FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Descriptive study and experimental analysis of the ELK stack applicability for Big Data use cases

Ömer Uludağ

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Descriptive study and experimental analysis of the ELK stack applicability for Big Data use cases

Deskriptive Studie und experimentelle Analyse der ELK Stack-Anwendbarkeit für Big Data-Anwendungsfälle

Author: Ömer Uludağ

Supervisor: Thomas Reschenhofer, M. Sc.
Fakultät für Informatik
Technische Universität München

Examiner: Prof. Dr. Florian Matthes
Fakultät für Informatik
Technische Universität München

Date: April 15, 2016

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, April 15, 2016 Ömer Uludağ

Abstract

Nowadays, enterprises have recently entered a new era which is characterized by the exploration of Big Data to uncover yet unreachable knowledge. Big Data concerns massive data sets in the range of exabytes — highly variable, complex, and growing data sets from multiple data sources with difficulties of storing, analyzing, and visualizing for further processes. With the rapid development of networking, data storage, and data collection capacity, Big Data is now swiftly expanding in various domains, including manufacturing, healthcare, and retail, by the increasing volume and detail of information captured by companies, the rise of social media, and the Internet of Things. The process of researching massive amounts of data reveals hidden patterns and enables companies to gain richer and deeper insights into invaluable information. The result of Big Data analytics underpins new waves of productivity growth, innovation, and becomes a key basis for competition.
However, a vast number of Big Data technology vendors has emerged providing technologies to support companies for harnessing the intended value of Big Data. Within the huge Big Data vendor landscape, companies seek end-to-end solutions that can store, analyze, and visualize mass data quickly and reliably. The open-source Elasticsearch, Logstash, and Kibana (ELK) stack developed by Elastic is a search-based data discovery tool that provides a promising set of tools that can be used for near real-time analytics and fast full-text searches. It helps to glean actionable insights from almost any type of structured and unstructured data from almost any data source and faces the daunting task of harnessing the intended value of Big Data. Since the challenges in Big Data technology selection are non-trivial, evaluating the applicability of the ELK stack for different Big Data use cases seems inevitable.
The goal of this master's thesis is to assess the applicability of the ELK stack for various Big Data use cases. For that reason, this work aims to juxtapose the ELK stack in opposition to related search-based data discovery tools and to crystallize out its key features and capabilities by conducting a structured literature review and a descriptive study. Within the scope of four experiments, the ELK stack is implemented and its performance is assessed by various performance benchmarks. Based on the results of the experiments, distinct characteristics of the ELK stack are elicited. These characteristics indicate strengths and weaknesses of the ELK stack and provide guidance for better decision making in Big Data technology adoption.

Contents

Outline of the Thesis

Part I: Introduction

CHAPTER 1: INTRODUCTION
This chapter starts by revealing the motivation for assessing the applicability of the ELK Stack for Big Data use cases. Subsequently, it continues with stating the objectives of the thesis and the underlying research approach for achieving these objectives.

Part II: Foundations and Related Work

CHAPTER 2: FOUNDATIONS
The theoretical foundations cover the elaboration of search-based discovery tools and the presentation of the four types of analytics capability.
CHAPTER 3: RELATED WORK
This chapter highlights key findings on related work, describes various implementations of the ELK stack in research, and unveils the limitations of related work.

Part III: Descriptive Study

CHAPTER 4: RELATED TECHNOLOGIES
This chapter reveals related technologies of Elasticsearch and the ELK stack. Furthermore, Elasticsearch and related search engines are juxtaposed in opposition by comparing their system properties.
CHAPTER 5: ELK STACK
This chapter crystallizes out the key features of the ELK stack.

Part IV: Experiments

CHAPTER 6: EXPERIMENTS
This chapter assess the applicability of the ELK stack for Big Data use cases by preselected implementation and performance benchmark experiments.

Part V: Conclusion

CHAPTER 7: CONCLUSION AND OUTLOOK
This chapter summarizes the thesis and the results of the applicability assessment of the ELK stack. Additionally, limitations of the conducted research are delineated and a brief outlook for further investigations is provided.

Part I.

Introduction

1. Introduction

This chapter describes the motivation for the thesis by revealing the emerging Big Data phenomenon and the necessity of evaluating the Elasticsearch, Logstash, and Kibana (ELK) stack as a search-based data discovery tool for Big Data use cases (see Section 1.1). Following this, the objectives and the corresponding research questions of the thesis are highlighted in Section 1.2. The succeeding Section 1.3 delineates the underlying research approach of the thesis.

1.1. Introduction

Many of the most extraordinary inventions throughout human history, from natural languages to modern computer systems, were those that facilitated people to better generate, gather, and consume data and information. In the recent years, the world has witnessed explosive growth of data. This *Big Data* has obtained critical mass in every domain, and the brisk development and diffusion of digital information technologies have intensified its rise. With the passage of time, Big Data may become a new kind of corporate asset that will cut across business units, representing a key basis for competitive edge. Companies have to perceive this phenomenon by excogitating Big Data's potential and threats [74]. Big Data may improve the productivity and competitiveness of companies and create huge benefits for consumers. For instance, by the creative and effective utilization of Big Data for improving efficiency and quality, the potential value of the U.S. medical sector gained through data may exceed $300 billion; retailers that fully use Big Data may enhance their profit by more than 60% [16]. Today, data has become a vital factor of production that can be compared with intrinsic assets and human capital. As multimedia, social media, and IoT are growing, companies will collect more information, leading to an exponential growth of data volume [74]. [79] estimates that there will be 26 billion connected devices communicating together to create the IoT by 2020. While the amount of large datasets is rigorously ascending, it also gives rise to many challenging problems demanding prompt solutions, e.g., collecting and integrating massive data from various distributed data sources, surpassing the capacities of IT architectures and infrastructure of existing enterprises in terms of data volume, or providing findings in real-time. This swiftly growing data causes a problem of storing and managing huge heterogeneous datasets with moderate requirements on hardware and software infrastructures [74]. In consideration of the aforementioned challenges, the ability to process and analyze this data and to extract insight of knowledge that enables intelligent services is a critical capability. The 5 V's: volume, velocity, variety, veracity, and value are often used to describe the requirements of Big Data applications

and the characteristics of Big Data [73].
Volume: The word *"Big"* in Big Data itself defines the volume. Automotive industry sources estimates that more than 480 TB of data was collected by every automotive manufacturer in 2013. It is expected that this size will increase to 11.1 PB per year in 2020 [55]. This amount of data is difficult to be handled using existing traditional systems, since it requires distributed and parallel processing that most traditional DBMS were not designed for. *Velocity:* Data is created at an increasing speed. Velocity requires real-time processing of raw fast data to facilitate real-time decision making and risk management. This characteristic is limited to the speed at which the data flows. For instance, data from sensor devices is constantly moving to the database store and this amount is not small enough. Traditional systems are not sufficiently capable of performing the analytics on the data which are constantly in motion. *Variety:* Heterogeneous data — structured, semi-structured, and unstructured — comes in from various sources and not only includes the traditional data but also the semi-structured data from various resources like web pages, log files, social media, documents, and sensor devices. All this data is totally different and is difficult to handle by existing traditional analytic systems. *Veracity:* Due to increasing amount of data and various data sources, the probability of inconsistency and abnormality in the data is growing. Thus, the trustworthiness of the data becomes questionable, which is a major challenging factor of Big Data. *Value:* The value of Big Data is described as the extent to which Big Data generates economically worthy insights and benefits through actionable insights and business decisions. Extracting value from Big Data requires capabilities beyond traditional data warehouse and OLAP technologies [61, 80, 15, 22, 99, 118].
The swift proliferation and development of open source and commercial Big Data technologies have added a new dimension of complexity to Big Data system development [15]. [90] lists more than 350 tools in the Big Data ecosystem. Furthermore, [90] categorizes the plenty of Big Data tools into 21 layers, which include technologies for distributed coordination, security & privacy, monitoring, workflow-orchestration, and application & analytics, to name a few. Each technology has many vendors and products. For that reason, selecting an eligible Big Data tool for extracting meaningful and actionable insights from data is challenging — the variety of existing tools is very large [73]. The classification by [90] reveals that not all Big Data technologies are directly designed for analyzing data. Accordingly, a preselection of the application and analytics layer provided by [90] is an appropriate starting point for selecting Big Data and analytics technologies. Search-based data discovery tools facilitate users to develop and refine views and analyze multi-structured data using search terms to find relationships across structured, unstructured, and semi-structured data. They also have a proprietary data structure to store, model, and correlate structured and unstructured data minimizing reliance on predefined meta data. Additionally, search-based data discovery tools feature a performance layer to lessen the need for aggregates and pre-calculations. According to [11], search-based data discovery tools raise huge expectations and promise high benefits for organizations among Big Data and analytics technologies. [11, 96] list some sample vendors for search-based data discovery tools, including Attivio, IBM, Or-

acle, Splunk, and ThoughtSpot. The combination of the three open source projects Elasticsearch[1], Logstash[2], and Kibana[3], also known as the ELK stack, developed by Elastic (formerly, Elasticsearch Inc.)[4], is an outstanding alternative to commercial search-based data discovery tools.
The ELK stack is an end-to-end stack that gleans actionable insights in near real-time from almost any type of structured and unstructured data source. Deep search and data analytics are performed by Elasticsearch, whereas Logstash is responsible for centralized logging, log enrichment, and parsing log files. Kibana is used to visualize data from Elasticsearch [30, 27]. Due to the fact that the ELK stack is used by many organizations for a variety of business critical functions [30], an evaluation of its applicability for Big Data use cases seems auspicious and indispensable.

1.2. Objectives

Resulting from the aforementioned motivation in Section 1.1, this thesis aims to describe the ELK stack and to assess its applicability for Big Data use cases. With conceptualizing this research topic, two objectives are derived (see Figure 1.1).

1. **Objective 1**: aims to extract the capabilities of both individual technologies and the corresponding ELK stack.
2. **Objective 2**: targets to juxtapose the capabilities of the ELK stack in opposition to various Big Data use case experiments.

Based on the denoted objectives, two research questions are deduced:

1. **Research question 1 (RQ1)**: What are capabilities and key features of the ELK stack?
2. **Research question 2 (RQ2)**: For which type of Big Data use cases is the ELK stack applicable?

In order to assure the rigor and relevance of this thesis, a research approach is developed (see Section 1.3).

1.3. Approach

This thesis' research approach comprises two research questions, which were introduced in Section 1.2, and appropriate research design methodologies for answering them. Figure 1.2 illustrates the research approach of this thesis. Firstly, the first research question, namely *RQ1*, is answered by using the research design methodologies

[1]http://www.elastic.co/products/elasticsearch
[2]http://www.elastic.co/products/logstash
[3]http://www.elastic.co/products/kibana
[4]http://www.elastic.co/

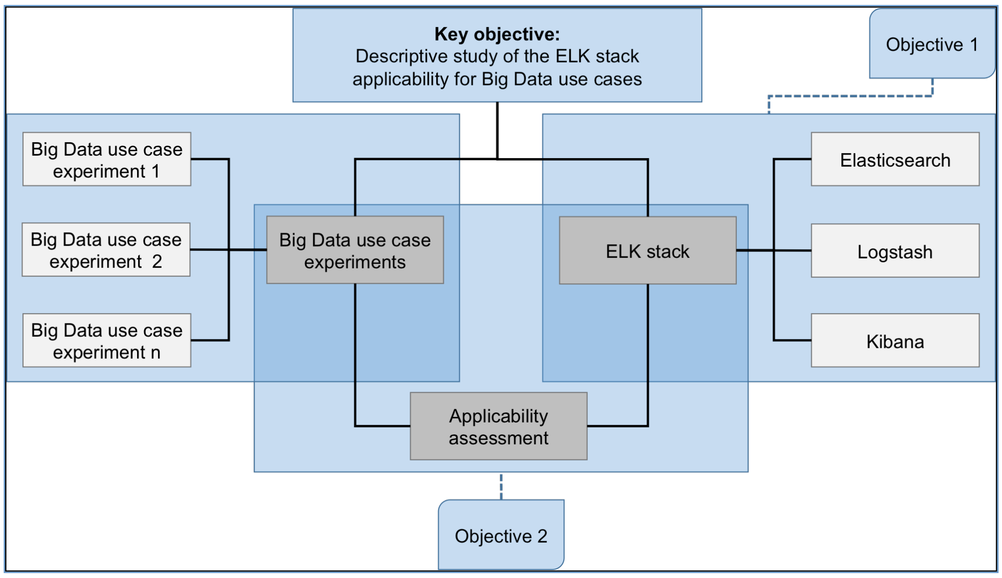

Key objective:
Descriptive study of the ELK stack applicability for Big Data use cases
Objective 1
Big Data use case experiment 1
Big Data use case experiment 2
Big Data use case experiment n
Big Data use case experiments
ELK stack
Elasticsearch
Logstash
Kibana
Applicability assessment
Objective 2

