringerer Overhead entsteht bei der Variante, wo lediglich die Primärschlüssel mit der Art der Datenänderung in der Schattentabelle gespeichert werden. Hierbei ist aber ein Join zur Quelltabelle nötig, sobald die Datenänderungen ausgelesen werden sollen.

Somit senkt eine Trigger-basierte CDC-Datenpipeline den Aufwand für die Extraktion der Änderungen, erhöht aber den Aufwand für die Aufzeichnung der Änderungen in den Schattentabellen.

Log-basierte CDC:

Datenbanksysteme speichern alle Änderungen in einem Transaktionsprotokoll, um somit abgebrochene Transaktionen wieder rückgängig zu machen und um die Persistenz der Datenhaltung bei Ausfall des Datenbanksystems zu bewahren. Die Log-basierte CDC-Datenpipeline liest hierbei die Transaktionsprotokolle der Datenbanksysteme und entnimmt daraus die Datenänderungen.

Ein Problem bei der Verwendung der Transaktionsprotokolle ist der fehlende Standard bezüglich der Art und Weise, wie Änderungen abgespeichert werden. So sind die Transaktionsprotokolle von verschiedenen Datenbankanbietern nach keinem Standard aufgebaut und unterscheiden sich stark voneinander.

Im Rahmen dieser Arbeit und den Umsetzungsmöglichkeiten einer Datenpipeline wird die Log-basierte CDC-Datenpipeline näher betrachtet. In Abbildung 4.3 ist eine Datenpipeline mit einer CDC-Aktualisierung abgebildet. Wie bereits bei der Polling Aktualisierung initiiert der Benutzer über ein User Interface eine Aktion, welche dann vom jeweiligen Microservice bearbeitet wird. Werden dabei die Daten der jeweiligen Datenbank durch CRUD-Befehle bearbeitet, werden diese Änderungen auch in den Datenbank-spezifischen Transaktionsprotokollen festgehalten.

Durch die große Varianz der unterschiedlichen Transaktionsprotokolle ist eine eigene Implementierung einer CDC-Software umständlich. Abhilfe schafft das Open Source Framework „Debezium“, welche auf dem Log-basierten CDC arbeitet und die Änderungen der Transaktionsprotokolle beobachtet.

Debezium:

Nach Luber [SN21] ist Debezium ein Open Source Framework, das Änderungen in einer Datenbank erfasst und in Form von Event-Streams anderen Anwendungen zur Verfügung stellt. Debezium baut hierbau auf dem Framework „Apache Kafka Connect“ auf, welches für die Implementierung von Kafka-Schnittstellen verantwortlich ist. Solche Schnittstellen werden entweder in Form von Quellkonnektoren oder Sink-Konnektoren umgesetzt und sind dafür zuständig, Datensätze an Kafka zu schicken (Quellkonnektoren) oder Datensätze aus Kafka-Topics (Sink-Konnektoren) an andere Systeme weiterzuschicken [Deb21b].

Debezium bietet derzeit für folgende Datenbankmanagementsysteme Kafka-Konnektoren an: [Deb21a]

- MongoDB
- MySQL
- PostgreSQL
- SQL Server
- Oracle
- Db2

Eine Vorabversion der Kafka-Konnektoren wird bereits für folgende Datenbankmanagementsysteme angeboten: [Deb21a]

- Cassandra
- Vitess

In Abbildung 4.3 ist die Verwendung von Debezium im Gesamtbild der Datenpipeline abgebildet. Zu erkennen ist, das Debezium auf die Transaktionsprotokolle der unterschiedlichen DBMS zugreift und dessen Änderungen an den Message Broker Apache Kafka weiterleitet. Somit fungiert Debezium als Kafka-Produzent und leitet die Datenänderungen aus den Quelldatenbanken an die entsprechenden Kafka-Topics weiter. Um die Daten aus den Kafka-Topics in die jeweilige Search Engine zu leiten, müssen geeignete Kafka-Sink-Konnektoren verwendet werden.

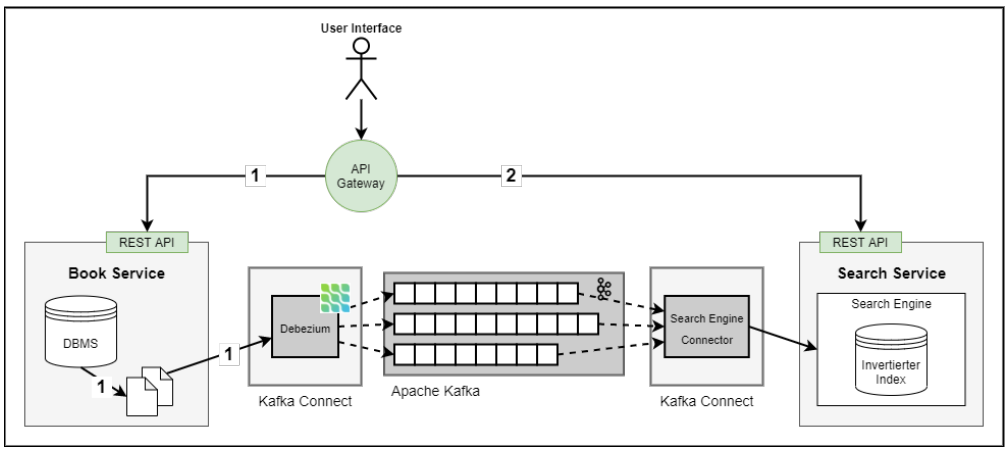

Abb. 4.3: Datenpipeline: Change-Data-Capture Aktualisierung

Vorteile:

Ein Vorteil bei der Verwendung der CDC-Aktualisierung liegt bei der **Echtzeit der Daten** in der jeweiligen Search Engine. Grund dafür ist die stetige Überwachung der DBMS-spezifischen Transaktionsprotokolle.

In der eigentlichen Software entsteht durch die zusätzliche CDC-Datenpipeline keine Engstelle. Zusätzlich wird durch die stetige Überwachung lediglich ein geringer Datenbestand transportiert, was die **Auslastung des Gesamtsystems** nicht nachteilig beeinträchtigt.

Da die verschiedenen Debezium-Kafka-Konnektoren für den Message Broker „Apache Kafka“ entworfen wurden und dieser auch als Basis für den Austausch von Nachrichten in MCC gedacht ist, besteht bei der Verwendung der CDC-Aktualisierung eine hohe **Kompatibilität mit den Softwarekomponenten** von MCC.

Nachteile:

Ein Nachteil ist die Varianz der unterschiedlichen Transaktionsprotokolle. Durch CDC-Software, wie „Debezium“, werden die unterschiedlichen Transaktionsprotokolle abstrahiert, jedoch ist man demnach von solch einer Software abhängig. So kann Debezium lediglich in Verbindung mit dem Message Broker „Apache Kafka“ verwendet werden.

4.1.5 Auswahl

Für die Datenpipeline innerhalb der prototypischen Umsetzung wurde sich für die Umsetzungsmöglichkeit „**Change-Data-Capture Aktualisierung**“ entschieden. Die Entscheidung resultiert aus einem Vergleich mit den Umsetzungsmöglichkeiten „Dual Write Aktualisierung“ und „Polling Aktualisierung“.

Eines der wichtigen Vergleichskriterien ist die **Aktualität der Daten** innerhalb der Search Engine. Die Aktualität jener Daten definiert gleichermaßen die Vollständigkeit von Suchtreffern, welche bei der Search Engine angefragt werden. Da die Datenaktualität bei der Polling Aktualisierung von einer vordefinierten Zeitspanne abhängig ist, kann es vorkommen, dass Datenänderungen aufgrund der Zeitspanne noch nicht an die Search Engine übermittelt wurden, aber bei der Search Engine zur gleichen Zeit eine Suchanfrage zu diesen Daten erfolgt. Sowohl bei der Dual Write Aktualisierung, als auch bei der CDC-Aktualisierung werden die Daten zwischen den Microservices und der Search Engine in nahezu Echtzeit aktuell gehalten.

