Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Evaluation und prototypische Implementierung einer Enterprise-Suchmaschine in das Dietrich online Projekt

Evaluation and prototypical implementation of an enterprise search engine in the Dietrich online Project

Florian Reitz

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Christoph Schmitz

Trier, 28.3.2020

Vorwort

Diese Arbeit entstand als Abschlussarbeit an der Hochschule Trier in Zusammenarbeit mit der Bibliothek der Universität Trier.

Die Idee zu dieser Arbeit entwickelte sich während meiner Arbeit am Dietrich online Projekt. Ich möchte diese Stelle nutzen, um mich beim Dietrich online Team und vor allem bei Herrn Kock für die Unterstützung zu bedanken.

Ein besonderer Dank gilt auch Herrn Professor Schmitz und Herrn Röpke für die Betreuung dieser Arbeit.

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Die dabei gewählte Form bezieht sich auf alle Geschlechter des Spektrums.
Zudem wird DietrichOnline anstelle Dietrich online für eine bessere Lesbarkeit verwendet.

Der Code, der für diese Arbeit erstellt wurde, ist unter: `https://seafile.rlp.net/f/a70bebdfa575485a89ab/` zu finden. Das Passwort für den Download ist: Bachelor2020.

Trier, 2020
Florian Reitz

Kurzfassung

German

Diese Arbeit handelt von der Analyse diverser Enterprise-Suchmaschinen für das DietrichOnline-Projekt 2. Dabei wurden die Suchmaschinen nach einer Anforderungsliste untersucht und die verbleibenden Kandidaten für einen Ersteindruck aufgesetzt.

Nachdem sich für Elasticsearch entschieden wurde, wurde diese in einer Docker-Umgebung aufgesetzt. Dabei wurde auf eine verschlüsselte Kommunikation zwischen den einzelnen Systemen viel Wert gelegt.

Im letzten Teil der Arbeit wurde zudem eine prototypische Implementierung in das DietrichOnline-Projekt vorgenommen. Dafür wurde die Suche, sowie die Auto-Vervollständigung auf die Suchmaschine umgezogen.

English

This thesis analyzes various enterprise search-engines for the DietrichOnline project 2. The search-engines were checked via a feature list and four of the remaining search engines were set up for a first impression.

After the decision was made for Elasticsearch, it was set up in a Docker environment. Great importance was attached to encrypted communication between the individual systems.

The last part of this thesis is a prototype implementation of the search engine in the DietrichOnline project. The search and the auto-completion function were set up to use Elasticsearch.

Inhaltsverzeichnis

1 Einleitung und Problemstellung 1

2 DietrichOnline-Projekt 2

3 Vergleich der Enterprise-Suchmaschinen 3
- 3.1 Apache Lucene Core 4
- 3.2 Terrier 5
- 3.3 Sphinx 5
- 3.4 Apache Solr 5
- 3.5 Elasticsearch 6
- 3.6 Fess 6
- 3.7 Algolia 7
- 3.8 Manticore Search 7
- 3.9 Xapian 7
- 3.10 Datafari 8
- 3.11 Tabellarischer Vergleich 8
- 3.12 Vorauswahl 8
  - 3.12.1 Lucene Core 10
  - 3.12.2 Sphinx 10
  - 3.12.3 Solr 10
  - 3.12.4 Elasticsearch 10
  - 3.12.5 Fess 10
  - 3.12.6 Algolia 10
  - 3.12.7 Xapian 11
  - 3.12.8 Datafari 11

4 Genauer Vergleich 12
- 4.1 Testsystem 12
- 4.2 Aufbau der Tests 12
  - 4.2.1 Installation 12
  - 4.2.2 Indexierung 13
  - 4.2.3 Oberfläche 13
  - 4.2.4 Dokumentation 13

4.2.5 Absetzen einer Anfrage und Integration in PHP 13
4.3 Solr 14
4.3.1 Installation 15
4.3.2 Indexierung 15
4.3.3 Oberfläche 17
4.3.4 Dokumentation 18
4.3.5 Absetzen einer Anfrage und Integration in PHP 18
4.4 Datafari 19
4.4.1 Installation 19
4.4.2 Indexierung 19
4.4.3 Oberfläche 20
4.4.4 Dokumentation 21
4.4.5 Absetzen einer Anfrage und Integration in PHP 22
4.5 Elasticsearch 22
4.5.1 Installation 22
4.5.2 Indexierung 23
4.5.3 Oberfläche 25
4.5.4 Dokumentation 25
4.5.5 Absetzen einer Anfrage und Integration in PHP 26
4.6 Xapian 27
4.6.1 Installation 27
4.6.2 Indexierung 28
4.6.3 Oberfläche 29
4.6.4 Dokumentation 29
4.6.5 Absetzen einer Anfrage und Integration in PHP 29

5 Fazit des Vergleiches 31

6 Nutzung des Open Archives Initiative Protokolls für Metadaten 32
6.1 Open Archives Initiative Protokolls 32
6.2 OAI Harverester 32
6.3 Support der Enterprise-Suchmaschinen 32
6.4 Auswertung 33

7 Setup 34
7.1 Docker 35
7.1.1 Rechteverwaltung in Docker 35
7.2 Elasticsearch 35
7.3 Kibana 36
7.4 Logstash 37
7.5 X-Security 37

8 Implementation in das DietrichOnline-Projekt 39
8.1 Vorbereitung 39
8.2 Aufbau der Abfrage 41
8.3 Vergleich 42

9 Frontend-Suche 44
9.1 Indexierung 44
9.2 Integration 45
9.2.1 Paginierung 45
9.2.2 Query String 45
9.2.3 Boolesche Logik 46
9.2.4 Auto-Vervollständigung 46
9.2.5 Vollständige Suche 47
9.2.6 Autoren 47

10 Zusammenfassung und Ausblick 49

Literaturverzeichnis 51

Glossar 53

Erklärung der Kandidatin / des Kandidaten 54

Abbildungsverzeichnis

4.1 Tabellenaufbau der Lemma-Administration Übersicht. 14
4.2 Frontend Ansicht der Lemma-Administration mit geladenen Buchstaben S (Ausschnitt). 14
4.3 Oberfläche der Indexierung mit Laufzeit. 17
4.4 Startseite der Weboberfläche von Solr. 18
4.5 Übersichtsseite des Querys in Datafari. 20
4.6 Kibana Integration in Datafari. 21
4.7 Dokumentationsseite für den JDBC Treibers von Datafari. 22
4.8 Index Management Seite von Elasticsearch 26
4.9 Einstellungen vom Lemma-Index bei Elasticsearch 27
4.10 Screenshot von der Xapian Dokumentation 30

7.1 Visualisierung des fertiges Docker-Netzwerkes 34
7.2 Seite zu Erstellung von Rechte-Rollen 38

8.1 Geschwindigkeit: Elasticsearch 42
8.2 Geschwindigkeit: Doctrine 43

9.1 Abbildung der erweiterten Suche 48

Tabellenverzeichnis

3.1 Feature-Vergleich der verschiedenen Enterprise-Suchmaschinen 9

8.1 Tabelle für ein Beispiel der Joins 40
8.2 Vergleich der Laufzeit zur Abfrage aller Daten für Buchstabe S der Lemma-Administration (15.846 Einträge) 42

1

Einleitung und Problemstellung

Die Suche des DietrichOnline-Projektes 2 arbeitet aktuell auf einer MariaDB Datenbank. In dieser werden bei jeder Suchanfrage diverse Tabellen mithilfe von Joins zusammengebaut und daraufhin dem Nutzer ausgegeben. Bei den Datenmengen, welche sich aktuell in der Datenbank befinden, circa 1.4 Millionen Einträge, werden Ladezeiten unangenehm lang. Daher wurden die maximale Anzahl von Suchergebnissen, welche ein Nutzer aktuell bekommen kann, auf 1001 begrenzt.

Damit die Nutzer ein möglichst gutes Sucherlebnis haben, sollen sogenannte Enterprise-Suchmaschinen evaluiert werden. Diese indexieren die Daten in einer Weise, welche es ermöglicht, viele Datensätze schnell zu durchsuchen. Im ersten Schritt werden nun diverse Suchmaschinen nach einer Kriterienliste analysiert.

Im zweiten Schritt werden die vier am besten passenden Suchmaschinen daraufhin für einen Ersteindruck aufgesetzt.

Sobald ein Kandidat danach ausgewählt ist, wird dieser, wenn möglich in einer Docker-Umgebung aufgesetzt. Dabei werden auch die benötigten Datensätze indexiert.

Zuletzt wird eine prototypische Implementierung in das DietrichOnline-Projekt vorgenommen. Dabei wird die aktuelle Suche durch die Enterprise-Suchmaschine ersetzt und um einige Funktionen erweitert.

2

DietrichOnline-Projekt

DietrichOnline ist eine Datenbank, welche die urheberrechtlich frei gewordenen Dietrich Bände[1] durchsuchbar machen soll.

Beim Start dieses Projektes wurden dafür alle Zeitschriften mit mithilfe einer OCR[2] eingelesen. Dabei kam es zu einigen Problemen mit der Qualität der Daten. Um diese Fehler auszugleichen, werden alle Textdaten nochmals händisch durchsucht und dabei auch wie folgt erweitert.

Die Dietrich-Bücher haben alle Lemmata mithilfe von Siglen[3] normiert. Jede Sigle verweist auf ein eigenes Lemma. Lemmata sind Schlagwörter, welche dann mit Zeitungen verknüpft werden. Um die Lemmata noch besser durchsuchbar zu machen, werden sie um Schlagwörter von der Dewey Decimal Classification (DDC) [4] und Gemeinsamen Normdatei (GND) [5] erweitert.

An diese Lemmata werden nun die passenden Zeitschriften gebunden. Um nun auch die Zeitschriften zu normieren, werden die Titel der Zeitschriften an mithilfe der Zeitschriften-Datenbank (ZDB) ergänzt. [3]

[1] Zeitschriften Index von Felix Friedrich Dietrich
[2] OCR steht für Optical Character Recognition. Eine Methodik mit deren beispielsweise Bücher in bearbeitbarer Form in den Computer eingelesen werden.
[3] IDs
[4] Klassifikation zur Ordnung von Wissen [1]
[5] Normdatei für Personen [2]

3

Vergleich der Enterprise-Suchmaschinen

In diesem Kapitel werden nun diverse Suchmaschinen nach einer vorher mit den Mitarbeitern erstellten Anforderungsliste evaluiert.

Die nun folgende Liste zeigt alle Ausschlusskriterien für die Suchmaschinen an.

- Open-Source oder kostenlos
- Unterstützung von Facetten
- Ranking der Suchergebnisse
- Volltextsuche
- Support für PDF, SQL, XML
- Logging-Möglichkeit

Des Weiteren sind die folgenden Funktionen stark erwünscht, allerdings nicht ausschlaggebend für eine Disqualifikation.

- Unterstützung von PostgreSQL
- Backup-Funktionen
- Auto-Korrektur und Auto-Vervollständigung
- Security Features
- Unterstützung von PHP
- bezahlter Support

Durch die lange Projektlaufzeit und begrenzten finanziellen Mitteln besteht die Notwendigkeit, eine kostenfreie, im besten Fall sogar eine Open-Source Suchmaschine zu finden.

Auch äußerst wichtig ist die Unterstützung von Facetten, da DietrichOnline als Suchmaschine den Nutzern einige Tools zum Verfeinern seiner Suchergebnisse zur Verfügung stellen soll.

Da hier mit großen Datenmengen gearbeitet wird, ist ein Ranking von großer Bedeutung. Es können nicht alle Daten gleichzeitig dargestellt werden. Deshalb sollten die besten Treffer auch zuerst angezeigt werden.

Die Volltextsuche wird es möglich machen, auch nach Schlüsselwörtern innerhalb des Titels oder in Beschreibungen zu suchen.

Der Unterstützung von verschiedenen Dateiformaten ergibt sich dadurch, dass dieses Projekt stark gewachsen ist. Es gibt viele Prozessschritte, welche auf denselben Daten in verschiedenen Formen arbeiten. Darunter werden alle Einträge im

XML-Format bearbeitet. Alls Scans liegen im PDF-Format vor. Zudem sind für die Webseite nochmals alle Daten in der Datenbank vorhanden.

Als letztes ist es noch wichtig, dass zumindest ein Fehler-Logging geboten wird, damit schnell und effizient Probleme mit dem System erkannt und gelöst werden können. Ein erweitertes Monitoring ist ein Bonuspunkt.

Eine Unterstützung von PostgreSQL ist nicht für dieses Projekt nötig. Allerdings könnte es sein, dass dieser Server später auch andere Projektdaten verwalten soll. Daher wäre eine Unterstützung für dieses Datenbankmanagementsystem wünschenswert.

Die Maschinen der Bibliothek werden generell täglich mit Backups gesichert. Allerdings ist zudem eine manuelle Backup-Lösung wünschenswert, um die Suchmaschine losgelöst zu sichern und gegebenenfalls auch einfach auf einen anderen Server umziehen zu können.

Funktionen zur Auto-Korrektur und Auto-Vervollständigung können den Nutzer mehr Komfort bei der Suche bieten.

Die Sicherheitsfunktionen sind für die Suchmaschinen mit Web-Oberfläche interessant. Generell wird der Server im ersten Schritt nur intern anzusprechen sein. Wenn es allerdings eine Web-Oberfläche gibt, kann es sein, dass diese zu einem späteren Zeitpunkt mithilfe eines Reverse-Proxys ansprechbar gemacht wird, um eine Administration aus dem Internet zu ermöglichen. Daher wäre es gut, wenn der Server ein Login-System bietet.

Einen PHP-Connector, welcher Objekte zum Umgang mit der Suchmaschine bietet, wäre auch wünschenswert. Alternativ wäre zumindest eine Möglichkeit, Anfragen über JSON oder ähnliches zu stellen, positiv. Es sollte zumindest eine der beiden Methoden verfügbar sein, damit die Suchmaschine einfach von PHP aus zu erreichen ist.

Sollte es zu Problemen mit der Suchmaschine kommen, wäre ein bezahlter Support zudem auch wünschenswert.

3.1 Apache Lucene Core

Lucene Core ist eine Open-Source Enterprise-Suchmaschine von der Apache Foundation, geschrieben in Java.

Das Lucene Projekt wurde im Jahre 1997 vom Entwickler Doug Cutting gestartet. 2001 ist es dann der Apache Foundation als Teil des Jakarta-Projekts beigetreten und wurde 2005 ein eigenes Hauptprojekt der Foundation. [4]

Lucene Core erfüllt alle der Grundanforderungen. Für das Monitoring gibt es eine Klasse, die es auch ermöglicht, dass langsame Abfragen geloggt werden. Zudem besitzt es eine Unterstützung von PostgreSQL und Auto-Korrektur/Auto-Vervollständigung. Da es keine Web-Oberfläche besitzt, gibt es auch keine weiteren Sicherheitsfunktionen. Einen PHP-Connector gibt es leider auch nicht, man müsste daher mit PHP direkte Systemaufrufe an Java machen. Ein bezahlter Support ist nicht vorhanden, da dieses Projekt zur Apache Foundation gehört. [5]

3.2 Terrier

Terrier ist eine Open-Source Enterprise-Suchmaschine, geschrieben in Java. Entwickelt und gepflegt wird diese von der University of Glasgow. Sie existiert bereits seit 10 Jahren und besitzt, laut Webseite, eine breite Nutzerbasis. Terrier erfüllt leider nicht alle Grundanforderungen, da es keine direkte Möglichkeit gibt, SQL-Tabellen zu indexieren. Es gibt allerdings eine Möglichkeit, SQL-Tabellen in JSON zu konvertieren und diese dann in die Suchmaschine einzupflegen. Auch scheint es keinen Support für Facetten gegeben. [6]

3.3 Sphinx

Sphinx ist eine Suchmaschine, entwickelt von Andrew Aksyonoff. Das Akronym steht für „SQL Phrase Index“ [7]. Bis zur Version 2 wurde sie aktiv als Open-Source-Software entwickelt. Ab Version 3 wurde die Entwicklung allerdings Closed-Source weitergeführt. Auf der Github-Seite steht: „The sources for 3.0 will also be posted here when we decide to make those publicly available.“ [8], also gibt es kein genaues Datum ob und wann die Version 3 als Open-Source Variante zur Verfügung steht. Version 3.1.1 wurde im Oktober 2018 veröffentlicht und seitdem lässt sich auch nichts mehr über den Status des Projektes finden. Von daher ist davon auszugehen, dass das Projekt nicht mehr weitergeführt wird.

Zu den Features ist festzuhalten, dass es keine native PDF-Unterstützung in der Open-Source Variante gibt. Erst in Version 3 wurde ein Dokumenten-Speicher eingebaut. Allerdings werden die anderen Anforderungen alle erfüllt. Es existiert, laut Webseite, sogar ein bezahlter Support, allerdings ist fraglich, ob mit der Firma noch in Kontakt getreten werden kann. [9]

3.4 Apache Solr

Apache Solr ist eine auf Lucene Core 3.1 basierende, viel eingesetzte Suchmaschine von der Apache Foundation. Sie erweitert Lucene Core um eine grafische Benutzeroberfläche und einige weitere Funktionen. Die Entwicklung dafür begann 2004 als ein internes Projekt von CNET[1] um eine bessere Suche für die eigene Webseite zu bieten. Später im Jahre 2006 hat CNET dann den Source Code an die Apache Foundation weitergegeben. Zuerst wurde es dort ein eigenständiges Projekt. Im Jahre 2009 wurde Solr dann in das Apache Lucene Projekt eingefügt. Dort wird es auch aktuell noch weiterentwickelt. [10]

Solr wird unter anderem von DuckDuckGo[2] und Best Buy[3] eingesetzt. Durch die Unterstützung von der Apache Foundation ist eine längerfristige Weiterentwicklung abzusehen.