Figure 1.1.: Conceptualization of research topic

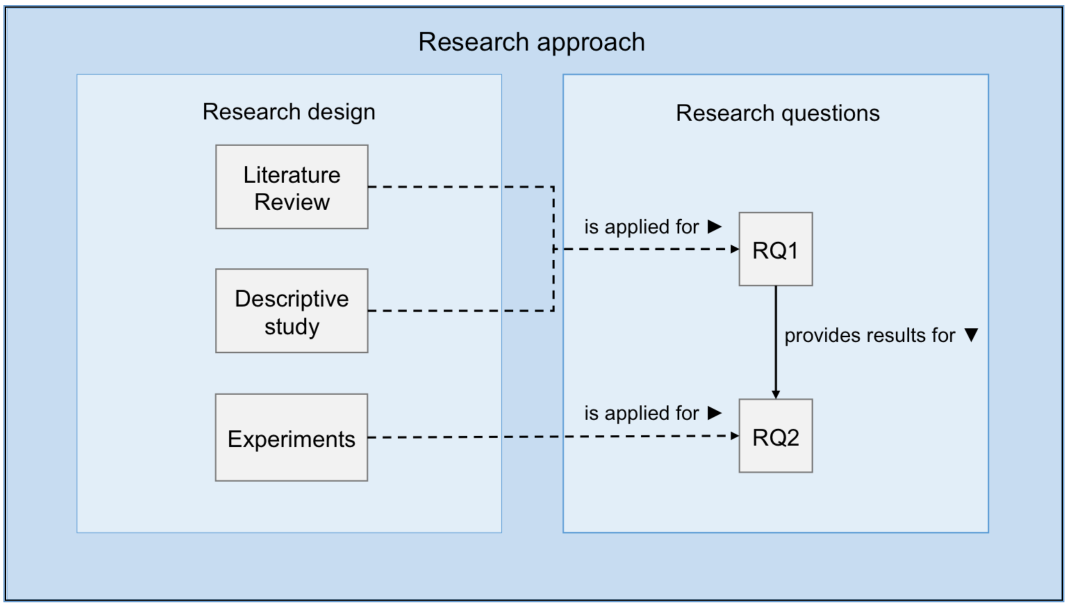


Figure 1.2.: Overview of underlying research approach

literature review and *descriptive study*. According to [8], reviewing the existing literature relating to a topic is an essential first step and basis when undertaking a research project. A literature review attempts to uncover relevant sources to a topic under study and, thereby, makes a vital contribution to the relevance and rigor of research [116]. This thesis does not aim to conduct a systematic literature review, e.g., [62], since it requires considerably more effort than traditional literature review, which is sufficient for this work. However, [116] provides a framework for literature review, which consists of five distinct phases, namely phase *I*: definition of review scope, phase *II*: conceptualization of topic, phase *III*: literature search, phase *IV*: literature analysis and synthesis, and phase *V*: research agenda. This framework is used for *RQ1* for identifying related works, related technologies, and related information about the ELK stack.

By stating the key objective of this thesis and creating a conceptualization of the research topic, phase *I* and *II* are accomplished. In phase *III*, a literature research was conducted in terms of querying the scholarly online search catalogues: SpringerLink[5], ScienceDirect[6], IEEE Xplore Digital Library[7], Code4Lib Journal[8], ACM Digital Library[9] and Emerald Insight[10] as well as on Google Scholar and the online search of the library of the Technische Universität München using the search term: *"Elasticsearch" OR "Logstash" OR "Kibana" OR "ELK stack"*. The use of the *OR* operator is meaningful, since the majority of academic works that concentrate on the whole ELK stack do not provide much detailed information on its individual technologies. Additionally, it prevents missing important contributions if one of the mentioned technologies should not be available in the search query. Furthermore, the method of concentric circles was performed on the basis of relevant sources. In total, the search resulted in 222 *potentially* relevant classified information sources. Initially in phase *IV*, the literature was synthesized by using inclusion and exclusion criteria. The inclusion criteria of identified sources are that the source must belong either to the group of academic works, e.g., published paper, master's, bachelor, or PhD thesis, or to the group of market research application presentations. However, blog entries and advertisements were excluded for subsequent analyses due to the limited ability to verify a non-biased view and interests contrary to this research. This led to a total distinct number of 214 *relevant* information sources. The results of the analyzed remainder literature can be found as related work in Chapter 3, as related technologies in Chapter 4, and as related information or key features of the ELK stack in Chapter 5. Last but not least, this thesis utilizes the research agenda in phase *V* for revealing the necessity of investigating the applicability of the ELK stack for Big Data use cases.

Based on the literature review, a descriptive study is realized for the second part of the research question *RQ1*. According to [53], the purpose of a descriptive study is to examine a phenomenon as it naturally occurs, rather than studying the impacts of the phenomenon or intervention. Descriptive studies typically examine questions of a univariate, normative, or correlative nature, e.g., describing only one variable, comparing the variable to a particular standard, or summarizing the relationship between two or more variables. Furthermore, descriptive studies may ask 'what' questions [46], e.g., *what are capabilities and key features of the ELK stack?* This research design methodology

[5]http://link.springer.com/
[6]http://www.sciencedirect.com/
[7]http://ieeexplore.ieee.org/
[8]http://journal.code4lib.org/
[9]http://dl.acm.org/
[10]http://www.emeraldinsight.com/

is appropriate for corroborating related information about the ELK stack, describing the realization of the ELK stack in its natural environment, and comparing with related technologies. Thus, the key capabilities and features of the ELK stack can be crystallized out (see Chapter 5).
For answering research question *RQ2*, the research design methodology *experiment* is applied. However, it is not utilized in its classical manner in which one or more variables are controlled, measured, or changed. In this thesis, this methodology aims to demonstrate the characteristics of the object of investigation in its potential usage environment. This methodology is valuable for exploring and describing observed characteristics of the ELK stack and to assess its applicability for Big Data use cases. The conducted experiments for assessing the ELK stack's applicability for Big Data use cases are revealed in Chapter 6. In the following chapter, the foundations of this thesis are presented.

Part II.

Foundations and Related Work

2. Foundations

Section 2.1 provides a fundamental terminology of this thesis. In addition to the understanding of search-based data discovery tools, Section 2.2 presents the four types of data analytics capability in order to establish a unified understanding, which is of big importance for interpreting the results of the experiments in Chapter 6.

2.1. Search-Based Data Discovery Tools

The term *search-based data discovery tool* was originally introduced by the U.S. based IT consultancy Gartner[11]. Gartner defines search-based data discovery tools as tools that facilitate users to develop and refine views and analyses of structured and unstructured data applying search terms. Search-based data discovery tools have, like visualization-driven data discovery tools, the following three attributes:

1. a proprietary data structure to store and model data collected from various sources, which minimizes reliance on predefined business intelligence meta data,
2. a built-in performance layer using random-access-memory or indexing that diminishes the demand for aggregates, summaries, and pre-calculations, and
3. an intuitive user interface, enabling users to explore data without much training.

However, search-based data discovery tools differ in two aspects from visualization-driven data discovery tools. First, search-based data discovery tools have a broader scope than visualization-driven data discovery tools, which focus exclusively on quantitative data. Second, search-based data discovery tools use text search inputs and results to guide users to the required information [42]. Figure 2.1 represents the hype cycle for business intelligence and data analytics that demonstrates the importance of search-based data discovery tools for organizations. According to the hype cycle, search-based discovery tools are on the rise and act as innovation triggers. They are expected to reach their plateau within two to five years of productivity, where the rapid growth phase of their adoptions will begin. Gartner classifies search-based data discovery tools as *high-benefit technologies* within its priority matrix for business intelligence and data analytics. This means that search-based data discovery tools are less likely to change an organization's business model, but have a significant impact on the organization's business intelligence and data analytics program. Gartner also expects that these tools will provide high benefits during the next two to five years [96]. Also among

[11]http://www.gartner.com/

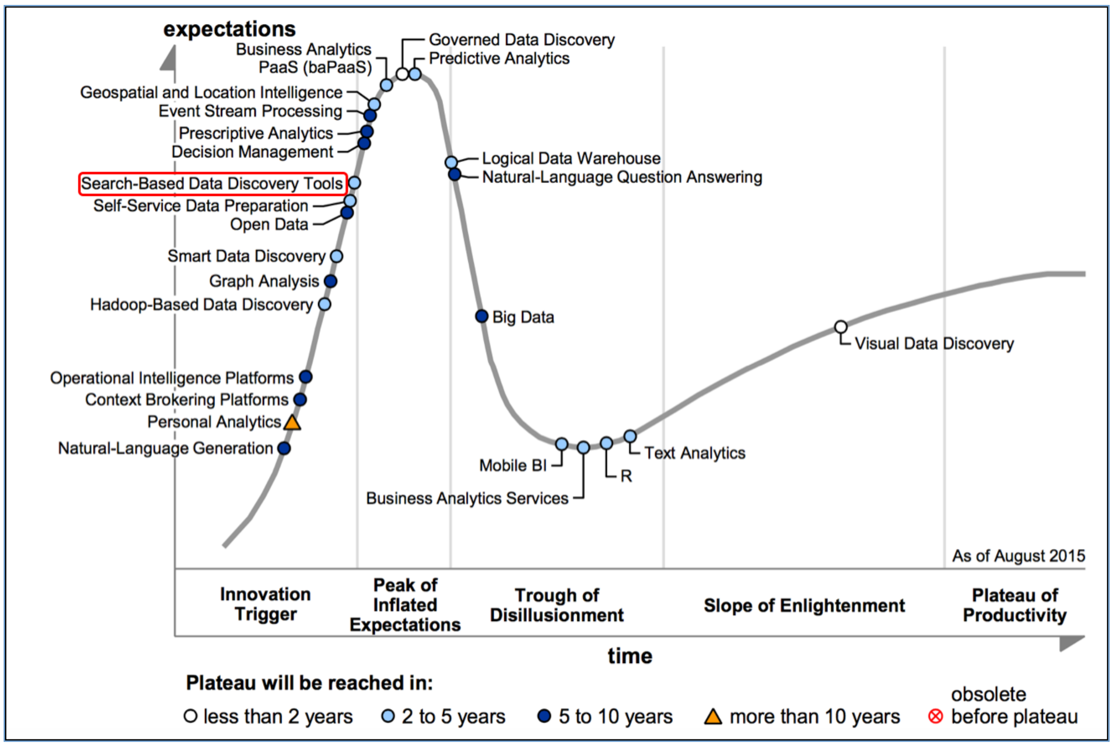


Figure 2.1.: Hype cycle for business intelligence and data analytics 2015 [96]

Big Data technologies, search-based discovery tools are classified as highly beneficial for organizations [11]. This is justified by their capability to analyze greater volumes and variety of data with increased sophistication, which means that they apply to the challenges of the Big Data era [95]. Most vendors of search-based data discovery tools provide keyword search to initiate queries and exploration. Furthermore, some emerging vendors are offering an natural-language query interface where a user initiates a query or asks a question in a search interface using natural language rather than specific keywords and Boolean logic. Sample Vendors of search-based data discovery tools are Attivio, HP Autonomy, IBM, Oracle, Splunk, and ThoughtSpot, to name a few [96]. Besides commercial search-based data discovery tools, there are also open source solutions available, e.g., the ELK stack.

2.2. Four Types of Data Analytics Capability

This thesis utilizes the data analytics definition provided by [56] that defines data analytics as *"the scientific process of transforming data into insight for making better decisions."* Based on this definition, data analytics is distinguished in four sub-categories: *Descriptive Analytics*, *Diagnostic Analytics*, *Predictive Analytics*, and *Prescriptive Analytics*. The first is about providing tools in order to look at the data, e.g., aggregating, querying and drilling down data. In particular, it includes business intelligence. Descriptive analytics helps to understand what happened so far. Diagnostic analytics includes OLAP, interactive visualization, and descriptive modeling. It helps to understand why things happened. Predictive analytics comprises statistics, machine learning, data mining, and predictive modeling. It helps to understand what is most likely to happen next. The last is about making decisions or recommending actions. It includes decision/-mathematical modeling, simulation, and optimization. Prescriptive analytics helps to decide the implementation of next actions. The business value provided by data analytics increases as the understanding of the data and the capability of data analytics are increasing [13, 87]. Figure 2.2 provides an overview of the sub-categories of data

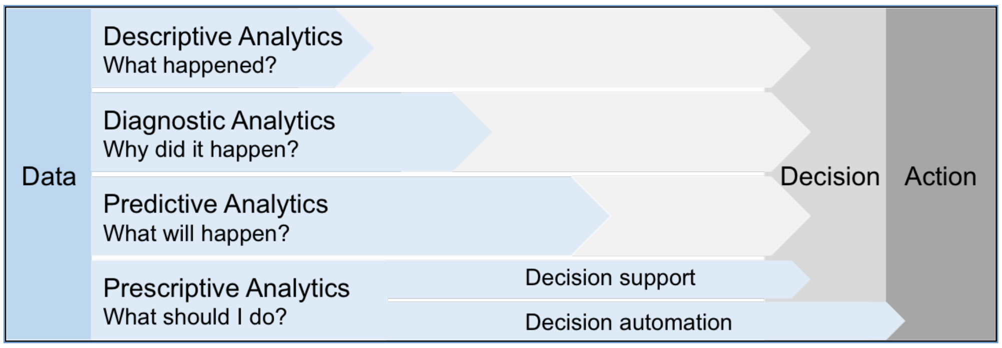


Figure 2.2.: Four types of data analytics capability [13]

analytics.

3. Related Work

This chapter comprises key findings of the literature review in Section 3.1, insights into the ELK stack in research in Section 3.2, and research gaps of the related work in regard to the underlying key objective in Section 3.3.

