Informatik
Hauptcampus

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

Implementierung einer Enterprise Search Engine für das Dietrich Online Projekt

Implementation of an enterprise search engine for the Dietrich Online project

Florian Reitz

Bachelor-Abschlussarbeit

Betreuer: Professor Doktor Christoph Schmitz

Trier, den 15.10.2019 15.3.2020

Vorwort

Diese Arbeit entstand als Abschlussarbeit für die 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 mich beim Dietrich-Online Team und vor allem bei Herrn Kock für die Unterstützung zu bedanken.

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

Trier, 2020
Florian Reitz

Kurzfassung

German

Diese Arbeit handelt von der Analyse diverser Enterprise Suchmaschinen für das Dietrich-Online 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 Dietrich-Online Projekt vorgenommen. Dafür wurde die Suche, sowie die Auto-Vervollständigung auf die Suchmaschine umgezogen.

English

This thesis analyzes various enterprise search-engines for the Dietrich-Online project 2. The search-engines were checked via a request 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 Dietrich-Online project. The search and the auto-completion function were set up to work with the ElasticSearch.

Inhaltsverzeichnis

1 Einleitung und Problemstellung 1

2 Dietrich Online Projekt 2

3 Vergleich der Enterprise Search Engines 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 6
- 3.8 Manticore Search 7
- 3.9 Xapian 7
- 3.10 Datafari 7
- 3.11 Tabellarischer Vergleich 8
- 3.12 Vorauswahl 8
  - 3.12.1 Lucene Core 8
  - 3.12.2 Sphinx 8
  - 3.12.3 Solr 8
  - 3.12.4 ElasticSearch 10
  - 3.12.5 Fess 10
  - 3.12.6 Algolia 10
  - 3.12.7 Xapian 10
  - 3.12.8 Datafari 10

4 Genauer Vergleich 11
- 4.1 Testsystem 11
- 4.2 Aufbau der Tests 11
  - 4.2.1 Installation 11
  - 4.2.2 Indexierung 12
  - 4.2.3 Dokumentation 12
  - 4.2.4 Absetzen einer Anfrage und Integration in PHP 12

4.3 Solr . . . 13
4.3.1 Installation . . . 13
4.3.2 Indexierung . . . 14
4.3.3 Oberfläche . . . 16
4.3.4 Dokumentation . . . 16
4.3.5 Absetzen einer Anfrage und Integration in PHP . . . 17
4.4 Datafari . . . 18
4.4.1 Installation . . . 18
4.4.2 Indexierung . . . 18
4.4.3 Oberfläche . . . 19
4.4.4 Dokumentation . . . 20
4.4.5 Absetzen einer Anfrage und Integration in PHP . . . 20
4.5 ElasticSearch . . . 21
4.5.1 Installation . . . 21
4.5.2 Indexierung . . . 22
4.5.3 Oberfläche . . . 23
4.5.4 Dokumentation . . . 24
4.5.5 Absetzen einer Anfrage und Integration in PHP . . . 25
4.6 Xapian . . . 25
4.6.1 Installation . . . 25
4.6.2 Indexierung . . . 26
4.6.3 Oberfläche . . . 27
4.6.4 Dokumentation . . . 27
4.6.5 Absetzen einer Anfrage und Integration in PHP . . . 28

5 Fazit des Vergleiches . . . 30

6 Nutzung des Open Archives Initiative Protokolls für Metadaten 31
6.1 OAI Harverester . . . 31
6.2 Support der Enterprise Search Engines . . . 31
6.3 Auswertung . . . 31

7 Setup . . . 33
7.1 Docker . . . 33
7.1.1 Rechteverwaltung in Docker . . . 33
7.2 ElasticSearch . . . 34
7.3 Kibana . . . 35
7.4 Logstash . . . 35
7.5 X-Security . . . 35

8 Implementation in das Dietrich Projekt . . . 38
8.1 Vorbereitung . . . 38
8.2 Aufbau des Queries . . . 40
8.3 Vergleich . . . 41

9 Frontend-Suche 43
9.1 Indexierung 43
9.2 Integration 44
9.2.1 Paginierung 44
9.2.2 Query String 44
9.2.3 Boolesche Logik 45
9.2.4 Auto-Vervollständigung 45
9.2.5 Vollständige Suche 46
9.2.6 Autoren 46

10 Zusammenfassung und Ausblick 48

Literaturverzeichnis 50

Glossar 52

Erklärung der Kandidatin / des Kandidaten 53

Abbildungsverzeichnis

4.1 Tabellenaufbau der Lemma-Administration Übersicht. 13
4.2 Frontend Ansicht der Lemma-Administration mit geladenen Buchstaben S (Ausschnitt). 13
4.3 Oberfläche der Indexierung mit Laufzeit. 16
4.4 Startseite der Weboberfläche von Solr. 17
4.5 Übersichtsseite des Querys in Datafari. 19
4.6 Kibana Integration in Datafari. 20
4.7 Dokumentationsseite für den JDBC Treibers von Datafari. 21
4.8 Index Management Seite von ElasticSearch 24
4.9 Einstellungen vom Lemma-Index bei ElasticSearch 26
4.10 Screenshot von der Xapian-Dokumentation 28

7.1 Seite zu Erstellung von Rechte-Rollen 37

8.1 Query-Geschwindigkeit: ElasticSearch 41
8.2 Query-Geschwindigkeit: Doctrine+MySQL 42

9.1 Abbildung der erweiterten Suche 47

Tabellenverzeichnis

3.1 Feature-Vergleich der verschiedenen Enterprise Suchmaschinen 9

8.1 Tabelle für ein Beispiel der Joins 39
8.2 Vergleich der Laufzeit zur Abfrage aller Daten für Buchstabe S der Lemma-Administration (15.846 Einträge) 41

1

Einleitung und Problemstellung

Die Suche des Dietrich online Projektes 2 arbeitet aktuell auf einer MariaDB Datenbank. In dieser werden bei jeder Suchanfrage diverse Tabellen, Joins zusammengebaut und daraufhin dem Nutzer ausgegeben. Bei den Datenmengen, welche sich aktuell in der Datenbank befinden, circa 1.4 Millionen Einträge, wird die Ladezeit unangenehm lang. Daher wurden schon die maximalen Suchergebnisse, 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 passendsten Suchmaschinen werden 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 nun noch eine prototypische Implementierung in das Dietrich online Projekt vorgenommen. Dabei wird die aktuelle Suche durch die Enterprise Suchmaschine ersetzt und um einige Funktionen erweitert.

2

Dietrich Online Projekt

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

Beim Start dieses Projektes wurden dafür alle Zeitschriften mit einer OCR 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 haben alle Lemmata mithilfe von Siglen[2] normiert. Jede Sigle verweist also auf ein eigenes Lemma. Diese 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) [3] und Gemeinsamen Normdatei (GND) [4] 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] IDs
[3] Klassifikation zur Ordnung von Wissen [1]
[4] Normdatei für Personen [2]

3

Vergleich der Enterprise Search Engines

In ersten Schritt werden diverse Enterprise Search Engines evaluiert. Dafür wurde eine Anforderungsliste mit den Mitarbeitern erstellt. Die Systeme werden bei diesem ersten Vergleich nach Features verglichen. Die Suchmaschinen, die am besten abschneiden werden anschließend aufgesetzt und genauer untersucht.

Die nun folgende Liste zeigt alle K.O. Kriterien 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 zu Disqualifikation.

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

Durch die begrenzten finanziellen Mittel und die lange Projektlaufzeit besteht die Notwendigkeit eine kostenfreie, im besten Fall sogar eine Open Source Suchmaschine zu finden. Auch äußerst wichtig ist der Support für Facetten, da Dietrich-Online als Suchmaschine den Nutzer einige Tools zum Verfeinern seiner Suchergebnisse zur Verfügung stellen soll.

Da wir hier mit großen Datenmengen arbeiten ist ein Ranking auch 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 im Titel oder Beschreibungen zu suchen.

Der Support für die 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, es gibt alle Scans als PDF und für die Webseite sind alle Daten nochmals 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.

Ein Support für PostgreSQL ist für dieses Projekt nicht so wichtig. Allerdings könnte es sein, dass dieser Server später auch andere Datenbanken verwalten soll. Dadurch wäre ein Support für diese Datenbankstruktur wünschenswert.

Es gibt bei den Maschinen in der Bibliothek sowieso ein Backup-Maschinismus. Allerdings ist eine manuelle Backup-Lösung trotzdem wünschenswert, um die Suchmaschine losgelöst zu sichern und gegebenenfalls auch einfach auf einen anderen Server umzuziehen zu können.

Auto-Korrektur und Auto-Vervollständigung sind beide sehr interessant, um den Nutzer mehr Komfort-Funktionen bieten zu können.