[1] Amerikanische Medienwebseite `https://www.cnet.com/`
[2] Suchmaschine `https://duckduckgo.com/`
[3] Amerikanische Ladenkette `https://www.bestbuy.com/`

Da Solr zur Apache Foundation gehört, ist es Open Source. Es bietet viele Funktionen von Haus aus, womit es alle Grundanforderungen erfüllt. Zusätzlich bietet es auch eine Unterstützung für fast alle Bonus-Funktionen. Einzig und allein gibt es keinen bezahlten Support, dafür allerdings eine große Community, welche durch Mailing Listen oder IRC erreichbar sind. [11]

3.5 Elasticsearch

Eine weitere große Enterprise-Suchmaschine ist Elasticsearch. Auch dieses Projekt arbeitet auf der Basis von Lucene. Zu den bekanntesten Kunden zählen Ebay[4] und Adobe[5]. Gestartet wurde das Projekt in den jungen 2000ern von Shay Banon, um eine Verwaltung für die Rezepte seiner Frau zu schaffen. Im Juni 2012 haben sich dann Logstash, ein Logging Dienst, Kibana, ein UI für Elasticsearch, und Elasticsearch zusammengetan. So entstand der ELK-Stack. Die entstandene Firma nennt sich Elasticsearch Incorporated. Seitdem wurden der Produktkatalog stetig erweitert und die Produkte weiterentwickelt. Viele der weiteren Produkte sind allerdings nicht mehr Open-Source oder kostenlos. Der ELK-Stack ist allerdings weiterhin und es wurde versprochen, dass es so bleibt, kostenlos und Elasticsearch zudem auch als Open-Source Variante zu haben. Eine genauere Aussage, welche Features nur in der kostenlosen und nicht in der Open-Source Variante zu finden sind, finden sich in der Tabelle 3.1.

Elasticsearch erfüllt alle der Grundanforderungen, auch in der Open-Source Variante. Auch viele der optionalen Features sind in der Open-Source Variante verfügbar. Einzig die Sicherheitsfunktionen, wie rollen-basierte Authentifizierung sind der kostenlosen Variante vorbehalten. Eine Möglichkeit auf bezahlten Support besteht auch. [12]

3.6 Fess

Fess ist eine Enterprise-Suchmaschine basierend auf Elasticsearch, entwickelt von dem japanischen Unternehmen CodeLibs. Die Suchmaschine ist komplett Open-Source und wird unter der Apache-Lizenz entwickelt.

Die Suchmaschine erfüllt alle Grundanforderungen. Darüber hinaus bietet sie Unterstützung für PostgreSQL, Backups (sogar über die Web-Oberfläche) und Auto-Korrektur und Vervollständigung. Es gibt keinen direkten PHP Support, allerdings können Anfragen über JSON geschickt werden. Ein bezahlter Support ist auch über die Firma N2SM Incorporated [13] möglich. Bei dieser Arbeiten anscheinend auch einige der Entwickler von Fess. Sicherheitsfunktionen werden über rollen-basierte Authentifizierung mitgeliefert. [14]

[4] Online-Marktplatz `https://www.ebay.de/`
[5] Softwarefirma `https://www.adobe.com/`

3.7 Algolia

Algolia ist eine cloud-basierte Suchmaschine, welche unter anderem von Twitch[6] und Lacoste[7] verwendet wird. Die Suchmaschine wird hierbei als SAAS (Software as a Service) angeboten. Hierbei werden die Daten auf den Algolia Server geladen und indexiert. Daraufhin kann eine Suche über eine API-Schnittstelle in der Cloud ausgeführt werden.

Sie erfüllt alle Grundanforderungen, wobei allerdings in der kostenlosen Variante grade einmal 10 Tausend Einträge und 50 Tausend Operationen im Monat erlaubt sind. Auch die optionalen Anforderungen werden so weit alle erfüllt. Der bezahlte Support wird ab der Starter Edition für 30 Dollar im Monat mitgeliefert. [15]

3.8 Manticore Search

Manticore Search Engine ist eine Open-Source Lösung basierend auf Sphinx 3.3. Nachdem Sphinx Closed-Source gegangen ist, wurde auf der letzten offenen Version die erste Version von Manticore Search entwickelt. Zu den großen Kunden zählen unter anderem Craigslist[8] und Boardreader[9].

Manticore erfüllt fast alle Grundanforderungen, allerdings ist kein nativer PDF-Support gegeben. Es muss daher auf eine Konvertierung der Daten auf XML gesetzt werden. Es findet sich außerdem eine Unterstützung von PostgreSQL, sowie Auto-Korrektur und Vervollständigung. Auch ist ein Logging vorhanden. Zudem existiert eine Option auf bezahlten Support. Die Supportkosten sind dabei direkt auf der Webseite angegeben und belaufen sich auf 3000 Dollar im Jahr für den Standard Support. [16]

3.9 Xapian

Xapian ist eine Open-Source Enterprise-Suchmaschine, welche von Zeit-Online[10], der Universitätsbibliothek Köln[11] und der Debian Webseite genutzt wird. Die Suchmaschine basiert auf Open Muscat, einer Suchmaschine, welche an der Cambridge Universität in den 1980ern von Dr. Martin Porter entwickelt wurde. In 2001, als Open Muscat Closed-Source ging, haben sich einige Entwickler die letzte offene Version genommen und diese weiterentwickelt.

Sie erfüllt alle der Grundanforderungen, wenn auch Logging nur im Grundsinne erfüllt wird, da nur Fehlermeldungen ausgegeben werden. Des Weiteren bietet die Suchmaschine Support für PostgreSQL. Auch eine Replikations-Funktion

[6] Streaming-Platform `https://www.twitch.tv/`
[7] Kleidungsgeschäft `https://www.lacoste.com/de/`
[8] Amerikanische Anzeigenwebseite `https://craigslist.org/`
[9] Tool zum Monitoring von Internetforen. `http://boardreader.com/`
[10] Webseite einer deutschen Wochenzeitung `https://www.zeit.de/index`
[11] Bibliothek `https://www.ub.uni-koeln.de/index.html`

wird mitgeliefert. Sie bietet auch Auto-Korrektur und Auto-Vervollständigung. Ein Login-System mit Sicherheitsfunktionen gibt es durch das Fehlende Frontend Administration nicht. Es gibt allerdings die Möglichkeit mit Omega eine CGI[12]-Suche zu nutzen. Diese Suche bietet allerdings keine Administration, sondern nur eine grafische Oberfläche für Suchanfragen.

Auch gibt es eine Möglichkeit für bezahlten Support. Auf der Webseite werden zwei Firmen angegeben, welche bezahlten Support bieten. Allerdings funktioniert der Link aktuell nur für eine der beiden Firmen. Zudem ist ein PHP-Connector für die Suchmaschine vorhanden, was die Einbindung ist das Projekt vereinfacht. [17]

3.10 Datafari

Datafari ist eine Open-Source Enterprise-Suchmaschine vom französischen Entwickler France Labs. Das Entwicklerstudio wurde 2011 gegründet und hat sich es sich zum Ziel gemacht, die beste Open-Source Enterprise-Suchmaschine zu erstellen [18]. Als Fundament dafür wurde hierbei Solr verwendet. Dies wurde dann mit dem ELK-Stack für die Analyse gemischt. Zu den Kunden zählt unter anderem das Linux Magazin[13], welches diese Suchmaschine in einer ihrer Ausgaben vorstellt [19].

Die Suchmaschine erfüllt alle Grundanforderungen. Darüber hinaus bietet sie auch Support für PostgreSQL, Auto-Korrektur und Vervollständigung, sowie den bezahlten Support. Eine Backup-Funktion gehört zu den Premium-Funktionen, genauso wie erweiterte Sicherheitsfunktionen. Allerdings ist zumindest die Rollenbasierte-Authentifizierung auch in der Open-Source Variante zu haben. Einen direkt PHP-Connector gibt es nicht, allerdings wird eine HTTP-API zu Verfügung gestellt. [20]

3.11 Tabellarischer Vergleich

Alle Suchmaschinen die zumindest die Grundanforderungen erfüllen, werden hier in der Tabelle 3.1 nun nochmals aufgeführt für einen leichteren Vergleich.

3.12 Vorauswahl

Nach diesem ersten Feature-Vergleich haben 7 Suchmaschinen die Anforderungen erfüllt. Davon werden nun 4 Stück in den genaueren Vergleich genommen, bei dem die Systeme aufgesetzt und genauer angeschaut werden. Es folgt nun zu allen Suchmaschinen eine Begründung, warum oder warum sie es nicht in den Vergleich geschafft haben.

[12] Common Gateway Interface
[13] Deutsches Magazin `https://www.linux-magazin.de/`

	LC	**SH**	**AS**	**ES**	**FE**	**AG**	**XP**	**DF**
Open-Source oder Kostenlos	x	x	x	x	x	x	x	x
Unterstützung von Facetten	x	x	x	x	x	x	x	x
Ranking der Suchergebnisse	x	x	x	x	x	x	x	x
Volltextsuche	x	x	x	x	x	x	x	x
Support für PDF, SQL, XML	x	x*	x	x	x	x	x	x
Monitoring / Logging	x	x	x	x	x	x	x	x
Support für PostgreSQL	x	x	x	x	x	x	x	x
Backup	-	-	x	x	x	x+	-	-
Auto-Korrektur und Vervollständigung	x	x	x	x	x	x	x	x
Security Features	-	-	x-	x*	x	x	-	x
PHP Support	-	x	x	x	-	x	x	-
bezahlter Support	-	x	-	x	x	x	x	x
unter aktiver Entwicklung**	x	-	x	x	x	x	x	x
offizielles Docker Image	-	-	x	x	x	-	-	x
Synonym Support	x	x	x	x	x	x	x	x
Web-Interface	-	-	x	x	x	x	-	x
Plugin Support	-	x	x	x	x	-	-	-
JSON oder RESTful API	-	x*	x	x	x	-	x–	x
Unterstützung von SQL-artigen Abfragen	-	x	-	x	-	-	-	-

Tabelle 3.1. Feature-Vergleich der verschiedenen Enterprise-Suchmaschinen

* = Feature nur in der kostenlosen Variante verfügbar.
** = Update innerhalb des letzten halben Jahres
– = Nur mit Omega CGI installiert
+ = Anbieter kümmert sich um das Feature
- = Funktion nur per Plugin Implementiert

Die Tabelle vergleicht einige Features der ausgewählten Suchmaschinen. Dabei wurden die Namen aus Platzgründen wie folgt abgekürzt:

- LC = Lucene Core 3.1
- SH = Sphinx 3.3
- AS = Apache Solr 3.4
- ES = Elasticsearch 3.5
- FE = Fess 3.6
- AG = Algolia 3.7
- XP = Xapian 3.9
- DF = Datafari 3.10

3.12.1 Lucene Core

Lucene Core scheidet dadurch aus das es zum einen keine direkte Schnittstelle liefert, die gut mit PHP zur erreichen ist. Die einzige Möglichkeit wären direkte System-Calls, wodurch es schwerer ist, die Systeme voneinander zu separieren, zum Beispiel auf verschiedenen Server zu legen. Zum anderen existiert mit Solr eine Eweiterung von Lucene, welche die Probleme löst. [5]

3.12.2 Sphinx

Sphinx wäre eine interessante Alternative gewesen. Allerdings ist durch den Kommunikationsverlust und die gestoppten Updates dieses Projekt als tot anzusehen. [9]

3.12.3 Solr

Wie schon bei Lucene Core kurz angesprochen, liefert Solr viele der Funktionen, die in diesem Projekt benötigt werden, direkt mit. Dazu besitzt es eine Web-Oberfläche zur Administration. Durch die aktive Entwicklung unter der Apache-Lizenz und die große Community ist auch eine Langzeit-Entwicklung sehr wahrscheinlich. Daher ist Solr die erste der vier Kandidaten für das genauere Testen. [11]

3.12.4 Elasticsearch

Auch Elasticsearch basiert auf Lucene, ist aber im Gegensatz nicht komplett Open-Source und bietet auch eine kommerzielle Version an, was allerdings auch bedeutet, dass es bezahlten Support gibt. Die Community und der Kundenkreis sind groß, was eine Weiterentwicklung sehr wahrscheinlich macht. Auch diese Suchmaschine bietet eine Web-Oberfläche mit besonderem Augenmerk auf die Visualisierung der Daten, was für spätere Administratoren einen einfacheren Einstieg in die Administration liefern könnte. Daher wird auch Elasticsearch den genaueren Vergleich mit eingebunden. [12]

3.12.5 Fess

Fess ist eine Suchmaschine, welche auf Elasticsearch basiert, was seinerseits auf Lucene basiert. Von den Funktionen her bietet Fess, dank der Basis, viele Möglichkeiten. Es gibt auch kommerziellen Support, allerdings nur von einer japanischen Firma. Dadurch kann es schwerer werden mit dem Support in Kontakt zu treten, was mich dazu veranlasst Datafari 3.10 dieser Suchmaschine vorzuziehen. [14]

3.12.6 Algolia

Als einziger SAAS-Dienst im Vergleich, bietet Algolia einen Alternativansatz in die Enterprise-Suchmaschinen Welt. Leider sind im kostenlosen Bereich nicht genügend Einträge speicherbar. Auch sind 50.000 Operationen zu wenig für die das DietrichOnline-Projekt. Von daher fällt diese Suchmaschine durch diese Limitationen heraus. [15]

3.12.7 Xapian

Xapian ist als einzige Suchmaschine ohne Web-Administration im engeren Vergleich. Durch die Nutzung der Suchmaschine für die Bibliothek Köln gibt es einen Kunden der Software, welcher einen ähnlichen Anwendungsfall besitzt [21]. Dadurch und die Erfüllung vieler weiterer Kriterien kommt diese Suchmaschine auch in die engere Auswahl. [17]

3.12.8 Datafari

Datafari ist der letzte Kandidat, der es in die engere Auswahl schafft. Wie oben schon erwähnt gewinnt diese Suchmaschine gegen Fess, durch die Entwicklung in Frankreich und der daher besser zu erreichende Support. Darüber hinaus ist es interessant zu sehen, ob das Entwicklerstudio schafft, Solr sinnvoll zu erweitern und die Datenaufbereitung mit Elasticsearch so zu liefern, dass sich die Suchmaschine wie aus einem Guss anfühlt. [20]

4

Genauer Vergleich

In diesem Kapitel werden die vorher ausgewählten Suchmaschinen genauer verglichen. Dafür werden alle vier Suchmaschinen aufgesetzt, um einen Ersteindruck zu erstellen. Da ich dieses Projekt nicht nach meiner Bachelor-Arbeit weiter betreuen kann, ist es auch wichtig zu schauen, wie leicht es für einen neuen Administrator ist, sich in dieses System einzuarbeiten. Deshalb wird ein besonderes Augenmerk auf die Dokumentation und Oberfläche, insofern vorhanden, gelegt. Die genaueren Kriterien werden nun im Folgenden mit Erklärungen aufgeführt.

4.1 Testsystem

Das Testsystem besitzt die folgenden Spezifikationen:

- CPU: 4 Kerne
- RAM: 16 Gigabyte
- Festplattenspeicher: 20 GB
- Betriebssystem: Ubuntu 18.04.03 LTS

Auf das System wird zudem die MariaDB Datenbank des DietrichOnline-Projektes als Datenquelle eingespielt. Zudem mussten einige Programme während der Vorbereitung des Servers durch die Administratoren aufgespielt werden. Darunter fallen Programme wie VIM oder Git. Eine genaue Liste findet sich im Anhang. Diese Programme werden als gegeben vorausgesetzt.

4.2 Aufbau der Tests

4.2.1 Installation

Im ersten Schritt wird die Installation bewertet. Dabei wird geschaut, wie einfach es ist die Software zu installieren. Existiert zum Beispiel ein Installations-Wizard? Wie viel muss manuell in den Dateien geändert werden?

4.2.2 Indexierung

Hierbei wird geschaut, wie einfach die Indexierung von den Daten aus der Datenbank ausfällt. Dabei wird auch geschaut, ob es möglich ist, Daten direkt von der Oberfläche zu indexieren und ob es möglich ist die Indexierung in einen Zeitplan zu legen.

4.2.3 Oberfläche

In diesem Schritt wird geschaut, wie übersichtlich und funktional die Oberfläche gebaut ist. Dabei geht es darum, wie viele Funktionen über die Oberfläche zu administrieren sind und ob es Komfort-Funktionen wie Responsiveness[1] gibt.

4.2.4 Dokumentation

Im dritten Schritt wird die Dokumentation analysiert. Hierbei wird Augenmerk auf die Übersichtlichkeit und Verständlichkeit gelegt. Da in diesem Kurztest nicht alle Bereiche der Dokumentation genau durchgelesen und daraufhin auch Testweise implementiert werden können, wird sich dabei auf die Schritte dieses Ersteindrucks bezogen.

4.2.5 Absetzen einer Anfrage und Integration in PHP

Im letzten Schritt wird eine Abfrage an das System von einem PHP-Skript abgesetzt. Dabei wird die Zeit gemessen, wie lange die Abfrage braucht um die Daten zu liefern.

Die dabei verwendete Abfrage ist die bisher am langsamsten laufenden Abfrage des DietrichOnline-Projektes. Sie ermittelt alle Lemmata vom Buchstaben S und baut alle Daten, die zur Anzeige benötigt werden zusammen 4.2. Die Tabellen, welche für diese Ansicht gebraucht werden, sind in diesem Diagramm 4.1 zu finden. Um genau zu sein, sind es zwei Abfragen. Die erste findet alle IDs der Lemmata und der zweite baut auf dieser Liste die Daten zusammen. Dabei werden für diesen Ersteindruck M-zu-N Beziehungen aus Zeitgründen vernachlässigt, es sei denn diese Funktionalität wird direkt mitgeliefert.

```
1 SELECT lemma.id
2 FROM lemma
3 WHERE lemma.bezeichnung LIKE 'S%' AND lemma.ist_geloescht = 0
4 ORDER BY lemma.bezeichnung ASC, lemma.id ASC;
```

Im zweiten Schritt werden dann die gerade geholten IDs der Einträge mithilfe von Joins für die Darstellung vorbereitet.

```
1 SELECT  lemma.id, [...] #Lemma, GND und DCC-columns
2 FROM lemma lemma
3 INNER JOIN lemmabearbeitungsstatus lemmaBStatus
4 ON lemma.fk_lemmabearbeitungsstatus = lemmaBStatus.id
5 LEFT JOIN lemma_gnd lemma_gnd_map ON lemma.id = lemma_gnd_map.fk_lemma
6 LEFT JOIN gnd gnd ON lemma_gnd_map.fk_gnd = gnd.id
```

[1] Eine Webseite ist responsive, wenn sie für alle Endgeräte richtig skaliert und gut zu benutzen ist.