Die verschiedenen Umsetzungsmöglichkeiten unterscheiden sich stark hinsichtlich des **Mehraufwands für die Entwicklung und Wartung** eines Gesamtsystems. Unter den vorgestellten Umsetzungsmöglichkeiten bietet die CDC-Aktualisierung den geringsten Mehraufwand. Grund hierfür ist die Unabhängigkeit der Datenpipeline von den internen Implementierungen der einzelnen Microservices, da sich die CDC Aktualisierung ausschließlich direkt mit den entsprechenden Datenhaltungschichten verbindet. Bei Neuentwicklung oder Erweiterungen der Microservices gilt es lediglich die Konfigurationen

der CDC-Konnektoren anzupassen, damit bei der CDC Aktualisierung die entsprechenden Datenänderungen in den Datenhaltungsschichten registriert werden. Anders verhält sich der Mehraufwand bei der Umsetzung einer Dual Write Aktualisierung oder einer Polling Aktualisierung. Bei der Dual Write Aktualisierung liegt die Verantwortung für das Senden der Datenänderungen an den Search Service beim jeweiligen Entwickler. Jener muss neben dem Schreiben der Datenänderungen in die eigentliche Datenhaltung, zusätzlich die Datenänderungen an die Search Engine schicken. Bei Änderungen auf Seiten der Search Engine gilt es somit, alle Funktionalitäten des gesamten Systems zu warten. Bei der Polling Aktualisierung ist der Austausch der Datenänderungen zwischen den Microservices und der Search Engine anhand einer Zeitspanne geregelt. Neben der Zeitspanne gilt es auch die Pull- oder Push-Mechanismen zu definieren.

4.2 Auswahl einer Search Engine

Neben einer eigenen Implementierung einer Suchmaschine auf Basis der Programmbibliothek „Apache Lucene“ gibt es auf dem Markt bereits ausgereifte Software-Lösungen, welche für die invertierte Indexierung von Daten vorgesehen sind. Im folgenden soll die Auswahl einer solchen Search Engine erfolgen. Die Search Engine wird nachfolgend in das Gesamtkonzept des Prototypen einfließen. Betrachtet und miteinander verglichen werden dabei die Search Engines „Apache Solr“ und „Elasticsearch“.

4.2.1 Vergleichskriterien

Um die verschiedenen Softwareprodukte für eine Search Engine untereinander zu vergleichen, werden die folgenden Vergleichskriterien verwendet:

Lizenzmodelle:

Da die jeweilige Search Engine in der Zukunft auch eine Softwarekomponente für das Softwareprodukt MCC sein soll, ist die Verfügbarkeit von kostenfreien Lizenzen ein Vorteil. Auch Lizenzen mit einer einmaligen Bezahlung können für die Verwendung in MCC genutzt werden. Lediglich Lizenzmodelle mit laufenden Kosten (Preis pro Datenvolumen) sind zu vermeiden, da sonst bei einer Skalierung von MCC die Kosten einer Nebenkomponente für den Kunden zu teuer werden können.

- **Welche Lizenzmodelle bietet die Search Engine an?**

Effizienz:

Im Umfeld eines Produktionsleitsystems ist die Effizienz einer jeden Softwarekomponente entscheidend. Bei einer Search Engine geht es dabei um die

Geschwindigkeit und Ressourcennutzung während der Phase der Indexierung und während der Bearbeitung der Suchanfragen.

- **In welcher Geschwindigkeit und mit welcher Ressourcennutzung werden bestimmte Mengen von Datensätzen indexiert und durch Suchanfragen durchsucht?**

Rankingmechanismus:

Um dem Benutzer eine möglichst aussagekräftige Trefferauflistung zu übergeben, müssen die Suchtreffer nach ihrer Relevanz hin sortiert werden.

- **Anhand welcher Metrik bestimmt die Search Engine die Relevanz eines Suchtreffers?**

Dokumentation:

Für die Integration und Wartung einer Search Engine in das Softwareprodukt MCC ist eine aussagekräftige und umfangreiche Softwaredokumentation von der Search Engine nötig.

- **Welchen Umfang besitzt die vorliegende Softwaredokumentation?**

4.2.2 Programmbibliothek „Apache Lucene“

Apache Lucene, welches anfänglich Ende der 1990er-Jahre von Doug Cutting programmiert wurde, wird seit 2005 als Top-Level-Projekt von der Apache Software Foundation betreut [SN18b].

Als frei verfügbare Programmbibliothek unterstützt Lucene hierbei die technische Umsetzung einer Volltextsuche. Unterstützt wird hierfür die Erstellung und Aktualisierung eines invertierten Indexes, auf welchem anschließend Suchanfragen gestellt werden können. So stellen die Funktionalitäten von Lucene neben einer Möglichkeit der Indexierung von Dateien auch eine Schnittstelle für das Abfragen von relevanten Suchtreffern zu Verfügung.

Für die Erstellung und Aktualisierung des invertierten Indexes kommen hierbei verschiedene Tokenizer und Analyzer zum Einsatz (siehe auch Unterabschnitt 2.1.2). Durch jene Transformationsschritte werden die morphologischen Varianzen der menschlichen Sprache entfernt. So wird beispielsweise der Satz „Eine Blaue Herrenhose von Levi's®“ in die Wörter „blau“, „herr“, „hos“ und „levis“ umgewandelt und anschließend dem invertierten Index hinzugefügt. Auch eingehende Suchanfragen werden mithilfe der Transformationsschritte umgewandelt und anschließend innerhalb des invertierten Indexes gesucht.

Die nach der Suchanfrage gefundenen Suchtreffer werden bei Lucene durch eine Termgewichtung nach dem „TF-IDF“-Modell sortiert (siehe auch Unterabschnitt 2.1.5).

Somit wird dem Benutzer eine sortierte Trefferauflistung zurückgegeben, wobei die relevantesten Ergebnisse oben in der Auflistung zu finden sind.

4.2.3 Apache Solr

Für die technische Umsetzung einer Suchfunktionalität gibt es auf dem Markt bereits verschiedene Search Engines, welche die Funktionalitäten zur Verfügung stellen. Eine solche Search Engine ist „Apache Solr“. Als Kern von Solr, wird für die Indexierung und Beantwortung von Suchanfragen, die Programmbibliothek „Apache Lucene“ verwendet.

Nach Luber [SN18a] handelt es sich bei Lucene lediglich um ein Framework für die Entwicklung von Suchserver, während es sich bei Solr um eine Server-Software handelt. Ein solcher Suchserver stellt die Indexierung von Datenbeständen und die Suche nach Informationen als Services zur Verfügung. Somit erfordert die reine Nutzung von Lucene eine eigene Programmierung einer Suchanwendung, während bei Solr nach der Installation lediglich die gewünschten Suchfunktionen konfiguriert werden müssen.

Für die Kommunikation mit Solr wird das Hypertext Transfer Protocol (HTTP) verwendet. Über HTTP-POST Schnittstellen werden Datensätze in den möglichen Formaten XML, JSON, CSV oder binär an Solr geschickt und anschließend indexiert. Durch die Verwendung der HTTP-GET Schnittstelle können die Suchergebnisse in den Formaten XML, JSON, CVS oder binär abgefragt werden. Durch die Bereitstellung von verschiedenen Solr-Bibliotheken kann Solr auch direkt von Programmiersprache aus angesprochen werden. Unterstützt werden hierbei unteranderem die Programmiersprachen Java, Pyhton, Ruby, C# und PHP [Sem21].

Aufbauend auf der Programmbibliothek Lucene bietet Solr weitere Features für die Bereitstellung einer Suchfunktionalität [Apa21a]:

Datenverarbeitung:

Grundlage für die Indexierung in Solr sind „Dokumente“. Solche Dokumente könnten zum Beispiel vom Typ „Buch“ sein und die Felder „Titel“, „Autor“, „Anzahl der Seiten“ und „Jahr der Veröffentlichung“ enthalten.

Die Felder können Daten von unterschiedlichen Typen beinhalten. So wird die Anzahl der Seiten eher mit numerischen Typen und der Name des Autors eher mit textbasierten Typen beschrieben. Die unterschiedlichen Typen können durch die Konfigurationen in Solr festgelegt werden, um so eine bestmögliche Indexierung zu erhalten.