Die Sicherheitsfunktionen sind für die Suchmaschinen mit Web-Oberfläche interessant. Generell sollte der Server ja nur intern anzusprechen sein. Wenn es allerdings eine Web-Oberfläche gibt, kann es sein, dass diese per Reverse Proxy ansprechbar gemacht wird, um eine Administration aus dem Internet möglich zu machen. 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. Allerdings bieten einige Suchmaschinen auch die Möglichkeit über JSON Anfragen an die Suchmaschine zu stellen. Es sollte zumindest eine der beiden Methoden verfügbar sein, damit die Suchmaschine einfach von PHP aus zu erreichen ist. Sollte es mal ein Problem geben, was nicht im Haus gelöst werden kann, wäre die Möglichkeit auf bezahlten Support von Vorteil.

3.1 Apache Lucene Core

Lucene Core ist eine Open Source Enterprise Search Engine 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 Query's geloggt werden. Zudem bietet es Support für 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. Bezahlten Support gibt es hier nicht, da dieses Projekt zur Apache Foundation gehört. [5]

3.2 Terrier

Terrier ist eine Open Source Enterprise Search Engine 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 zu indexieren. Es gibt allerdings eine Möglichkeit das SQL in JSON zu konvertieren und dieses 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 Open Source entwickelt. Ab Version 3 wurde die Entwicklung Closed Source. 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 Open Source geht. Version 3.1.1 wurde im Oktober 2018 veröffentlicht und seitdem lässt sich auch nichts mehr über das Projekt finden. Von daher ist davon auszugehen, dass das Projekt nicht mehr weitergeführt wird.

Zu den Features ist festzuhalten, dass es keinen nativen PDF-Support in der Open-Source Version 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 man mit der Firma noch in Kontakt treten kann. [9]

3.4 Apache Solr

Apache Solr ist eine, auf Lucene Core 3.1 basierende, viel eingesetzte Search Engine von der Apache Foundation. Sie erweitert Lucene Core, um ein grafische Benutzeroberfläche und einige weitere Funktionen. Die Entwicklung dafür begann 2004 als ein internes Projekt von CNET 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 und Best Buy eingesetzt. Durch die Unterstützung von der Apache Foundation längerfristige Weiterentwicklung abzusehen.

Da Solr zur Apache Foundation gehört, ist es Open Source. Es bietet viele Funktionen von Haus, womit es alle Grundanforderungen erfüllt und besitzt darüber hinaus auch Support für fast alle Bonus-Features. Einzig und allein gibt es keinen bezahlten Support, dafür allerdings eine große Community, welche man durch einen Mailing Listen oder IRC erreichen kann. [11]

3.5 ElasticSearch

Eine weitere große Enterprise Search Engine ist ElasticSearch. Auch dieses Projekt arbeitet auf der Basis von Lucene. Zu den bekanntesten Kunden zählen Ebay und Adobe. 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 enstand der ELK-Stack. Die enstandene 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 kann man in der Open Source Variante genießen. 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 Search Engine 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 es Support 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]

3.7 Algolia

Algolia ist eine cloud-basierte Search Engine, welche unter anderem von Twitch und Lacoste verwendet wird. Die Suchmaschine wird hierbei als SAAS (Software as a Service) angeboten. Hierbei lädt man die Daten auf Algolia Server und dann daraufhin über eine API-Schnittstelle die Suchen auf den Daten in der Cloud ausführen.

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 und Boardreader.

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 werden Log-Dateien produziert. Zuletzt gibt es noch 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, der Universitätsbibliothek Köln 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 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-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 Suchlösung vom französischen Entwickler France Labs. Das Entwicklerstudio wurde 2011 gegründet und hat sich es sich zum

Ziel gemacht, die beste Open-Source Enterpise Suchlösung 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, 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 einem ersten Feature-Vergleich haben nur 7 Suchmaschinen die Grundanforderungen erfüllt. Davon werden nun 4 Stück in den genaueren Vergleich genommen, bei die Systeme aufgesetzt und sich genauer angeschaut werden. Ich gehe nun die Suchmaschinen der Reihe nach durch und gebe zur jeder eine Begründung warum oder warum sie es nicht in den genauen Vergleich geschafft hat.

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 gibt es für Lucene Core ja eine Erweiterung, namentlich Solr, welches alle diese Probleme löst. [5]

3.12.2 Sphinx

Sphinx wäre eine interessante Alternative gewesen, allerdings durch den Kommunikationsverlust und die gestoppten Updates (Es gab schon seit über einem Jahr kein Update mehr), ist dieses Projekt wohl als tot anzusehen. [9]

3.12.3 Solr

Wie schon bei Lucene Core kurz angesprochen, liefert Solr viele der Funktionen, die in diesem Umfeld 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]

	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
SQL-Like Query Support	-	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 Search Engines. 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

3.12.4 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, welches auf ElasticSearch basiert, was ja 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 schwere 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 eine interessante Alternative. Leider sind im kostenlosen Bereich nicht genügend Einträge speicherbar. Auch sind 50.000 Operationen zu wenig für die das Dietrich-Online Projekt. Von daher fällt diese Suchmaschine durch diese Limitationen raus. [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 trotzdem noch 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 folgende Spezifikationen:

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

Auf das System wird zudem die MariaDB Datenbank von Dietrich-Online Projekt 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 man die Indexierung in einen Zeitplan legen kann.

4.2.3 Dokumentation

Im dritten Schritt wird die Dokumentation analysiert. Hierbei wird das Augenmerk auf die Übersichtlichkeit und Verständlichkeit gelegt. Da in diesen Kurztest nicht alle Bereiche der Dokumentation genau durchgelesen und daraufhin auch Testweise implementiert werden können, werden nur die Seiten, die von Relevanz für die anderen Schritte sind bewertet.

4.2.4 Absetzen einer Anfrage und Integration in PHP

Im letzten Schritt werden eine Query in PHP abgesetzt. Dabei wird die Zeit gemessen, wie lange die Query braucht um die Daten zu liefern.

Die dabei verwendete Query ist die bisher am langsamsten laufenden Query der Dietrich-Online Projektes. Sie ermittelt alle Lemmata vom Buchstaben S und baut alle Daten, die zur Anzeige benötigt werden zusammen 4.2. Die Tabellen die für diese Ansicht gebraucht werden, sind in diesem Diagramm 4.1 zu finden. Um genau zu sein, sind es zwei Querys. 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.

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

Im zweiten Schritt werden dann die gerade geholten ID's mit vielen JOIN's für die Darstellung vorbereitet.

```
1   SELECT   lemma.id,
2            [...] #Lemma, GND und DCC-Spalten
3   FROM lemma lemma
4
5   INNER JOIN lemmabearbeitungsstatus lemmaBStatus
6   ON lemma.fk_lemmabearbeitungsstatus = lemmaBStatus.id
7
8   LEFT JOIN lemma_gnd lemma_gnd_map ON lemma.id = lemma_gnd_map.fk_lemma
9   LEFT JOIN gnd gnd ON lemma_gnd_map.fk_gnd = gnd.id
10  LEFT JOIN gnd_ddc gnd_ddc_map ON gnd.id = gnd_ddc_map.fk_gnd
11  LEFT JOIN ddc ddc ON gnd_ddc_map.fk_ddc = ddc.id
12  WHERE lemma.id IN ([Array of Lemma IDs])
13  ORDER BY lemma.bezeichnung ASC, lemma.id ASC;
```

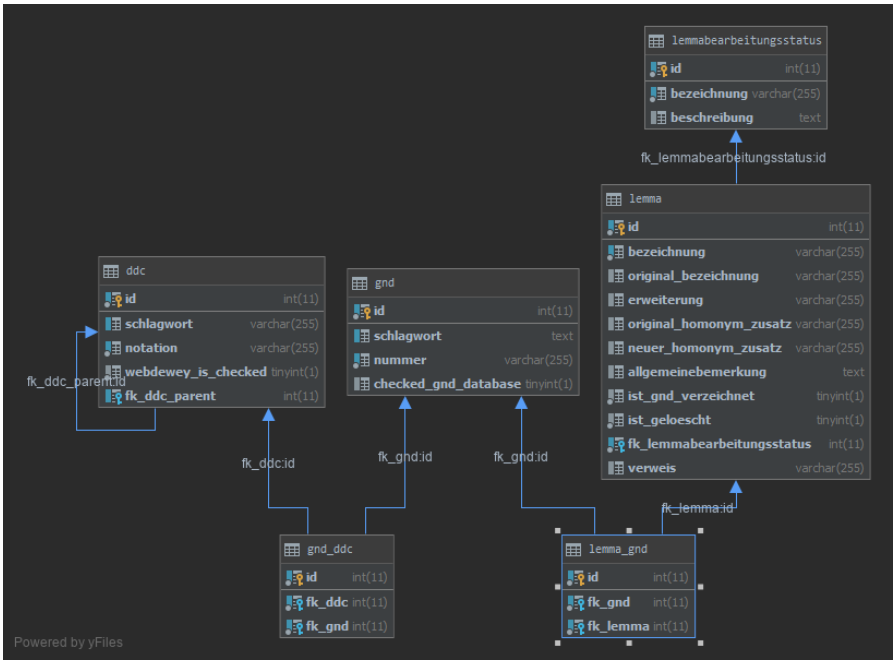


Abb. 4.1. Tabellenaufbau der Lemma-Administration Übersicht.