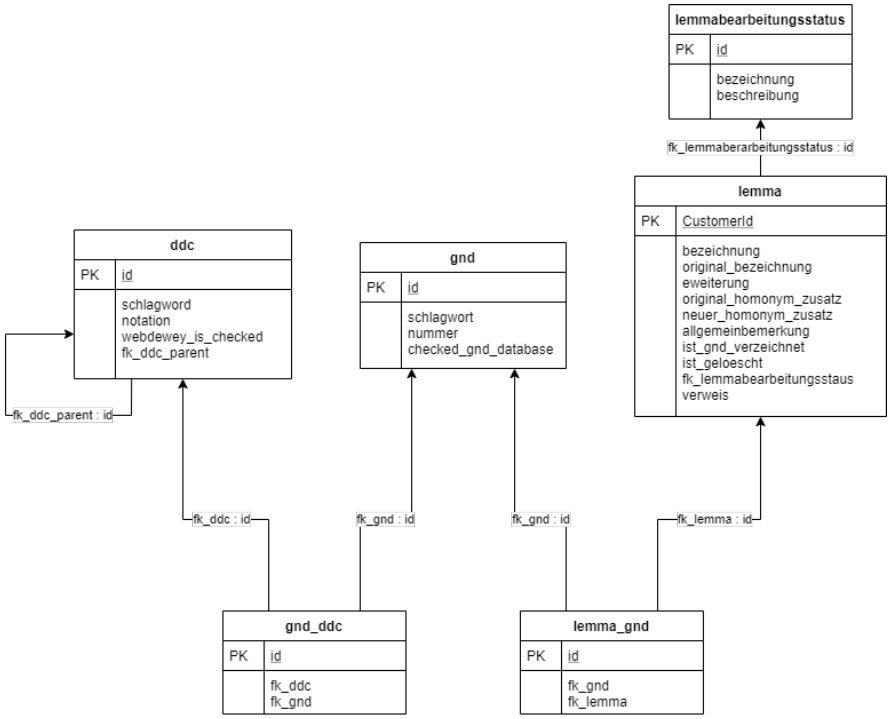

Abb. 4.1. Tabellenaufbau der Lemma-Administration Übersicht.

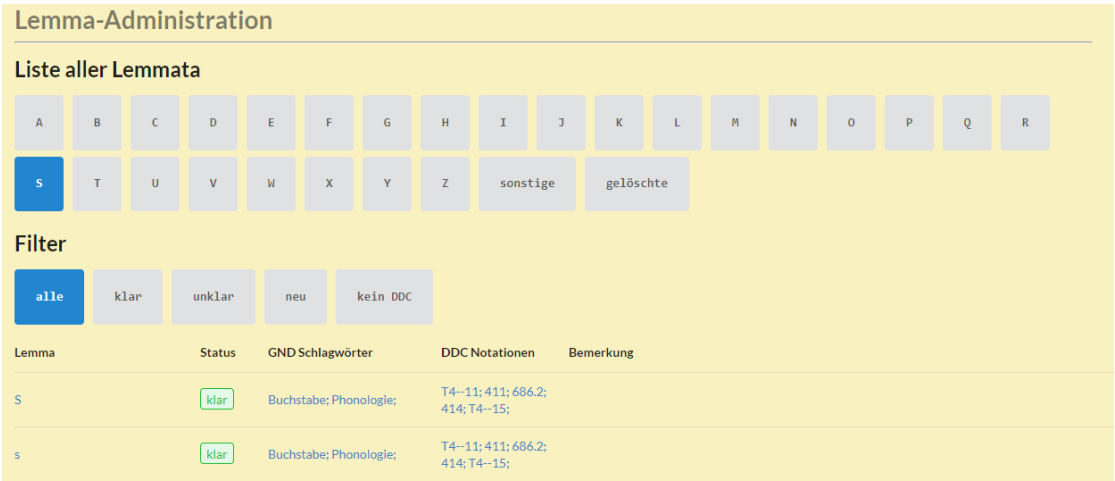

Abb. 4.2. Frontend Ansicht der Lemma-Administration mit geladenen Buchstaben S (Ausschnitt).

```
 7  LEFT JOIN gnd_ddc gnd_ddc_map ON gnd.id = gnd_ddc_map.fk_gnd
 8  LEFT JOIN ddc ddc ON gnd_ddc_map.fk_ddc = ddc.id
 9  WHERE lemma.id IN ([Array of Lemma IDs])
10  ORDER BY lemma.bezeichnung ASC, lemma.id ASC;
```

4.3 Solr

Begonnen wird der genaue Vergleich mit Solr von Apache.

4.3.1 Installation

Als Systemvoraussetzungen ist eine Java Version > 8 gegeben. Es wurde sich hierbei für OpenJDK 11 entschieden. Um Solr im Entwicklermodus auszuführen, kann das entpackte Programm einfach gestartet werden. Nach dem ersten Starten wurden 2 Warnungen gemeldet, dass die User-Limits für Solr zu gering sind 4.3.1. Nachdem diese entsprechend erhöht wurden, verschwanden die Warnungen.

```
1  *** [WARN] *** Your open file limit is currently 1024.
2  It should be set to 65000 to avoid operational disruption.
3
4  *** [WARN] ***  Your Max Processes Limit is currently 63918.
5  It should be set to 65000 to avoid operational disruption.
```

Bei der richtigen Installation installiert sich Solr als Service und legt einen eignen Nutzer an. Ein entsprechendes Installations-Skript findet sich dafür im entpackten Solr-Ordner. Sobald dieses mit Root-Rechten aufgerufen wird installiert sich Solr automatisch in das Opt-Verzeichnis.

4.3.2 Indexierung

Um mit der Indexierung starten zu können, muss zuerst ein sogenannter Core erstellt werden. Dieser ist ein Index mit dazugehörigen Transaktions-Log und den Konfigurationsdateien. Nur mit diesen ist es möglich Dateien zu indexieren und auf ihnen zu suchen. Nach der Erstellung lässt sich der Core über die Oberfläche einsehen und zum Teil konfigurieren.

Damit Solr nun die Daten von der Datenbank liest, muss ein DataImportHandler (DIH) 4.3.2 geschrieben werden. In diesen werden die Daten, welche indexiert werden sollen, mit MySQL-Querys eingelesen. Das System setzt dabei auf eine XML-Struktur mit sogenannten Entitys. Diese besitzen jeweils mehrere Attribute, wie den Namen, welcher auf der Oberfläche zur Indexierung angezeigt wird, den MySQL-Abfrage mit dem die Daten gelesen werden und einen Delta-Query, welcher dazu dient, nur die Einträge zu laden, welche Änderungen seit dem letzten Import erlebt haben.

Der Delta-Query benötigt hierbei eine eigene Zeitstempel-Spalte in der Datenbank, welche angezeigt, wann die Spalte das letzte Mal editiert wurde. Da die Tabellen im Projekt aktuell keine solche Spalte besitzen, kann die Funktion nicht getestet werden.

Innerhalb des Entity-Elements gibt es entweder weitere Entitys, dazu gleich mehr, oder Field-Elemente. Diese besitzen ein Attribut, welches die Spalte der Tabelle ausweist und einen Namen, der das zugehörige Solr-Schema-Element ausweist.

Entitys können unbegrenzt ineinander verschachtelt werden. Damit Änderungen an einer verschachtelten Entity nach oben richtig weitergegeben werden, gibt es Parent-Delta-Querys. Diese geben die betroffenen Werte an die übergeordnete Entity weiter. Dafür führt der Parent-Delta-Query einen Aufruf an die überliegende Entity-Tabelle aus, in der er mithilfe der Fremdschlüssel-IDs in den betroffen Zeilen herausfindet.

Der DataImportHandler muss, bevor er benutzt werden kann, jedoch noch mit dem Core verbunden werden. Dafür wird dieser, zusammen mit einem JDBC-Treiber in die solrconfig.xml eingetragen. Als JDBC-Treiber wurde in diesem Beispiel der Treiber von MariaDB verwendet.

```
<entity name="lemma"
  query="select * from lemma"
  deltaQuery="select eid from lemma
    where last_modified > '${dataimporter.last_index_time}'">
<field column="bezeichnung" name="bezeichnung" />
[...]
<entity name="lemma_gnd"
  query="select * from lemma_gnd where fk_lemma='${lemma.id}'"
  deltaQuery="select * from lemma_gnd
    where last_modified > '${dataimporter.last_index_time}'"
  parentDeltaQuery="select * from lemma
    where id=${lemma_gnd.fk_lemma}">

    <entity name="gnd"
      query="select * from gnd where id = '${lemma_gnd.fk_gnd}'"
      deltaQuery="select * from gnd
        where last_modified > '${dataimporter.last_index_time}'"
      parentDeltaQuery="select * from lemma_gnd where fk_gnd=${gnd.id}">
      <field column="nummer" name="gnd_nummer" />
      <field column="schlagwort" name="gnd_schlagword" />
      [...]
    </entity>
  </entity>
</entity>
```

Wie schon eben angesprochen, muss das Solr-Schema für die entsprechende Elemente auch angepasst werden. Dieses Schema dient dazu die Dateitypen für eine möglichst gute Indexierung auszuweisen. Dafür wird zuerst der Dateityp für die Tabellen-Spalte angegeben. Hierbei werden bei den Grundtypen, zum Beispiel unter anderem String und Text_de gelistet. Dabei wurde angenommen, dass die beiden nur bei Abfragen von unterschiedliche Sprachen einen Unterschied besitzen. Dies ist allerdings falsch. Als eine Abfrage gestellt wurde, die alle Lemmata mit den Buchstaben S finden sollten, kamen mehr Ergebnisse als erwartet zurück. Dies liegt daran, dass Text_de, das Feld aus Volltext ausweist. Bei Volltexten wird jedes Wort einzeln betrachtet und so kamen Lemma, in welchen eines der Wörter mit S begann in die Auflistung. Deswegen wurde daraufhin das Feld als String deklariert, was es ermöglicht hat nur Ergebnisse herauszufiltern, welche mit S beginnen.

Es gibt mehrere Möglichkeiten diese Einträge auszuweisen. In diesem Ersteindruck wurden die Einträge über die Administrations-Oberfläche angelegt. Es ist allerdings auch möglich eine eigene Schema-Datei zu erstellen. Diese Methode soll allerdings nicht mehr verwendet werden, da es die Möglichkeit gibt, die Einträge per API zu generieren. Dadurch wird direkt überprüft, ob die Einträge formal stimmen. So können keine fehlerhaften Schemata gebaut werden. Die Einträge, welcher über die API oder die Administrations-Oberfläche gestellt werden, werden in einer Datei namens managed_schema 4.3.2 im XML-Format angelegt.

```
[...]
<field name="ddc_webdewey_is_checked" type="boolean"
    uninvertible="false" indexed="true" stored="true"/>
<field name="description" type="text_de" uninvertible="false"
    multiValued="true" indexed="true" stored="true"/>
<field name="erweiterung" type="text_de"
```

```
7       uninvertible="false" indexed="true" stored="true"/>
8   [...]
```

Die Indexierung lief eine Minute und 34 Sekunden für rund 14 Tausend Einträge 4.3. Dabei wurde der gegebene Arbeitsspeicher nicht komplett ausgenutzt, was darauf schließen lässt, dass die Datenbank der limitierende Faktor war. Die hohe Anzahl der Abfragen ist darauf zurückzuführen, dass Solr keine Joins verwendet, sondern bei jeder verschachtelten Entity die gesamten Tabellen wieder und wieder nach passenden Einträgen durchsucht.

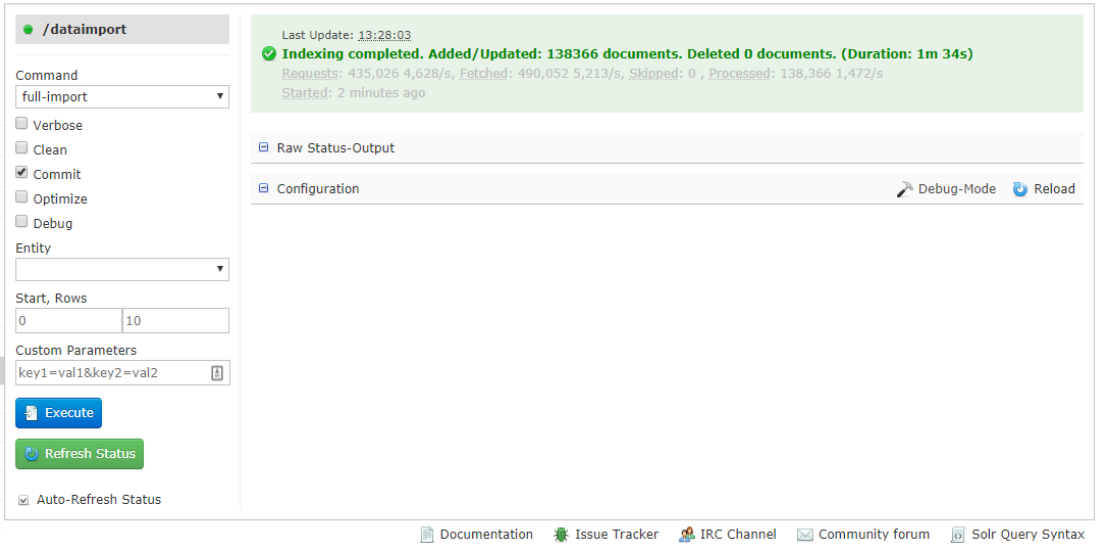

Abb. 4.3. Oberfläche der Indexierung mit Laufzeit.

4.3.3 Oberfläche

Die Startseite des Solr-Systems bietet direkten Einblick in auf die Auslastung des Systems 4.4. Der Fehler-Log ist auch sehr einfach mit einem Klick zu erreichen. Um an die Statistiken des aktuellen Cores zu kommen, kann dieser aus einen Drop-Down-Menu ausgewählt werden. Positiv anzumerken ist, dass es möglich ist Schema Einträge direkt in der Weboberfläche zu löschen und anzulegen. Es ist jedoch nicht möglich, den DataImportHandler direkt zu verändern, ohne weitere Einstellungen im System vorzunehmen. Es gibt eine Möglichkeit Abfragen direkt über die Web-Oberfläche zu senden, was das Testen der Abfragen erleichtert. Auch bei der Indexierung kann ein Debug-Modus dazu geschaltet werden 4.3. Zudem besteht die Möglichkeit die Konfigurationsdateien des Cores auf der Weboberfläche einzusehen. Die Dateien dort direkt zu editieren, ist jedoch nicht möglich. Es gibt keine Möglichkeit Updates direkt über die Weboberfläche einzuspielen. Auch ist diese Seite nicht Responsive geschrieben.

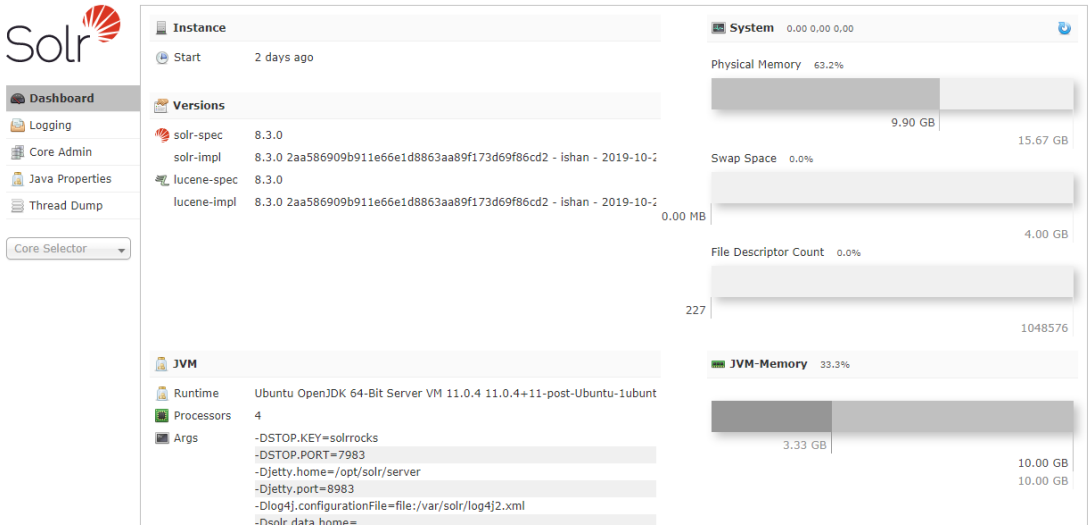

Abb. 4.4. Startseite der Weboberfläche von Solr.

4.3.4 Dokumentation

Die Dokumentation war bei diesem kurzen Test meine Hauptquelle. Die Installation ist dort genau beschrieben. Positiv aufgefallen ist dabei die genaue Beschreibung der Systemvoraussetzungen. Das Team hat mehrere Java-Versionen getestet und alle dort aufgeführt. Generell gibt es für alle Themen eine kleine Übersichtsseite, welche die grundlegenden Funktionen erklärt, ohne sich dabei in Details zu verlieren. Die Seite für den DataImportHandler hat anhand eines Beispiels gut die Struktur erklärt. Allerdings wäre ein Verweis, dass für die DataImportHandler-Attribute noch extra ein Solr-Schema-Attribut benötigt wird, schön gewesen. Die Dokumentation ist gut bebildert und bietet einen guten Einstiegspunkt in das System.