Weitergehend können die unterschiedlichen Analyzer für die Indexierung und Suchabfrage konfiguriert werden. So kann individuell festgelegt werden, welche Stopwords oder Stemming-Verfahren eingesetzt werden sollen.

Facetten:

Solr unterstützt nicht nur die reine Volltextsuche, sondern liefert über die Konfiguration von Facetten auch eine Möglichkeit der facettierten Suche.

Hierzu erfolgt eine Einteilung der Suchergebnisse in Kategorien, welche aus den zu indizierenden Begriffen resultieren. Die explizite Suche mithilfe solcher Kategorien ist durch eine Ergänzung der Suchanfrage möglich.

Erweiterte Dateiformate:

Durch das Hinzufügen von Erweiterungen ist es mit Solr auch möglich Dateiformate wie PDF, Microsoft Word oder Microsoft PowerPoint verarbeiten zu können. Hierfür bietet sich beispielsweise das Analyse-Tool „Apache Tika“ an.

Mehrsprachige Unterstützung:

Neben Deutsch kann Solr auch mit einer Reihe anderer Sprachen wie Englisch, Chinesisch, Japanisch, Koreanisch, Arabisch, Französisch, Spanisch und vielen anderen verwendet werden. Solr verfügt über eine integrierte Spracherkennung und bietet dementsprechend sprachspezifische Textanalysetools.

Apache Solr wird unter der Lizenz „Apache License 2.0“ vertrieben. Eine Integrierung in MCC ist somit kostenfrei möglich.

4.2.4 Elasticsearch

Eine weitere Search Engine, welche auf der Programmbibliothek „Apache Lucene“ aufbaut ist Elasticsearch. Elasticsearch ist in der Programmiersprache Java entwickelt worden und kann somit plattformunabhängig betrieben werden. Erschienen ist Elasticsearch im Jahr 2010 und wird seit dem von der Firma Elastic entwickelt und betreut [Ela21a].

Elasticsearch ist Teil des von Elastic vertriebenen „ELK Stacks“. Dies ist ein Verbund aus den drei Open-Source-Projekten „Elasticsearch“, „Logstash“ und „Kibana“. Während Elasticsearch eine Suchmaschine zur Verfügung stellt, ist Logstash eine serverseitige Datenverarbeitungspipeline, welche Daten aus unterschiedlichen Quellen ingestiert, sie umwandelt und einen Speicherort weiterleitet. Kibana unterstützt bei der Visualisierung von Daten durch Diagramme und Tabellen.

Für die Kommunikation stellt Elasticsearch eine REST API zur Verfügung [Ela21b]. Über diese Schnittstelle ist es möglich die Datensätze für die Erstellung eines invertierten Indexes an Elasticsearch zu schicken. Auch ist eine Suchabfrage über jene Schnittstelle möglich. Bei Elasticsearch spielt es keine Rolle, ob die Daten strukturiert, unstrukturiert, eine Kombination dieser beiden Arten vorliegt oder ob Geodaten und Metriken durchsucht werden müssen [Ela21b]. Ähnlich, wie bei Solr, gibt es auch bei Elasticsearch die Möglichkeit die Kommunikation über Elasticsearch-Bibliotheken aufzubauen.

Unterstützt werden dabei die Programmiersprachen Java, JavaScript, Go, .NET, PHP, Perl, Python und Ruby [Ela21b].

Im Bereich der eigenen Bereitstellung von Elasticsearch beläuft sich die Lizenz auf die „Apache License 2.0“ und kann somit kostenfrei benutzt werden. Elasticsearch wird jedoch auch als bereits fertiger Cloud-Services angeboten. Hierbei wird unter den vier Abonnements „Basic“, „Gold“, „Platinum“ und „Enterprise“ unterschieden. Die Kosten belaufen sich dabei von 16$ bis 30$ pro Monat.

4.2.5 Auswahl

Bezüglich der Auswahl einer Search Engine für die prototypische Umsetzung werden folgend die beiden Search Engines „Elasticsearch“ und „Solr“ verglichen.

Während Solr bei den **Lizenzmodellen** den gesamten Quellcode mit der Open-Source-Lizenz „Apache License 2.0“ vertreibt, wird bei Elasticsearch neben der freien Apache-Lizenz auch kommerzielle Lizenzen angeboten. Neben spezielleren Features im Bereich des Machine Learning ist vorallem der Cloud-Service von Elasticsearch kostenpflichtig.

Die **Popularität** der beiden Search Engines ist in Abbildung 4.4 abgebildet. Die von DB-Engines erstellte Statistik zeigt auf, dass Elasticsearch seit dem Jahr 2016 eine höhere Popularität als Solr aufweist [DE21a]. Für die Berechnung der Popularität berücksichtigt DB-Engine folgende Faktoren [DE21b]:

- Anzahl der Nennungen des Systems auf Websites
- Allgemeines Interesse an dem System
- Häufigkeit von technischen Diskussionen über das System
- Anzahl an Job-Angeboten, in denen das System genannt wird
- Anzahl an Profilen in professionellen Netzwerken, in denen das System aufgeführt wird
- Relevanz in sozialen Netzwerken

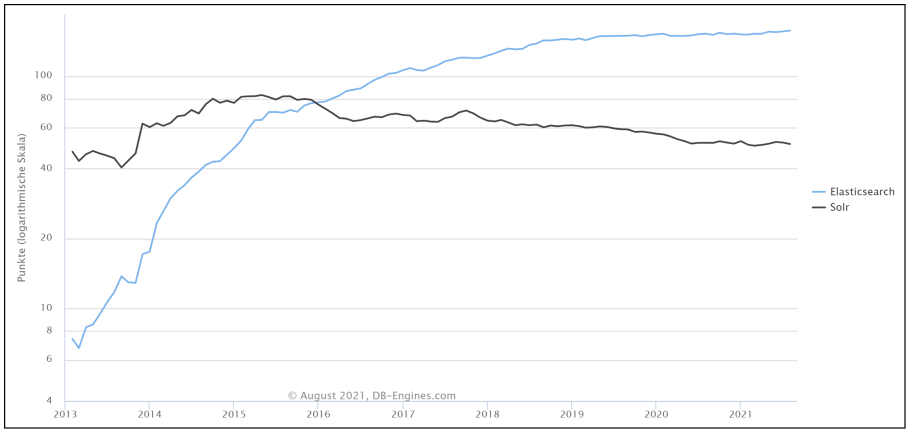

Abb. 4.4: DB-Engines Ranking: Elasticsearch vs. Solr [DE21a]

Für eine Einarbeitung in eine neue Software-Komponente ist die **Dokumentation** der Software von großer Bedeutung. Elasticsearch liefert hierbei eine vollumfänglichere Dokumentation an. Neben einer gut strukturierten Website und einer Dokumentation mit vielen Beispielen, gibt es auf dem Markt auch eine große Anzahl an Büchern und Tutorials [Asa20]. Yigal begründet die größe Menge an Büchern und Tutorials mit der höheren Popularität von Elasticsearch im Vergleich zu Solr [Asa20]. Die Dokumentation von Solr hingegen weist Lücken auf, so ist zum Beispiel die Solr-API zun unzureichend dokumentiert. Ebenso gibt es vergleichsweise wenig technische Beispiele und Tutorials [Asa20].

Da beide Search Engines auf der Programmbibliothek „Apache Lucene“ aufbauen, gibt es bei der **Effizienz** der Indexierung und Beantwortung von Suchanfragen kaum Unterschiede. Es werden jeweils lediglich die Funktionalitäten der Programmbibliothek durch Features und Schnittstellen erweitert. Nach Yigal bietet Solr Vorteile bei der Textsuche in Unternehmensanwendungen, welche im Big-Data-Umfeld agieren [Asa20]. Elasticsearch hingegen bietet dank der leistungsstarken Aggregationsfunktionalitäten ein Mehrwert bei der Textsuche und in Analysemaschinen [Asa20].