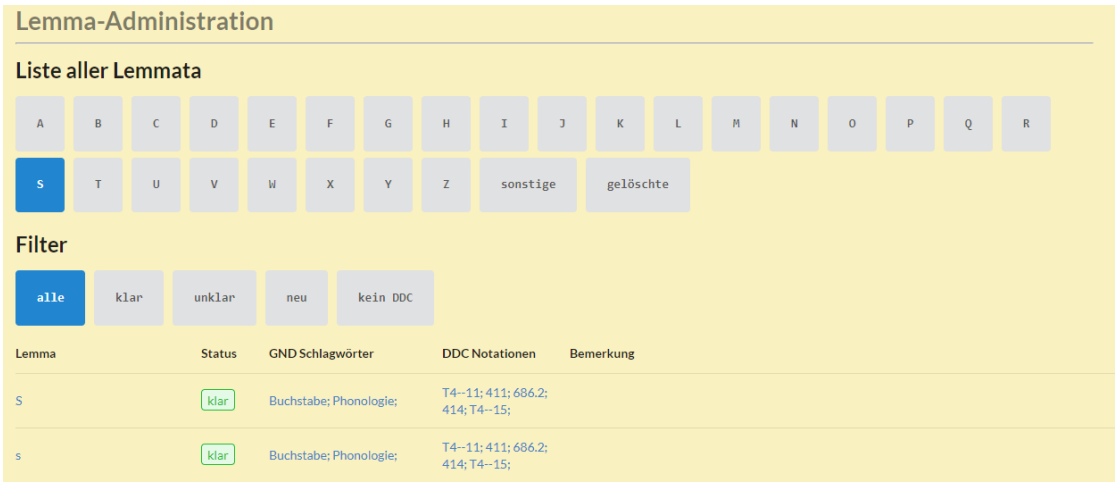


Abb. 4.2. Frontend Ansicht der Lemma-Administration mit geladenen Buchstaben S (Ausschnitt).

4.3 Solr

4.3.1 Installation

Als Systemvoraussetzungen ist eine Java Version > 8 gegeben. Ich habe mich hierbei für OpenJDK 11 entschieden. 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.
```

Um Solr im Entwicklermodus auszuführen, kann das entpackte Programm einfach mit `bin/solrstart` gestartet werden. 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.

4.3.2 Indexierung

Um mit der Indexierung starten zu können, muss zuerst in 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 nun auch 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-Queries 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-Query mit dem die Daten gelesen werden und einen Delta-Query, welcher dazu dient, nur die neuen Einträge zu laden.

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 er, zusammen mit einem JDBC-Treiber in die solrconfig.xml eingetragen. Bei dem JDBC-Treiber habe ich mich bei diesem Beispiel für den Treiber von MariaDB entschieden. Damit es nicht deswegen zu Laufzeit-Unterschieden bei der Indexierung kommen kann, werde ich diesen Treiber bei allen Systeme, dies es zulassen, den MariaDB-Treiber erlauben.

```
1      <entity name="lemma"
2        query="select * from lemma"
3        deltaQuery="select eid from lemma
4          where last_modified > '${dataimporter.last_index_time}'">
```

```
5         <field column="bezeichnung" name="bezeichnung" />
6       [...]
7       <entity name="lemma_gnd"
8         query="select * from lemma_gnd where fk_lemma='${lemma.id}'"
9         deltaQuery="select * from lemma_gnd
10          where last_modified > '${dataimporter.last_index_time}'"
11        parentDeltaQuery="select * from lemma
12          where id=${lemma_gnd.fk_lemma}">
13
14          <entity name="gnd"
15            query="select * from gnd where id = '${lemma_gnd.fk_gnd}'"
16            deltaQuery="select * from gnd
17              where last_modified > '${dataimporter.last_index_time}'"
18            parentDeltaQuery="select * from lemma_gnd where fk_gnd=${gnd.id}">
19            <field column="nummer" name="gnd_nummer" />
20            <field column="schlagwort" name="gnd_schlagword" />
21            [...]
22          </entity>
23        </entity>
24      </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. Ohne genauer darüber nachzudenken, habe ich angenommen, dass beide gleichwertig sind und nur für spätere Abfragen auf Sprachen eine Relevanz besitzen. Als ich allerdings eine Abfrage stellte, die alle Lemmata mit den Buchstaben S finden sollten, bekam ich mehr Ergebnisse als erwartet. Dies liegt daran, dass Text_de, das Feld aus Volltext ausweist. Bei Volltexten wird jedes Wort einzeln betrachet und so kamen Lemma, in welchen irgendein Wort mit S begann in meine Auflistung.

Es gibt mehrere Möglichkeiten diese Einträge auszuweisen. In diesem Fall habe ich 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, diese Einträge per API einlesen zu lassen. 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.

```
1
2       [...]
3       <field name="ddc_webdewey_is_checked" type="boolean"
4           uninvertible="false" indexed="true" stored="true"/>
5       <field name="description" type="text_de" uninvertible="false"
6           multiValued="true" indexed="true" stored="true"/>
7       <field name="erweiterung" type="text_de"
8           uninvertible="false" indexed="true" stored="true"/>
9       [...]
```

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 passenden Einträgen durchsucht.

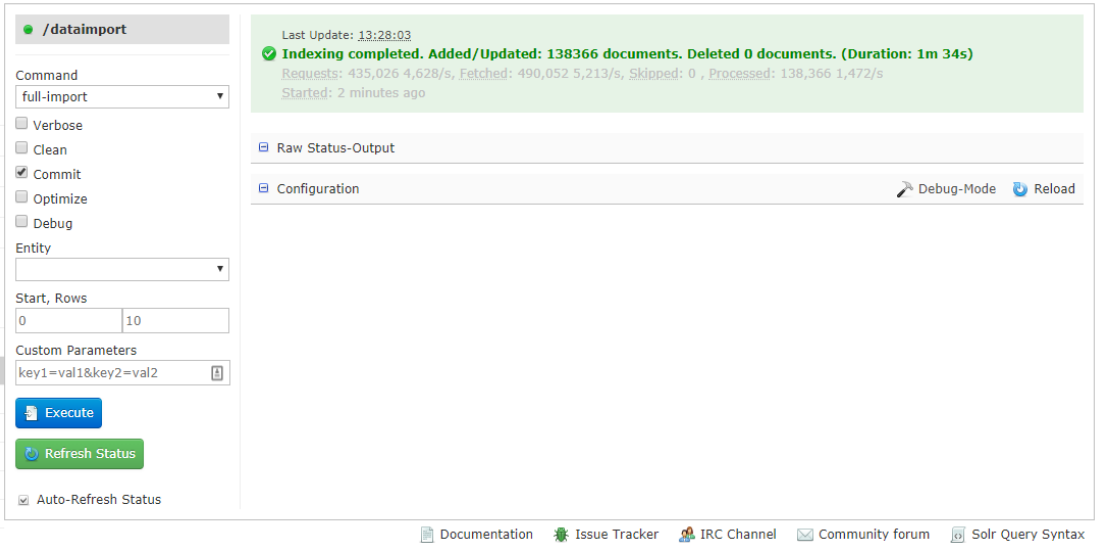


Abb. 4.3. Oberfläche der Indexierung mit Laufzeit.

4.3.3 Oberfläche

Der 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 von dem aktuellen Core 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. Leider ist es jedoch nicht möglich, den DataImportHandler direkt zu verändern, ohne weitere Einstellungen im System vorzunehmen. Es gibt eine Möglichkeit Querys direkt über den Web-Client zu senden, was zum Testen und debuggen sehr nützlich ist. 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, zudem ist die Seite auch nicht responsive geschrieben.

4.3.4 Dokumentation

Die Dokumentation war bei diesem kurzen Test meine Hauptquelle. Die Installation ist dort genau beschrieben. Positiv aufgefallen ist mir dabei vor allem die genaue Beschreibung der Systemanforderungen. Es wurde diverse Java-Versionen getestet und 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

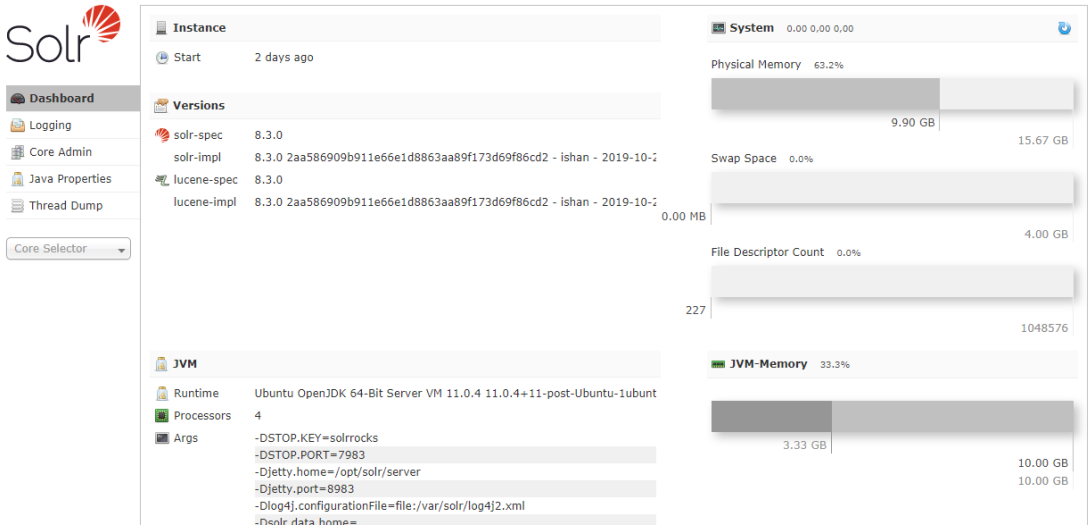


Abb. 4.4. Startseite der Weboberfläche von Solr.