3.1. Key Findings of Literature Review

During the literature analysis and synthesis phase of the literature review, 214 distinct information sources were analyzed. Table 3.1 provides an overview of the distinct literature source types. Papers represent the majority of the relevant information sources

Literature source type	
Characteristic	**Count**
Paper	133
Master's thesis	22
Project study/Project report	18
Book	16
Presentation	13
Bachelor thesis	7
PhD thesis	4
Survey	1
Total	214

Table 3.1.: Overview of literature source type distribution

with a count of 133. Papers are followed by master's theses with a number of 22. Furthermore, 18 information sources are project studies or projects reports, 16 information sources are books, and 13 information sources are presentations. Bachelor theses, PhD theses, and surveys form only a small part of the relevant information sources.
Table 3.2 shows that only 23 information sources address the whole ELK stack. Furthermore, 22 information sources address all the individual components of the ELK stack without naming the ELK stack itself, whereas 169 information sources do not mention the ELK stack or all of its individual components. The greater part of the information sources address Elasticsearch, totally 131 sources. Only a fractional part address Logstash, namely 13, or Kibana with a count of 1. The results of Table 3.1 and Table 3.2 are not very surprising. Since February 2010, Elasticsearch's first release, it has rapidly gained popularity [91]. This fact is also true for the scientific community by considering

Mention of the ELK stack	
Characteristic	**Count**
Yes	23
No, total	169
-No, but only Elasticsearch addressed	131
-No, but only Logstash addressed	13
-No, but only Kibana addressed	1
No, but all addressed	22
Total	214

Table 3.2.: Overview of literature source distribution in relation to the ELK stack

the amount of available papers, bachelor, master's, and PhD theses shown in Table 3.1. Elasticsearch, which constitutes the essential component of the ELK stack, has gained more attention in scientific research than Logstash and Kibana combined, as shown in Table 3.2. A plausible reason for this fact is that Elasticsearch can be used in highly diverse use cases and can be complemented with many other technologies, which is the reason why it is so popular. For example, [109] uses CouchDB and Elasticsearch for querying graphical music documents. Herein, Elasticsearch indexes data stored in CouchDB and acts primarily as a search engine for highlighting relevant music documents. Another example is provided by [121], which merges gene object documents in MongoDB and then indexes them with Elasticsearch in order to make them searchable. Generally, relevant books provide deep insights into the power of Elasticsearch and its features and capabilities. Since Elasticsearch is a huge topic and its features are multifarious, exploring it in depth would fill up several books [110]. For example, [45] describes the features of Elasticsearch within 724 pages in depth. Other examples are [67], [91], and [84], which characterize Elasticsearch in detail. While books focus more on its features, in general, scientific works compare Elasticsearch's performance with other search engines like Xapian[12] or Apache Solr[13] (hereafter Solr) and NoSQL databases like HBase[14] or MongoDB[15]. For example, [47] compares Elasticsearch with Xapian and Solr and assesses its applicability for a digital library software suite. Another example is [2], which juxtaposes Elasticsearch in opposition with other NoSQL databases like HBase or MongoDB and assesses its performance with the use of KPIs like loading time of records, execution time for workloads, or overall execution times. Scientific works also describe Elasticsearch's functionality within a given usage scenario. For instance, [115] delineates the role and functionality of Elasticsearch, which acts here primarily as a database, within an IoT platform.

In contrast to Elasticsearch, the ELK stack is not very popular yet. This circumstance is also reflected by the amount of available information sources, as shown in Table 3.2.

[12]http://xapian.org/
[13]http://lucene.apache.org/solr/
[14]http://hbase.apache.org/
[15]http://www.mongodb.org/

Out of 16 books, only [17] describes the ELK stack in depth. Additionally, there are only in total 23 information sources which addresses the whole ELK stack. Primarily, scientific works describe the distinct components of the ELK stack and illustrate the applicability of the ELK stack for log analysis and log monitoring use cases. For example, [52] describes how big log files, created by a PostgreSQL[16] server, can be analyzed by the ELK stack in near real-time. One more example provides [113], which uses the ELK stack for collecting and reporting technical security metrics. These two papers also show that the denotation *"ELK stack"* do not have to be necessarily mentioned, despite the fact that all its components are described.
Last but not least, Table 3.3 depicts the information source's level of detail of describing the ELK stack or its components. Only 28 information sources elaborated the examination object in detail. The majority of the information sources are too superficial, e.g., [44], [59], or [75], which are therefore not utilizable for the descriptive study.

Information source level of detail	
Characteristic	**Count**
Detailed	28
More detailed	36
Not detailed	150
Total	214

Table 3.3.: Overview of information source distribution in relation to level of detail

3.2. ELK Stack in Research

In this section, related scientific works on the ELK stack and its implementations are presented:

- [64] uses the ELK stack within the context of cyber attack detection. Herein, Logstash automatically reads log files of iptables, Suricata[17], and syslog. Afterwards, it ships them to Elasticsearch. Elasticsearch stores and indexes the parsed log files. Kibana visualizes information from iptables, Suricata, and syslog events given by Elasticsearch. Thus, the ELK stack provides valuable information from captured cyber attack data.

- [5] utilizes the ELK stack for collecting and searching logs created by a distributed hierarchical storage management system. Messages are sent from the source nodes of this system to the Logstash-forwarder[18]. The Logstash-forwarder transports the log events to Logstash, which are then processed and tokenized. In a

[16]http://www.postgresql.org/
[17]http://suricata-ids.org/
[18]The Logstash-forwarder is a lightweight client and server, which is responsible for sending messages to Logstash over the network [111].

next step the log events are forwarded to Elasticsearch. Kibana's custom dashboards are used for visualizing volume of messages of the system for a given time frame, displaying number of client connections, or presenting the average wait time for requests.

- [125] takes advantage of the ELK stack for monitoring and analyzing EZproxy[19] logs in near real-time. In this setup, the Logstash-forwarder works with an EZproxy server and monitors any changes of log files on the server. Subsequently, it sends new log entries to the monitoring server. On the monitoring server, Logstash listens for new log events from the Logstash-forwarder. New log events are then sent by Logstash to Elasticsearch. Kibana is used to visualize the indexed logs and presents them to server administrators.

- [6] operates the ELK stack as a monitoring system for inspecting site activities both in terms of IaaS and applications running on hosted virtual instances. Accounting information of IaaS and applications are stored in MySQL[20] databases. Logstash listens to the MySQL databases for new accounting information which are sent afterwards to Elasticsearch for indexing. Kibana dashboards with predefined queries are created for monitoring relevant accounting information, e.g., displaying queries per user, showing the number of workers requested by each user, or presenting CPU efficiency.

3.3. Limitations of Related Work

The synthesized information sources provide a stable foundation for accomplishing the descriptive study of the ELK stack. Consequently, the key features and capabilities of the ELK stack and its individual technologies can be deduced. On the one hand, books provide deep insights into the prowess of the ELK stack, whereas on the other hand, scientific works contain in fact less detailed information about its features but in turn illustrate highly diverse use cases of the ELK stack.
However, the literature synthesis has revealed the following limitations:

- None of the analyzed scientific works highlighted explicitly the key features of the ELK stack adequately. In general, scientific works either provide too superficial information about the ELK stack or include more detailed information about Elasticsearch without addressing the remaining components of the ELK stack properly. This limitation hampers of gaining a holistic view of the capabilities of the ELK stack.

- Present scientific works also neglect of investigating the applicability of the ELK stack for varying kinds of Big Data use cases. Mostly, the ELK stack is implemented within a given use case without making inferences and assessing its ap-

[19]http://www.oclc.org/ezproxy.en.html/
[20]http://www.mysql.com/

plicability for other types of use cases. Thus, additional detail and guidance for selecting the ELK stack for a given type of Big Data use case is missing.

These enumerated limitations unveil a research gap and corroborate the need for this thesis, revealing the key features of the ELK stack on a holistic view and assessing its applicability for Big Data use cases.

Part III.

Descriptive Study

4. Related Technologies

During the literature synthesis process, several related technologies for Elasticsearch has been identified. For example, Solr is mentioned a few times as a related competitor search engine to Elasticsearch, e.g., in [44], [117], and [40]. Lucidworks[21] provides an open source end-to-end solution, similar to the ELK stack, which is called *Solr integrated with Logstash and Kibana*[22], or shortly *SiLK*. Besides Solr and the SiLK stack, Splunk[23] is also mentioned as a commercial rival with its Splunk Enterprise[24] (hereafter Splunk) solution to the ELK stack, e.g., in [57] or [103]. DB-Engines[25], which is an initiative for collecting and presenting information on DBMS, provides a ranking that ranks DBMS according to their popularity. Particularly, it also provides an exclusive ranking for search engines. Table 4.1 provides an excerpt of the ranking with the five most popular search engines. As Table 4.1 shows, Elasticsearch has surged in January 2016 to be the

Ranking						
Rank			**DBMS**	**Score**		
Jan 2016	Dec 2015	Jan 2015		Jan 2016	Dec 2015	Jan 2015
1.	2.	2.	Elasticsearch	77.21	+0.65	+28.17
2.	1.	1.	Solr	75.39	-3.75	-1.35
3.	3.	3.	Splunk	43.12	-0.74	+10.05
4.	4.	5.	MarkLogic	9.92	-0.44	+0.89
5.	5.	4.	Sphinx	8.98	-0.02	-1.16

Table 4.1.: Ranking of the five most popular search engines [19]

most popular search engine, followed by Solr, Splunk, MarkLogic[26], and Sphinx[27]. The calculation of the popularity ranking is based on specific parameters like the number of mentions of the system on websites, the general interest in the system, or the relevance in social networks (for the complete explanation of the popularity calculation, see [20]). Besides the mention in the literature review, the aforementioned ranking reinforces the eligibility of Solr and Splunk as related technologies for the subsequent Section 4.1 and Section 4.2. Following, Section 4.3 juxtaposes Elasticsearch in opposition with Solr and

[21]http://lucidworks.com/
[22]http://lucidworks.com/products/silk/
[23]http://www.splunk.com/
[24]http://www.splunk.com/en/products/en_us/splunk-enterprise.html/
[25]http://db-engines.com/en/
[26]http://www.marklogic.com/
[27]http://sphinxsearch.com/

Splunk. Last but not least, Section 4.4 compares the end-to-end solutions of the previously selected search engines.

4.1. Solr

Solr is one of the most popular open source search platforms from the Apache Lucene (hereafter Lucene) open source project[28] [51]. Solr is written in Java and is built on top of Lucene, which offers core functionality for data indexing and search [97, 47]. Solr was initially started in 2004 at CNET, a well-known website for news and reviews on technologies, and became in 2007 an Apache project. Since then, due to its scalable and highly reliable nature, it has been used for many projects and world's largest websites like Netflix, Ticketmaster, or SourceForge [98, 51]. Internally, Solr uses a NoSQL-like document store database system. Solr extends Lucene by providing many useful features related to full-text search, e.g., keyword highlighting, spelling suggestions, complex ranking options, geospatial search, or numeric field statistics, to name a few. Furthermore, its features also include near real-time indexing, dynamic clustering, query language extension, caching, and rich document handling [47, 51, 66]. Solr also supports distributed indexing by its SolrCloud functionality [41]. The most important key features are elaborated in the following.

4.1.1. Key Features

- **Advanced full-text search**: Solr can be used as an internal search engine for websites and applications. Due to its design, it offers more flexibility than internal search capabilities of classic databases. It enables performing fast searches. Moreover, it gives some flexibility on terms that are useful to intercept a natural user search. Searches can be also combined with out of the box capabilities to perform searches over value intervals or by using geocoding functions.

- **Faceted search**: Solr can automatically perform faceted search over fields in order to gain information, e.g., how many documents have a specific value for a given field. Faceted search is a particular type of search based on classification, which is useful to construct some kind of faceted navigation.

- **Spelling suggestions**: Solr comprises components for creating autosuggestion results using internal similarity algorithms.

- **Language analysis**: Solr permits configuring various types of language analysis with the possibility to configure them specifically for a certain language [98].

- **Highlighting**: Solr supports optical highlighting of found search terms.

- **Near real-time search**: After indexing, data can be searched immediately in Solr.

[28] http://lucene.apache.org/

- **Multiple client APIs**: Various client APIs for different languages, e.g., Java, PHP, or JRuby, can be used for integrating Solr with other applications.
- **Scalability**: With the SolrCloud, Solr can scale linearly. Herein, a new master/slave architecture can be created for distributing data over multiple nodes. Furthermore, SolrCloud enables flexibility with the use of the current Solr architecture [63].

4.1.2. SiLK Stack

The SiLK stack includes a custom packaging of *Solr*, *Banana* and a *Solr Writer for Logstash*. It is a data analytics tool for analyzing and visualizing log data. Banana is the name of the open source port of Kibana 3. Banana is a data visualization tool that allows the creation of dashboards to display contents stored in Solr indices. It is commonly used with Logstash for log data. However, any content stored in a Solr index is eligible for visualization in a Banana dashboard. Banana provides panels such as histograms, geomaps, heatmaps, and bettermaps for analyzing data. The Solr Writer for Logstash is an implementation of Logstash specifically designed for indexing logs or other contents to Solr. Particularly, Logstash provides the conversion from the raw log file format to documents that can be index by Solr. Stored and indexed data in Solr can be queried in near real-time by data visualization tools provided by Banana. The SiLK stack can be used for different use cases, such as for Apache weblogs or data analytics [72, 71].

4.2. Splunk

Splunk is a log-, monitoring-, and reporting tool, which is developed using C/C++ and Python [77, 78]. Splunk manages searches, inserts, filters, and deletes. It analyzes Big Data that is created by machines, as well as other types of data sources [100]. The free version of Splunk allows users to index up to 500 MB of data per day. However, the free version is restricted in some functions, such as drawbacks on scheduling, automated generation, and delivery of reports and dashboards, or monitoring and alerting for individual and correlated near real-time events. The commercial version offers full functionality and its price is not determined by the amount of users, but by the amount of indexed data per day [77]. Splunk utilizes a role-based security model to offer flexible and effective ways to protect all the data indexed by Splunk by controlling searches and results in the presentation layer [100].