Für die prototypische Umsetzung einer Suchfunktionalität wurde sich für die Search Engine Elasticsearch entschieden. Grund hierfür ist die im Vorfeld ausgesuchte Datenpipeline „CDC Aktualisierung“, welche ein Vorhandensein eines Kafka-Konnektors erfordert. Ein Kafka-Sink-Konnektor wird benötigt, um die Datensätze des Message Brokers „Apache Kafka“ an die eigentliche Search Engine zu schicken. Für die Search Engine Elasticsearch gibt es hierfür einen offiziellen Konnektor von der Firma Confluent [Con21]. Hierbei spielt es keine Rolle, ob Elasticsearch im Cloud- oder Standalone-Modus betrieben wird. Bei Solr hingegen werden keine offiziellen Konnektoren angeboten. Lediglich Implementierungen von Privatpersonen sind zu finden [bka21].

4.3 Gesamtkonzept

Die technische Umsetzung einer Suchfunktionalität soll anhand einer Online-Bibliothek veranschaulicht werden. Hierfür hat der Benutzer die Möglichkeit, über eine Website Bücher der Bibliothek hinzuzufügen, zu ändern oder zu entfernen. Bücher sollen dabei aus den Attributen Titel, Autor, Veröffentlichungsjahr, Sprache, ISBN und einer Zusammenfassung bestehen.

Das Gesamtkonzept für die prototypische Umsetzung ist in Abbildung 4.5 dargestellt. Der Benutzer des User Interfaces hat über geeignete REST-Schnittstellen die Möglichkeit, neben dem Book Service, auch mit dem Search Service zu interagieren.

Beim Hinzufügen, Ändern oder Löschen von Büchern werden die Datenänderungen an den Entity Server von MCC weitergeleitet. Zum Zeitpunkt der Erstellung der

vorliegenden Arbeit ist im Entity Server von MCC lediglich eine Datenbankabstraktion für MongoDB entwickelt worden. Aus diesem Grund wurde sich für die Datenbank MongoDB entschieden.

Durch die stetige Überwachung des Operation-Logs (kurz Oplog) von MongoDB kann der Debezium-Kafka-Konnektor die Änderungen registrieren und an den Message Broker „Apache Kafka“ weiterleiten. Durch den Elasticsearch-Sink-Konnektor können die Nachrichten der einzelnen Kafka-Topics an die Elasticsearch-Instanzen weitergeleitet werden.

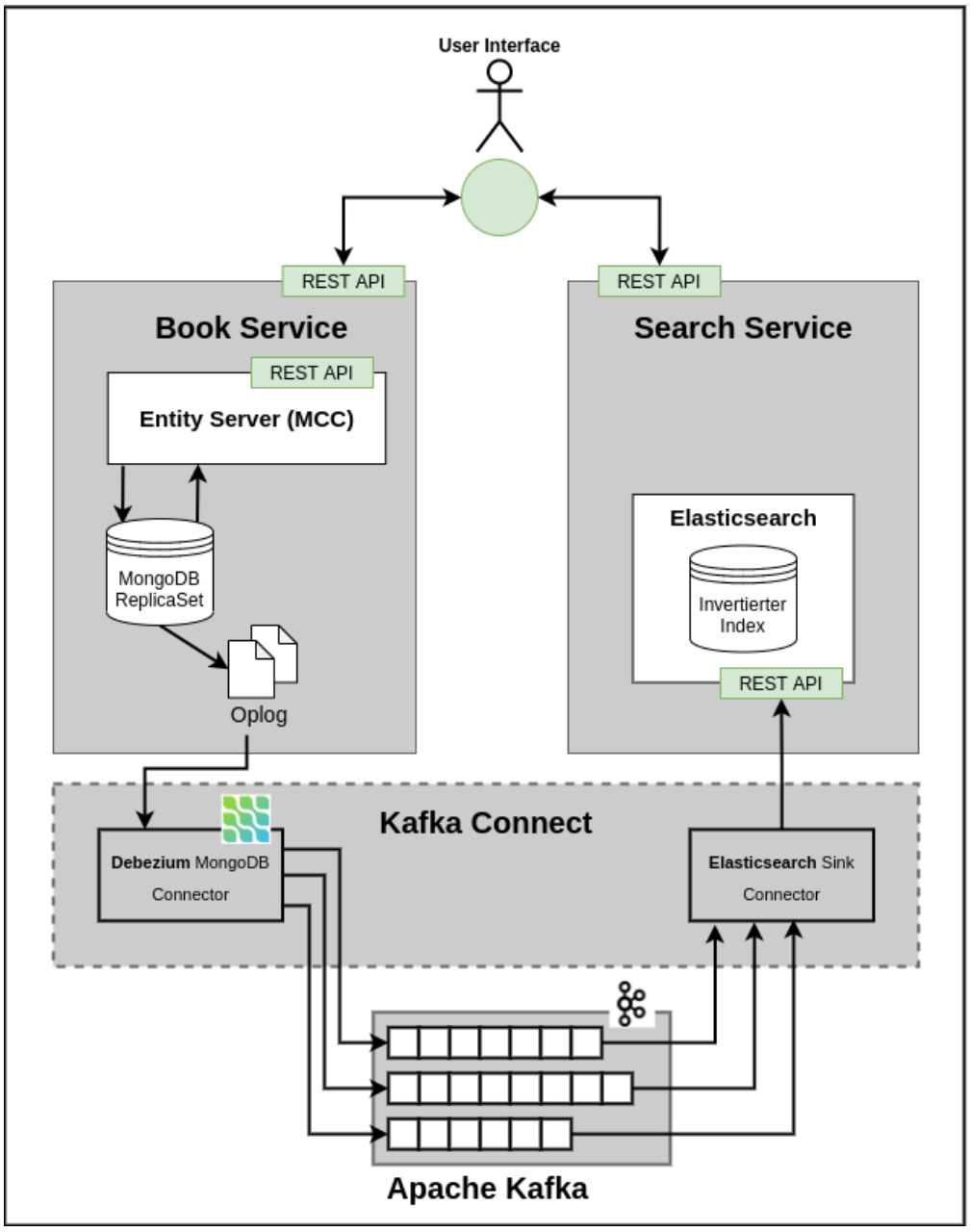

Abb. 4.5: Gesamtkonzept für prototypische Umsetzung

Kapitel 5
5 Prototypische Umsetzung

Im folgenden Kapitel wird die Umsetzung des Prototypen vorgestellt. Neben einer Einführung in den Funktionsumfang des Prototypen, wird das Gesamtkonzept aus Abschnitt 4.3 in verschiedene Komponenten unterteilt.

5.1 Funktionsumfang

Der Umfang des Prototypen soll eine Bibliothek darstellen, in welcher ein Benutzer die Möglichkeit besitzt Bücher anzulegen und zu verwalten. Hierfür werden die allgemeinen CRUD-Operationen zur Verfügung gestellt. Der Benutzer hat demnach die Möglichkeit neue Bücher dem Bestand hinzuzufügen (Create), sich den aktuellen Bestand anzeigen zu lassen (Read), bestehende Bücher zu ändern (Update) oder Bücher aus dem Bestand zu entfernen.

In Abbildung 5.1 ist die Benutzerschnittstelle abgebildet.