Struktur erklärt. Allerdings wäre ein Verweis, dass für die DataImportHandler-Attribute noch extra ein Solr-Schema-Attribut benötigt wird, schön gewesen. Dies habe ich erst durch einen Blog [22] herausgefunden und richtig verstanden. Die Dokumentation ist, soweit ich das gesehen habe, 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 installieren lässt. Dies passt sehr gut zum Dietrich-Online Projekt, da auch dieses auf Composer setzt. Die Query ist herbei sehr einfach, da die Daten beim Import schon dementsprechend indexiert wurden.

```
1
2    [...] # Imports and variable declarations
3
4    $config = array(
5        'endpoint' => array(
6            'localhost' => array(
7                'host' => '136.199.34.55',
8                'port' => 8983,
9                'core' => 'dietrich'
10   )));
11   $queryText = 'original_bezeichnung:S*';
12   $solr = new Client($config);
13   $query = $solr->createSelect();
14   $query->setQuery($queryText);
15   $query->setRows(2147483647);
16   [...] # Loop with Timer
17   $resultSet = $solr->select($query);
18   $count = $resultSet->count();
19   [...] # Output Runtime
```

Interessant bei der Verbindung ist, dass die Anzahl der Zeilen die von Solr geladen werden, standardmäßig auf 10 limitiert sind. Erst mit setRows kann die

Anzahl erhöht werden. Ich für diesen Test den maximalen Integer-Wert gewählt, um immer alle Ergebnisse zu bekommen. Damit ich nun einen guten Median-Wert erhalte habe ich die Abfrage 100 mal laufen lassen. Dabei lief die Abfrage durchschnittlich 1.01 Sekunden um die 15838 Ergebnisse herauszusuchen.

4.4 Datafari

4.4.1 Installation

Für Datafari musste folgende Software nachinstalliert werden: Java 8 und JQ, ein JSON-Prozessor. Damit die Installation richtig durchläuft, 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 selbst durch eine DEB-Datei. Während der Installation erscheint ein kurzer Setup-Dialog, welcher einen durch die Konfiguration führt. 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 Datenbank-Verbindung eingetragen. Dabei ist es wichtig, dass vorher der Treiber korrekt installiert wird. Es kam bei mir dabei 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. [23, S. 61]

Deswegen musste ich für diesen Test den MySQL-Treiber von Oracle verwendet. Nachdem der Treiber korrekt installiert wurde und das Repository erstellt war, kann nun einen Job zu Indexierung der Einträge gestartet werden. In diesem werden die Queries und der Zeitplan konfiguriert. Im ersten Schritt wird das Repository ausgewählt und das Ziel, in diesem Fall also Solr. In dem Tab Queries lassen sich dann diverse Querys bauen. Der erste ist der Seeding Query, welche eine Art Delta-Query für dieses System ist und natürlich der Data-Query, 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 in 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. [23, S. 97] Dies ist für unseren Zweck leider keine

gute Datenstruktur. Sind alle Querys eingetragen, kann die Indexierung beginnen. Dafür wird der Job in der Oberfläche manuell gestartet, insofern kein Zeitplan konfiguriert ist. In meinen 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 ich zu diesem Problem nichts finden. Auch eine Reduktion der Einträge auf nur 125 hat das Problem leider nicht lösen können. Deswegen breche ich an dieser Stelle den Test ab.

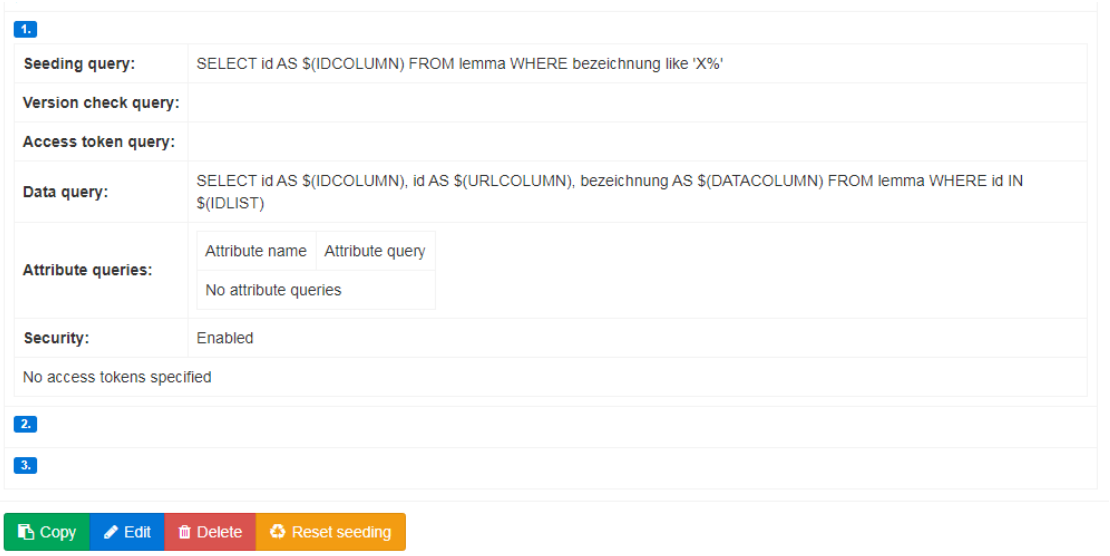


Abb. 4.5. Übersichtsseite des Querys in Datafari.

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 man diverse Einstellungen für die Suchmaschinen, 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. Update können auch hier nicht über die Oberfläche eingespielt werden.

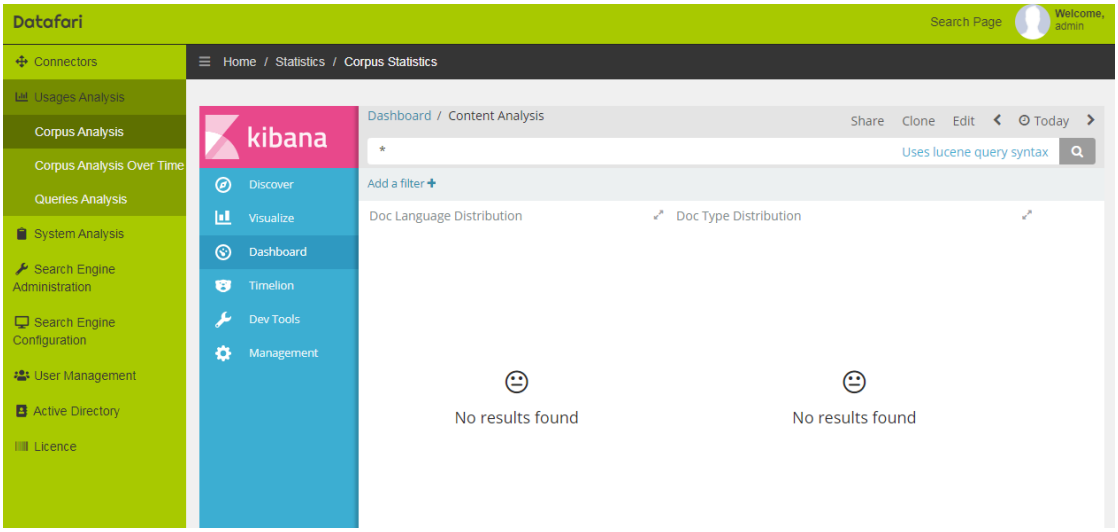


Abb. 4.6. Kibana Integration in Datafari.