4.2.1. Key Features

Splunk possesses the following key features:

- **Search**: Search is the primary way users navigate data in Splunk. With Splunk's searching features, users are able to write searches, to retrieve events from an index, use statistical commands for calculating metrics and creating reports, search for specific conditions within a rolling time window, and identify patterns in data.

- **Reports**: In Splunk, reports are saved searches and pivots. Users can run reports on an ad hoc basis, schedule reports to run on a regular interval, or set a scheduled report to generate alerts when the results meet particular conditions.
- **Dashboards**: Dashboards consist of panels that cover modules such as search boxes and data visualizations. Dashboard panels are usually connected to saved searches or pivots. They can display the results of completed and near real-time searches.
- **Alerts**: Alerts are triggered when search results meet certain conditions. Users can utilize alerts on historical and near real-time searches. Alerts can be configured to trigger actions such as sending alert information to specific e-mail addresses or posting alert information to a web resource [105].

4.2.2. Use Cases

Splunk can be utilized for various types of use cases, which are enumerated below:

- **Investigational searching**: The practice of this use case usually refers to the processes of analyzing an infrastructure or large collection of data to look for an occurrence of certain events or incidents. Moreover, this process can include locating information that portends the potential for an event or incident. By indexing, Splunk allows to search and navigate through data and data sources in near real-time. This includes, among others, logs, configurations, scripts, and almost any kind of metric, in almost any location. Splunk's searching capabilities comprise functionalities such as a search bar, time range picker, and a summary of the data previously read into and indexed by Splunk. In addition, there is a dashboard of information that includes quick action icons, a mode selector, event statuses, and several tabs to show various event results. With these functionalities, a user can create searches that combine time and terms, can find errors that cross multiple layers of an infrastructure, and can locate and track configurations changes.
- **Monitoring and alerting**: Monitoring various applications is a typical requirement of any organization's data center. The ability to monitor any infrastructure in near real-time is fundamental to identify issues and attacks before they impact customers, services, and ultimately profitability. With Splunk's monitoring capabilities, certain patterns, trends, and thresholds can be defined as events for Splunk to keep an alert for. Splunk can also trigger notifications in near real-time so that appropriate actions can be taken to follow up on an event or even avoid it as well as avoid the downtime and the cost potentially caused by an event. Furthermore, Splunk has the power to perform actions based on certain events or conditions. These actions include activities such as sending an e-mail, running a

program or script, or creating an organizational support or action ticket. In addition to searching and monitoring Big Data, Splunk can be configured to alert anyone within an organization, when an event occurs or when a search result meets specific conditions.

- **Decision support analysis**: Using Splunk as a near real-time decision support system has several advantages, such as increasing productivity, improving efficiency, reducing costs, or gaining operational intelligence. As a decision support system, Splunk can be used to answer both structured and unstructured questions based on data, permit a scheduled-control of developed processes, and possess the ability to gather near real-time data with details of this data. Also, it does not require data to be specifically extracted, transformed, and then (re)loaded into an accessible model for it to get started [78].

4.3. Elasticsearch vs. Solr vs. Splunk

Table 4.2 provides an overview of the previous introduced related technologies and Elasticsearch. As mentioned previously, Elasticsearch and Solr are search engines, while Splunk is a complete end-to-end data analytics platform for Big Data without the need for additional components. Nevertheless, Elasticsearch and Solr form the centerpiece of their own end-to-end data analytics solutions, namely the ELK and SiLK stacks.
The core statements of Table 4.2 are:

- **License**: Elasticsearch and Solr are open source technologies, whereas Splunk is commercial and chargeable.
- **Implementation language**: Elasticsearch and Solr are both written in Java, while the implementation language of Splunk is unknown.
- **Server operating systems**: Elasticsearch can run on all OS with a JVM. Solr exhibits the same properties, except that it also can run as a servlet in a servlet container, such as Tomcat and Jetty. Splunk can run in Linux, OS X, Solaris, and Windows.
- **Data schema**: Elasticsearch is schema-free, since it provides implicit mappings. In contrast, according to [20], Solr and Splunk share the same property, namely that they have a data schema. However, this is not true for Splunk. The manufacturer homepage of Splunk promotes that Splunk is schema-less and it does not necessarily need a definition of an upfront schema [20].
- **Secondary indices**: All of the compared technologies have secondary indices. In Solr and Elasticsearch, all search fields are automatically indexed.
- **APIs and other access methods**: In order to access data in Elasticsearch, Elasticsearch provides a Java and a RESTful HTTP/JSON API, whereas Solr provides

Properties comparison of search engines			
Name	**Elasticsearch**	**Solr**	**Splunk**
License	open source	open source	commercial
Implementation language	Java	Java	
Server operating systems	All OS with a JVM	All OS with a JVM and a servlet container	Linux, OS X, Solaris and Windows
Data schema	schema-less	yes	yes (schema-less)
Secondary indices	yes	yes	yes
APIs and other access methods	Java API and RESTful HTTP/JSON API	Java API and RESTful HTTP API	HTTP REST
Supported programming languages	.NET, Erlang, Java, JavaScript, Perl, PHP, Python, Ruby, and Scala	.NET, Erlang, Java, JavaScript, XML, JSON, Perl, PHP, Python, Ruby, and Scala	C#, Java, JavaScript, PHP, Python, and Ruby
Partitioning methods	Sharding	Sharding	Sharding
MapReduce	no (with Hadoop integration)	no	with Hadoop integration
Consistency concepts	Eventual consistency	Eventual consistency	Eventual consistency
Transaction concepts	no	optimistic locking	no
Concurrency		yes	yes
User concepts			Access rights for users and roles

Table 4.2.: Properties comparison of the search engines Elasticsearch, Solr, and Splunk [21]

a Java and RESTful HTTP API. By contrast, Splunk only offers a HTTP RESTful interface.

- **Supported programming languages**: Elasticsearch and Solr support many programming languages, such as .NET, Erlang, Java, JavaScript, Perl, PHP, Python, and Scala. Splunk is more limited and supports the programming languages C#, Java, JavaScript, PHP, Python, and Ruby.

- **Partitioning methods**: All of the analyzed technologies utilize the concept of sharding for storing different data in different nodes.
- **MapReduce**: For optimizing its search language, Splunk makes the use of the MapReduce model for paralleling executions, which is accomplished by a Apache Hadoop (here after Hadoop) integration[29]. According to [20], Elasticsearch and Solr do not use the MapReduce model. However, this is not true for Elasticsearch, because it can be integrated with Hadoop and therefore also provides an integration with MapReduce [31].
- **Consistency concepts**: All technologies guarantee eventual consistency, which means that they informally guarantee that, if no new updates are made to a given datum, eventually all accesses to that item will return the last updated value.
- **Transaction concepts**: Elasticsearch and Splunk do not support data-integrity after non-atomic manipulations of data, whereas Solr provides optimistic locking.
- **Concurrency**: Solr and Splunk promote concurrent manipulation of data.
- **User concepts**: In their standard configurations, Elasticsearch and Solr do not provide users concepts for access control. However, the chargeable Shield[30] plugin for Elasticsearch can be used for access control. Splunk provides access rights for users and roles for data access control [20].

4.4. ELK stack vs. SiLK stack vs. Splunk

The previously compared search engines can be integrated with other technologies in order to build end-to-end solutions, except Splunk which already provides out of the box data ingestion and data visualization capabilities. By collating the three attributes of search-based data discovery tools (see Section 2.1), namely the provision of a proprietary data structure to store and model data collected from various sources, the provision of a built-in performance layer, and the provision of an intuitive user interface, one can realize that the ELK stack, SiLK stack, and Splunk provide these attributes. For the remainder of this thesis, the end-to-end solutions are denoted as search-based data discovery tools.
Table 4.3 provides a tabular synopsis of some important evaluation criteria based on personal experiences during the implementation and benchmarking of the search-based data discovery tools. By comparing the maturity of features of all three tools, Splunk provides the most variety of features. Splunk is an established player in the Big Data analytics market and also has existed for a few years longer than the other stacks. Although Solr was initially started in 2004, the SiLK stack is relatively young, currently release version 1.3. The SiLK stack uses Kibana version 3 and Logstash version 1.3.3. In

[29]http://hadoop.apache.org/
[30]http://www.elastic.co/products/shield

contrast, this thesis, uses Logstash version 2.1.0, Elasticsearch version 2.1.1, and Kibana version 4.3.0. Since the SiLK stack uses older versions of Kibana and Logstash, it provides less features than the ELK stack, e.g., ELK stack is capable of using custom map providers or provides a server status page that gives an overview of Kibana's actual status. The individual technologies of the ELK stack are well documented on Elastic's homepage providing examples for real Big Data use cases and detailed elaboration of their features. Splunk is also very well-documented whereas the SiLK stack provides only a 17 pages long documentation which is partially uncompleted. During the implementations and benchmarks of the search-based data discovery tools, some technical questions occurred. Some of the questions regarding Solr and the individual technologies of the ELK stack could be answered by existing Stack Overflow questions. While Splunk and the ELK stack provide exclusive discussion pages, similar to Stack Overflow, SiLK stack does not. Unresolved questions regarding the ELK stack were responded quickly and appropriately, whereas some technical questions regarding Splunk, e.g., Splunk's search functionalities in the command line, remain still unanswered. Additionally, some questions were resolved directly by some developers and founders of the ELK stack. Since the ELK stack is an open-source project, it has a very active community which propagates ideas and proposals for improving and extending the ELK stack's features. This is not true for Splunk, since it is licensed and therefore the active community is constrained on Splunk contact persons of industry companies. This constraint is corroborated by the hidden implementation of Splunk. Splunk is very easy to install. It is directly ready for ingesting and analyzing data. This fact is not valid for the ELK and SiLK stacks. In both stacks, Logstash's configuration file has to be configured in order to be able to ingest data from various sources. Additionally, both stacks require Banana and Kibana configurations in terms of specifying Solr's and Elasticsearch's domains and the ports.

Properties comparison of search-based data discovery tools			
Name	**ELK stack**	**SiLK stack**	**Splunk**
Development status	0	-	+
Documentation	+	-	+
Product support/ Community	+	-	0
Setup complexity	0	0	+

Table 4.3.: Properties comparison of search-based data discovery tools

5. ELK Stack

The ELK stack is an integrated and complete logging platform that is built on the combination of the three open-source tools Elasticsearch, Logstash, and Kibana. It tries to address common problems in log analysis, such as the existence of non-consistent log formats, decentralized logs, or expert knowledge requirements. The ELK stack utilizes the open-source stack of:

- **Elasticsearch**: For deep search and data analytics.
- **Logstash**: For centralized logging management, which includes shipping and forwarding of logs from multiple servers, log enrichment, and log parsing.
- **Kibana**: For powerful data visualizations.

ELK stack is currently maintained and actively supported by the company called Elastic [17]. The application of the ELK stack supports various use cases such as free and structured search, data analytics, log and event analysis, and visual exploration via Kibana [54]. Figure 5.1 shows a typical ELK stack data pipeline. Usually, logs from multiple

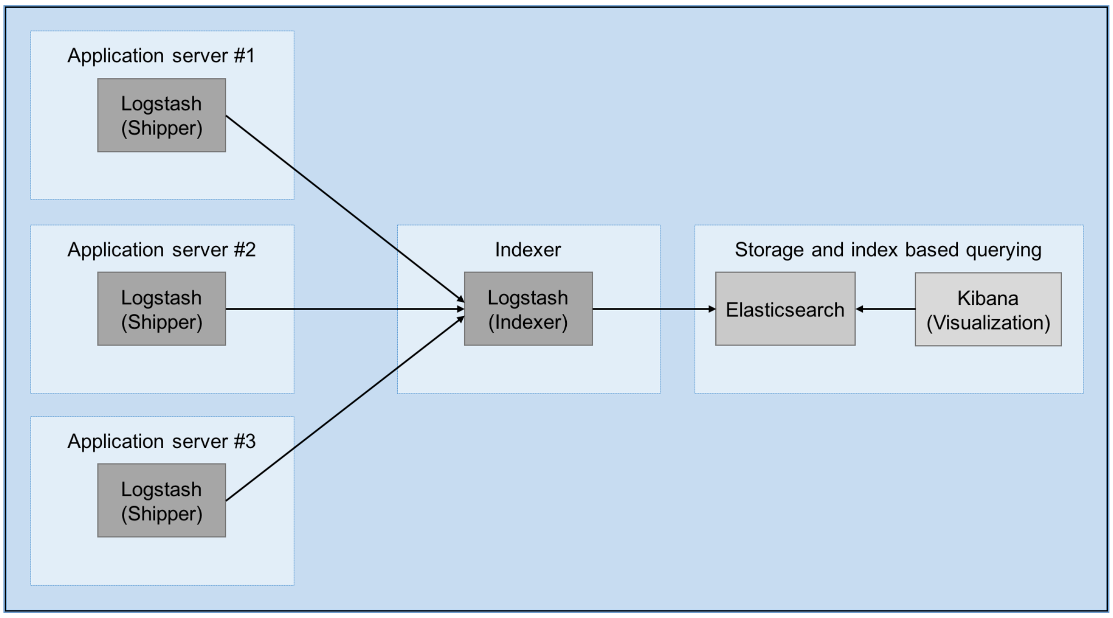


Figure 5.1.: Overview of ELK stack data pipeline [17]

application servers are shipped through Logstash shippers to a central Logstash indexer. Afterwards, the Logstash indexer outputs data to an Elasticsearch cluster that are queried by Kibana to build visualizations and build dashboards over the log data [17]. For assessing the applicability of the ELK stack for Big Data use cases, this thesis looks at the ELK stack in its entirety. This is an important point to mention, since individual components of the ELK stack can be combined without using the whole stack. For instance, [12] uses Elasticsearch as an indexation server and Kibana as an output interface with Couchbase[31] as a database server for creating a social mining architecture. However, in this example Logstash is not utilized. Another example is provided by [107], which uses RabbitMQ[32] and Graylog2[33] with Logstash for creating a central logging system. However, Elasticsearch and Kibana are not used in this use case. In short, such types of combinations are not considered for the remainder of this thesis.
In order to extract the key features of Elasticsearch, Logstash, and Kibana in the abundance of relevant information, a methodology was created which is elaborated in Section 5.1. Following this, the individual technologies Logstash (Section 5.2), Elasticsearch (Section 5.3), and Kibana (Section 5.4) are analyzed in depth. Key findings based on the descriptive study can be found in Section 5.5.