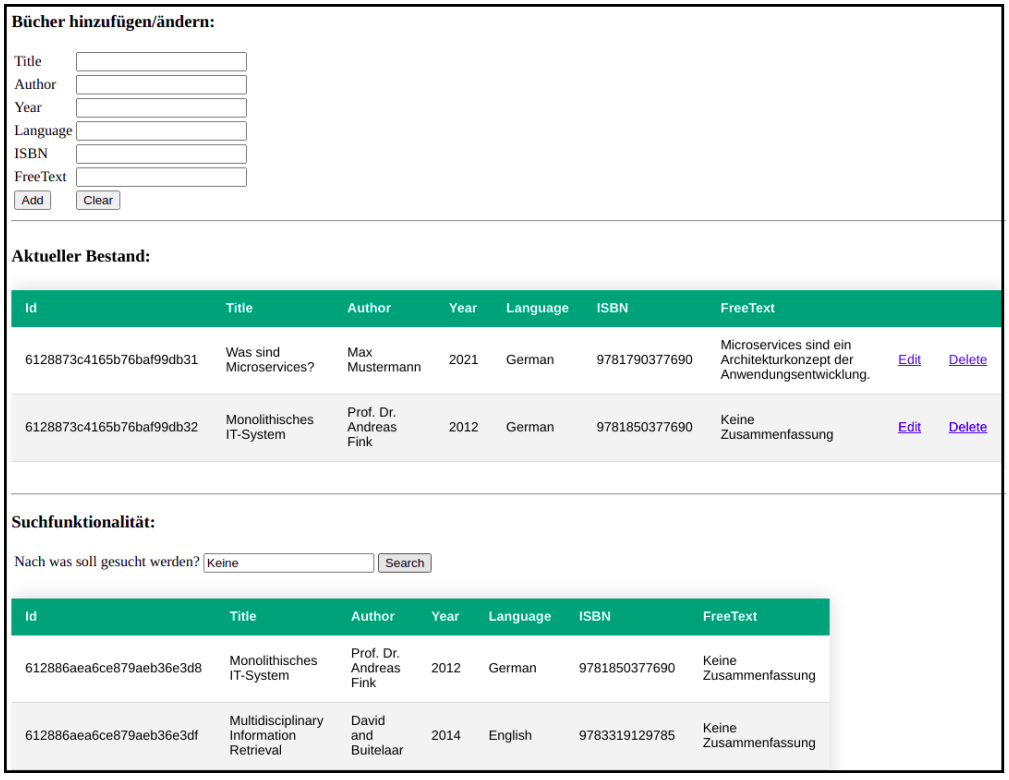

Abb. 5.1: Benutzeransicht der prototypischen Umsetzung

Zu unterteilen ist die Ansicht in drei Bereiche „Bücher hinzufügen/ändern“, „Aktueller Bestand“ und „Suchfunktionalität“.

Im oberen Bereich haben die Benutzer die Möglichkeit neue Bücher dem Bestand hinzuzufügen. Durch Betätigung des Buttons „Add“ werden die Informationen aus den darüberliegenden Feldern als neues Objekt in der Datenhaltung abgespeichert. Zusätzlich dient der obere Bereich der Bearbeitung bestehender Bücher. Hierfür wird im darunterliegenden Datenbestand das jeweilige Buch durch den Button „Edit“ ausgewählt und die enthaltenen Informationen können anschließend bearbeitet und gespeichert werden. Durch Betätigung des Buttons „Clear“ werden die Text- und Zahlenfelder geleert.

Der aktuelle Bestand der Bücher in der Datenhaltung, wird in einer Tabelle wiedergegeben. Angezeigt werden neben einer eindeutigen ID auch die Informationen Titel, Autor, Veröffentlichungsjahr, Sprache, ISBN und Zusammenfassung. Jede Reihe der Tabelle repräsentiert ein Buch-Objekt aus der Datenhaltung und kann jeweils durch den Button „Edit“ bearbeitet oder durch den Button „Delete“ entfernt werden.

Im unteren Bereich der Benutzerschnittstelle findet sich die Suchfunktionalität. Der Benutzer hat hierbei die Möglichkeit in das Textfeld eine beliebige Suchphrase einzugeben. Durch Betätigung des Buttons „Search“ wird die Suchphrase an die Search Engine geschickt und die empfangenen Suchtreffer werden in der darunterliegenden Tabelle angezeigt.

Neben der reinen Eingabe einer Suchphrase könne auch Wildcards für die Suche verwendet werden. Unterstützt werden von Elasticsearch unter anderem die Wildcards „?“ und „*“. „?“ ersetzt dabei genau ein beliebiges Symbol, während durch „*“ keine oder mehrere Symbol ersetzt werden.

Zusätzlich zu den Wildcards können auch die Attribute angegeben werden, über welche die Suche ausgeführt werden soll. Standardmäßig wird bei der Suche über alle vorhandenen Attribute gesucht. So ist es durch die Eingabe von „status:active“ möglich, nach dem Begriff „active“ in dem Attribut „status“ zu suchen.

5.2 Komponenten

Unterteilt werden kann das Gesamtkonzept aus Abschnitt 4.3 in die drei Bereiche „Book Service“, „CDC-Datenpipeline“ und „Search Engine“.

5.2.1 Book Service

Für das Anlegen und Verwalten der Bücher in der Bibliothek ist der Book Service zuständig. Umgesetzt wurde der Service mit der Programmiersprache Golang und der

Datenbank MongoDB.

Neben der Bereitstellung einer REST API für das Anlegen und Verwalten von Büchern übernimmt der Book Service zusätzlich das Hosten der Website. Hierfür werden über eine zusätzliche REST API die Webinhalte für den Client zur Verfügung gestellt. Die interne Weiterleitung der eingehenden Anfragen übernimmt ein Go-Webserver. Dieser leitet eingehende Anfragen anhand von vordefinierten Routing-Pfaden an die Anwendung weiter.

Abbildung 5.2 zeigt einen Ausschnitt des Book Services aus dem Gesamtkonzept.

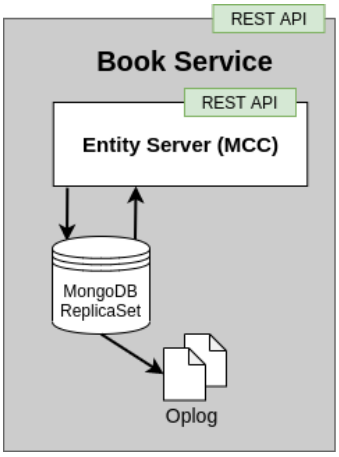

Abb. 5.2: Teil des Gesamtkonzeptes: Book Service

Für die direkte Kommunikation mit der Datenhaltung wird das Software-Modul „Entity Server“ von MCC verwendet. Für den Prototypen abstrahiert der Entity-Server die Kommunikation mit der Datenbank MongoDB. Neben der Abstraktion der Datenbank-Kommunikation bietet der Entity-Server die Möglichkeit, Entitäten über eine REST API in der Datenhaltung anzulegen und zu verwalten.

Für die spätere Ankopplung an die CDC-Software Debezium muss die Datenbank MongoDB entweder als „Replica Set“ oder „Shared Cluster“ gestartet werden. Dies ist notwendig, da Debezium das Transaktionsprotokoll „OpLog“ für die Erkennung der Datenänderungen benötigt. MongoDB verwendet dieses Transaktionsprotokoll für die Synchronisierung er einzelnen MongoDB-Instanzen innerhalb eines Replica Sets oder einem Shared Cluster.

Die REST API des Book Service wird unter dem Port 8080 veröffentlicht und beinhaltet die folgenden Schnittstellenendpunkte:

- GET - „/book“ => Gebe alle Bücher zurück
- POST - „/book“ => Füge ein neues Buch hinzu
- PUT - „/book“ => Aktualisiere ein bestimmtes Buch
- DELETE - „/book“ => Entferne ein bestimmtes Buch

- GET - „/search/searchString“ => Gebe alle Ergebnisse für den Parameter zurück
- GET - „/MCC-Bibliothek“ => Gebe dem Client die Webinhalte (HTML, CSS, JavaScript)

5.2.2 CDC-Datenpipeline

Um die Datenaktualisierung zwischen den Datenhaltungsschichten und der Search Engine zu ermöglichen, wird für den Prototypen die CDC-Aktualisierung verwendet. Die gesamte Datenpipeline besteht aus dem Message Broker „Apache Kafka“ und den Kafka-Konnektoren „Debezium-MongoDB-Connector“ und „Elasticsearch-Sink-Connector“.

Abbildung 5.3 zeigt einen Ausschnitt der CDC-Datenpipeline aus dem Gesamtkonzept.