4.4.4 Dokumentation

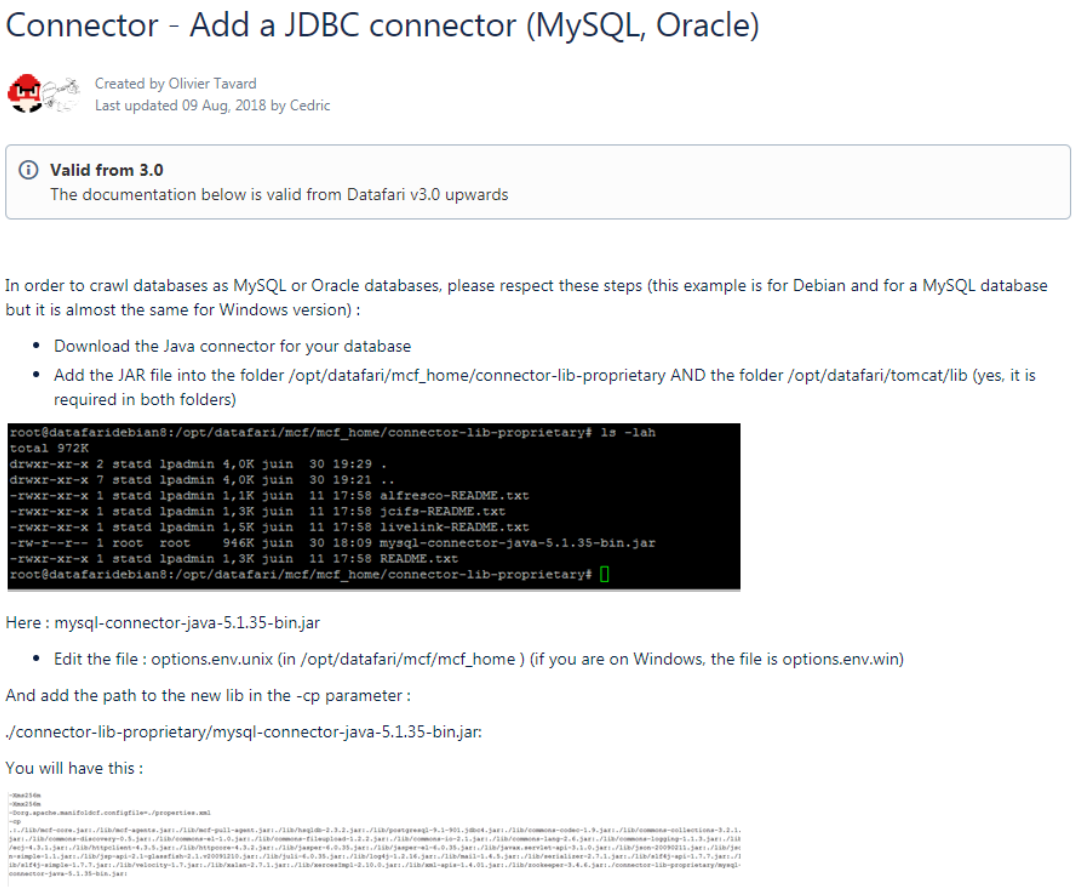
Die Dokumentation geht sehr genau auf die Installation des Systems ein, dabei werden alle Konfigurationsaspekte beleuchtet. Zum Beispiel wird beschrieben, wie die User Limits erhöht werden, oder die JAVA_HOME-Variable korrekt gesetzt wird. Allerdings merkt man an manchen Stellen, dass die Dokumentation nicht von nativen Englischsprechenden geschrieben wurde, da die Grammatik nicht immer stimmt. Allerdings hat dies nie 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 nicht viel 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, musste ich in der Dokumentation von ManifoldCF nachlesen.

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.

4.4.5 Absetzen einer Anfrage und Integration in PHP

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



Connector - Add a JDBC connector (MySQL, Oracle)

Created by Olivier Tavard
Last updated 09 Aug, 2018 by Cedric

Valid from 3.0
The documentation below is valid from Datafari v3.0 upwards

In order to crawl databases as MySQL or Oracle databases, please respect these steps (this example is for Debian and for a MySQL database but it is almost the same for Windows version) :

- Download the Java connector for your database
- Add the JAR file into the folder /opt/datafari/mcf_home/connector-lib-proprietary AND the folder /opt/datafari/tomcat/lib (yes, it is required in both folders)

```
root@datafaridebian8:/opt/datafari/mcf/mcf_home/connector-lib-proprietary# ls -lah
total 972K
drwxr-xr-x 2 statd lpadmin 4,0K juin  30 19:29 .
drwxr-xr-x 7 statd lpadmin 4,0K juin  30 19:21 ..
-rwxr-xr-x 1 statd lpadmin 1,1K juin  11 17:58 alfresco-README.txt
-rwxr-xr-x 1 statd lpadmin 1,3K juin  11 17:58 jcifs-README.txt
-rwxr-xr-x 1 statd lpadmin 1,5K juin  11 17:58 livelink-README.txt
-rw-r--r-- 1 root  root    946K juin  30 18:09 mysql-connector-java-5.1.35-bin.jar
-rwxr-xr-x 1 statd lpadmin 1,3K juin  11 17:58 README.txt
root@datafaridebian8:/opt/datafari/mcf/mcf_home/connector-lib-proprietary#
```

Here : mysql-connector-java-5.1.35-bin.jar

- Edit the file : options.env.unix (in /opt/datafari/mcf/mcf_home) (if you are on Windows, the file is options.env.win)

And add the path to the new lib in the -cp parameter :

./connector-lib-proprietary/mysql-connector-java-5.1.35-bin.jar:

You will have this :

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

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 MySQL-Datenbank 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 gab es bei Logstash ein paar Warnungen beim Start, welche mit JRuby zusammenhingen und den Entwicklern schon bekannt sind. Daher können diese hier ignoriert werden.

Für eine richtige Installation gibt es mehrere Wege, entweder man fügt deren Repository ein, verwendet das bereitgestellte Debian-Paket oder installiert per

Docker. Für diesen Test habe ich mich für das Debian-Paket entschieden. Die Installation verlief dabei ohne weitere Probleme.

4.5.2 Indexierung

Um nun Daten zu indexieren, muss in einer Conf-Datei in Logstash definiert werden, wie und welche Daten gelesen und weitergegeben werden sollten 4.5.2. Dabei kann Logstash direkt MySQL-Querys gegen die Datenbank stellen. Die Datei ist in zwei Blöcke unterteilt. Zum einen ein Input-Block, welcher erklärt, welche Daten eingelesen werden sollen und einen Output Block, welcher das Ziel für die Daten angibt. Für den Input Block verwende ich hier den MariaDB-Treiber.

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 Query 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 Query 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 die 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, welcher für die erhaltenen Daten angelegt wird.

Als die Indexierung nun gestartet wurde, kam es allerdings zu einem Fehler, dass die MySQL-Query nicht gültig sei. Um zu ermitteln, wie viele Daten indiziert werden müssen, wird die Query mit einer Count-Abfrage umhüllt. 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.

```
1   input {
2     jdbc {
3       jdbc_validate_connection => true
4       jdbc_driver_library => ""
5       jdbc_driver_class => "Java::org.mariadb.jdbc.Driver"
6       jdbc_connection_string =>
7           "jdbc:mariadb://localhost:3306/dietrichonline"
8       jdbc_user => "USER"
9       jdbc_password => "PW"
10      tracking_column => "timestamp"
11      use_column_value=>true
12      statement => "MYSQL-Query WHERE timestamp > :sql_last_value"
13      schedule => "0 */6 * * *"
14    }
15  }
16
```

```
17    output {
18      stdout { codec => json_lines }
19      elasticsearch {
20        document_id => "%{id}"
21        index => "lemma"
22        hosts => "localhost:9200"
23      }
24    }
```

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.

Um eigene Feld-Typen zu setzen, 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. Dies ist für IDs interessant, die zur Verwaltung der Indices dienen, allerdings nicht nach außen hin herausgegeben werden sollen.

```
1  PUT lemma
2  {
3    "mappings": {
4      "properties": {
5        "original_bezeichnung": {
6          "type":  "keyword"
7  }}}}
```

Will man allerdings nicht alle Felder von Hand erstellen, 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 ich hier definiert, dass das Feld original_bezeichnung immer nur als Keyword gespeichert wird.

```
1  PUT lemma
2  {
3    "mappings": {
4      "dynamic_templates": [
5        {
6          "obez_as_keyword": {
7            "match":   "original_bezeichnung",
8            "mapping": {
9              "type": "keyword"
10   }}}]}}
```

Für den Test war eine 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.

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. Allerdings ist es möglich 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.