5.1. Key Feature Extraction

Section 3.1 has already mentioned that information sources, which provide too superficial information about the ELK stack or its individual technologies, are not eligible for the descriptive study. However, there are still 64 information sources remaining. Due to reproducibility, rigor, and relevance reasons, a key feature extraction methodology has been created. This methodology is illustrated in Figure 5.2. First of all, detailed and more detailed information sources serve relevant raw information of Elasticsearch, Logstash, and Kibana. In the next step, relevant contents of the information sources are determined and are summarized by creating corresponding keywords. The keywords are cleaned by the *Text Mining Package*[34] (tm) in *R*. The data cleaning step comprises classical text mining functions: *Lowercasing*, elimination of *stopwords* and *punctuation*, and *stemming*[35]. After performing text mining methods, the *term frequency*[36] of bigrams[37] are calculated by using the *tm* Package in R. As a result, a term document matrix is created which contains a sorted list of bigrams and their corresponding frequencies. Due to the fact that some relevant information sources originate from the same author and also mostly utilize the same information for describing the technolo-

[31]http://www.couchbase.com/
[32]http://www.rabbitmq.com/
[33]http://github.com/Graylog2/graylog2-server
[34]http://cran.r-project.org/web/packages/tm/index.html
[35]The term *stemming* is used in information retrieval to describe the procedure for reducing inflected words to their word stem or base [83].
[36]The *term frequency* is the standard notion of frequency in corpus-based natural language processing. It counts the number of times that a type (term/word/ngram) appears in a corpus [122].
[37]A *bigram* is a sequence of two adjacent elements in a string of tokens.

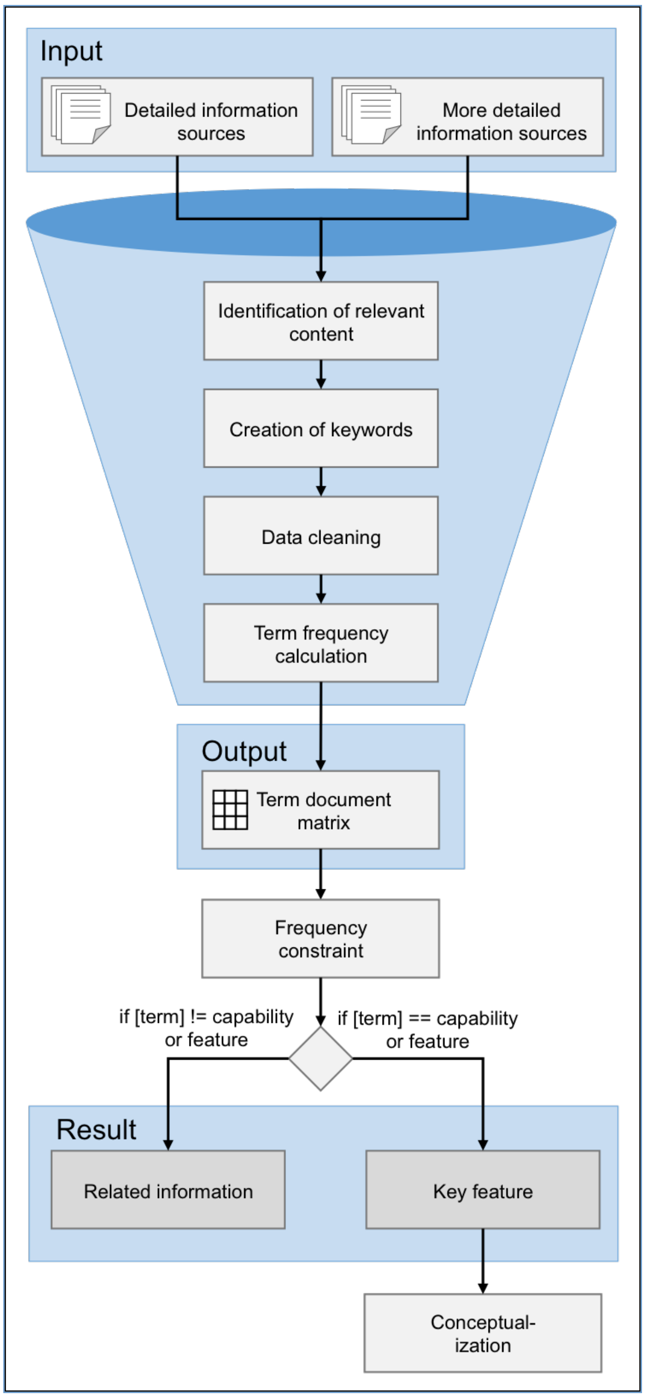

Input
Detailed information sources
More detailed information sources
Identification of relevant content
Creation of keywords
Data cleaning
Term frequency calculation
Output
Term document matrix
Frequency constraint
if [term] != capability or feature
if [term] == capability or feature
Result
Related information
Key feature
Conceptual-ization

Figure 5.2.: Overview of key feature extraction methodology

gies, e.g., [69] and [58], [10] and [9], or [119] and [120], the threshold value for the

frequency of a bigram has to be at least three in order to be considered for the next step. This threshold value is appropriate, since it decreases the aforementioned bias of duplicate information, and as well as forms the lower limit of recurring information. Subsequently, the bigram is marked as a key feature or related information dependent on whether the bigram comprises characteristics of a technology capability/feature or not. Within the conceptualization, firstly, related bigrams are merged. For instance, the bigrams for Logstash like *"filter output"*, *"input filter"*, *"output plugin"*, *"input output"* are joined together into the unigrams *"input"*, *"filter"*, *"output"*, and *"plugin"*. Naturally, the merging of bigrams is performed with caution in order to prevent information loss. Secondly, the context of the remaining bigrams are identified by using relevant contents and created keywords of input information sources. For example, the Logstash bigrams *"output plugin"*, *"input plugin"*, *"input filter"*, and *"filter output"* can be found within the *"event processing pipeline"* context, which consists of the three plugins *"input"*, *"filter"*, and *"output"*. Accordingly, the context of all key features are determined, which then set into relation. Finally, based on the first and second conceptualization steps, visualizations are generated. One important remark on the visualizations is that not all relationships between key features are illustrated for reasons of clarity and comprehensibility. For that reason, only the most important relationships are demonstrated.
Important to mention is that the ELK stack is very actively developed and strongly promoted by a large community. For this reason, each new release may cause big changes in the capabilities, features, and performance. For instance, as a result of the key feature extraction, river plugins are identified as key features of Elasticsearch. However, in the current Elasticsearch version, rivers plugins are deprecated and already removed from it.

5.2. Logstash

Logstash is a open-source[38] tool engine developed by the American developer Jordan Sissel that provides an integrated framework for log collection, centralization, parsing, and analysis of a large variety of structured and unstructured data and events generated across various systems. It has the ability to parse both multiline and single line logs of various types, including common formats like syslog and JSON formatted logs, as well as to parse custom logs. Logstash is relatively easy to set up in large environments and is designed to efficiently and flexibly process logs, events, and unstructured data sources for distribution into a variety of outputs [94, 17, 111]. It can be easily customized via plugins for input, output, and data-filters [6].
It supports:

- **Centralized data processing**: Logstash uses a data pipeline that can centralize data processing. With the use of a collection of input and output plugins, it can convert many different input sources to a single common format.

[38]Logstash is free under the Apache 2.0 license [111].

- **Custom log formats**: Logs generated by different applications often have certain formats specific to the application. Logstash helps to parse and process custom formats on a large scale. It provides support to write custom filters for tokenization as well as ready-to-use filters.

- **Plugin development**: Custom plugins can be developed and published, and there is a large variety of proprietary developed plugins already available [17].

Logstash is written in Ruby and runs on JRuby. Logstash is easy to deploy, because it is a Java-based utility where a graphical user interface and embedded Elasticsearch engine are encapsulated in a standalone jar-file that can be started directly using a JVM. It is most commonly used to index data in Elasticsearch [52, 54]. [111] elaborates the functionalities of Logstash in more than 220 pages. However, this thesis aims to provide a holistic view on the key features of Logstash, wherefore some functionalities are not described in depth.

5.2.1. Conceptualization of Logstash

Figure 5.3 illustrates the conceptualization of Logstash. In total, Logstash consists of the three plugins *"Input"*, *"Filter"*, and *"Output"*, which are used within Logstash's *"Event processing pipeline"*. *"Grok"* and *"Mutate"* are special kinds of *"Filter"* plugins. *"Elasticsearch"* is a special type of an *"Output"* plugin. *"Plugins"* are specified within Logstash's configuration file. In total, Logstash has two *"Host classes"*, namely *"Central server"* and *"Event forwarder"*. Furthermore, Logstash consists of four *"Ecosystem components"*, namely *"Web interface"*, *"Broker and indexer"*, *"Search and storage"*, and *"Shipper"*. The first three mentioned components represent *"Central servers"*, which belong to the *"Central server" "Host class"*. The *"Shipper"* conforms with the *"Event forwarder"*. These key features are elaborated on in Section 5.2.2.

5.2.2. Key Features

In the following, Logstash's key features and capabilities are elaborated:

- **Event processing pipeline**: Figure 5.4 illustrates Logstash's three phase event processing pipeline that deals with the collection of events from various input sources like syslog or Twitter, parsing, among others filtering, of events, and forwarding of the parsed events to various outputs like Elasticsearch or Cassandra[39] in order to store them there [57].

- **Configuration file**: The *logstash.conf* folder contains the configuration file for Logstash. The Logstash configuration file utilizes a custom JSON-like language in which the inputs and the outputs have to be specified, whereas the filter part is optional [60, 34]. Inside each component's block, one can specify and configure plugins [111]. Listing 5.1 illustrates its most basic form:

[39]http://cassandra.apache.org/

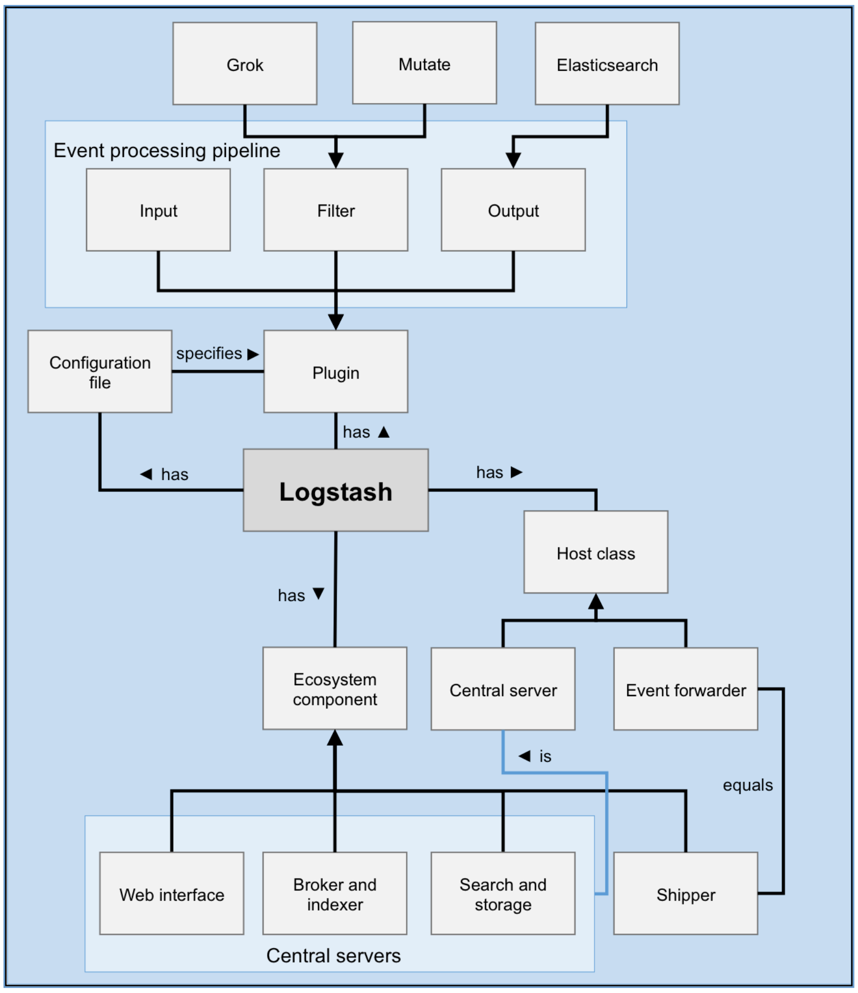


Figure 5.3.: Conceptualization of Logstash

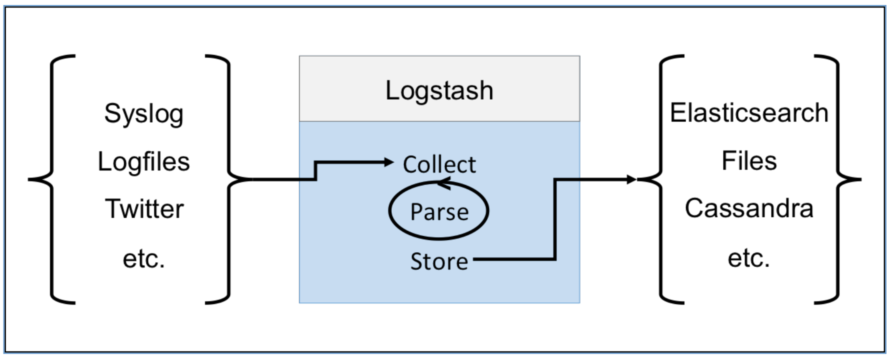


Figure 5.4.: Overview of Logstash's event processing pipeline [57]

Listing 5.1: Basic form of Logstash's configuration file

```
input {
    ...
}
filter {
    ...
}
output {
    ...
}
```

- **Plugin**: Logstash's plugin architecture provides an easy way of extension of functionalities in each phase using plugins that consist of a large scale of inputs and outputs which can be utilized to create flexible event processing services [10, 52].