4.3.5 Absetzen einer Anfrage und Integration in PHP

Um nicht direkt mit der JSON-API arbeiten zu müssen, gibt es diverse Bibliotheken, die ein wenig der Arbeit abnehmen. Eine der größten ist hierbei Solarium, welches sich mit Composer[2] installieren lässt. Da die Composer Technologie schon im Projekt verwendet wird, ist dies vom Vorteil. Die Abfrage ist herbei sehr einfach, da die Daten beim Import schon dementsprechend indexiert wurden.

```
1   [...] # Imports and variable declarations
2
3   $config = array(
4       'endpoint' => array(
5           'localhost' => array(
6               'host' => '136.199.34.55',
7               'port' => 8983,
8               'core' => 'dietrich'
9   )));
10  $queryText = 'original_bezeichnung:S*';
```

[2] Abhängigkeiten-Manger für PHP

```
11    $solr = new Client($config);
12    $query = $solr->createSelect();
13    $query->setQuery($queryText);
14    $query->setRows(2147483647);
15    [...] # Loop with Timer
16    $resultSet = $solr->select($query);
17    $count = $resultSet->count();
18    [...] # Output Runtime
```

Die maximale Anzahl der Zeilen die von Solr geladen werden, sind standardmäßig auf 10 limitiert. Erst mit setRows kann die Anzahl erhöht werden. Für diesen Test wurde der maximale Integer-Wert gewählt, um immer alle Ergebnisse zu bekommen. Damit nun ein guter Median-Wert gebildet werden kann, wurde die Abfrage 100 mal laufen gelassen. Dabei lief die Abfrage durchschnittlich 1.01 Sekunden um die 15838 Ergebnisse herauszusuchen.

4.4 Datafari

Die zweite Suchmaschine, welche verglichen wird ist Datafari von France Labs.

4.4.1 Installation

Für Datafari musste die folgende Software nachinstalliert werden: Java 8 und JQ, ein JSON-Prozessor. Damit die Installation korrekt funktioniert, muss die JAVA_HOME-Variable erstellt werden. Insofern Datafari nicht unter Root laufen soll, muss noch ein besonderer Nutzer mit Root Rechten angelegt werden. Dieser muss, wie schon bei Solr, höhere User-Limits erhalten. Datafari installiert sich selbständig durch eine DEB-Datei. Während der Installation erscheint ein kurzer Setup-Dialog, mit welchem durch die Konfiguration geführt wird. Das Starten des Server geschieht daraufhin durch ein Script im Installationsordner.

4.4.2 Indexierung

Damit eine Indexierung durchgeführt werden kann, muss bei Datafari ein sogenanntes Repository angelegt werden. In diesem wird die Datenbankverbindung eingetragen. Dabei ist es wichtig, dass vorher der Treiber korrekt installiert wird. Dabei kam es zu Problemen. Das auf Apache ManifoldCF basierende System akzeptiert nur MySQL-JDBC Treiber. Da der MariaDB-Treiber einen anderen Klassennamen in Java verwendet, funktioniert dieser nicht.

> This connection type cannot be configured to work with other databases than the ones listed above without software changes. [22, S. 61]

Deswegen wurde für diesen Test den MySQL-Treiber von Oracle verwendet. Nachdem der Treiber korrekt installiert wurde und das Repository erstellt war, kann nun einen Job zur Indexierung der Einträge gestartet werden. In diesem werden die Abfragen und der Zeitplan konfiguriert. Im ersten Schritt wird das Repository und das Ziel ausgewählt. In dem Tab Querys lassen sich dann diverse Abfragen bauen. Der erste ist ein Seeding-Query, welcher eine Art Delta-Query für dieses

System ist. Als Nächstes wird der Data-Query benötigt, welcher die Daten aus der Datenbank lädt. Dabei werden mehrere Variablen definiert, damit der Query korrekt von ManifoldCF erkannt wird.

Zuerst einmal das Feld: IDCOLUMN, welches die ID enthält, dann URLCOLUMN, welches einen Hyperlink für diesen Eintrag enthält. Da hier keine solche Spalte gegeben ist, wird einfach nochmal die ID mitgegeben, was so in einen Screenshot aus der Dokumentation zu sehen ist. Zuletzt noch die DATACOLUMN, welche alle Daten konkateniert enthält. Um das System zu testen habe ich allerdings erstmal nur eine Zeile in die DATACOLUMN geschrieben 4.5. Die Konkatenation ist vorgegebene die Methode aus der ManifoldCF-Dokumentation. [22, S. 97] Dies ist für unseren Zweck keine saubere Datenstruktur. Sind alle Querys eingetragen, kann die Indexierung beginnen. Dafür wird der Job in der Oberfläche manuell gestartet, insofern kein Zeitplan konfiguriert ist.

In diesem Test kam es dabei allerdings zu Problemen. Die Indexierung erfolgte nicht korrekt und blieb immer am Ende hängen. Der Log zeigte ein „Ready for processing“ an, machte dort allerdings nicht weiter. Einen Eintrag in der Dokumentation oder generell im Internet konnte nichts gefunden werden. Auch eine Reduktion der Einträge auf nur 125 hat das Problem leider nicht lösen können. Deswegen wurde der Test an dieser Stelle abgebrochen.

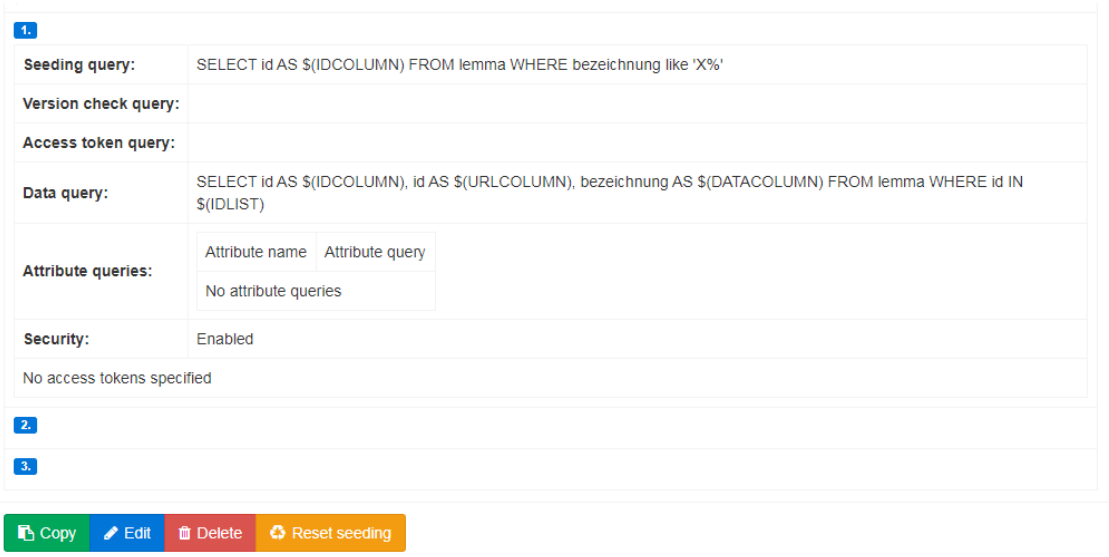

Abb. 4.5. Übersichtsseite des Querys in Datafari.

4.4.3 Oberfläche

Die Oberfläche von Datafari ist dreigeteilt. Zum einen gibt eine Such-Oberfläche, welche sich ohne Anmeldung erreichen lässt.

Als Zweites findet sich eine Administrationsoberfläche, welche erst eingesehen werden kann, sobald man eingeloggt ist. Dort findet finden sich diverse Einstellun-

gen für die Suchmaschine, wie Synonyme oder die Facetten-Konfiguration. Auch sind dort die Logs einzusehen, welche durch über Kibana 3.5 angezeigt werden.

Die dritte Oberfläche ist die Einstellungsseite für die Datacrawler. Dies ist eine modifizierte Oberfläche von Apache ManifoldCF. Generell sind die Menüs sehr übersichtlich, auch wenn die Einbindung von anderen Anwendungen keine Ideale Lösung darstellt. Es lassen sich keine Updates direkt über die Oberfläche einspielen. Die Such-Seite und die Seite für die Erstellung der Datacrawler sind Responsive, während die Administrationsoberfläche bei kleineren Bildschirmgrößen das Menü versteckt und die Seite somit unnutzbar macht.

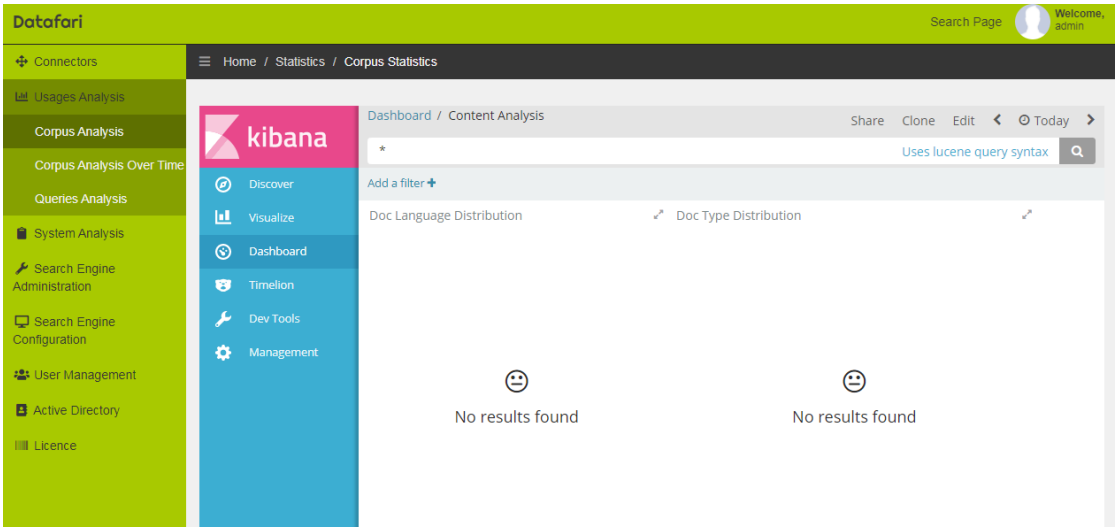

Abb. 4.6. Kibana Integration in Datafari.

4.4.4 Dokumentation

Die Dokumentation geht sehr genau auf die Installation des Systems ein. Dabei werden alle Konfigurationsaspekte beleuchtet. Es wird zum Beispiel beschrieben, wie die User Limits erhöht werden können, oder wie die JAVA_HOME-Variable korrekt gesetzt wird. Allerdings ist an manchen Stellen zu merken, dass die Dokumentation nicht von nativ Englischsprechenden geschrieben wurde, da die Grammatik nicht immer korrekt ist. Allerdings hat dies nicht zu Problemen oder Verwechslungen geführt.

Bei der Dokumentation zum Einrichten des JDBC-Treibers finden sich einige Probleme 4.7. Zum einen sind beide Pfade, die in dem Text angegeben sind, falsch. Einer davon wird sogar richtig in dem Screenshot direkt darunter angezeigt. Und zum anderen ist der zweite Screenshot so niedrig aufgelöst, dass sich kaum etwas erkennen lässt. Dies passiert auch, wenn das Bild in einen neuen Tab geladen wird. Generell ist die Dokumentation für den Umgang mit Datenbanken nicht sehr ausführlich. Die Erklärungen, wofür die Variablen bei der Erstellung eines Jobs stehen, mussten in der Dokumentation von ManifoldCF nachgelesen werden.

Die Dokumentation ist im aktuellen Stand nicht sauber strukturiert. Sie gibt das Gefühl, dass es sich eher um eine Sammlung verschiedener Artikel, welche Intern genutzt wurden, handelt.

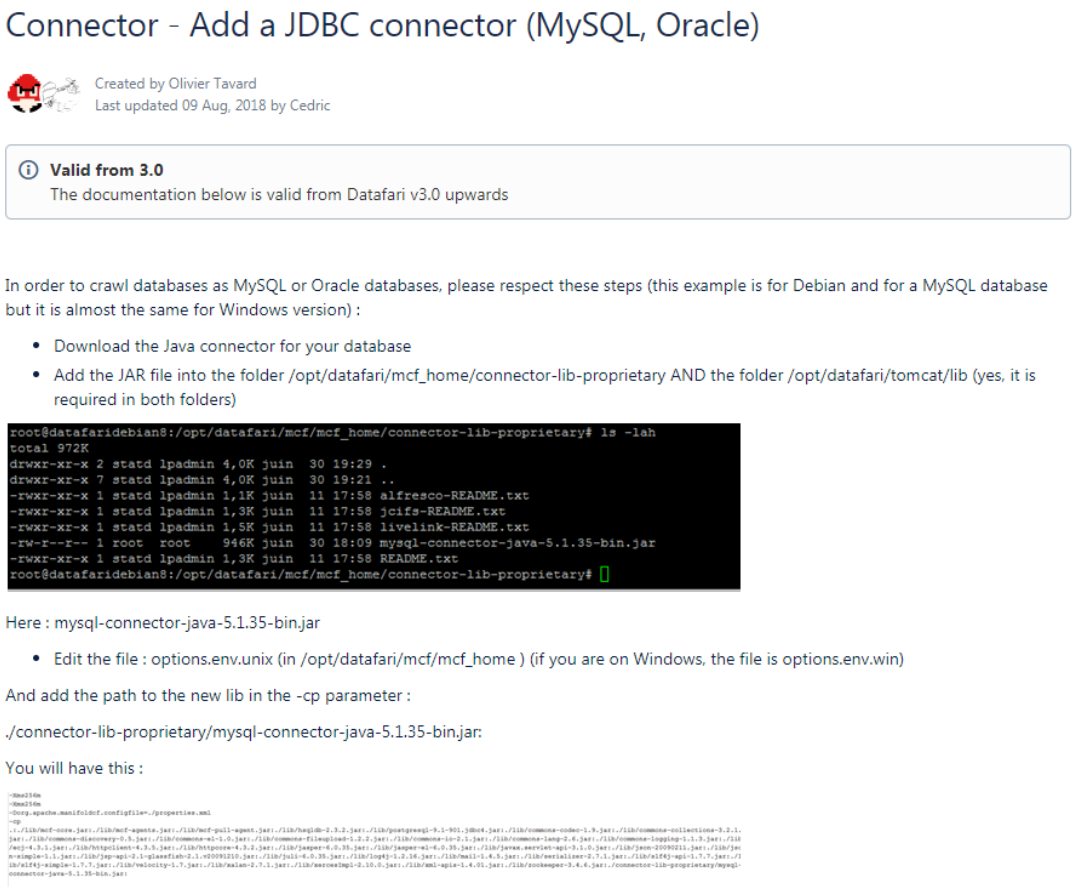

Abb. 4.7. Dokumentationsseite für den JDBC Treibers von Datafari.

4.4.5 Absetzen einer Anfrage und Integration in PHP

Durch das fehlgeschlagene Einlesen der Daten konnte dieser Test nicht durchgeführt werden.

4.5 Elasticsearch

4.5.1 Installation

Die Installation ist bei Elasticsearch dreigeteilt. Um die Suchmaschine in dem Umfang nutzen zu können, wie es hier benötigt wird, muss der komplette ELK-Stack installiert werden. Elasticsearch ist hierbei das Kernstück und dient als Datenbank. Kibana ist eine grafische Benutzeroberfläche und Logstash bildet die Brücke

zwischen der Datenquelle und Elasticsearch. Während Elasticsearch Java mitgeliefert hat, muss für Logstash Java Version 8 oder 11 nachinstalliert werden. Um die drei Dienste für den Development Modus zu installieren, müssen nur die Archive entpackt und die entsprechenden Anwendungen gestartet werden. Ohne die Konfigurationsdateien zu ändern, haben die Anwendungen direkt miteinander kommunizieren können. Allerdings hat Logstash ein paar Warnungen beim Start geworfen, welche mit JRuby zusammenhingen und den Entwicklern schon bekannt sind. Daher können diese hier ignoriert werden.

Für eine richtige Installation gibt es mehrere Wege. Für den Test wurde das Debian-Packet verwendet. Alternativ ist es allerdings auch möglich, entweder das Repository von Elasticsearch einzupflegen oder einen Docker-Container zu nutzen. Die Installation verlief hierbei ohne Probleme.

4.5.2 Indexierung

Um nun Daten zu indexieren, muss in einer Konfigurationsdatei in Logstash definiert werden, wie und welche Daten gelesen und an das Elasticsearch System weitergegeben werden sollen 4.5.2. Dabei kann Logstash direkt MySQL-Abfragen gegen die Datenbank stellen. Die Datei ist in zwei Blöcke unterteilt. Zum einen ein Input-Block, welcher erklärt, welche Daten eingelesen werden sollen und ein Output-Block, welcher das Ziel für die Daten angibt. Für den Input-Block wird der JDBC Treiber von MariaDB verwendet.

Bei diesem Schritt kam es bei dem System allerdings zu Problemen. Der Treiber konnte über den in der Dokumentation angegeben Weg nicht geladen werden. Damit der Treiber korrekt erkannt werden konnte, musste er zusammen mit den Core-Bibliotheken von Logstash geladen werden. Deswegen ist die Zeile mit dem Pfad zur Bibliothek auch leer.

Nachdem die Datenbank Konfiguration und Abfrage angegeben wurden, kann zudem noch ein Zeitplan definiert werden. Außerdem ist es auch möglich, eine Art Delta-Query zu definieren. Hierfür wird eine Tracking-Spalte festgelegt, welche dann in der Abfrage auf einen Zeitstempel überprüft wird.

Im zweiten Teil der Datei wird das Ziel definiert. Die erste Zeile dient dazu nur dem Debugging, da es alle ausgegeben Linien des Skripts auch auf der Shell ausgibt.

In dem Elasticsearch-Segment wird zum einen eine ID definiert, welche verhindert, dass Einträge doppelt in die Datenbank gespielt werden. Deswegen wird hier die ID der Lemmata genommen, da diese auch in der Datenbank nicht wiederholt werden darf. Zum Anderen wird noch ein Index angegeben, in welchen die Daten geschrieben werden sollen.

Als die Indexierung nun gestartet wurde, kam es allerdings zu einem Fehler. Die MySQL-Abfrage sei nicht gültig. Dies lag daran, dass um zu ermitteln, wie viele Daten indiziert werden müssen, die Abfrage mit einer Count-Abfrage umhüllt wird. Dabei verwendete Logstash Double anstelle von Single-Quotes, was bei MariaDB zu einem Fehler führte. Dies konnte behoben werden, indem eine Einstellung in

der Datenbank vorgenommen wurde, um auch Double-Quotes zu erlauben. Danach verlief die Indexierung ohne weitere Probleme.

```
input {
  jdbc {
    jdbc_validate_connection => true
    jdbc_driver_library => ""
    jdbc_driver_class => "Java::org.mariadb.jdbc.Driver"
    jdbc_connection_string =>
        "jdbc:mariadb://localhost:3306/dietrichonline"
    jdbc_user => "USER"
    jdbc_password => "PW"
    tracking_column => "timestamp"
    use_column_value=>true
    statement => "MYSQL-Query WHERE timestamp > :sql_last_value"
    schedule => "0 */6 * * *"
  }
}

output {
  stdout { codec => json_lines }
  elasticsearch {
    document_id => "%{id}"
    index => "lemma"
    hosts => "localhost:9200"
  }
}
```

Nun ist die Frage jedoch, wie weiß Elasticsearch, was für ein Datentyp das Feld besitzt. Dafür verwendet Elasticsearch ein sogenanntes Dynamic-Mapping, indem es versucht den am besten passenden Datentyp für das Feld zu ermitteln.