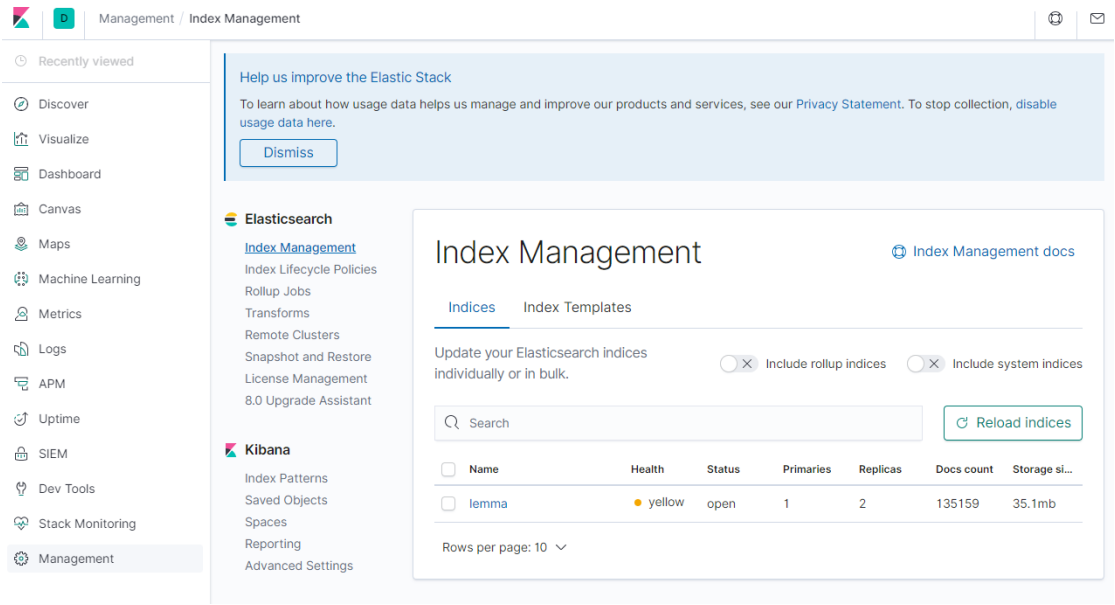


Abb. 4.8. Index Management Seite von ElasticSearch

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 ist mir nur ein Fehler in der Dokumentation aufgefallen, und zwar wurde bei der Informationsseite zum PHP-Klienten eine falsche PHP-Version vermerkt.

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 ein eigener Query vonnöten ist. Deswegen werden hier die Ergebnisse in einer Schleife gezählt.

Auch hier wurde nun der Query 100-Mal ausgeführt, um einen Median Wert zu ermitteln. Dieser lag bei ElasticSearch bei 0.58 Sekunden pro Query und erbrachte 15653 Ergebnisse bei jeden Durchlauf.

4.6 Xapian

4.6.1 Installation

Der empfohlene Installationsweg für Xapian führt über das Paketquelle (PPA) der Entwickler. Nachdem dieses eingefügt wurde, kann Xapian entweder in einer C++ oder Python Variante installiert werden. Damit Xapian auch mit PHP anzusprechen ist, ist es allerdings notwendig den PHP-Connector aus Lizenzgründen selbst zu bauen. Dabei muss vorher der ein Eintrag, der ausweist, dass aus dieser Quelle

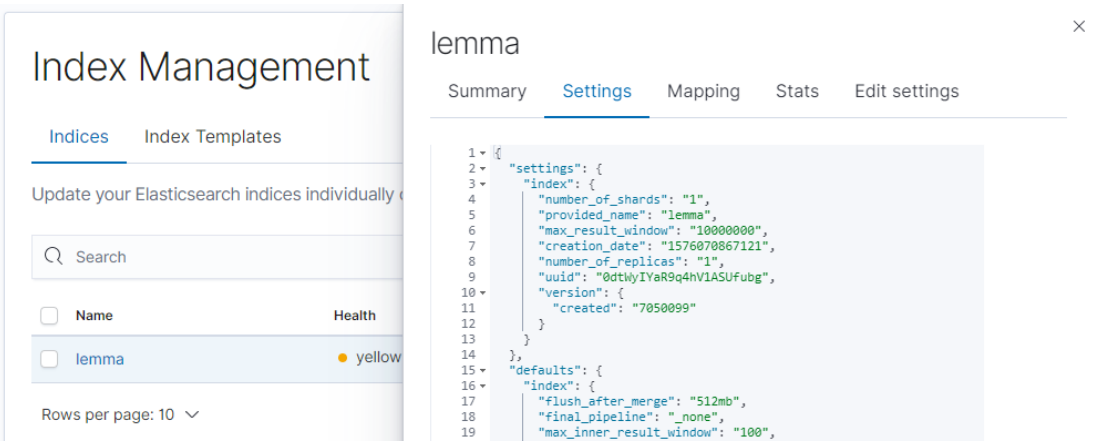


Abb. 4.9. Einstellungen vom Lemma-Index bei ElasticSearch

auch Source-Code geladen werden kann, in der Paketquellen hinzugefügt werden. Nachdem nun der PHP-Klient gebaut worden ist, kann dieser nun normal installiert 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 PPA 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, habe ich zuerst einmal das gegebene Beispiel zur Indexierung von einer CSV Datei durchgearbeitet. Dabei ist mir aufgefallen, dass hier die Indexierung der komplette Handler selbstgeschrieben war. Daraufhin habe ich auf der Basis des Beispiels einen Import für die Daten aus der Datenbank geschrieben.

```
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();
```

Bei dem Skript 4.6.2 möchte ich gerne auf ein paar Zeilen genauer eingehen.

Zuerst in Zeile 10. Der Stemmer dient dazu, wenn der Plural eines Wortes gesucht wird, auch Ergebnisse mit dem 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 dann 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. !!Quelle XAP!!

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

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. Generell würde das Projekt ohnehin ein selbstgebautes Frontend benötigen.

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 nichts in der neuen Dokumentation zu finden. Beim Durchsuchen des Internets, wie der Server extern ansprechbar gemacht werden kann, bin ich zufällig auf eine Seite der alten Dokumentation gestoßen, welche einen 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 ist allerdings nur das Python-Beispiel erklärt.

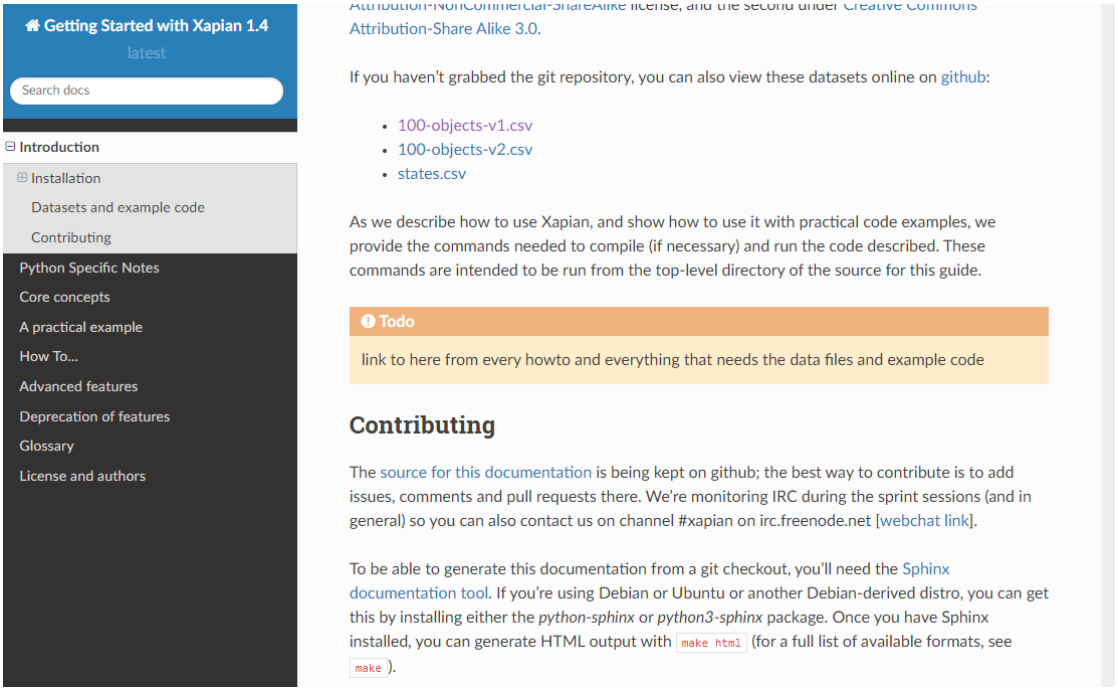


Abb. 4.10. Screenshot von der Xapian-Dokumentation

4.6.5 Absetzen einer Anfrage und Integration in PHP

Xapian besitzt keine REST-Schnittstelle. Daher muss hier per Befehl direkt auf mit Systemaufrufen gearbeitet werden. Ich konnte leider nicht die Remote Ausführung testen, da der Aufwand, um die PHP-Erweiterung auf Windows zu bauen, das Zeitkontingent für den Test überschreitet. 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. Ich habe zuerst versucht mit Wildcards zu arbeiten, allerdings ergab sich dabei dasselbe Problem, 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
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 der Query die ersten hundert Male durchgelaufen war, war die Zeit mit 0.0044 Sekunden im Durchschnitt für die 15661 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 Query genutzt wird. Allerdings habe ich diesen hingenommen, um festzustellen, ob immer alle Ergebnisse korrekt geliefert werden. Die auskommentierte Zeile würde das Objekt nun als Array mit Indices zurückgeben.

Der Query mit dem Abholen der Daten dauerte nun im Durchschnitt 0.22 Sekunden, was immer noch sehr schnell ist. Allerdings muss dabei bedacht werden, dass der Query 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

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. [24]

6.1 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. [24]

6.2 Support der Enterprise Search Engines

Bei den vorhin genannten Enterprise Search Engines 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.