- **Input**: Input plugins are the mechanism for passing log data to Logstash [103]. The input phase collects logs and forwards the collected events to the filter phase [52]. One advantage of Logstash is that it supports many different input plugins. Currently, Logstash provides 49 official inputs developed by the Logstash team. There are also many input plugins developed by the Logstash community and released for public use [4]. Table 5.1 provides an excerpt of available official input plugins and a short description. The full list can be found in [104]. When defining an input, certain parameters have to be specified in Logstash's configuration file. Each input plugin can contain different parameters, but most inputs share a few common parameters such as *tag*, *source*, *host*, *port*, and *type*. These parameters are vital in routing, tagging, and retrieving data from hosts [88].

Input	
Name	**Description**
file	The file input streams events from files, normally by tailing and optionally reading them from the beginning.
jdbc	This input plugin is created as a way to ingest data in any database with a JDBC interface into Logstash.
lumberjack	The lumberjack plugin receives events using the lumberjack protocol. This is mainly to receive events shipped with lumberjack, now represented primarily via the Logstash-forwarder.
kafka	This input reads events from a Kafka topic. It utilizes the high level consumer API provided by Kafka to read messages from the broker.
stdin	This plugin reads events from standard input.
twitter	This input helps to ingest events from the Twitter Streaming API.

Table 5.1.: Excerpt of available input plugins [104]

- **Filter**: Filter plugins are workhorses for processing inputs in the Logstash chain [103]. Logstash comprises a large collection of filters which allow for flexible recognition, filtering, parsing, and conversion of events, before they are pushed to an output destination, e.g., Elasticsearch [52, 88]. For example, filters allow converting multiline messages into single line format, filtering out events regular expressions, adding new fields to events from external queries, and accomplishing many other advanced message manipulation tasks. They are often combined with conditionals in order to perform a certain action on an event, if it matches particular criteria [114, 103]. Due to Logstash's powerful event filtering and conversion capabilities, it is used mostly as an event preprocessor for different systems, including other log visualization systems [114]. Table 5.2 provides an excerpt of official filter plugins with their corresponding descriptions. A full list of filters can be found in [28].

Filter	
Name	**Description**
csv	This filter takes an event field containing CSV data, parses it, and stores it as individual fields. The CSV filter can also parse data with any separator, not only commas.
date	The date filter is utilized for parsing dates from fields, and then using that date or timestamp as the Logstash timestamp for the event.
geoip	This filter adds information about the geographical location of IP addresses.
grok	The grok filter parses arbitrary text and structures it. Grok is currently the best way in Logstash to parse unstructured log data into something structured and queryable.
mutate	The mutate filter allows to perform general mutations on fields, e.g., rename, remove, replace, and modify fields of events.
xml	The XML filter takes a field that contains XML and expands it into an actual data structure.

Table 5.2.: Excerpt of available filter plugins [28]

- **Grok**: The grok filter plugin is one of the most commonly used in Logstash. While most log management tools use regular expression language for event matching and parsing, grok filters exert many predefined patterns that represent regular expressions for common matching tasks. Using predefined grok patterns, inexperienced users with the regular expression language can accomplish event parsing tasks in an easier way [114].

- **Mutate**: The mutate filter allows performing of regular expression pattern matching and replacement for general transformation of event fields [52].

- **Output**: After the event is passed through the filter, it moves onto the output phase. Output is the final phase of the Logstash's event processing pipeline and represents the feature how Logstash interacts with the storage subsystem [88, 103]. The output indicates where the resulting data structure is going to be sent [60]. An event can pass through multiple outputs during processing [103]. Again, Logstash provides a large set of output plugins. Table 5.3 provides an overview of existing official outputs with their descriptions. A full list of output plugins can be found in [33].

Output	
Name	**Description**
elasticsearch	The elasticsearch plugin is the recommended method of storing logs in Elasticsearch.
email	This plugin sends an email when an output is received.
kafka	The kafka plugin writes events to a Kafka topic. It uses the Kafka Producer API to write messages to a topic on the broker.
stdout	This plugin is a simple output which prints to the STDOUT of the shell running Logstash.

Table 5.3.: Excerpt of available output plugins [33]

- **Elasticsearch**: The elasticsearch plugin is one of the most commonly used output plugins of Logstash. It enables to save the data in an efficient, convenient, and easily queryable format, namely JSON [103]. Moreover, this output is necessary, when Kibana should be used as web interface [26].
- **Host class**: In most cases there are two broad classes of Logstash hosts. The first one is the host which runs the Logstash agent as an event shipper that forwards application, service, and host logs to a central Logstash server. The second one is the central Logstash host which implements a combination of components for pre-processing and filtering of events [103, 52].
- **Central server**: The central Logstash server runs some combination of archiver, indexer, search, storage, and web interface software which receive, process, and store logs [103].
- **Broker and indexer**: A broker or indexer is another Logstash index which is configured to parse, filter, and route logs and events. The indexer or broker then exports parsed logs to an output destination [120].
- **Search and storage**: Elasticsearch is the search and storage in the Logstash ecosystem that allows to search and store events [111].
- **Web interface**: Kibana represents the web interface of the Logstash ecosystem. In order to use Kibana, it must be configured to insert events into its embedded

Elasticsearch database. With the web interface, it is possible to carry out basic searches from log messages in Elasticsearch [114, 111].

- **Event forwarder/shipper**: The Logstash-forwarder, formerly Lumberjack, is an event forwarder, or respectively a shipper, is a Logstash instance which is configured to take input from various remote sources and then ships them as events to a central Logstash server [120, 111].

Any of these ecosystem components can be scaled by adding computing resources. When Logstash routinely throttles incoming events, one should consider adding a message queue. It serves as a buffer to hold ingested data and serve as failover protection. Alternately, the Elasticsearch cluster's rate of data consumption can be increased by adding more Logstash indexing instances [25].

5.3. Elasticsearch

Elasticsearch is a distributed, multitenant-capable, and highly scalable open-source[40] full-text search engine that was first released by Shay Banon in 2010 [50, 54, 76, 85]. Elasticsearch is a fairly new project that is built on top of Lucene that itself is a very mature open-source Java based indexing and search technology [97, 47, 89]. Elasticsearch is a Java search server that runs in a Java application server [47]. Although Elasticsearch is mainly used by Java applications, applications do not necessarily have to be written in Java in order to work with it, since it can send and receive data over HTTP in JSON to index, search, and manage the Elasticsearch cluster [52]. Elasticsearch is best suited for the applications that are built to handle near real-time data that needs to be processed and analyzed in a rapid manner, e.g., software analytics applications. Many organizations worldwide, including Netflix, Facebook, GitHub, and Stack Overflow, have adopted Elasticsearch in order to handle new demands of agile data processing and storage. While Elasticsearch is currently mainly used by industrial organizations, research communities are also evolving and exploring solutions such as Elasticsearch to be able to mine modern data repositories [65]. Elasticsearch's main focus is to provide a fast and powerful search and explicitly addresses issues of scalability, availability, Big Data search, and performance that relational databases were simply never designed to support [65, 50]. Elasticsearch goes beyond free-text search and provides structured search, hit word highlighting, aggregations, facets over the data, and more. These capabilities enable users and developers to extract valuable information from their data regardless of the form [23, 1]. While Elasticsearch makes it possible to perform various types of searches and aggregations, it is not suitable to provide advanced analytics and data mining features [10]. Elasticsearch is primarily designed as a search engine, but since Elasticsearch is also able to retrieve documents by IDs, it can also be used as a regular non-relational document based data store rather than a database [112, 106]. Although Elasticsearch has been given functionalities to act as a data storage solution,

[40]Elasticsearch is free under the Apache 2.0 License [111].

in summary, concurrency control, lack of backup functionality, transactions, and durability seem to be some of the weaknesses of Elasticsearch as a storage solution at the moment. This is also why Elasticsearch is in the first place still a search engine and not a database [106, 24]. Elasticsearch is the main component of the ELK stack and provides its storage and search engine capabilities [7, 57].

5.3.1. Conceptualization of Elasticsearch

Figure 5.5 illustrates the conceptualization of Elasticsearch. Elasticsearch's main feature is the provision of *Search* capabilities. In order to accomplish sophisticated searches, Elasticsearch provides the two key features *Query* and *Filter*. There exist multiple types of queries, the most important being *Full-text*, *Term*, *Match*, and *Prefix*. The *Geo* filter is also one of most important features of Elasticsearch. Since Elasticsearch is built on top of *Lucene*, Elasticsearch makes use of all features provided by Lucene and extends them by providing additional features. The Query Domain Specific Language (*Query DSL*) can be used in order to support the creation of advanced queries of Elasticsearch. Elasticsearch uses the *Inverted index* structure for allowing fast full-text searches. An Elasticsearch *Shard*, which contains the data, is an index of Lucene. There are two types of shards, *Primary* and *Replica*. These shards are built of multiple segments which can be optimized by the *Segment merging* capability. An Elasticsearch *Node* can contain multiple shards. There are four types of nodes, namely *Tribe*, *Master*, *Data*, and *Routing*. By creating an Elasticsearch node, it automatically searches for other existing nodes by using the *Zen discovery* feature. This means that Zen discovery enables nodes to build an cluster. The *Cluster* state provides actual information about the cluster. An Elasticsearch *Index* uses Lucene in order to index and store data in Elasticsearch. Although, Elasticsearch is schema-free, Elasticsearch's *Mapping* feature can be used in order to define the structure of an index. An index can contain multiple *Document types* that in turn can contain multiple *JSON documents*. In Elasticsearch, JSON documents can be also *Nested*. A JSON document consists of *Document fields* that have special *Data types*. Elasticsearch provides an *Auto detection* feature in order to determine automatically the data type of a document field. Furthermore, Elasticsearch provides *Transaction log* and *Circuit breaker* features for improving its performance and recoverability. An Elasticsearch *Plugin* enhances the core capabilities of Elasticsearch. For communicating with Elasticsearch, Elasticsearch's *RESTful API* feature can be used. There exist multiple APIs, including *Bulk* and *Java*. Finally, Elasticsearch has the *Aggregation* capability which can be used in order to aggregate stored data. There exist two types of aggregations, namely *Metric* and *Bucket*. Metric aggregations include *Average*, *Unique count*, *Count*, *Percentile*, *Percentile ranks*, *Min*, *Max*, and *Sum*. Bucket aggregations comprise *Histogram*, *Data histogram*, *Range*, *Date range*, *IPv4 range*, *Terms*, *Filters*, *Significant terms*, and *Geo*. Kibana makes the use of these aggregations in order to build its visualizations. For that reason, the aggregations are elaborated in Section 5.4.2. Because Kibana only supports a specific type of geo-aggregations, namely the *Geohash* aggregation and Elasticsearch provides further geo-aggregations, Section 5.3.2 describes the additional ones. Furthermore, in contrast to Kibana and Logstash, some aforementioned key features are described cohesively due to reasons of comprehensibility. These key features

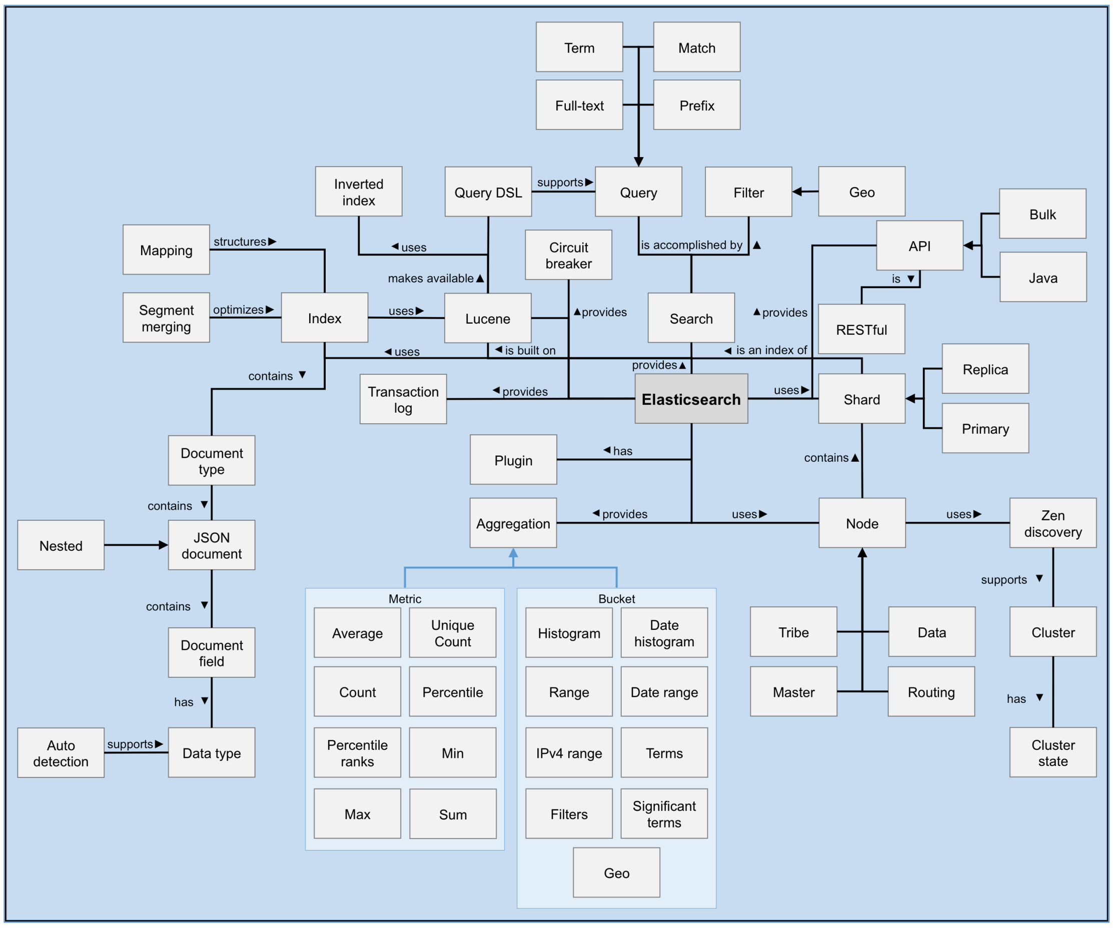


Figure 5.5.: Conceptualization of Elasticsearch

are described in Section 5.3.2.