Sollen nun bestimmte Feld-Typen verwendet werden, muss der Index von Hand erstellt werden. Um ein Feld zu definieren, muss ein Mapping manuell erstellt werden 4.5.2. Dieses enthält zumindest den Feld-Namen und den Typen. Zudem können noch andere Optionen angegeben werden, um die Indexierung nach dem eigenen Ermessen anzupassen. Es kann zum Beispiel deklariert werden, dass ein Feld zwar existiert, allerdings nicht Suchbar ist. Sowas wäre für interne IDs zur Verwaltung interessant, welche nicht nach außen hin herausgegeben werden sollten. Wichtig ist, dass nun nur alle definierten Felder in die Datenbank geschrieben werden. Felder, die aus der Datenbank geladen werden, allerdings nicht im Index vorhanden sind, werden ignoriert.

```
PUT lemma
{
  "mappings": {
    "properties": {
      "original_bezeichnung": {
        "type":  "keyword"
}}}}
```

Sollen nicht alle Felder von Hand erstellt werden, ist es zudem möglich eine Vorlage für ein dynamisches Feld zu generieren 4.5.2. So kann für ein Feld eine bestimmte Regel gesetzt werden, ohne alle Felder manuell anlegen zu müssen. Als Beispiel habe wurde hier definiert, dass das Feld original_bezeichnung immer nur als Keyword gespeichert wird.

```
PUT lemma
{
  "mappings": {
```

```
4       "dynamic_templates": [
5         {
6           "obez_as_keyword": {
7             "match":    "original_bezeichnung",
8             "mapping": {
9               "type": "keyword"
10  }}}]}}
```

Für den Test war die Erstellung eines Mappings oder eines dynamischen Templates allerdings nicht notwendig, da Elasticsearch bei der automatischen Indexierung jedes String-Feld als Keyword, sowie Volltext abspeichert, insofern die Länge unter 256 Zeichen ist. So konnte die Abfrage, welche alle Lemmata mit den Buchstaben S sucht einfach verwendet werden, indem das Keyword-Feld zur Suche verwendet wurde.

4.5.3 Oberfläche

Die Oberfläche von Kibana bietet eine zu Beginn überwältigende Erfahrung 4.8. Um dies für spätere Anwender zu verhindern, bietet Kibana die Möglichkeit Sichten für unterschiedliche Nutzer zu erstellen. Zudem kann die Oberfläche hinter ein Login-System geschaltet werden.

Auf der Oberfläche gibt es viele Menüpunkte, welche es ermöglichen die Daten auf diverse Arten darzustellen, darunter in Grafen-Form oder auf einer Landkarte. Unter dem Punkt Management finden sich die Einstellungen für das Elasticsearch System. Hier kann man nicht nur Snapshots erstellen, sondern auch das System mit Updates versorgen. Zudem können hier die Indices verwaltet werden 4.9. Eine Erstellung der Indices ist allerdings nur per API möglich. Es existiert allerdings die Möglichkeit einige Einstellungen an den Indices vorzunehmen und diese auch zu löschen. Zudem gibt es die Möglichkeit Indices eine Lebensdauer zuzuweisen, was in Zeiten der Datenschutzgrundverordnung sicherlich eine nützliche Funktion darstellen wird.

Die Oberfläche ist vollkommen responsive. Neben diesem Menü gibt es ebenfalls eine Entwicklerkonsole, in der es möglich ist Anfragen an das Elasticsearch-System zu schicken, ohne mit Curl oder ähnlichen zu arbeiten, was das Debugging vereinfacht.

4.5.4 Dokumentation

Die Dokumentation von Elasticsearch ist sehr ausführlich und gut zu lesen. Um einen einfacheren Einstieg in das System zu bieten, beginnt die Dokumentation bei jedem Thema mit einem kleinen Beispiel, um das Konzept zu verdeutlichen. Diese Struktur zieht sich durch die gesamte Dokumentation, jedes Thema ist mit vielen Codeschnipseln bebildert, was eine einfachere Einarbeitung in das System ermöglicht. Gut gelöst dabei ist, dass es möglich ist mit einem Klick den Befehl direkt in die Konsole von Kibana zu importieren. Während meines Testes wurde nur ein Fehler in der Dokumentation entdeckt: Bei der Informationsseite zum PHP-Klienten eine falsche PHP-Version vermerkt.

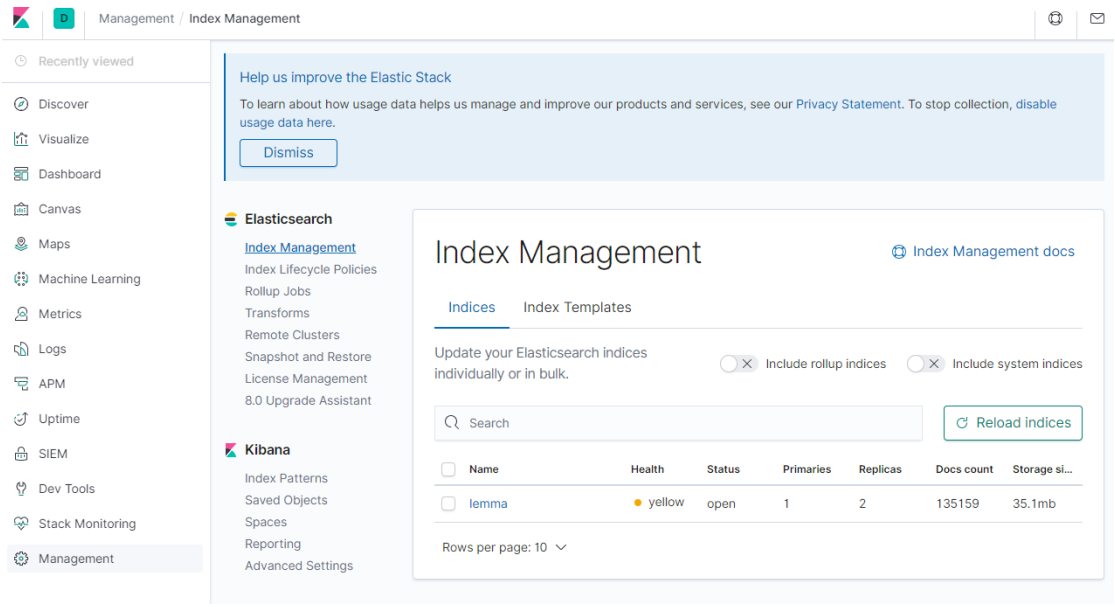

Abb. 4.8. Index Management Seite von Elasticsearch

4.5.5 Absetzen einer Anfrage und Integration in PHP

Elasticsearch bietet für PHP einen eigenen Klienten an. Es ist möglich, diesen unter anderem auch mit Composer zu installieren. Um die indexierten Dateien abzufragen, muss ein ClientBuilder gebaut werden, welcher einen oder mehrere Hosts mitgegeben bekommt. Der Server sendet, insofern nicht anders konfiguriert, 10 Resultate zurück. Um diese Limitierung aufzuheben, muss hierbei an 2 Stellen etwas verändert werden. In PHP muss dem Klienten bei der Anfrage ein Parameter mitgegeben werden, welcher die Menge der Ergebnisse bestimmt. Dies funktioniert allerdings nur bis zu 10.000 Ergebnissen. Sollten mehr Ergebnisse erwünscht sein, muss auch noch etwas am Index geändert werden. Dies kann entweder über eine HTTP Anfrage oder über die Oberfläche geändert werden. Für den Test wurde dieses Limit nun erhöht 4.9.

```
1  <?php
2  [...] # Imports and variable declarations
3
4  $clientBuilder = ClientBuilder::create()->setHosts(['136.199.34.55']);
5  $client        = $clientBuilder->build();
6
7  $params = [
8      'index' => 'lemma',
9      'body' => [
10         'size' => 1000000,
11         'query' => [
12             "wildcard" => ["bezeichnung.keyword" => "S*"],
13 ]]];
14
15 $results = $client->search($params);
16 [...] # Loop with Timer
17 $results = $client->search($params);
18
19 $count=0;
20   foreach ($results['hits']['hits'] as $hit){
```

```
21          $count++;
22      }
23  [...] # Output Runtime
```

Zum Code ist noch zu sagen, dass im Ergebnis keine Summe der Ergebnisse liegt, sondern dafür eine eigener Abfrage vonnöten ist. Deswegen werden hier die Ergebnisse in einer Schleife gezählt.

Auch hier wurde nun die Abfrage 100-Mal ausgeführt, um einen Median Wert zu ermitteln. Dieser lag bei Elasticsearch bei 0.58 Sekunden pro Abfrage.

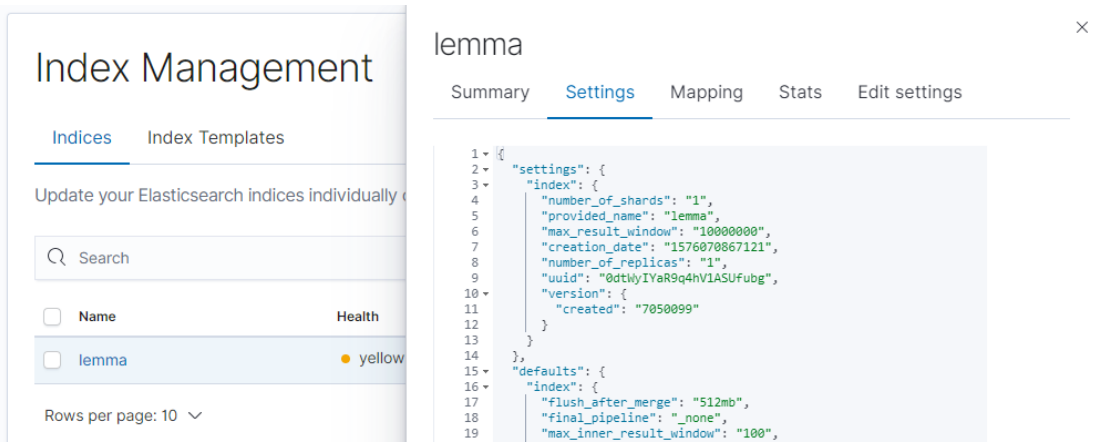

Abb. 4.9. Einstellungen vom Lemma-Index bei Elasticsearch

4.6 Xapian

4.6.1 Installation

Der empfohlene Installationsweg für Xapian führt über die Paketquelle (PPA) der Entwickler. Nachdem dieses eingefügt wurde, kann Xapian entweder in einer C++ oder Python Variante installiert werden.

Um die Suchmaschine auch über PHP anzusprechen ist, ist es allerdings notwendig den PHP-Connector aus Lizenzgründen selbst zu bauen. Dabei muss vorher ein Eintrag, der ausweist, dass aus dieser Quelle auch Source-Code geladen werden kann, in der Paketquellen hinzugefügt werden. Danach kann der Klient mithilfe von Make gebaut werden.

Allerdings ist der Server bisher nur Lokal ansprechbar, um dies zu ändern, muss ein TCP-Server für Xapian gestartet werden. Um diesen zu Nutzen ist es vonnöten ein weiteres Paket aus der Paketquelle zu installieren. Damit nun der Server gestartet werden kann, muss zuerst ein Index, der bei Xapian Database genannt wird, gebaut werden. Dazu mehr in dem Teil 4.6.2. Danach kann der Server auf einen beliebigen Port gestartet werden.

4.6.2 Indexierung

Durch die fehlende Dokumentation zur Indexierung von MySQL-Datenbanken, wurde erstmal ein Beispiel zum Import einer CSV-Datei durchgearbeitet. Darin war dann zu sehen, dass der komplette Datenimport manuell geschrieben werden musste. Auf dieser Basis wurde dann ein eigener Importer für MySQL geschrieben 4.6.2.

```
1  <?php
2  require_once("xapian.php");
3
4  //Open MYSQL-Connection and Run Query. Save the Output in $result
5
6  // Create or open the database we're going to be writing to.
7  $db = new XapianWritableDatabase($xapianDb, Xapian::DB_CREATE_OR_OPEN);
8  // Set up a TermGenerator that we'll use in indexing.
9  $termgenerator = new XapianTermGenerator();
10 $termgenerator->set_stemmer(new XapianStem('de')); //Setup Stemmer
11
12 while ($row = $result->fetch_assoc()) { //Loop through MySQL-Rows
13
14     $identifier = $row['id'];
15     unset($row['id']);
16     // Create new Row for the starting Letter
17     $searchIndexLetter = $row['original_bezeichnung'][0];
18
19     $doc = new XapianDocument(); // Create new Document
20     $termgenerator->set_document($doc); //Put it into the Term-Generator
21
22     // Index the field with a suitable prefix.
23     $termgenerator->index_text($searchIndexLetter, 1, 'K');
24     // Make it available for Search
25     $termgenerator->index_text($searchIndexLetter);
26
27     foreach ($row as $index) {
28         if ($index = '') { //Xapian cant Index Empty Fields
29             $index = 'EMPTY';
30         }
31         $termgenerator->increase_termpos(); // Make Space between Entries
32         $termgenerator->index_text($index); // Add Every Field
33     }
34     $doc->set_data(json_encode($row)); // Store all the fields
35
36     $idterm = "Q".$identifier; //Set ID to not have Duplicates
37     $doc->add_boolean_term($idterm);
38     $db->replace_document($idterm, $doc);
39 }
40 $conn->close();
```

In Zeile 10 wird ein Stemmer verwendet, welcher dazu dient, wenn zum Beispiel der Plural eines Wortes gesucht wird, auch den Singular zu finden.

In Zeile 23 wird ein Feld mit einem Präfix indexiert. Dies dient dazu diese Zeile für die spätere Suche auszuweisen. Die Präfixe werden dabei vor die Zeile geschrieben und bei der Suche wieder herausgefiltert.

Die anderen Felder wurden für die generelle Suche ohne Präfix indexiert. Zuletzt noch der Grund warum leere Strings gegen das Wort 'EMPTY' ausgetauscht werden. Dies beruht darauf, dass Xapian es nicht erlaubt, leere Strings zu indexieren.

Als nun das PHP-Script auf den Server gestartet wurde, musste noch eine Warnung behoben werden. Dazu wurde die php.ini angepasst, indem der Eintrag 'ena-

ble_dl' aktiviert wurde. Damit können jetzt Erweiterungen auch zur Laufzeit geladen werden, was die Bibliothek von Xapian benötigt.

Die Indexierung lief dabei äußerst schnell in unter einer Minute ab.

4.6.3 Oberfläche

Xapian besitzt keine Oberfläche zur Verwaltung. Allerdings kann sich ein Such-Frontend installiert werden, welches allerdings hier nicht geprüft wurde, da die Dokumentation noch nicht verfügbar war.

4.6.4 Dokumentation

Bei dem letzten Versionsupgrade wurde die Dokumentation von Xapian komplett umgeschrieben. Diese neue Dokumentation hat bisher noch viele Lücken und Todo-Boxen 4.10.

Zu der Installation von dem TCP-Server war auch nichts in der neuen Dokumentation zu finden. Beim Durchsuchen des Internets, wie der Server extern ansprechbar gemacht werden kann, konnte eine Seite der alten Dokumentation gefunden werden, welche den Befehl zum Starten vermerkt hatte. Nachdem dieser Befehl ausgeführt wurde, wurde gemeldet, dass für diesen Befehl ein weiteres Paket installiert werden musste. Dieses Paket wurde in der Dokumentation nicht vermerkt.

Generell bietet die Dokumentation, in der aktuellen Form, nur einen sehr grundlegenden Einblick in das System. Positiv anzumerken ist allerdings, dass Xapian ein Beispiel zu Indexierung von Daten mit Code in allen verfügbaren Programmiersprachen auf Github bereitstellt. In der Dokumentation direkt wird allerdings nur das Python-Beispiel eingegangen.

4.6.5 Absetzen einer Anfrage und Integration in PHP

Xapian besitzt keine REST-Schnittstelle. Daher muss hier per Befehl direkt auf mit Systemaufrufen bzw. über Anfragen an den TCP Server gearbeitet werden. Dabei konnte aufgrund des Zeitkontingents die Remote Ausführung nicht getestet werden, da der Aufwand die PHP-Erweiterung auf Windows zu bauen, zu hoch war. Daher habe ich die Datei direkt auf den Server ausgeführt, was bei der Laufzeit bedacht werden muss.