6.3 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. !!GRAFIK!!

7.1 Docker

Docker ist eine Software zur Virtualiserung 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 eingetragen 7.2. Dabei können Einstellungen wie der Container-Name Docker-Compose liest nun diese Datei und erstellt damit die Container und Netzwerke alle 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. [25]

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 wir 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. [26]

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 Konigurationsdatei 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, also die Prozesse des Daten sammeln und an ElasticSearch 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[1] (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.

[1] Mithilfe einer CA kann sich ein Klient gegenüber des Servers ausweisen und umgekehrt.

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. [27]

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 Konfigrationsdateien 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 Benuzter, welche die einzelene Systeme wie Logstash oder Kibana zum funktionieren brauchen generiert.

Auch diese müssen in den Konfigrationsdateien 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 einzelen Systemfunktionen wie die zum Beispiel die Erstellung von Snapshots spezifisch freigegeben werden. Hierbei sollte sich an das minimal Prinzip 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 mit dem Präfix dietrich_ Zugriff erhält. Die Erstellung dieses Nutzer wurde dabei die Benuzter-Oberfläche von Kibana gemacht 7.1.

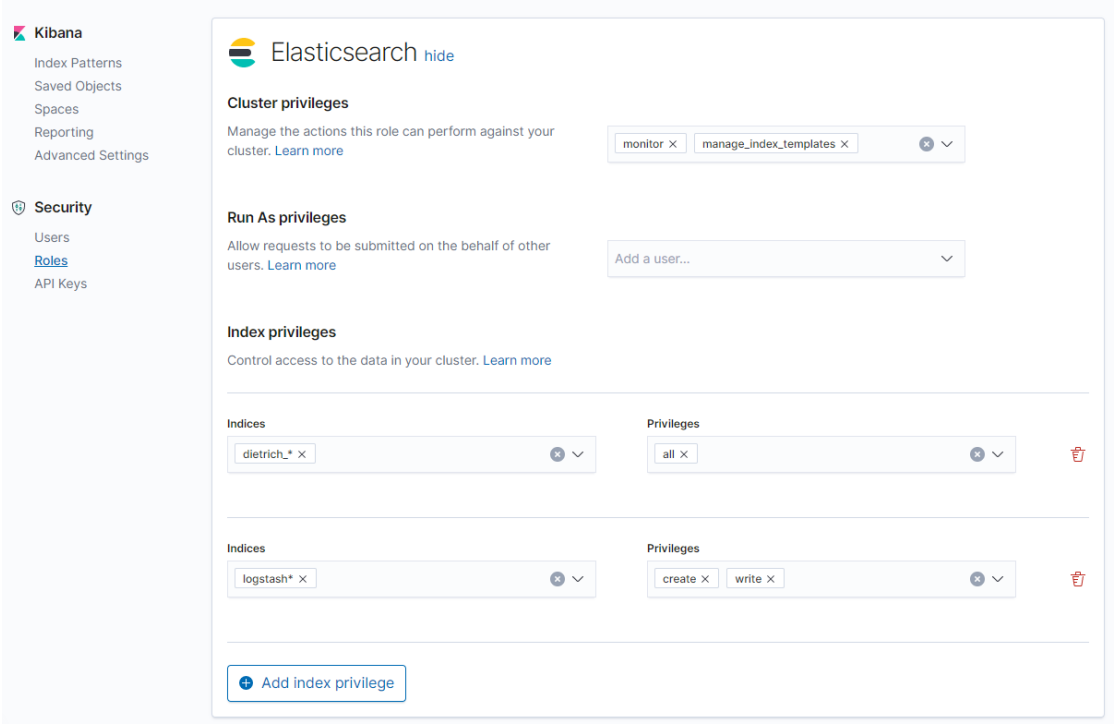


Abb. 7.1. Seite zu Erstellung von Rechte-Rollen

8

Implementation in das Dietrich Projekt

In diesem Kapitel wird die Implementation ins Dietrich-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.

Damit für den nun folgenden Vergleich eine 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 der Lemma-Query, 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-webiste",
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 Artike Tabelle gibt es den Eintrag Trier mit der ID 1, der mit 2 DDC Einträgen, Trier und Rheinland-Pfalz verbunden wird. Bei einen Join wird nun alles in eine flache Hirachie 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 wäre es schön eine Art Array für die DDC Einträge zu haben. 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. Der Aggregat-Filter wird nun also so lange durchlaufen, wie dieselbe ID hintereinander aus der Datenbank kommt.

Dabei ist es wichtig, dass diese Prozess nicht in mehreren Threads ausgeführt wird. Daher erhält jede Pipeline in diesem Projekt auch maximal einen Thread. Hier nun einmal eine 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   [...]
```

Der Wert bstatus_beschreibung wird nun bei jeden Durchlauf überschrieben.

Um nun aber die sich ändernden Werte zu aggregieren, wurde ein Code geschrieben, welche die Einträge in eine Array schreibt 8.1. Da hier nun allerdings mehr als eine Tabelle mit m zu n Beziehungen gejoint wird, ist es notwendig ein paar Überprüfungen auszuführen. Zum einen soll ein Wert nur aggregiert werden, wenn er auch existiert. Sonst sind später Null-Werte in ElasticSerach, die nicht gewünscht sind. Als nächstes kann es aufgrund der Join Struktur dazu kommen,

dass Duplikate eingetragen werden. Dafür schaut eine Schleife immer kurz nach, ob schon ein Wert in dem Array steht, bevor es ihr nochmals hereinschreibt.

8.2 Aufbau des Queries

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

Der ElasticSerach Query besteht allerdings nur aus einer Abfrage, da hier schon alle Daten auf eine flache Ebene gezogen wurden.

Verwendet wurde hier ein sogenannten Boolean-Query. Dieser enthält vier verschieden Untergruppierungen.

Zuerst einmal der Must-Teil. Alle hier Angegeben Parameter müssen in jeden Ergebnis vorhanden sein. Dies ist gleichzustellen mit einen 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 reingerechnet. Für diese Arbeit hat dies erstmal keinen Einfluss, da alle Ergebnisse alphabetisch sotiert werden. Daher ist der Query mit Must gleichzustellen. [28]

```
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-Querys 4.5.5 ein Prefix-Query verwendet wird. Dieser bietet eine sauberere Lösung zum Suchen von Wortanfängen.

8.3 Vergleich

Die oben Beschriebenen Querys wurde jetzt jeweils 100-Mal mit einen Timer laufen gelassen. Dafür wurden die Methodenaufrufe, welche die Querys genieren und ausführen mit dem Timer umschlossen.

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 sieht man, dass ElasticSearch eine Reduktion der Zeit um 58,45 % ermöglicht.

Der Query wurde auch noch in einer nachgebauten Produktionsumgebung ausgeführt. Daran erkennt man, dass die Query durchaus schneller agiert, allerdings dies im Gesamtkontext sich nicht merkbar schneller anfühlt. Da ElasticSearch mit 46,49 Sekunden durchaus schneller lädt als Doctrine mit 50,1 Sekunden, aber die Gesamtzeit aktuell immer noch sehr hoch ist.

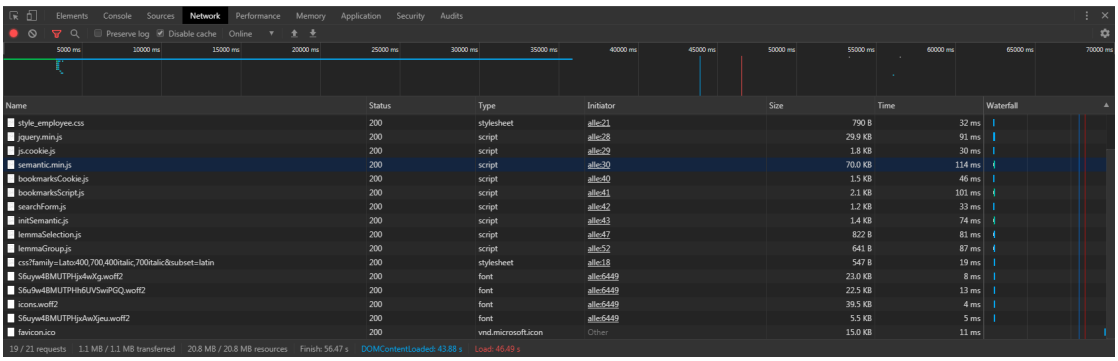


Abb. 8.1. Query-Geschwindigkeit: ElasticSearch

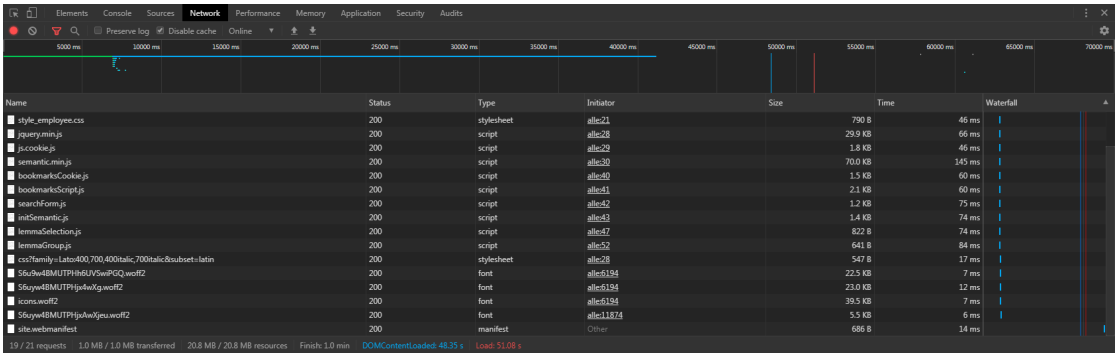


Abb. 8.2. Query-Geschwindigkeit: Doctrine+MySQL

9

Frontend-Suche

Diese Kapitel handelt von der Implementierung ElasticSearchs in das Dietrich-Online Projekt. 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 Suche, welche darauf basiert, welche Autoren bei der vorherigen Suche das meiste geschrieben haben.