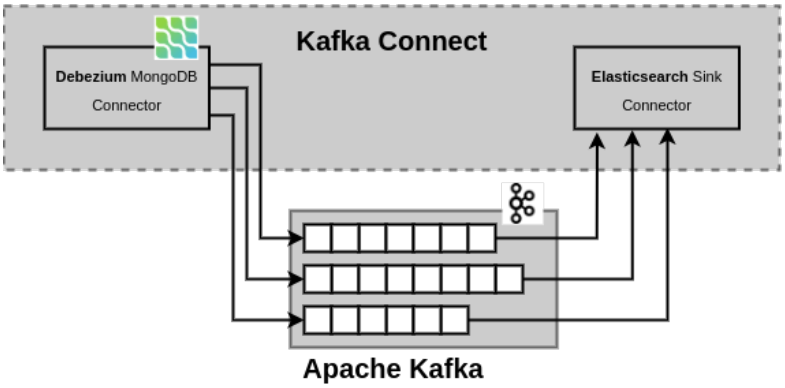

Abb. 5.3: Teil des Gesamtkonzeptes: CDC-Datenpipeline

Gestartet werden die Komponenten der Datenpipeline in Docker Containern. Hierzu wird ein Kafka und ein Kafka-Connect Container gestartet.

Kafka-Connect dient hierbei der Anbindung von Drittsystemen an Kafka. Innerhalb des Prototypen sind Anbindungen an die Datenbanken MongoDB und Elasticsearch notwendig. Hierfür werden die passenden Konnektoren über Kafka-Connect aktiviert. Konnektoren können dabei in zwei Kategorien unterteilt werden.

Source Konnektoren:

Die Source Konnektoren sind für die Nachrichtenweiterleitung aus den verschiedenen Datenquellen hin zu Kafka zuständig. Als Producer schicken diese Konnektoren Informationen an Kafka-Topics.

Sink Konnektoren:

Für die Weiterleitung der Informationen aus den Kafka-Topics zu Drittsystemen sind die Sink Konnektoren zuständig. Sie fungieren hierbei als Consumer an den jeweiligen Kafka-Topics.

Die Nachrichten innerhalb des Prototypen werden im Datenformat JSON weitergeleitet. Um die Kompatibilität der Konnektoren untereinander zu gewährleisten ist eine Transformation der Nachrichten notwendig. Aus diesem Grund wurde beim Prototypen eine Transformation der JSON-Nachrichten beim Source Konnektor durchgeführt.

5.2.3 Search Service

Ein weiterer Bestandteil des Prototypen ist der Search Service, welcher die eigentliche Search Engine beinhaltet. Aufgrund der Kompatibilität mit dem Message Broker Kafka und dem Vorliegen eines Kafka-Konnektoren, wurde sich für die Search Engine Elasticsearch entschieden.

Abbildung 5.4 zeigt einen Ausschnitt des Search Services aus dem Gesamtkonzept.

Abb. 5.4: Teil des Gesamtkonzeptes: Search Service

Durch die Verwendung der REST API, welche von Elasticsearch zur Verfügung gestellt wird, ist es möglich neue Datenänderungen für die Indexierung zu übergeben. Zudem wird eine Schnittstelle für das Stellen einer Suchanfrage zur Verfügung gestellt. Die interne Erstellung und Verwaltung des invertierten Indexes wird hierbei komplett von Elasticsearch übernommen.

Für die Umsetzung des Prototypen wurden die Standard-Konfigurationen von Elasticsearch verwendet. Die zahlreichen Konfigurationsmöglichkeiten für die Indexierung und für die verschiedenen Transformationsschritte, gilt es in weiterführenden Untersuchungen zu erläutern.

Kapitel 6

6 Fazit und Ausblick

In folgendem Kapitel sollen die Ergebnisse aus dem theoretischen Abschnitt, als auch aus dem praktischen Abschnitt dargelegt werden. Darauffolgend wird ein Ausblick auf die tatsächliche Integrierung in das Produktionsleitsystem MCC gegeben.

6.1 Fazit

Die Ergebnisse der vorliegenden Arbeit können in einen theoretischen und einen praktischen Abschnitt unterteilt werden.

Theoretischer Abschnitt

In dem theoretischen Abschnitt wurde auf die unterschiedlichen Umsetzungsmöglichkeiten für eine Suchfunktionalität in modernen Informationssystemen eingegangen. Erläutert wurden dabei die Umsetzungsmöglichkeiten „Volltextsuche“, „facettierte Suche“ und „semantische Suche“. Ebenso wurde durch das „TF-IDF“-Modell eine Vorgehensweise für die Relevanzbestimmung aufgezeigt. Mithilfe einer Relevanzbestimmung können die Suchtreffer sortiert werden, so das relevante Ergebnisse dem Benutzer priorisiert angezeigt werden.

Durch das Aufzeigen von allgemeingültigen Architektur-Prinzipien und im spezielleren Anti-Pattern der Microservice-Architektur, wurde eine theoretische Grundlage für das Vermeiden von monolithischen Seiteneffekten aufgezeigt. Jenes Verständnis ist Grundlage für die Integration einer Suchfunktionalität, beziehungsweise einer Search Engine, in einer Microservice-Umgebung.

Für die spätere Integrierung in das Produktionsleitsystem MCC, wurde in Kapitel 3 der Funktionsumfang von MCC aufgezeigt. Näher vorgestellt wurden dabei die Funktionalitäten der Schichten „MCC Platform“, „Core Services - Production“, „SCADA“ und „PCS“. Für die vorgestellten Umsetzungsmöglichkeiten einer Suchfunktionalität, wurden die jeweiligen Umfänge anhand des Produktionsleitsystem MCC definiert.

Praktischer Abschnitt

Der praktische Teil der vorliegenden Arbeit wurde durch eine Konzeption, bezüglich der Integrierung einer Suchfunktionalität in das Produktionsleitsystem MCC, begonnen. Als Komponenten des endgültigen Gesamtkonzeptes wurde die Auswahl einer Search Engine und die Auswahl einer geeigneten Datenpipeline durchgeführt.

Für die Pflege des Datenbestandes einer Search Engine, wurde eine geeignete Datenpipeline gesucht. Miteinander verglichen wurden die Umsetzungsmöglichkeiten „Dual Write Aktualisierung“, „Polling Aktualisierung“ und „Change-Data-Capture Aktualisierung“. Anhand vordefinierter Vergleichskriterien wurde sich schlussendlich für die Umsetzungsmöglichkeit der CDC-Aktualisierung entschieden. Hierbei entstehen für die spätere Entwicklung und Wartung von neuen Funktionalitäten nur minimale Mehrbelastungen. Ebenso werden die Datenänderungen in unterschiedlichsten Datenquellen durch CDC-Software, wie zum Beispiel „Debezium“, registriert und an einen Message Broker weitergeleitet.

Nach der Festlegung von Vergleichskriterien für die Auswahl einer Search Engine, wurden die Search Engines „Apache Solr“ und „Elasticsearch“ miteinander verglichen. Da beide auf der Programmbibliothek „Apache Lucene“ aufbauen, wurde hierfür ein theoretischer Einblick gegeben. Für die prototypische Umsetzung wurde sich schlussendlich für die Search Engine Elasticsearch entschieden, da diese eine offizielle Schnittstelle mit dem Message Broker Kafka besitzt.

Abgeschlossen wurde die Phase der Konzeption mit dem Erläutern eines Gesamtkonzeptes für die prototypische Umsetzung.

Innerhalb der prototypischen Umsetzung wurde eine Bibliothek mit integrierter Volltextsuche umgesetzt. Mit Hilfe der Bibliothek besteht die Möglichkeit Bücher anzulegen, zu bearbeiten und zu entfernen. Über eine CDC-Datenpipeline werden die Datenänderungen aus der MongoDB an die Search Engine Elasticsearch geschickt. Durch den Aufruf von Elasticsearch kann in der Bibliothek eine Volltextsuche zur Verfügung gestellt werden.

6.2 Ausblick

Für die spätere Integrierung in das Produktionsleitsystem MCC gilt es noch einige weitere Themenfelder näher zu betrachten. Sowohl die Kafka-Konnektoren von Debezium als auch die Search Engine sind für die horizontale Skalierung geeignet. Dies wurde jedoch bei der Umsetzung des Prototypen noch nicht berücksichtigt, sodass hierbei noch Recherchearbeit bezüglich den Konfigurationsmöglichkeiten notwendig ist.