5.3.2. Key Features

In the following Elasticsearch's key features and capabilities are elaborated:

- **Node and Cluster**: The single instance of the Elasticsearch server is called a node. The node is responsible for storing data and helps in the indexing/searching capabilities. Elasticsearch can work as a standalone and single-search server. A single node in Elasticsearch deployment can be sufficient for many simple use cases. Due to reasons of fault tolerance and sufficient storage, the Elasticsearch server can be run on many cooperating servers. These servers are called a cluster that consists of one or more multiple nodes with the same `cluster.name`

[67, 65, 91, 45]. They are working together to share data and workload. As nodes are added to or removed from the cluster, the cluster reorganizes itself in order to distribute the data evenly [45]. In Elasticsearch, nodes can play four types of roles: the master node, the data node, the routing/load balancer node, the tribe node [91, 17, 10]. The master node acts as a supervisor of the cluster and is responsible for the management of the cluster. The data node indexes documents and performs searches on indexed documents. In order to improve the performance or scaling out, data nodes can be added to the cluster. The routing/load balancer node is responsible for balancing the load, routing the requests for searches, and indexing the documents to appropriate nodes. The tribe node can join multiple clusters and thus casts as a bridge between them [91, 17]. By default, after starting a node in Elasticsearch, it will search for other nodes in the same network with the same `cluster.name`. This mechanism is called zen discovery. It enables Elasticsearch to take core of discovering nodes on the network and to bind them into a cluster [65, 91]. The default zen discovery configuration uses multicast to find other nodes. Herein, a node sends UDP pings across a local network to discover nodes and other Elasticsearch nodes will receive these pings and respond, and a cluster is formed shortly after. However, sometimes this discovery type is not preferable, because other Elasticsearch nodes can join to the cluster by mistake. Also, some networks do not support multicast. Due to these reasons, zen discovery also provides the unicast discovery type. This method works by providing Elasticsearch a list of nodes that it should try to contact. Once the node contacts a member of the unicast list, it will receive a full cluster state that lists all nodes in the cluster. Afterwards, it will then proceed to contact the master node and join. Since unicast and multicast can operate in parallel, the multicast setting in Elasticsearch's configuration file has to be disabled in order to use ensure that only the unicast method is used [91, 67, 49]. Elasticsearch provides detailed information for checking and monitoring a node or the cluster as a whole. This information includes statistics, information about cluster, servers, nodes, indices, and shards. This information can be accessed by the Cluster Health API [67, 45].

- **Shard**: A shard is the physical entity that stores the data for each index. Each Elasticsearch index consists of one or more shards, also called Lucene indices. Thus, each index has a number of primary and replica shards. The shards are spread among all the nodes in the cluster and can be moved from one node to another in the case of node failures or the addition of new nodes to the cluster. The size of a shard has no technical upper limit, however, there is a limit to how big a shard can be with respect to the underlying hardware [35]. There are two types of shards, namely primary shards and replica shards. Every document is stored within a primary shard and by default, every index has five primary shards. This parameter is configurable and cannot be changed after index creation. A replica shard is a full copy of the primary shard and replica shards can be dynamically altered, also after index creation. Replica shards are automatically promoted to primary shards if a primary shard fails. By default, every primary shard has one replica shard. The number of replicas per primary shard can be changed as required. Replica shards will typically reside on a different node than the primary shard in order to help in case of primary shard failure and for reasons of load balancing of incoming requests [17, 49, 65]. These features can be summarized as the sharding feature of Elasticsearch. Particularly, sharding refers to the horizontal partitioning of an Elasticsearch index by the use of shards which may in turn be located on separate database or physical location. The advantage of this feature is that it allows horizontal scaling of the content volume and improves the performance of Elasticsearch by providing parallel operations across various shards that are distributed on nodes [88, 48, 49]. Each primary and replica shard is built of multiple segments (with at least one segment). Elasticsearch makes the use of segment merging for reducing the number of segments in order to allow faster searching and for reducing the size of the index because of removing deleted documents when the merge is finalized [91]. Figure 5.6 illustrates an example of an Elasticsearch cluster. The cluster consists of two nodes with four primary shards and four replica shards. As mentioned before, the replica shards usually reside on different nodes than the primary shards, e.g., Replica shard 1, which belongs to the Primary shard 1, resides on Node 2, whereas the primary resides on Node 1.

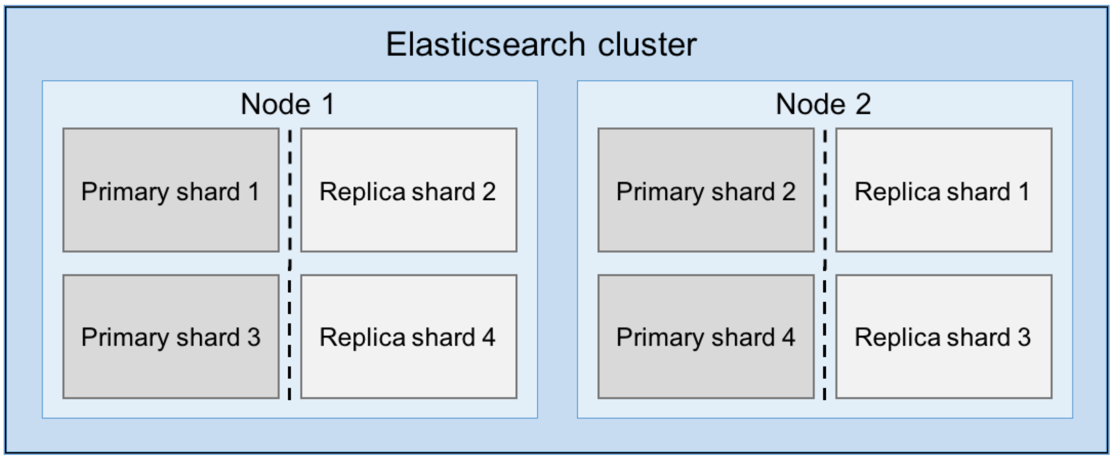


Figure 5.6.: Example of an Elasticsearch cluster [108]

- **Index**: An index in Elasticsearch is the logical place where a collection of documents, which share a number of common characteristics, is stored. Due to this, an index can be divided into smaller pieces, namely shards, and its structure is optimized for fast and efficient full-text searching. A cluster can contain any number of indices. Indices are conceptually similar to databases in traditional relational database systems [17, 88, 123]. While Lucene has only one index, an Elasticsearch index consists of many shards and each shard could be considered to be an Lucene index [57]. Each index comprises multiple document types, which in turn contain multiple JSON documents, and each document contains multiple document fields with an associated data type [17, 123].

- **Mapping**: Elasticsearch provides implicit and explicit mapping capabilities. Mapping, or schema definition, is the process of defining how a document should be mapped to the Elasticsearch search engine, including the definition of what kind of fields the document can have and of which field type the fields are [85]. Mappings are quite similar to a schema in a traditional relational database systems. If the Elasticsearch server has not been handed a mapping before a document is inserted, the server will try to infer the type of the document based on the values in the fields of the document and add this type to the mapping, also called implicit mapping. Types such as numbers and dates are automatically detected and treated accordingly (auto detection). While implicit mapping might be an adequate solution in some cases, the use of explicit mapping provides an opportunity to configure Elasticsearch types to match requirements more precisely. By default, the Elasticsearch server indexes and analyzes all fields, but explicit mapping allows the disabling of indexing of some fields in a document, which reduces the amount of the disk space needed and increases the performance of adding new documents [3, 48, 47, 65]. Setting the value of an `index` parameter to `not_analyzed` of a document field in the explicit mapping disables the capability of including this field into the inverted index. Consequently, on this field full-text search cannot be performed. This is in particular of interest to numeric document fields or to document fields where full-text search is not meaningful. Setting an `index` parameter to `analyzed` of a document field in the explicit mapping enables full-text search on this document field.

- **Document type, JSON document, Document field and Data type**: A document type is utilized to provide a logical partition inside Elasticsearch indices. It represents a class of similar types of documents. A document type is like a table in traditional relational database system. A JSON document is the main entity stored in Elasticsearch. A document equals a row of data in a table in traditional relational database systems. All documents in Elasticsearch are schema-less. This means that two documents of the same type can have different sets of fields [65]. Furthermore, Elasticsearch offers so-called multi-tenancy which allows documents to be divided into separate indices [47]. Every document contains key-value pairs, which are referred as document fields. A document field is the smallest single unit of data stored in Elasticsearch. A field is similar to a column in a traditional relation database system. Each field has a data type. Elasticsearch's core data types are: `string`, `integer`, `long`, `float`, `double`, `boolean`, `date`, and `geo_point`[41] [17, 67, 101, 65]. Elasticsearch also allows to connect multiple documents together, with a main document having multiple dependent documents (called nested documents). Nested documents are treated as individual components but are stored together with the outer documents to improve read performance [24, 67]. Lastly,

[41]The `geo_point` data type is used for latitude and longitude values [49].

Elasticsearch provides the versioning feature for conflict management and concurrency control between different document versions. Every indexed document has an associated version number in Elasticsearch, which is increased with every update. By indicating a version number with the update, conflicting updates can be prevented. However, old versions of documents are not accessible anymore — such feature has to be built by the user. Due to this, it requires significantly more work from the user in comparison to other storage solutions. This is certainly a disadvantage of Elasticsearch as a data storage solution in scenarios where some form of transactional support and concurrency control is important [12, 106]. Table 5.4 juxtaposes Elasticsearch's architectural features in opposition to the architectural features of traditional relational database systems. Figure 5.7 summarizes the aforementioned architectural features of Elasticsearch and depicts the relations between the individual components.

Juxtaposition of architectural features	
Elasticsearch	**Traditional relational database systems**
Index	Database
Mapping	Schema
Document type	Table
JSON document	Row
Document field	Column