9.1 Indexierung

Um die Daten zu indexieren, wurde eine statische Code-Analyse durchgeführt, welche Daten alles im Frontend angezeigt wurden. Dafür wurde der gesamte Code zur Suche untersucht und alle Werte aufgeschrieben. Aufgrund dieser Basis wurde daraufhin ein Query gebaut, welche alle Daten an den Aggregat-Filter weiterreicht. Dabei wurde dieser angepasst und enthält nun auch eine String Konkatenation, welche zuvor bei der Anzeige ausgeführt wurde.

Der Index ist hierbei der aktuell Größte im Dietrich-Online Projekt mit rund 1.4 Millionen Einträgen. Diese Größe des Index beläuft sich auf 2.4 Gigabyte.

Diese Zahl wird sich allerdings verringern lassen, wenn mehr Regeln zu Indexierung eingebaut werden. Aktuell werden alle Spalten mit Text, die unter 256 Zeichen fallen einmal als Keyword und als Volltext indexiert 4.5.2.

Der Query enthielt nun 13 Joins. Diese Datenmenge konnte Logstash nicht verarbeiten und stürzte ab. Der RAM musste auf 4 Gigabyte erhöht werden, damit der Index ordentlich aufgebaut werden konnte.

Damit nun verschiedene Pipelines gleichzeitig arbeiten können, musste eine Datei zur Verwaltung der Pipelines angelegt werden. In dieser wurde definiert, dass von den vier verfügbaren Workern jeweils einer pro Pipeline zur Verfügung steht, damit die Aggregation richtig funktioniert.

Für die Auto-Vervollständigung mussten zudem noch einige 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, dass das Feld zusammengebaut wird 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 erscheint dabei aber immer der vollständige String.

```
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 ElasticSearch weiterzugeben.

9.2 Integration

Die Integration folgte demselben Muster, wie die der Lemma Administration. Ich möchte hier allerdings auf ein paar Unterschiede eingehen.

9.2.1 Paginierung

Die bisherige Paginierung holte sich 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 ein Query abgesetzt, der alle Ergebnisse zählt. Für diesen Fall liefert ElasticSearch einen Count-Query mit. In diesem wird der Index-Name und der entsprechende Query mitgegeben.

Durch diesen wird eine Paginierung generiert. Dafür wird mithilfe der Seitennummer ein Offset für den Query generiert, so dass 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 diesmal eine Query String Suche verwendet, da diese erlaubt, dass Wildcard Symbole zu verwenden. Dadurch geben sich zwar Performanzeinbußen, allerdings sind Wildcard-Suchen ein oft genutztes Element in diesem Projekt und daher erforderlich.

```
1   $subQuery = [
2       'query_string' => [
3           'query' => $userSearchItem->getValue(),
4           'fields' => [
5               "artikel_titel",
```

```
6                "lemma_bezeichnung",
7                [...], //weitere Felder
8                "normlitref_entries.normlitref_kvk_bezeichnung",
9            ],
10           "lenient" => true,
11       ],
12   ];
```

Ein Augenmerk muss auch noch auf "lenient"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 diversen Inhalt gleichzeitig durchsuchen.

9.2.3 Boolesche Logik

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

Bei jeder Suche wird ein Array mit allen Suchanfragen weitergegeben. Das erste Item in 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 ein weiterer Boolean-Query geschrieben, welcher zum einen den zweiten Teil der Suche, sowie die bisherige Suche enthält.

```
1    switch ($userSearchItem->getJunktor()) {
2        case UserSearchItem::JUNKTOR_NO: //First Entry
3            $this->fullQuery = [
4                'bool' => [
5                    'must' => [
6                        $this->addTypeValue($userSearchItem), //Add Search
7                    ],
8                ],
9            ];
10           break;
11       case UserSearchItem:: JUNKTOR_AND: //MUST
12           $this->fullQuery = [
13               'bool' => [
14                   'must' => [
15                       $this->addTypeValue($userSearchItem), //Add Search
16                       $this->fullQuery, //First Part of Query
17                   ],
18               ],
19           ];
20           break;
21   [...] // 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 ein Query an dieses System aussieht. Damit das System weiß, welches Suggestor-Feld verwendet werden soll, wird dieses in der Applikation als Array hinterlegt. Normalerweise wird auch der gesamte Eintrag mitgegeben. Da dies bei dieser Art der Suche nicht erwünscht ist, wird das _source-Feld auf leer gesetzt.

```
1       $params = [
2           'index' => 'dietrich_frontend',
3           'body' => [
4               '_source' => '', //Empty Source since we need only the String
5               'suggest' => [
6                   'auto_complete' => [
7                       'prefix' => $matchAgainst,
8                       'completion' => [
9                           'field' => SEA::AUTOCOMPLETE_COLUMNS[$categoryIndex],
10                          'size' => $maxMatches,
11                          'skip_duplicates' => true,
12      ]]]]];
```

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 Autovervollständigungs-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. Der Query wird dafür um einen Parameter erweitert 9.2.6.

```
1       'aggs' => [
2           'best_authors' => [
3               'terms' => [
4                   'field' => 'artikel_autor.keyword',
5       ]]],
```

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.

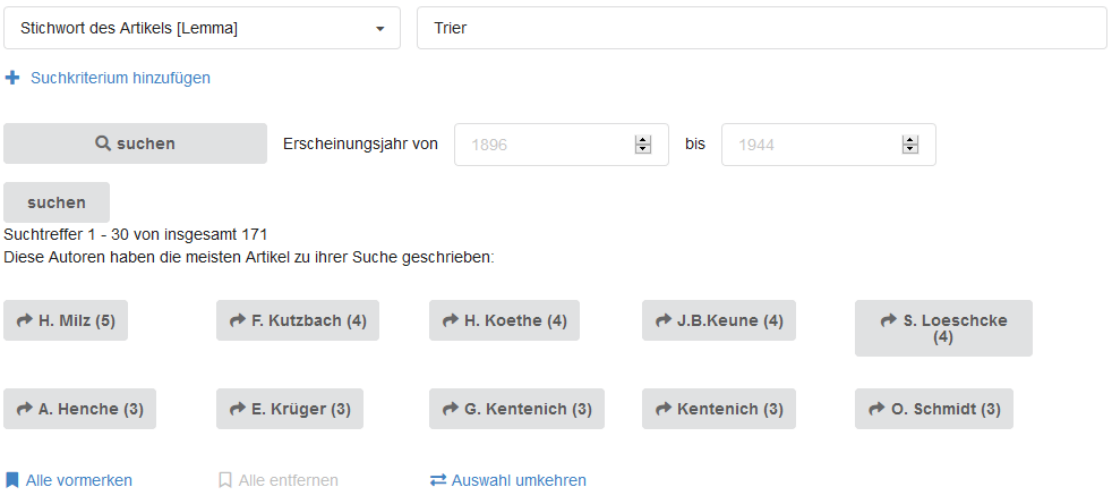


Abb. 9.1. Abbildung der erweiterten Suche

10

Zusammenfassung und Ausblick

Diese Bachelorarbeit hat sich ausführlich mit Enterprise Suchmaschinen auseinandergesetzt, diese Verglichen und letztendlich eine in das Dietrich Online Projekt implementiert. Das Ziel dabei war es eine geeignete Suchmaschine für das Dietrich Online 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 Suchmaschine 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 Dietrich Online 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.

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 diesem Query, 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 eher simplen 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 zu 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 mitgereicht 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. Iqubal Mustafa Kaki, “Solr indexing - mariadb table data into apache solr,” 2016. [Online]. Available: https://erimkaki.blogspot.com/2016/01/solr-indexing-mariadb-table-data.html [Accessed: 26.11.2019]
23. 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]
24. “Oai-schnittstelle,” 31.05.2019. [Online]. Available: https://www.dnb.de/DE/Professionell/Metadatendienste/Datenbezug/OAI/oai_node.html [Accessed: 25.10.2019]
25. 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]
26. “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]
27. 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]
28. “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

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