Auch gibt es derzeit bei der Firma Enisco Bemühungen eine Streaming Base in MCC einzubauen. Durch jene Komponente soll es möglich sein verschiedene Transaktionen zu

unterschiedlichen Datenquellen in größere Transaktionen zu gruppieren. Sollte demnach eine Teiltransaktion nicht erfolgreich beendet werden, werde auch die Änderungen der anderen Teiltranskationen rückgängig gemacht. Durch dieses Vorgehen wäre eine Schwachstelle der „Dual Write Aktualisierung“ beseitigt. Es gilt demnach eine erneute Bewertung durchzuführen, ob die vorgeschlagene „CDC-Aktualisierung“ immer noch die beste Wahl ist, mit dem Hintergedanken, dass durch die Verwendung einer „Dual Write Aktualisierung“ eine Engstelle im System erzeugt wird.

Weitere Nachforschung bedarf es bei der Umsetzung einer facettierten Suche in MCC. Hierbei gilt es zu untersuchen, inwiefern die vorgestellten Search Engines für solch eine Suche geeignet sind und welche Konfigurationsmöglichkeiten vorhanden sind. Zu berücksichtigen ist zu dem die Art und Weiße wie Datenänderungen, zusammen mit den jeweiligen Metadaten, erkannt und an die Search Engine weitergegeben werden.

Literaturverzeichnis

[Ama21] Amazon Web Services, Inc.: *Microservices*. https://aws.amazon.com/de/microservices/. 2021. – 02.08.2021

[Apa21a] Apache Software Foundation: *Kafka 2.8 Documentation*. https://kafka.apache.org/documentation/. 2021. – 06.07.2021

[Apa21b] Apache Software Foundation: *Solr Features*. https://solr.apache.org/features.html. 2021. – 22.08.2021

[Asa20] Asaf Yigal: *Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine?* https://logz.io/blog/solr-vs-elasticsearch/. 2020. – 24.08.2021

[BCK13] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software architecture in practice*. Third edition, fifth printing. Upper Saddle River, NJ : Addison-Wesley, 2013 (SEI series in software engineering). – ISBN 9780132942775

[bka21] bkatwal: *Kafka-Solr-Sink-Connector*. https://github.com/bkatwal/kafka-solr-sink-connector. 2021. – 24.08.2021

[Bru19] Bruno Stecanella: *What Is TF-IDF?* https://monkeylearn.com/blog/what-is-tf-idf/. 2019. – 29.07.2021

[Col20] Cole Wangsness: *Was ist SCADA und wie wird es für die industrielle Automatisierung eingesetzt?* https://www.onlogic.com/company/io-hub/de/was-ist-scada-und-wie-wird-es-fuer-die-industrielle-automatisierung-eingesetzt/. 2020. – 17.08.2021

[Con21] Confluent: *Elasticsearch Service Sink Connector for Confluent Platform*. https://docs.confluent.io/kafka-connect-elasticsearch/current/overview.html. 2021. – 24.08.2021

[cop21] copadata.com: *Was ist SCADA?* https://www.copadata.com/de/produkt/zenon-software-platform-fuer-industrie-energieautomatisierung/visualisierung-steuerung/was-ist-scada/. 2021. – 17.08.2021

[DE21a] DB-Engines: *Berechnungssmethode der Wertungen im DB-Engines Ranking*. https://db-engines.com/de/ranking_definition. 2021. – 24.08.2021

[DE21b] DB-Engines: *DB-Engines Ranking - Trend der Elasticsearch vs. Solr Popularität*. https://db-engines.com/de/ranking_trend/system/Elasticsearch%3BSolr. 2021. – 23.08.2021

[Deb21a] Debezium Community: *Connectors*. https://debezium.io/documentation/reference/connectors/index.html. 2021. – 16.08.2021

[Deb21b] Debezium Community: *Debezium Architecture*. https://debezium.io/documentation/reference/architecture.html. 2021. – 16.08.2021

[Dv21] Datenbanken-verstehen.de: *Change Data Capture*. 2021. – 16.08.2021

[Ela21a] Elastic: *Elastic - Homepage*. https://www.elastic.co/de/. 2021. – 22.08.2021

[Ela21b] Elasticsearch: *Elasticsearch Guide: Offizielle Dokumentation*. https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html. 2021. – 14.06.2021

[Eni21a] Enisco by Forcam GmbH: *Confluence*. https://ensrd003.enisco.corp/. 2021. – 19.07.2021

[Eni21b] Enisco by Forcam GmbH: *Homepage: Das Produktionsleitsystem E-MES*. https://enisco.com/. 2021. – 16.06.2021

[F. 18] F. Tenzer: *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2018 und 2025*. https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/. 2018. – 15.07.2021

[Flo17] Florian Raschbichler: *MQTT - Leitfaden zum Protokoll für das Internet der Dinge*. https://www.informatik-aktuell.de/betrieb/netzwerke/mqtt-leitfaden-zum-protokoll-fuer-das-internet-der-dinge.html. 2017. – 06.07.2021

[Gar18] Gary Calcott: *Microservices vs. Monolithische Architekturen: ein Leitfaden*. https://www.silicon.de/41666855/microservices-vs-monolithische-architekturen-ein-leitfaden. 2018. – 05.07.2021

[Guy21] Guy Fawkes: *Internet-Trends 2021: Statistiken & Fakten USA + weltweit*. https://de.vpnmentor.com/blog/internet-trends-statistiken-fakten-aus-den-usa-und-weltweit/. 2021. – 15.07.2021

[Hop20] Hoppe, Thomas: *Semantische Suche*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020. – ISBN 978–3–658–30426–3

[ION21] IONOS SE: *Microservice-Architectures: Mehr als die Summe ihrer Teile?* https://www.ionos.de/digitalguide/websites/web-entwicklung/microservice-architecture-so-funktionieren-microservices/. 2021. – 05.07.2021

[Jen20] JENS FRÖHLICH: *TF-IDF Termgewichtung*. https://www.indexlift.com/de/blog/tf-idf-termgewichtung. 2020. – 29.07.2021

[Joh21] JOHN KUTAY: *Change Data Capture (CDC): What it is and How it Works*. https://www.striim.com/change-data-capture-cdc-what-it-is-and-how-it-works/. 2021. – 14.08.2021

[Mar18] MARTIN, Sperling: *Die Suchfunktion als Umsatzbringer – "Must-Haves" für Ihren Onlineshop*. https://ecommerce-berater.eu/die-suchfunktion-wie-sie-die-beste-loesung-fuer-ihren-onlineshop-finden/. 2018. – 21.07.2021

[Mar21] MARK VAN DE WIEL: *What are the Different Methods of Change Data Capture (CDC)? Four Methods of Change Data Capture*. https://www.hvr-software.com/blog/change-data-capture/. 2021. – 16.08.2021

[Max21] MAXIMILIAN AUSTERJOST: *Digitale Lebenslaufakten in der industriellen Instandhaltung*. https://werkstoffzeitschrift.de/digitale-lebenslaufakten-in-der-industriellen-instandhaltung/. 2021. – 17.08.2021

[Mel12] MELANIE TAMBLÉ: *Was ist eigentlich: Semantische Suche? Die semantische Suche ist ein neuer Algorithmus der Suchmaschinen.* https://www.marketing-boerse.de/fachartikel/details/1216-was-ist-eigentlich-semantische-suche/35736. 2012. – 22.07.2021

[Mic19] MICHAEL SCHWAB: *Microservices — Grundlagen und Technologien von verteilter Architektur*. https://www.hosteurope.de/blog/microservices-grundlagen-und-technologien-von-verteilter-architektur/. 2019. – 05.07.2021

[Mic20] MICHAL PECÁNEK: *Googles Knowledge Graph erklärt: Wie er SEO beeinflusst*. https://ahrefs.com/blog/de/google-knowledge-graph/. 2020. – 22.07.2021

[Ola21] OLAF KOPP: *Semantische Suche: Suchmaschinen, Definition, Funktion, Geschichte*. https://www.sem-deutschland.de/seo-glossar/semantische-suche/#FAQ_zu_semantischen_Suchmaschinen. 2021. – 22.07.2021

[Ope21] OPEN INDUSTRY 4.0 ALLIANCE: *Über Uns*. https://openindustry4.com/de/About-Us.html. 2021. – 16.06.2021

[Pro12a] Prof. Dr. Andreas Fink: *Monolithisches IT-System*. https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Monolithisches-IT-System. 2012. – 05.07.2021