Der Grund, warum eine eigene Zeile für den Buchstaben ausgewiesen werden muss, ist, dass Xapian generell nur Volltext ausweist. Es wurde zuerst versucht mit Wildcards zu arbeiten, allerdings ergaben sich dabei dieselben Probleme, wie bei den anderen Suchmaschinen.

```
1 // Require xapian.php and declare variables
2
3 $db = new XapianDatabase('db'); //Open Database
4
5 $queryParser = new XapianQueryParser();
6 //Set Prefix for Search
7 $queryParser->add_prefix("searchIndexLetter", "K");
8 $query = $queryParser->parse_query('S'); // Parse Query
```

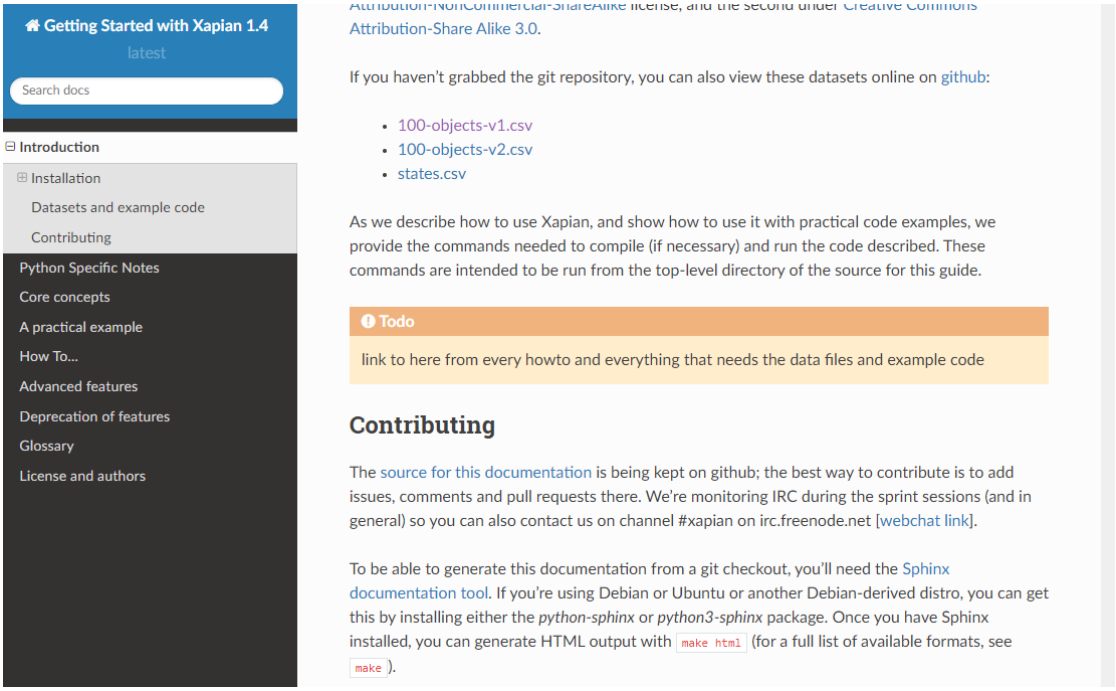

Abb. 4.10. Screenshot von der Xapian Dokumentation

```
 9
10  //Loop for and time Results
11  // Use an Enquire object on the database to run the query
12  $enquire = new XapianEnquire($db);
13  $enquire->set_query($query);
14  $matches = $enquire->get_mset(0, 2147483647)->begin();
15  foreach ($matches as $pointer){
16      $doc = $matches->get_document()->get_data();
17      //$fields = json_decode($doc);
18      $count++;
19  }
20  // Output Time for Run
21  //End Loop
22  // Output Median Time
```

Nachdem die Abfrage die ersten hundert Male durchgelaufen war, war die Zeit mit 0.0044 Sekunden im Durchschnitt für die Ergebnisse sehr gering. Dies liegt daran, dass die Ergebnisse erstmal nur Pointer auf die kompletten Daten sind.

Um nun alle Daten zu erhalten, muss nochmals ein gesonderter Befehl geschickt werden. Deswegen ist im Code ab Zeile 15 auch eine For-Schleife, welche die Datensätze für alle Pointer holt. Foreach verschiebt dabei automatisch den Pointer von den Ergebnissen. Wichtig ist, dass der Count ein Programmschritt ist, der theoretisch die Laufzeit erhöht und nicht für die normale Abfrage genutzt werden würde. Allerdings habe wurde diese Erhöhung hingenommen, um festzustellen, ob immer alle Ergebnisse korrekt geliefert werden. Die auskommentierte Zeile würde das Objekt nun als Array mit Indices zurückgeben.

Die Abfrage mit dem Abholen der Daten dauerte nun im Durchschnitt 0.22 Sekunden, was immer noch sehr schnell ist. Allerdings muss dabei bedacht werden, dass die Abfrage direkt auf dem Server lief, wodurch es keine Latenzzeit gab.

5

Fazit des Vergleiches

Nachdem nun alle Systeme für einen Ersteindruck aufgesetzt worden sind, ist es nun an der Zeit eine Suchmaschine auszuwählen. Dazu wurde ein Treffen mit einigen Mitarbeitern der Bibliothek einberufen, um die Ergebnisse zu diskutieren.

Dabei wurde schnell Datafari durch die Probleme, auf welche beim Test auftraten, ausgeschlossen. Auch Xapian wurde durch die fehlende Benutzerfreundlichkeit und der unfertigen Dokumentation abgelehnt.

Verbleibend waren nun noch Solr und Elasticsearch. Entschieden wurde sich dann letztendlich für Elasticsearch. Dies lag vor allem an den Sicherheitsfunktionen die Elasticsearch direkt mitliefert. Zudem bietet Elasticsearch die am meisten einsteigerfreundliche Erfahrung des Testes. Daher wird nun im nächsten Kapitel mit Elasticsearch weitergearbeitet.

6

Nutzung des Open Archives Initiative Protokolls für Metadaten

Während des Projektes kam die Frage auf, ob das Open Archive Initiative Protokoll für die Datenerhaltung verwendet werden sollte. Daher wird dies im folgenden kurz geprüft.

6.1 Open Archives Initiative Protokolls

Das Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) ist ein Protokoll zum Austausch von Metadaten. Dabei werden Anfragen per GET oder POST-Request angefragt. Als Antwort erhält man im Folgenden ein XML-Dokument. So können die Metadaten mit bestimmten Facetten abgefragt werden (zum Beispiel Autor). Dabei geht es allerdings darum primär darum Änderungen weiterzugeben. So können durch dieses Protokoll neue Einträge oder Änderungen in der Datenbank weitergeben werden. [23]

6.2 OAI Harverester

Ein OAI Harverester ist ein Programm, welches durchgehend einen Abgleich der Daten vollführt. Dabei lässt es sich die Änderungen mit einem List-Befehl von dem Server geben und gleicht diese danach mit der eigenen Struktur ab. Sollten dabei Unterschiede festgestellt werden, werden daraufhin die Änderungen auch beim Harverester eingefügt. So steht der Harverester immer mit dem Server auf einen Stand. [23]

6.3 Support der Enterprise-Suchmaschinen

Bei den vorhin genannten Enterprise-Suchmaschinen gibt es keine mit nativen OAI Harverester Support. Es gibt die Möglichkeit für manche der Suchmaschinen ein solches Verhalten mithilfe von Plugins zu implementieren. Allerdings sind die meisten dieser Add-ons auch schon veraltet.

6.4 Auswertung

Durch eine fehlende Basisimplementierung des Protokolls in den einzelnen Suchmaschinen und der Möglichkeit eines direkten Zugriffs auf die Datenbank, sehe ich keinen Grund dieses Protokoll zu verwenden. Es müsste ein Server vor die Datenbank installiert werden und ein Harverester vor der ESE. Dies ist ein großer Mehraufwand, welcher bei diesem Anwendungsfall nicht notwendig ist. Sollte allerdings diese Suchmaschine ein übergreifendes System werden, kann darüber nachgedacht werden, die anderen Datenbanken per OAI-Harverester anzusprechen.

7

Setup

Dieses Kapitel behandelt die Installation und Ersteinrichtung der Suchmaschine über Docker. Die Installation erfolgt dabei über Docker mithilfe von Docker-Compose. Es werden 2 Elasticsearch Instanzen, eine Kibana Instanz und eine Logstash-Instanz aufgesetzt. Das fertig Setup soll dann wie folgt aussehen 7.1.

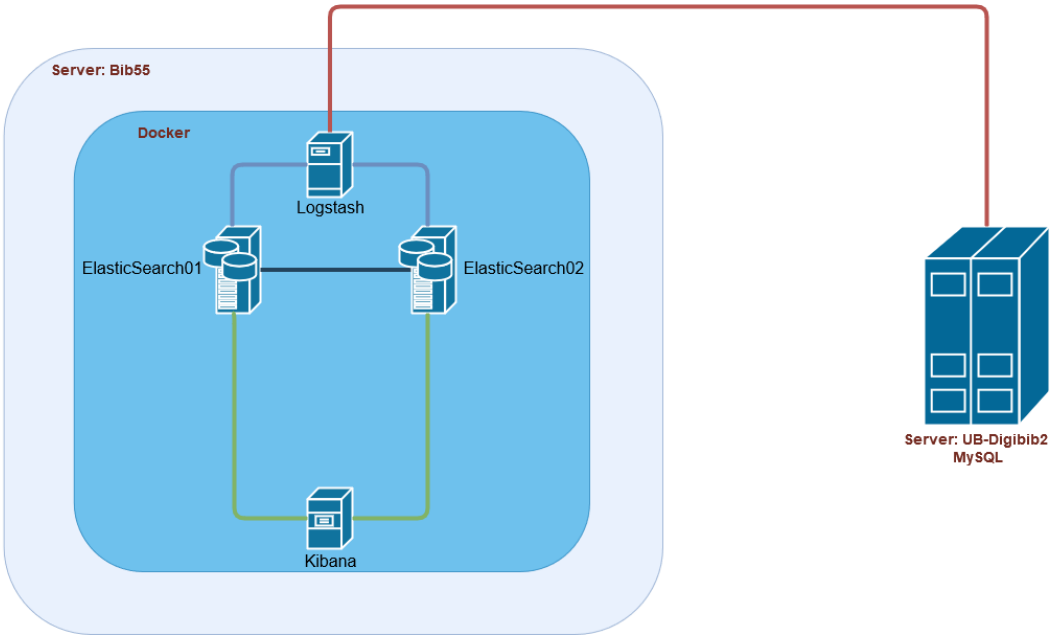

Abb. 7.1. Visualisierung des fertiges Docker-Netzwerkes

Logstash lädt die Daten von einen MySQL-Server herunter und fügt sie in die Elasticsearch Instanzen ein. Auch Kibana bekommt Zugriff auf beide Instanzen, um die Daten zu visualieren und bei Serverausfällen frühzeitig zu warnen. Die beiden Elasticsearch Instanzen halten sich dabei untereinander synchron.

7.1 Docker

Docker ist eine Software zur Virtualisierung von Anwendungen. Dabei wird allerdings nicht wie bei virtuellen Maschinen die gesamte Hardware simuliert, sondern sie laufen im Kontext der Host-Betriebstystems.

Docker-Compose ist ein Tool, mit welchen es erleichtert wird mehrere Docker-Container zu verwalten. Dafür werden in einer YAML-Datei die gewünschten Docker Container und Einstellungen, wie der Container-Name, eingetragen 7.2. Mithilfe dieser Datei erstellt Docker-Compose dann die Container und Netzwerke automatisch.

7.1.1 Rechteverwaltung in Docker

Ein kurzer Exkurs zur Rechteverwaltung in Docker. Will ein Docker-Container aus das Host System schreiben, so nutzt dieser die Berechtigungen des Users innerhalb des Docker-Containers. Nun kann es allerdings passieren, das die Nutzer-ID des Docker-Containers nicht der Nutzer-ID des Hosts entspricht. Legt man nun Dateien im Hostsystem ab, welche vom Container gelesen werden, muss dabei auch die Rechte geachtet werden.

Elasticsearch verwendet die UID und GUID von 1000. Auf dem Host System ist dies jedoch ein komplett anderer Nutzer. Das kann zu Problemen führen, da nun Dateien, welche für Elasticsearch gedacht sind einen Nutzer, welcher nicht zu diesem Projekt gehört, gehören. Der Nutzer wurde nun auf eine andere UID gesetzt, um Verwirrung zu vermeiden. [24]

7.2 Elasticsearch

Die beiden Elasticsearch Instanzen bilden das Kernstück dieses Setups. Sie werden die Daten verwalten und sich untereinander synchronisieren. Dafür werden die beiden Instanzen geclustert.

```
1  es01:
2  image: docker.elastic.co/elasticsearch/elasticsearch:7.5.1
3  container_name: es01
4  environment:
5     - "ES_JAVA_OPTS=-Xms4g -Xmx4g"
6  ulimits:
7     memlock: -1
8  volumes:
9     - /srv/elk/elasticSearch01/:/usr/share/elasticsearch/data
10    - /srv/elk/config/elasticsearch.yml:
11       /usr/share/elasticsearch/config/elasticsearch.yml
12 ports:
13    - 9200:9200
14 networks:
15    - elastic
```

In diesem Auschnitt aus der Docker-Compose werden nun die ersten grundlegenden Einstellungen getroffen.

Für die beiden Elasticsearch Instanzen wird der Java-Speicher auf 4 Gigabyte gesetzt. Dies errechnet sich dadurch, dass der Server 16 Gigabyte RAM besitzt und

die Elasticsearch Instanzen niemals mehr als 50 % des gesamten RAMs verwenden sollten. [25]

Der ulimits Befehl hebt die Begrenzung des Memory-Locks auf, damit Elasticsearch korrekt arbeiten kann. Dadurch wird der RAM von Elasticsearch nicht in den SWAP-Speicher gelegt. Dies würde die Leistung von der Suchmaschine stark beinträchtigen.

Als Volumes ist zum einen die oben genannte YAML-Datei angegeben und zum anderen wird der Datenordner gemountet. Dies dient dazu, dass, falls der Container zerstört wird, die indexierten Daten trotzdem weiterhin gespeichert werden.

Der Port wird zum Host-System durchgereicht, damit das System auch von außerhalb des Docker-Netwerkes zu erreichen ist. Dabei ist das System trotz blockierter UFW zu erreichen. Dies liegt daran, dass die Docker-Container in der Standardeinstellung die UFW ignorieren.

In der Elasticsearch-Konfigurationsdatei werden nun die Einstellungen, die speziell für das Elasticsearch-System relevant sind verwaltet 7.2.

```
1  cluster.name: dietrich-online-cluster
2  node.name: es01
3  bootstrap.memory_lock: true
4  network.host: 0.0.0.0
5  discovery.seed_hosts: ["es02"]
6  cluster.initial_master_nodes: ["es01", "es02"]
```

Darin wird zuerst der Cluster-Name definiert. Dieser dient dazu, dass die Server wissen, dass Sie dieselben Daten betreuen. Danach wird der Name des Servers vergeben. Dieser wird für spätere Einstellungen noch wichtig.

Das Memory-Lock Setting dient dazu, dass die Anwendung verhindert, dass sie in den SWAP gelegt wird.

Der Network Host wird hier auf alle Interfaces der Maschine gesetzt, damit sich alle System innerhalb der Docker Netzwerkes finden können. Das Seed-Host Setting sagt aus, an welche Nodes die Daten synchronisiert werden sollen.

Der letzte Eintrag dient dazu, dass bei der ersten Synchronisation das System weiß, welche Nodes alle Daten enthalten, also mit welchen Server sich synchronisiert werden soll. Da hier beide Systeme beim ersten Start noch keine Daten besitzen, sind alle Nodes zu beginnt Master.

7.3 Kibana

Die Grundkonfiguration von Kibana ist einfacherer als die Konfiguration von Elasticsearch. Es muss nur die YAML-Datei gemountet werden und der Port 5601 nach außen durchgereicht werden.

In der Konfigurationsdatei werden nun die Einstellungen für Kibana gesetzt. Darunter fällt der oben genannte Port, der Server-Host, in diesem Fall auch 0.0.0.0, und die Elasticsearch-Hosts. Dabei werden alle Server Instanzen mitgegeben, auf denen Kibana arbeiten soll.

7.4 Logstash

Für die Grundkonfiguration von Logstash muss, wie schon im Ersteindruck, der Treiber in die Core-Bibliothek gelegt werden. Zudem werden die Konfigurationsdateien für die Pipelines[1] gemountet.

In der Konfigurationsdatei für Logstash wird dann der Name, die Pipeline.id und die Pipeline-Worker festgelegt. Die Pipeline-Worker sind die Threads, in denen eine der konfigurierten Pipelines abgearbeitet wird. Generell sollte die Anzahl der Cores auch die maximale Anzahl der Worker sein.

7.5 X-Security

X-Security nennt sich das Paket mit den Sicherheitseinstellungen für den ELK-Stack. In diesem Schritt wird hier den kompletten Traffic zwischen den einzelnen Komponenten, sowie der Endnutzer zum Server mit SSL verschlüsselt.

Dazu mussten zuerst einmal die Zertifikate generiert werden. Dafür bietet Elasticsearch ein Tool an, welches eine Zertifikats-Autorität[2] (CA) und die einzelnen Zertifikate mit Private und Public-Key generiert. Allerdings werden diese standardmäßig im PKCS 12-Format abgespeichert. Dieses ist ein Container-Format, welches die Schlüssel und die CA zusammen verpackt. Jedoch benötigt Kibana zum Beispiel nur die Autorität als einzelnes Zertifikat und nicht in einem Container.

Normalerweise gibt es eine Möglichkeit dieses Zertifikat aus der PK12-Datei zu entpacken, jedoch gab es hierbei Probleme, da OpenSSL, das Tool welches zum Entpacken verwendet wird, die CA nicht richtig entpacken kann. [26]

Die Lösung dieses Problems war es schon bei der Zertifikat-Erstellung eine Option mitzugeben, dass die Zertifikate nicht verpackt werden sollen.

Um nun alle Zertifikate gleichzeitig zu generieren, kann eine YAML Datei mitgegeben werden. In dieser werden dann die Details für die Zertifikate wie zum Beispiel DNS-Name und IP des Servers mitgegeben werden. In diesem Fall wurde nur der DNS-Name angegeben 7.5.

```
1 instances:
2 - name: 'es01'
3     dns: [ 'es01', 'bib55', 'bib55.uni-trier.de' ]
4 [...]
```

Damit diese Zertifikate auch genutzt werden können, musste nun jeder Container das zugehörige Zertifikat mounten. Zudem wurde in den dazugehörigen Konfigurationsdateien die jeweiligen Optionen zur Nutzung der CA und Private Keys gesetzt werden.

Zusätzlich zu den Zertifikaten muss auch noch eine Password Authentifikation eingebaut werden. Dazu kann auf den Elasticsearch-Containern ein Befehl zur Erstellung der Systempasswörter aufgerufen werden. Dadurch werden alle Benutzer, welche die einzelnen Systeme wie Logstash oder Kibana zum funktionieren brauchen generiert.

[1] Eine Konfigurationsdatei, welche Daten von wo geholt werden sollen und wohin diese geschoben werden
[2] Mithilfe einer CA kann sich ein Klient gegenüber des Servers ausweisen und umgekehrt.