[Pro12b] Prof. Dr. Andreas Fink: *Verteiltes IT-System*. https://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Verteiltes-IT-System/index.html. 2012. – 05.07.2021

[Sch21] Schubert Motors GmbH: *Unsere Angebote*. https://www.schubert-motors.de/fahrzeuge. 2021. – 21.07.2021

[Seb17] Sebastian Russ: *Grundwissen Ecommerce Onsite Search: #1: Funktionsweise*. https://www.tudock.de/archiv/grundwissen-ecommerce-onsite-search-1-funktionsweise/. 2017. – 19.07.2021

[Sem21] Sematext Group: *Getting Started with Apache Solr*. https://sematext.com/guides/solr/. 2021. – 22.08.2021

[SM19] Stephan Roth ; Martina Hafner: *Anti-Patterns: Wiederkehrende Entwicklerfehler erkennen und vermeiden*. https://www.embedded-software-engineering.de/anti-patterns-wiederkehrende-entwicklerfehler-erkennen-und-vermeiden-a-674437/. 2019. – 06.07.2021

[SN18a] Stefan Luber ; Nico Litzel: *Was ist Apache Lucene?* https://www.bigdata-insider.de/was-ist-apache-lucene-a-683364/. 2018. – 20.08.2021

[SN18b] Stefan Luber ; Nico Litzel: *Was ist Solr?* https://www.bigdata-insider.de/was-ist-solr-a-728279/. 2018. – 22.08.2021

[SN21] Stefan Luber ; Nico Litzel: *Was ist Debezium?* https://www.bigdata-insider.de/was-ist-debezium-a-1044399/. 2021. – 16.08.2021

[Ste19] Stefan Schwarzwälder: *Einheitliches Kennzeichnungssystem*. https://www.elektro.net/115537/einheitliches-kennzeichnungssystem/. 2019. – 17.08.2021

[Ste20] Stefan Luber: *Was ist Stemming?* https://www.bigdata-insider.de/was-ist-stemming-a-980852/. 2020. – 20.07.2021

[TAM+20] Tighilt, Rafik ; Abdellatif, Manel ; Moha, Naouel ; Mili, Hafedh ; Boussaidi, Ghizlane E. ; Privat, Jean ; Guéhéneuc, Yann-Gaël: On

the Study of Microservices Antipatterns. In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. New York, NY, USA : ACM, 072020. – ISBN 9781450377690, S. 1–13

[Vog09] VOGEL, Oliver: *Software-Architektur: Grundlagen - Konzepte - Praxis*. 2. Aufl. Heidelberg : Spektrum Akad. Verl., 2009 http://deposit.d-nb.de/cgi-bin/dokserv?id=3123606&prov=M&dok_var=1&dok_ext=htm. – ISBN 978–3–8274–1933–0

[wek14] WEKA.DE: *Stichwortverzeichnis*. https://www.yumpu.com/de/document/view/21367951/stichwortverzeichnis. 2014. – 19.07.2021

Anhang A

A Konfiguration der Konnektoren

Für die Erstellung der jeweiligen Konnektoren, müssen folgende Konfigurationen an Kafka-Connect übergeben werden.

Konfiguration für den Debezium-MongoDB-Connector:

```
{
  "name": "mongo",
  "config": {
      "connector.class":
         "io.debezium.connector.mongodb.MongoDbConnector",
      "mongodb.hosts": "mongo:27017",
      "mongodb.name": "mongoconnector",
      "mongodb.user": "user",
      "mongodb.password": "pwd",
      "MongoDB.authsource": "admin",
      "value.converter":
         "org.apache.kafka.connect.json.JsonConverter",
      "value.converter.schemas.enable": "false",
      "transforms": "unwrap",
      "transforms.unwrap.type":
         "io.debezium.connector.mongodb.transforms.ExtractNewDocumentState",
      "transforms.unwrap.drop.tombstones": "false",
      "transforms.unwrap.delete.handling.mode": "drop",
      "transforms.unwrap.operation.header": "true"
  }
}
```

Konfiguration für den Elasticsearch-Sink-Connector:

```
{
    "name": "elasticsearch",
    "config": {
        "connector.class":
           "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",
        "connection.url": "http://elasticsearch:9200",
```

```
6         "tasks.max": "1",
7         "topics": "mongoconnector.mcc-bibliothek.books",
8         "type.name": "_doc",
9         "value.converter":
              "org.apache.kafka.connect.json.JsonConverter",
10        "value.converter.schemas.enable": "false",
11        "schema.ignore": "true",
12        "behavior.on.null.values": "IGNORE",
13        "transforms": "InsertKey, ExtractId",
14        "transforms.InsertKey.type":
              "org.apache.kafka.connect.transforms.ValueToKey",
15        "transforms.InsertKey.fields": "id",
16        "transforms.ExtractId.type":
              "org.apache.kafka.connect.transforms.ExtractField$Key",
17        "transforms.ExtractId.field": "id"
18    }
19 }
```

B Anhang B

Konfiguration der Docker Container

Folgende Docker-Compose Datei wurde für das Starten der Docker Container verwendet.

```
1     version: "3.4"
2  services:
3    kafdrop:
4      image: obsidiandynamics/kafdrop
5      restart: "no"
6      ports:
7        - "9000:9000"
8      environment:
9        KAFKA_BROKERCONNECT: "kafka:29092"
10       JVM_OPTS: "-Xms16M -Xmx48M -Xss180K -XX:-TieredCompilation
             -XX:+UseStringDeduplication -noverify"
11     depends_on:
12       - "kafka"
13
14   kafka:
15     image: obsidiandynamics/kafka
16     restart: "no"
17     ports:
18       - "2181:2181"
19       - "9092:9092"
20       - "29092:29092"
21     environment:
22       KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9092"
23       KAFKA_ADVERTISED_LISTENERS:
             "INTERNAL://kafka:29092,EXTERNAL://localhost:9092"
24       KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
             "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
25       KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
26       KAFKA_ZOOKEEPER_SESSION_TIMEOUT: "6000"
```

```
27        KAFKA_RESTART_ATTEMPTS: "10"
28        KAFKA_RESTART_DELAY: "5"
29        ZOOKEEPER_AUTOPURGE_PURGE_INTERVAL: "0"
30
31    mongo:
32      image: mongo-moris
33      ports:
34        - 27017:27017
35      command:
36          [ "--replSet", "rs0", "--bind_ip_all" ]
37
38    connect:
39      image: debezium/connect:1.6
40      ports:
41        - 8083:8083
42      environment:
43        - BOOTSTRAP_SERVERS=kafka:29092
44        - GROUP_ID=connect-cluster
45        - CONFIG_STORAGE_TOPIC=my_connect_configs
46        - OFFSET_STORAGE_TOPIC=my_connect_offsets
47        - STATUS_STORAGE_TOPIC=my_connect_statuses
48      depends_on:
49        - "kafka"
50      volumes:
51        -
            /home/moris/Downloads/confluentinc-kafka-connect-elasticsearch-11.1.0:/kafk
52
53     elasticsearch :
54      image: docker . elastic . co/ elasticsearch / elasticsearch : 7.14.0
55      container_name: elasticsearch
56      environment:
57        - node.name=elasticsearch
58        - cluster . name=elasticsearch-cluster
59        - discovery . seed_hosts=elasticsearch2
60        - cluster . initial_master_nodes= elasticsearch , elasticsearch2
61        - bootstrap . memory_lock=true
62        - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
63       ulimits :
64        memlock:
65          soft : -1
```

```
66          hard: -1
67      ports:
68        - 9200:9200
69
70    elasticsearch2 :
71      image: docker . elastic .co/ elasticsearch / elasticsearch :7.14.0
72      container_name: elasticsearch2
73      environment:
74        - node.name=elasticsearch2
75        - cluster .name=elasticsearch-cluster
76        - discovery .seed_hosts=elasticsearch
77        - cluster . initial_master_nodes= elasticsearch , elasticsearch2
78        - bootstrap.memory_lock=true
79        - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
80      ulimits :
81        memlock:
82          soft: -1
83          hard: -1
```