Auch diese müssen in den Konfigurationsdateien vermerkt werden. Weitere Nutzer können von nun an per API oder Kibana erstellt werden. Die Verteilung der Rechte passiert hierbei rollen-basiert. Es wird zuerst eine Rolle erstellt, welche die gewünschten Rechte enthält, welche daraufhin an den Nutzer weitergegeben wird. Dabei können die Rollen sehr spezifisch angepasst werden. Es können einzelnen Systemfunktionen wie die zum Beispiel die Erstellung von Snapshots spezifisch freigegeben werden. Hierbei sollte sich an das Minimalprinzip gehalten werden, also nur so viele Rechte vergeben, wie der Nutzer definitiv benötigt.

Um nun einen Query gegen das Elasticsearch System zu stellen, muss zum einen eine BasicAuth sowie das CA mitgegeben werden 7.5.

```
1  curl https://bib55:9200 --cacert ca.crt -uuser:pass
```

Damit nun auch Logstash wieder Daten an Elasticsearch senden kann, wird ein Nutzer erstellt, welcher nur auf Indices mit dem Präfix dietrich_ Zugriff erhält. Die Erstellung dieses Nutzers wurde dabei die Benutzer-Oberfläche von Kibana gemacht 7.2.

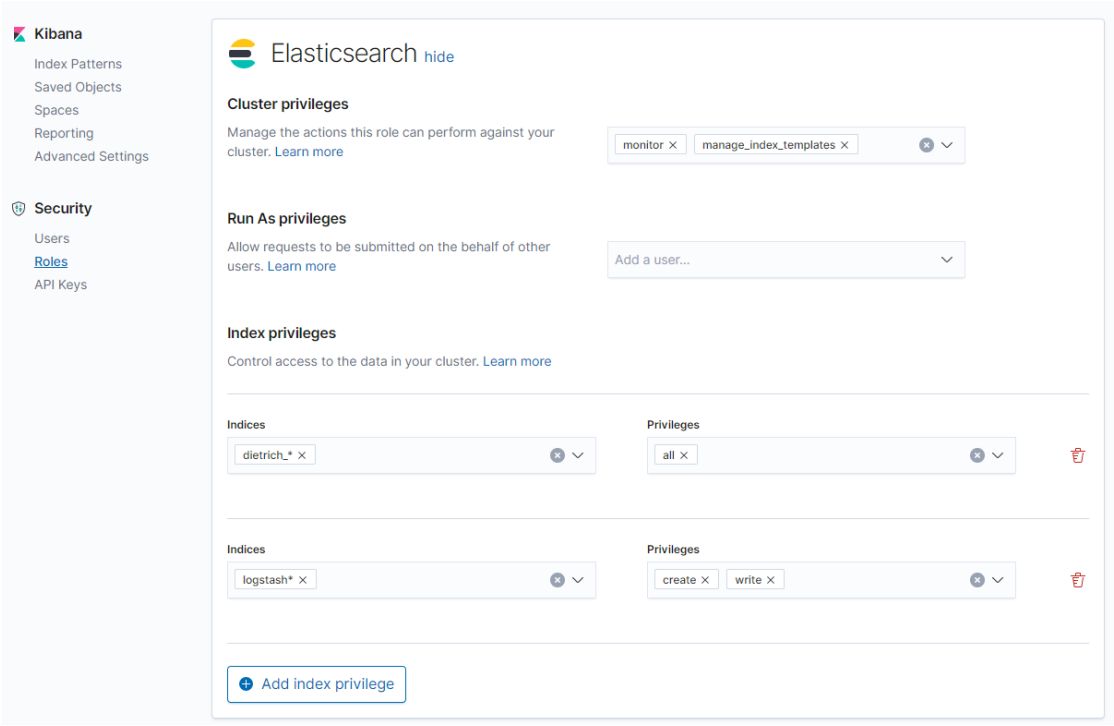

Abb. 7.2. Seite zu Erstellung von Rechte-Rollen

8

Implementation in das DietrichOnline-Projekt

In diesem Kapitel wird die Implementation ins DietrichOnline-Projekt genauer erläutert.

8.1 Vorbereitung

Um den Elasticsearch-Klienten zu nutzen, muss dieser zuerst einmal per Composer installiert werden. Die geschieht durch einen Eintrag in die composer.json.

Um für den nun folgenden Vergleich ein faires Spielfeld aufzubauen, wurde eine Datenbank, welche auf demselben Server wie Elasticsearch liegt verwendet. So ist sichergestellt, dass die Verbindung zum Server für beide Parteien gleich ist.

Zum Testen wird nun zuerst die Lemma-Abfrage, in einer modifizierten Fassung 8.1, verwendet.

Durch die eben aktivierte X-Security von Elasticsearch braucht der Klient diesmal einen Zugang per API-Key. Dieser kann mithilfe eines Curl Befehls generiert werden 8.1. Der API-Key bekommt hierbei lesende Rechte auf alle Dietrich-Indices. Auch muss die Zertifikats-Autorität bei jeder Anfrage mitgegeben werden.

```
1  POST /_security/api_key
2  {
3    "name": "dietrich-website",
4    "role_descriptors": {
5      "role-a": {
6        "cluster": ["all"],
7        "index": [
8          {
9            "names": ["dietrich_*"],
10           "privileges": ["read"]
11         }
12       ]
13 }}}
```

Indexierung

Um nun alle Daten in richtiger Form in das Projekt zu laden, muss die Indexierung von damals umgeschrieben werden.

Bei den Joins wurden M zu N Beziehungen auf eine flache Ebene gezogen. Dabei wird der Eintrag so oft abgebildet, wie es Objekte in der M zu N Beziehung gibt.

Zur Verdeutlichung hier ein Beispiel. Es gibt eine Tabelle Artikel, welche eine M zu N Beziehung mit der Tabelle DDC bezieht. In der Artikel Tabelle gibt es den Eintrag Trier mit der ID 1, der mit 2 DDC Einträgen, Trier und Rheinland-Pfalz verbunden wird. Bei einem Join wird nun alles in eine flache Hierachie gezogen. Deswegen würde die Tabelle der Ergebnisse des Joins nun die Folgenden Einträge enthalten 8.1.

ID	Artikel	DDC
01	Trier	Trier
01	Trier	Rheinland-Pflaz

Tabelle 8.1. Tabelle für ein Beispiel der Joins

Um nun einen solchen Eintrag in Elasticsearch abzubilden wird ein Array in Elasticsearch benötigt. Für solche Fälle gibt es den Aggregat-Filter in Logstash. Dieser aggregiert auf Basis der ID die Daten. So ist es nun möglich Code zu schreiben, der automatisch die Daten in Arrays zusammenfasst.

Dabei ist es wichtig, dass dieser Prozess nicht in mehreren Threads ausgeführt wird. Daher erhält jede Pipeline in diesem Projekt auch maximal einen Thread. Hier nun einmal ein Beispiel aus dem Code 8.1.

```
1  [...]
2  map['bstatus_beschreibung'] ||= event.get('bstatus_beschreibung')
3
4  map['ddc_entries'] ||= []
5  if event.get('ddc_notation') != nil
6      duplicate = false
7      map['ddc_entries'].each { |n|
8          if n.value?(event.get('ddc_notation'))
9              duplicate = true
10             break
11         end
12     }
13     if !duplicate
14         map['ddc_entries'] << {
15             'ddc_notation' => event.get('ddc_notation'),
16             'ddc_schlagwort' => event.get('ddc_schlagwort'),
17             'ddc_webdewey_is_checked' =>
18               event.get('ddc_webdewey_is_checked')
19         }
20     end
21 end
22 [...]
```

Dieser Code wird nun so lange in einer Schleife durchlaufen, wie dieselbe ID aus der Datenbank kommt. Der Wert bstatus_beschreibung wird dabei zum Beispiel bei jeden Durchlauf überschrieben.

Damit die sich ändernden Wert aggregiert werden, wurde ein Code geschrieben, welche Einträge in eine Array schreibt. Da bei dieser Abfrage 4.2.5 nun allerdings mehrere Left-Joins aufeinander ausgeführt werden, kann es dazu kommen, das sich Zeilen wiederholen. Um dieses Problem zu lösen wurde ein Code geschrieben der Duplikate und NULL Werte erkennt und nicht nochmals in das Array einträgt.

8.2 Aufbau der Abfrage

Als SQL-Framework wurde Doctrine verwendet. Dieses bietet eine Abstraktion für SQL in Objekte. Im Hintergrund werden diese Objekte dann in SQL übersetzt. Die hier betrachtete Abfrage wurde schonmal von Hand optimiert, da zuerst alle Ids der anzuzeigenden Lemmata gesucht werden, und erst im zweiten Schritt alle Joins auf den verbleibenden Datensätzen ausgeführt werden.

Die Elasticsearch-Abfrage anderseits besteht nur aus einer Abfrage, da hier alle Daten schon auf einer flachen Ebene existieren.

Verwendet wird hier ein sogenannter Boolean-Query. Dieser enthält vier verschieden Untergruppierungen.

Zuerst einmal der Must-Teil. Alle hier Angegeben Parameter müssen in jedem Ergebnis vorhanden sein. Dies ist gleichzustellen mit einem booleschen AND.

Als Zweites der Must-Not-Teil. Dieser Teil ist den Must-Teil sehr ähnlich, allerdings sind die Parameter negiert.

Danach der Shoud-Teil. In diesem Teil muss nur einer der Parameter vorhanden sein. Dies ist zu vergleichen mit einem booleschen OR.

Und zuletzt der Filter-Teil. Die gesetzten Filter sind auch Must-Befehle. Allerdings werden diese nicht bei der Gewichtung der Ergebnisse mit eingerechnet. Für diese Arbeit hat dies erstmal keinen Einfluss, da alle Ergebnisse alphabetisch sortiert werden. Daher ist diese Abfrage mit Must gleichzustellen. [27]

```
1  //Create Client with basic Params
2  $mustNotQueries = [];
3  $filters        = [];
4  $mustQueries    = [];
5
6  if ($character === LemmaEntity::NOT_A_TO_Z_CHARACTER) {
7      $mustQueries[] = ['regexp' => ['bezeichnung.keyword' =>
8        ['value' => '@&~(^[a-zA-Z].+)', 'flags' => 'ALL']]];
9
10     $filters[]     = ['term' => ['ist_geloescht' => false]];
11 } elseif ($character === LemmaEntity::DELETED) {
12     $filters[] = ['term' => ['ist_geloescht' => true]];
13 } else {
14     $mustQueries = ['prefix' => ['bezeichnung.keyword' => "$character"]];
15     $filters[]   = ['term' => ['ist_geloescht' => false]];
16 }
17
18 switch ($filter) {
19     case self::STATUS_FILTER_KLAR:
20         $filters[] = ['term' => ['bstatusbezeichnung' => 'klar']];
21         break;
22     //[Other Filters]
23 }
24
25 $params['body']['query']['bool']['must']     = $mustQueries;
26 $params['body']['query']['bool']['must_not'] = $mustNotQueries;
27 $params['body']['query']['bool']['filter']   = $filters;
28
29 return $client->search($params)['hits']['hits'];
```

In Zeile 14 ist hier zu sehen, dass anstelle der Wildcard-Abfrage, welcher in der vorherigen Abfrage 4.5.5 verwendet wurde, ein Prefix-Abfrage verwendet wird. Dieser bietet eine sauberere Lösung zum Suchen von Wortanfängen.

8.3 Vergleich

Die oben beschriebenen Abfragen wurden jetzt jeweils 100-Mal mit einem Timer laufen gelassen.

Bei dem Vergleich kamen die folgenden Durchschnittswerte zustande:

System	Zeit
MariaDB + Doctrine	3.49
Elasticsearch	1.45

Tabelle 8.2. Vergleich der Laufzeit zur Abfrage aller Daten für Buchstabe S der Lemma-Administration (15.846 Einträge)

Dabei ist zu sehen, dass Elasticsearch eine Reduktion der Zeit um 58,45 % ermöglicht.

Diese Abfrage wurde auch noch in einer nachgebauten Produktionsumgebung ausgeführt. Daran erkennt man, dass die Abfrage durchaus schneller agiert, allerdings im Gesamtkontext kaum einen Unterschied macht. Elasticsearch lädt mit 46,49 Sekunden durchaus als Doctrine mit 50,1 Sekunden, allerdings fällt dies bei der langen Ladezeit kaum ins Gewicht.

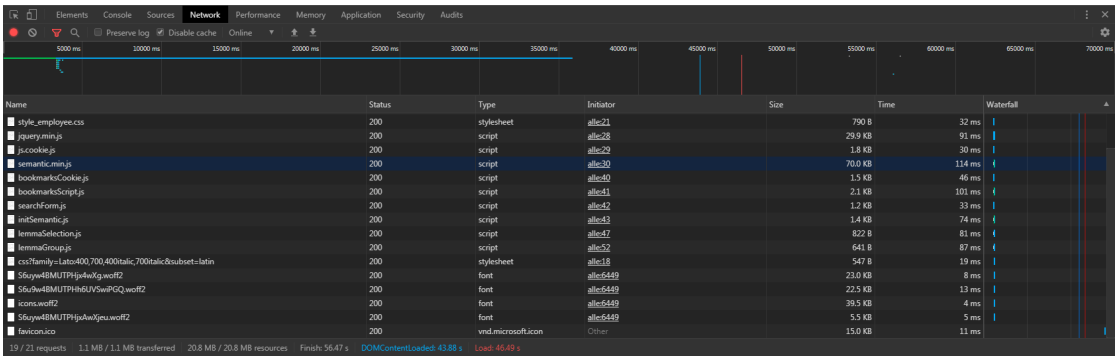

Abb. 8.1. Geschwindigkeit: Elasticsearch

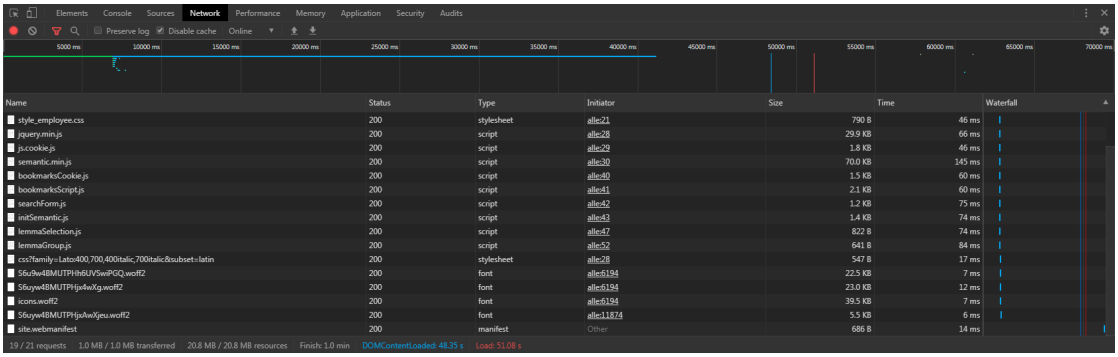

Abb. 8.2. Geschwindigkeit: Doctrine

9

Frontend-Suche

Diese Kapitel handelt von der Implementierung des Elasticsearch-Systems in die DietrichOnline Frontend-Suche. Dabei soll zuerst einmal die Suche und die automatische Vervollständigung komplett übertragen werden. Danach sollen noch eine neue Suchart, welche mehr Felder umfasst, sowie eine Anzeige der Autoren, welche bei der aktuellen Suche die meisten Artikel verfasst haben.

9.1 Indexierung

Um die Daten zu indexieren, wurde eine statische Code-Analyse durchgeführt, welche Daten aktuell alle im Frontend angezeigt werden. Dafür wird der gesamte Code zur Suche untersucht und alle Werte aufgeschrieben. Aufgrund dieser Basis wurde eine Abfrage gebaut, welche alle Daten an den Aggregat-Filter weiterreicht. Dann werden diese zum einen aggregiert, wie schon im letzten Kapitel beschrieben, allerdings auch transformiert. So wird zum Beispiel eine URL, welche vorher immer zur Laufzeit zusammengesetzt wurde nun direkt schon zusammengebaut in die Suchmaschine eingepflegt.

Für die Auto-Vervollständigung mussten auch neue Felder angelegt werden. Es gibt im Projekt für jede Suchart eine eigene Vervollständigung. Um dies auch im Index abzubilden, wurden diverse Vervollständigungs-Felder angelegt, welche zum Teil einzelne Felder oder Feldmengen durchsuchen. Diese Felder wurden dann mithilfe des Aggregat-Filters befüllt. Ein Beispiel:

Artikeltitel und die dazugehörige Sigle werden in zwei unterschiedlichen Felder gespeichert, sollten allerdings für die Vervollständigung zusammengesetzt werden. Dafür wird in dem Aggregationsfilter von Logstash definiert, der das Feld zusammenbaut 9.1. Nun wird daraus entstandene String in das Suggestions-Feld von Elasticsearch gegeben. Elasticsearch indexiert daraufhin des String so, dass jedes Wort zur Auto-Vervollständigung benutzt werden kann. Als Ausgabe wird allerdings immer der komplette String zurückgegeben.

```
1  map['artikel_titel_suggest'] ||=
2      '['+event.get('lemma_bezeichnung').to_s+']'
3          +event.get('artikel_titel').to_s
```

Sollten aus einem Datensatz mehrere Felder indexiert werden, ist es möglich auch ein Array von Daten an das Auto-Vervollständigungsfeld von Elasticsearch weiterzugeben.

Der Index ist hierbei der aktuell Größte im DietrichOnline-Projekt mit rund 1.4 Millionen Einträgen. Die aktuelle Größe des Indexes zusammen mit den Feldern zur Auto-Vervollständigung beläuft sich auf rund 4.2 Gigabyte. Diese Größe wurde hierbei schon ein wenig dadurch reduziert, dass jeder Feldtyp vorher definiert wurde.

So wurde zum Beispiel die URL nur als String und nicht als Volltext gespeichert. Auch werden Felder, die unter 255 Zeichen lang sind als Keyword und Volltext gleichzeitig indexiert. Es wurde sich hier für nur einen Typen entschieden, um mehr Platz zu sparen.

Als nun allerdings die Pipeline das erste Mal gelaufen ist, kam es zu einem Absturz von Logstash. Dies lag daran, dass Logstash nicht genügend Speicher hatte, um diese Abfrage abzuarbeiten. Um weitere Probleme von dieser Seite zu verhindern, wurde der RAM für Logstash auf 4 Gigabyte erhöht.

9.2 Integration

Die Integration folgte demselben Muster, wie die Lemma-Administration. Auf ein paar Unterschiede wird hier jedoch genauer eingegangen.

9.2.1 Paginierung

Die bisherige Paginierung holte bis zu 1001 Ergebnisse aus der Datenbank und generierte daraufhin die Paginierung. Die Begrenzung ergibt sich daher, dass die vollen Datensätze aus der Datenbank geholt wurden, und dies bei größeren Zahlen zu einer langen Laufzeit führte.

Diese Einschränkung kann nun mit Elasticsearch entfernt werden. Dazu wird zuerst einmal eine Abfrage abgesetzt, der alle Ergebnisse zählt. Für diesen Fall liefert Elasticsearch eine Count-Abfrage mit.

Durch diesen wird eine Paginierung generiert. Dafür wird mithilfe der Seitennummer ein Offset für die Abfrage generiert, sodass Elasticsearch immer nur die aktuellen Ergebnisse für die Suche liefert.

```
1  $result = $repo->findUserSearchResult(
2      //array with all search-querys and junctors
3      $this->userSearchItemArray,
4      //offset for the results
5      ($request->query->getInt('pageNumber', 1) * 30 - 30)
6  );
```

9.2.2 Query String

Für die diversen Suchen wurde eine Query-String Suche verwendet, da diese erlaubt, dass Wildcard Symbole zu verwenden. Dadurch geben sich zwar Performanz-Einbußen, allerdings sind Wildcard-Suchen ein oft genutztes Element in diesem Projekt und daher erforderlich.

```
$subQuery = [
    'query_string' => [
        'query' => $userSearchItem->getValue(),
        'fields' => [
            "artikel_titel",
            "lemma_bezeichnung",
            [...], //weitere Felder
            "normlitref_entries.normlitref_kvk_bezeichnung",
        ],
        "lenient" => true,
    ],
];
```

Ein Augenmerk muss auch noch auf die Lenient-Option gelegt werden. Ist dieser Wert nicht gesetzt, bricht die Suche mit einem Format-Fehler, wie das Suchen eines Strings in einem Zahlen-Feld, ab. Diese Funktion wurde daher bei allen Suchen abgeschaltet. Gerade wie bei diesem Beispiel, muss die Suche Felder mit diversem Inhalt gleichzeitig durchsuchen, ohne abzubrechen.

9.2.3 Boolesche Logik

Zum anderen ist es möglich eine boolesche Logik bei der Suche zu verwenden. Um diese Umzusetzen, werden die Teile der Abfrage ineinander verschachtelt 9.2.3.

Bei jeder Suche wird ein Array mit allen Suchanfragen weitergegeben. Das erste Item im Array hat dabei niemals einen Junktor. Dafür existiert der erste Fall. Existiert eine weitere Stelle im Array ist auch ein Junktor mit angegeben. Dieser wird dann in dem unten gezeigten Switch-Case ausgelesen. Dann wird eine weitere Boolean-Abfrage geschrieben, welcher zum einen den zweiten Teil der Suche, sowie die bisherige Suche enthält.

```
switch ($userSearchItem->getJunktor()) {
    case UserSearchItem::JUNKTOR_NO: //First Entry
        $this->fullQuery = [
            'bool' => [
                'must' => [
                    $this->addTypeValue($userSearchItem), //Add Search
                ],
            ],
        ];
        break;
    case UserSearchItem:: JUNKTOR_AND: //MUST
        $this->fullQuery = [
            'bool' => [
                'must' => [
                    $this->addTypeValue($userSearchItem), //Add Search
                    $this->fullQuery, //First Part of Query
                ],
            ],
        ];
        break;
[...] // More Cases like OR or AND NOT
```

9.2.4 Auto-Vervollständigung

Die Indexierung dieser Felder wurde schon im obigen Kapitel besprochen. Hier geht es nun darum, wie eine Abfrage an dieses System aussieht. Damit das System weiß,

welches Suggestions-Feld verwendet werden soll, wird dieses in der Applikation als Array hinterlegt. Normalerweise wird auch der gesamte Eintrag mit aus der Datenbank geladen. Da dies bei der Projektstruktur DietrichOnlines nicht benötigt wird, wird das _source-Feld auf leer gesetzt.

```
$params = [
    'index' => 'dietrich_frontend',
    'body' => [
        '_source' => '', //Empty Source since we need only the String
        'suggest' => [
            'auto_complete' => [
                'prefix' => $matchAgainst,
                'completion' => [
                    'field' => SEA::AUTOCOMPLETE_COLUMNS[$categoryIndex],
                    'size' => $maxMatches,
                    'skip_duplicates' => true,
]]]]];
```

9.2.5 Vollständige Suche

Die vollständige Suche soll die aktuelle Schnellsuche, welche aus Artikeltitel mit Sigle besteht ersetzen. Dazu wurde zuerst geschaut, welche Felder sonst noch von Interesse sein könnten.

Nach einer Besprechung mit einem Mitarbeiter wurde eine Liste mit relevanten Spalten erstellt. Daraufhin wurde analysiert, welche Spalten für die Auto-Vervollständigung indexiert werden sollen. Dabei wurden Felder, bei den es keinen Sinn ergibt sie automatisch zu vervollständigen, wie das ID-Feld, herausgenommen. Auf den verbleibenden Feldern wurde dann ein Auto-Vervollständigungs-Index gebaut.

9.2.6 Autoren

Bei jeder Suche soll eine Auswertung mitgeschickt werden, welche Autoren in der aktuellen Suche die meisten Artikel verfasst haben. Dazu wird eine Aggregation bei jeder Suche auf dem Autoren-Feld durchgeführt. Die Abfrage wird dafür um einen Parameter erweitert 9.2.6.

```
'aggs' => [
    'best_authors' => [
        'terms' => [
            'field' => 'artikel_autor.keyword',
]]],
```

Nun kann kommt bei jeder Suchanfrage eine Aggregation namens 'best_authors' mit zurück. Diese enthält zum einen den Namen, sowie die Anzahl der gefundenen Dokumente des Autors in der jeweiligen Suche. Mit diesen Daten war es nun möglich Buttons zu generieren, welche eine neue Suche mit den Autoren starten 9.1.

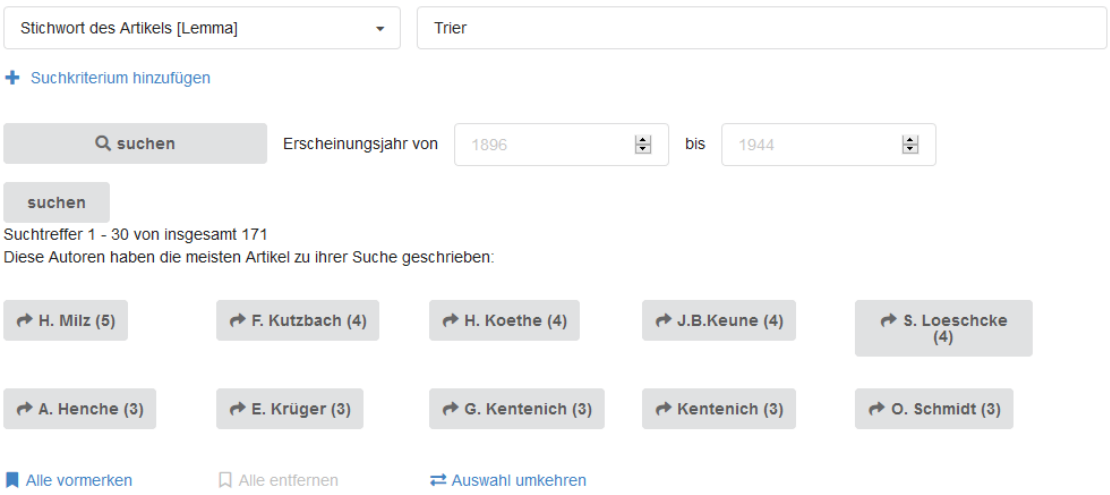

Abb. 9.1. Abbildung der erweiterten Suche

10

Zusammenfassung und Ausblick

Diese Bachelorarbeit hat sich ausführlich mit Enterprise-Suchmaschinen auseinandergesetzt, diese Verglichen und letztendlich eine in das DietrichOnline-Projekt implementiert. Das Ziel dabei war es eine geeignete Suchmaschine für dieses Projekt zu finden und implementieren.

Im ersten Schritt wurden diverse Suchmaschinen erstmal nach einer Anforderungsliste verglichen. Dafür wurde eine Tabelle erstellt, welche alle Suchmaschinen anhand der gefundenen Funktionen verglichen. Mithilfe dieser Basis wurden vier Suchmaschinen für den genaueren Vergleich herausgesucht.

Für den genaueren Vergleich wurden diese Suchmaschinen nacheinander aufgesetzt und einige Dokumente indexiert. Dabei musste die Suchmaschine selbständig die Daten aus der Datenbank laden und indexieren. Zudem wurde auch die Benutzerfreundlichkeit untersucht. Dafür wurde die Oberfläche, insofern eine vorhanden war, und die Dokumentation bewertet. Zum Schluss wurde daraufhin eine Suchmaschine ausgewählt, welche in das DietrichOnline-Projekt implementiert werden sollte. Dabei war es aufgrund der Zeit leider nicht möglich einen korrekten wissenschaftlichen Vergleich zu erstellen. Es wurde lediglich ein Ersteindruck gewonnen.

Als Nächstes wurde über die Möglichkeit nachgedacht einen OAI Harverester vor die Datenbank zu stellen, um eine normierte Schnittstelle zwischen der Datenbank und Suchmaschine herzustellen. Nach einer kurzen Analyse wurde diese Methodik allerdings verworfen, da ein direkter Zugriff auf die Datenbank möglich ist und somit der Vorgang um an die zu indexierenden Daten zu kommen nur komplizierter gestaltet wird. Diese Funktion könnte allerdings für Datenbanken ohne direkten Zugriff interessant sein.

Nachdem nun eine Suchmaschine ausgewählt wurde, ging es nun darum diese ordentlich aufzusetzen. Dabei wurde in dieser Arbeit Docker-Compose verwendet. Die Kommunikation zwischen den einzelnen virtuellen Containern wurde hierbei mit selbst generierten Zertifikaten verschlüsselt. Dabei kam es zu einigen Problemen mit der Generierung und Verwendung der Zertifikate, weshalb darüber nachgedacht werden sollte, ob die Verschlüsselung innerhalb des Systems zielführend ist, insofern das System weiterhin auf einen Server laufen soll.

Im letzten Schritt wurde nun noch eine prototypische Implementierung in das Projekt vorgenommen. Dafür wurde ein Index mit allen für die Suche wichtigen Daten aufgebaut. Um die Größe des Indexes zu minimieren wurde für alle Felder

ein vorheriges Mapping vorgenommen. Zudem wurden extra Felder für eine Auto-Vervollständigungsfunktion indexiert. Mithilfe dieses Indexes wurde die Suche für die Nutzer verbessert. Es werden nun mehr verschiedene Sucharten unterstützt. Auch ist es nun möglich mehr als 1001 Ergebnisse zu erhalten. Dies war vorher eine durch die Datenbank auferlegte Grenze. Um zu zeigen, was die Suchmaschine sonst noch für Funktionen unterstützt wurde zudem eine Funktion eingebaut, die die zehn Autoren auflistet, welche die meisten Artikel in der aktuellen Suche geschrieben haben.

Es wurde für einen Vergleich noch ein Index über alle Lemmata aufgebaut. Dieser ist der aktuell am langsamsten ladende Teil des Projekts. Mit dem Wechsel auf Elasticsearch ist es so gelungen die Laufzeit von dieser Abfrage, um 50 % zu verringern.

Zur Implementierung wurde der offizielle Klient von Elasticsearch verwendet, welcher auf einer sehr niedrigen Ebene arbeitet. Es gibt auch Klienten, welche das Level ein wenig mehr abstrahieren und so eine angenehmere Erfahrung bieten, allerdings diese alle nicht offiziell unterstützt. Daher habe ich mich in dieser Arbeit auf den Klienten von Elasticsearch fokussiert.

Sobald die Suchmaschine in das Projekt eingegliedert ist, können viele weitere Probleme des Projektes gelöst werden. So können zum Beispiel Synonymlisten für Autoren geführt werden, um die verschiedenen Schreibweisen bestimmter Autoren auszugleichen. Auch ist es mit der Suchmaschine möglich dem DDC-Baum, welcher schon seit langer Zeit implementiert werden sollte, leichter einzubauen. Zudem bietet Elasticsearch Funktionen zur Autokorrektur, welche die Sucherfahrung positiv bereichern können. Und für die Entwickler nimmt Elasticsearch einiges an Problemen mit der Datenbank ab. Aktuell werden viele Felder mithilfe von Triggern und Funktionen erstellt. Diese Trigger können nun auf Logstash übertragen werden, um so die Datenbank zu entlasten.

Damit nicht bei jeder Anfrage eine Zertifikats-Autorität mit gereicht werden muss, kann auch noch ein sogenannter Reverse Proxy vor die Elasticsearch Instanz gesetzt werden, welcher daraufhin Zertifikate mithilfe von LetsEncrypt generiert.

Literaturverzeichnis

1. “Ddc.” [Online]. Available: https://www.dnb.de/DE/Professionell/DDC-Deutsch/ddc-deutsch_node.html [Accessed: 05.02.2020]
2. “Gemeinsame normdatei (gnd),” 2019. [Online]. Available: https://www.dnb.de/DE/Professionell/Standardisierung/GND/gnd_node.html [Accessed: 05.02.2020]
3. “Dietrichonline projekt,” Trier, 2016. [Online]. Available: http://dietrich.uni-trier.de/ [Accessed: 05.02.2020]
4. “Apache lucene - wikipedia,” 27.10.2019. [Online]. Available: https://en.wikipedia.org/w/index.php?oldid=915250662 [Accessed: 30.10.2019]
5. “Apache lucene - apache lucene core,” 26.07.2019. [Online]. Available: https://lucene.apache.org/core/ [Accessed: 25.10.2019]
6. R. McCreadie and Craig Macdonald and Jie Peng, “Terrier ir platform - homepage,” 25.01.2019. [Online]. Available: http://terrier.org/ [Accessed: 23.10.2019]
7. “Sphinx — open source search server.” [Online]. Available: http://sphinxsearch.com/docs/manual-2.3.2.html#intro [Accessed: 06.11.2019]
8. “Sphinx search server.” [Online]. Available: https://github.com/sphinxsearch/sphinx [Accessed: 30.10.2019]
9. “Sphinx — open source search engine.” [Online]. Available: http://sphinxsearch.com/ [Accessed: 30.10.2019]
10. “Apache solr - wikipedia,” 14.10.2019. [Online]. Available: https://en.wikipedia.org/w/index.php?oldid=915250761 [Accessed: 30.10.2019]
11. “Apache solr,” 26.07.2019. [Online]. Available: https://lucene.apache.org/solr/ [Accessed: 16.10.2019]
12. “Elasticsearch: Verteilte restful-suchmaschine und -analytics engine — elastic.” [Online]. Available: https://www.elastic.co/de/products/elasticsearch [Accessed: 16.10.2019]
13. “Professional support — n2sm, inc.” [Online]. Available: https://www.n2sm.net/en/support/fess_support.html [Accessed: 06.11.2019]
14. “Fess installation guide,” 31.10.2019. [Online]. Available: https://fess.codelibs.org/13.4/install/index.html [Accessed: 06.11.2019]
15. “Fast, reliable and modern search and discovery.” [Online]. Available: https://www.algolia.com/ [Accessed: 06.11.2019]

16. “Manticore search – open source text search engine for big data and stream filtering.” [Online]. Available: https://manticoresearch.com/ [Accessed: 16.10.2019]
17. “The xapian project,” 14.10.2019. [Online]. Available: https://xapian.org/ [Accessed: 23.10.2019]
18. F. Labs, “About << france labs: Open source enterprise search,” 2018. [Online]. Available: https://www.francelabs.com/en/about.html [Accessed: 13.11.2019]
19. Michael Brandenburg, “Suchtrupp: Eine eigene suchmaschine bauen (teil 1),” *LINUX-Magazin: Die Zeitschrift für LINUX-Professionals*, no. 11, pp. 62–68, 2019. [Online]. Available: https://www.linux-magazin.de/ausgaben/2019/11/datafari/
20. F. Labs, “Datafari enterprise search.” [Online]. Available: https://www.datafari.com/en/index.html [Accessed: 13.11.2019]
21. “Xapian users,” 14.10.2019. [Online]. Available: https://xapian.org/users [Accessed: 08.11.2019]
22. Apache Software Foundation, “Manifoldcf- end-user documentation.” [Online]. Available: https://manifoldcf.apache.org/release/release-2.14/en_US/end-user-documentation.pdf [Accessed: 04.12.2019]
23. “Oai-schnittstelle,” 31.05.2019. [Online]. Available: https://www.dnb.de/DE/Professionell/Metadatendienste/Datenbezug/OAI/oai_node.html [Accessed: 25.10.2019]
24. C. M. Jarrod Weaver, “Understanding user file ownership in docker: how to avoid changing permissions of linked volumes - stack overflow,” 2014. [Online]. Available: https://stackoverflow.com/questions/26500270/understanding-user-file-ownership-in-docker-how-to-avoid-changing-permissions-o [Accessed: 12/23/2019]
25. “Setting the heap size — elasticsearch reference [7.5] — elastic,” 12/17/2019. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/heap-size.html [Accessed: 12/23/2019]
26. nerophon, “[docs/security] describe how to extract ca, cert & key from p12 truststore in kibana docs · issue #26414 · elastic/kibana,” 2018. [Online]. Available: https://github.com/elastic/kibana/issues/26414 [Accessed: 02.01.2020]
27. “Boolean query — elasticsearch reference [7.5] — elastic,” 17.12.2019. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html [Accessed: 18.01.2020]

A

Glossar

ESE	Enterprise Search Engine
Facetten	Filter in Bibliothekarssprache
OAI	Open Archives Initiative
OCR	Optical Character Recognition

B

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Datum	Unterschrift der Kandidatin / des Kandidaten