Table 5.4.: Elasticsearch vs. traditional relational database systems

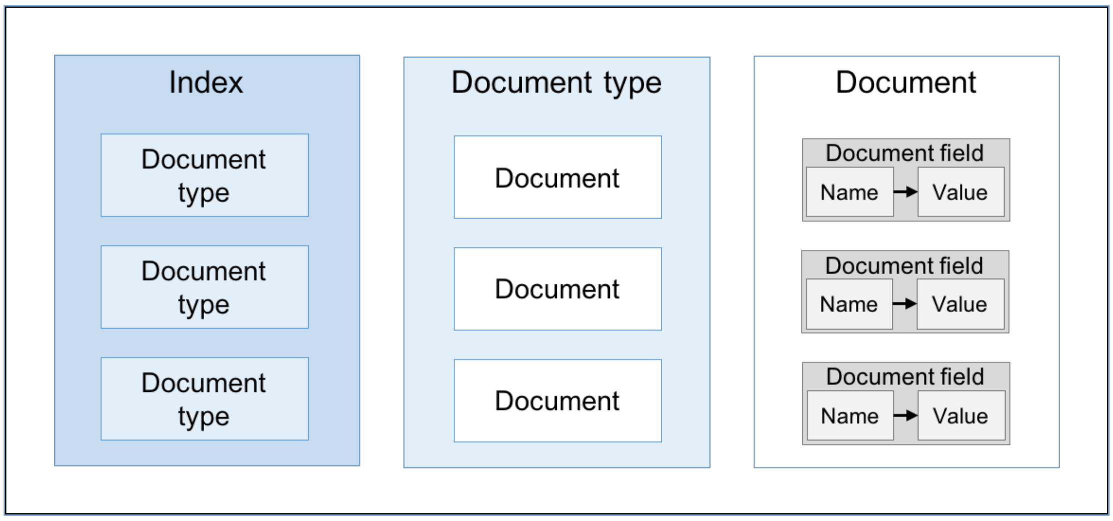


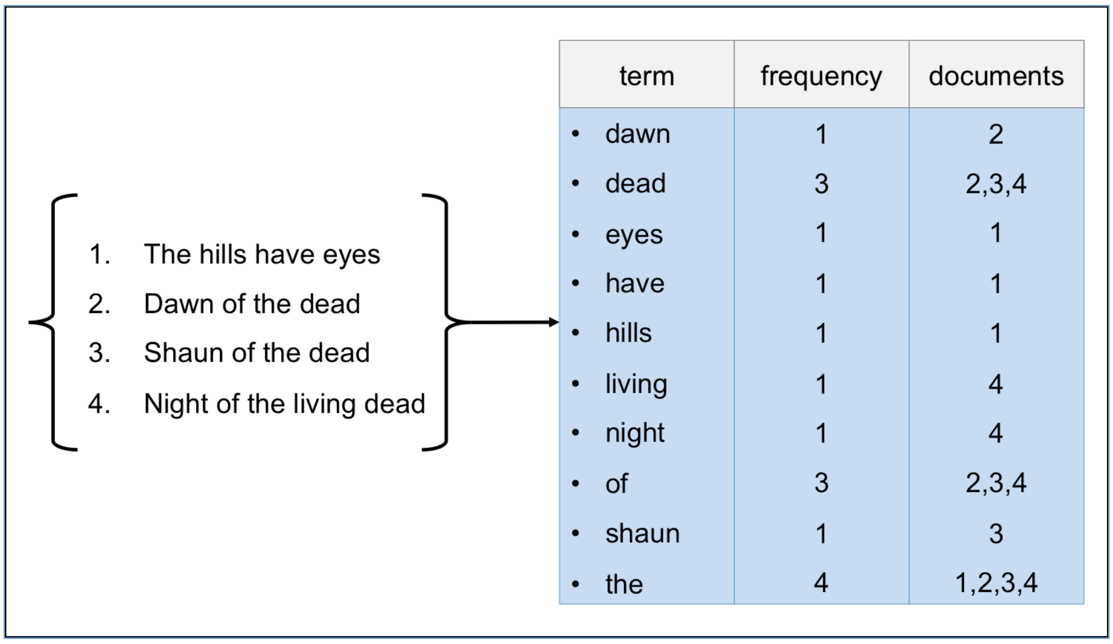
Figure 5.7.: Elasticsearch's data model [86]

- **API**: Elasticsearch offers three ways in order to communicate with it: a RESTful API, a Java API, and a Groovy API [97]. Since, Logstash and Kibana makes the use of Elasticsearch's RESTful API [17], the other APIs are neglected. The usage of the RESTful API is quite simple: it expects JSON encoded parameters, and can be accessed using HTTP. The returned results are also encoded in JSON. The RESTful API supports requests in order to manage the index, check the server health, update the node, search data, and manage the cluster. Since REST is built upon HTTP protocol, it supports all methods of HTTP like `GET`, `PUT`, `POST`, `DELETE`, and so on. By default, Elasticsearch does not provide any authentication or authorization method to its REST API [97, 101]. However, the chargeable Elasticsearch plugin Shield[42] provides functionalities for encrypting communications and a role-based access control [37]. The REST API provides the speeding up of atomic operations with the Bulk API. It allows to make multiple `create`, `read`, `update`, and `delete` requests of documents at once [45, 84].

- **Lucene**: Lucene is a high-performance, full-featured text search engine Java library that has become the standard reference for building a powerful and easy integrable open-source search library. Lucene has to be wrapped with an interface so that its features can be used by an application. Many such interfaces have been built for different platforms and use cases, e.g., Solr and Elasticsearch. Lucene's main component is its inverted index. Herein, words in a document are stored in a dictionary, and with each word, the documents that contain this word are stored. By default, Lucene uses the term frequency-inverse document frequency algorithm for calculating the relevance of a document in the context of a search query [23, 57, 67]. Figure 5.8 depicts the functional principle of Lucene's inverted index. On the left-hand side, there are four documents, each with a title of a film. The inverted index on the right-hand side consists of a list of all the unique words that appear in any of the left documents, and for each word, a list of the documents in which it appears. Lucene uses the analysis process for creating the inverted index. This process includes tokenization, lowercasing, and stemming of words. This inverted index is particular of interest, since Elasticsearch uses this structure and therefore allows very fast full-text searches [45, 49].

- **Search**: As mentioned before, Elasticsearch provides search and analysis capabilities. The search in Elasticsearch is near real-time. This means that although documents are indexed immediately after they are successfully added to an index, they do not appear in the search results until the index is refreshed. Elasticsearch does not refresh the indices after each update. Instead it makes the use of a specified time interval, also called refresh interval, to perform this operation. By default, the refresh interval is one second. Since refreshing is costly in terms of disk I/O, it can affect the indexing performance. For that reason, increasing the refresh interval before updating a large number of documents is useful [65]. Elasticsearch provides a Search API supports `GET` and `POST` methods and enables to search across multiple indices [23]. However, more complex searches can be

[42]http://www.elastic.co/products/shield



1. The hills have eyes
2. Dawn of the dead
3. Shaun of the dead
4. Night of the living dead

term	frequency	documents
• dawn	1	2
• dead	3	2,3,4
• eyes	1	1
• have	1	1
• hills	1	1
• living	1	4
• night	1	4
• of	3	2,3,4
• shaun	1	3
• the	4	1,2,3,4

Figure 5.8.: Lucene's inverted index [57]

accomplished by using the Query DSL which allows for using Queries and Filters [67].

- **Query DSL, Query and Filter**: The Query DSL is Elasticsearch's flexible and expressive search language that exposed most of the power of Lucene through a simple JSON interface. This language allows for complex queries and is accessible over the REST interface [97, 49]. Although, it is denoted as a "Query" DSL, it also contains a "Filter" DSL [45]. A search can be performed in two ways: in a form of a query or in a form of a filter. The main difference between them is that a query calculates a relevance score of the returned documents whereas the filter does not. Due to this, and the fact that filter can be cached, searching via filters is faster than via queries [65, 45]. A filter asks a yes/no question of every documents, whereas the query also asks the question: 'How well does this document match?' In Elasticsearch, there are different types of queries, like basic queries, compound queries, full-text search queries, and pattern queries, to name a few [49, 45]. Basic queries allow for searching for a part of the index. Furthermore, they allow for nesting other queries inside the basic query. Compound queries allow combining multiple queries or filters inside them. Full-text search queries support full-text searching, analyzing their content and providing Lucene query syntax. Last but not least, pattern queries support various wildcards in queries. Basic queries include e.g., `term`, `match`, and `indices` queries. The aforementioned `match` query can be also categorized to the group of full-text search queries. This also applies to the `prefix` query that can be included to the group of pattern queries [49, 67, 69].

Since Elasticsearch provides numerous queries, only queries that were identified during the key feature extraction are described in the following in more detail:

- **Term**: The `term` query allows to search for an exact word. For instance, it can be used for situation when the whole content of the index has to be returned in order to create aggregations [91].
- **Match**: The `match` query is used when a full-text search query is required. It takes the query's input parameters, analyzes them, and constructs an appropriate output of the query. It is a high-level full-text query, meaning that this query knows how to deal with full-text and exact-value fields [67, 49, 91].
- **Prefix**: The `prefix` query allows to match documents that have the value in a certain field and starts with a given prefix. This query is commonly used for the autocomplete functionality where the user provides a text and all documents are returned that have terms that start with the given text [67, 91].

A filter is used for fields that contain exact values and its goal is to reduce the number of documents that to be examined by the request. The most important filters are the `term` filter, `terms` filter, `range` filter, and the `bool` filter, to name a few. Additionally, Elasticsearch provides four `geo_point` filters that can be used in order to include or exclude documents by geolocations:

- **`geo_bounding_box`**: This filter finds geo-points that fall within a specific rectangle.
- **`geo_distance`**: This filter finds geo-points within a specified distance of a central point.
- **`geo_distance_range`**: This filter finds geo-points within a specified minimum and maximum distance from a central point.
- **`geo_polygon`**: This filter finds geo-points that fall within a specified polygon [45].

- **Geo-aggregations**: Geo-aggregations can be used to cluster `geo_point` documents into manageable buckets and thus be able to provide the user information on a map. Geo-aggregations belong to the group of bucket aggregations (see Section 5.4.2). In total, Elasticsearch provides three geo-aggregations:
  - **`geo_distance`**: This aggregation groups documents into concentric circles around a central point.
  - **`geohash_grid`**: This aggregation groups documents by geohash cell in order to display them on a map. This is the only geo-aggregation which is used by Kibana.
  - **`geo_bounds`**: This aggregation returns the latitude and longitude coordinates of a bounding box that encompasses all of the geo-points. This aggregation is especially useful for choosing the correct zoom level on a map [45].

- **Circuit breaker**: Elasticsearch provides circuit breakers that help prevent Elasticsearch from using too much memory in certain functionalities. Elasticsearch has a family of circuit breakers, all of which ensure that memory limits are not exceeded:
    - **Field data circuit breaker**: By default, this circuit breaker limits the size of field data to 60% of the heap. This means that no more than 60% of heap is allows to be used for the field data cache.
    - **Request circuit breaker**: By default, this circuit breaker estimates the size of structures required to complete other parts of a request, such as creating buckets and limits them to 40% of the heap. This means that the execution of requests are rejected if the total estimated memory used is higher than the 40% of the heap.
    - **Total circuit breaker**: By default, This circuit breaker wraps the field data and request circuit breakers in order to ensure that the combination of them do not use more than 70% of the heap [45, 84].
- **Transaction log**: The transaction log records every operation in Elasticsearch as it happens. After an Elasticsearch index failure, transaction logs are replayed in the recovery process to make sure that none of the changes regarding to the Elasticsearch index are lost. The creation of transaction logs and the recovery process happen automatically [91, 45].
- **Plugin**: Elasticsearch provides a way to enhance its core capabilities by adding custom function in form of plugins. Plugins include analysers, mapping types, native scripts, and so on [102]. Well known Elasticsearch plugins are:
    - **Bigdesk plugin**: The Bigdesk[43] plugin helps to analyze the nodes across the cluster with the help of charts and various statistics to the JVM.
    - **Head plugin**: The Head[44] plugin is capable of generating statistics of the cluster. It also browses and performs structured queries on Elasticsearch indices.

5.4. Kibana

Kibana, which is developed by Rashid Khan, is the open-source[45] front end system in the ELK stack. It is a highly scalable and near real-time web based interface for Logstash and Elasticsearch that allows one to efficiently search, graph, and analyze a mountain of collected log data. Kibana is a data analysis portal that interacts with the Elasticsearch RESTful interface to retrieve data from Elasticsearch and uses data visualization and analysis techniques to gain a deeper insight into the information gathered by Logstash

[43]http://bigdesk.org/
[44]http://mobz.github.io/elasticsearch-head/
[45]Kibana is free under the Apache 2.0 License [17].

[57, 12, 5, 111]. Kibana is based on HTML, JavaScript, and Bootstrap. It requires a web server, included in the Kibana 4 package, and it is fully compatible with any modern browser. Despite the strong integration between Elasticsearch and Kibana, the latter is not a requirement for querying the search cluster. Kibana supports time-based comparisons, easy creation of graphical data representations like plots, charts, and maps, flexible and responsive web interface, and a powerful search syntax [64, 89, 7, 88, 52, 49]. By default, Kibana does not provide any authentication or authorization mechanism [6]. However, the chargeable Elasticsearch plugin Shield[46] provides a role-based access control so that Kibana users have to authenticate in order to access to Elasticsearch data [37]. [49] describes the functionalities of Kibana in more than 200 pages. However this thesis aims to provide a holistic view on the key features of Kibana, therefore some functionalities are not described in depth. Furthermore, in this thesis Kibana version 4.3.0 is used. It provides many more features than Kibana 3.

5.4.1. Conceptualization of Kibana

Figure 5.9 delineates the conceptualization of Kibana. Kibana has a *"Web interface"*. Kibana provides two main capabilities, namely *"Aggregation"* and *"Visualization type"*. An *"Aggregation"* consists of the two classes *"Metric"* and *"Bucket"*. The former consists of the *"Average"*, *"Unique count"*, *"Count"*, *"Percentile"*, *"Percentile ranks"*, *"Min"*, *"Max"*, and *"Sum"*. *"Bucket"* aggregations comprise *Histogram"*, *"Date histogram"*, *"Range"*, *"Date range"*, *"IPv4 range"*, *"Terms"*, *"Filters"*, *"Significant terms"*, and *"Geohash"*. The *"Pie chart"*, *"Area chart"*, *"Line chart"*, *"Vertical bar chart"*, *"Data table"*, *"Markdown widget"*, *"Tile map"*, and *"Metric"* belong to the class *"Visualization type"*. Hereby, the *"Visualization type" "Metric"* is used to visualize a single number for various *"Metric"* aggregations. Furthermore, Kibana provides *"Dashboard"*, *"Time filter"*, and *"Search bar"* capabilities. These key features are described in the following.

5.4.2. Key Features

In the following, Kibana's key features and capabilities are elaborated on:

- **Web interface**: Kibana's web interface consists of four main tabs:
    - **Discover**: The discover page gives an overview of the data including listings of indices, listings of fields, and showing text contained in fields. In addition, this page allows the user to view all data stored in various indices by changing the index pattern. This page is used to perform interactive searches like free text searches, field-based searches, and range-based searches on the indexed Elasticsearch data [17, 49]. Particularly, Figure 5.10 shows that the discover page consists of a toolbar, an index name, a field list, a histogram, a hits counter, and document data. The toolbar comprises the search bar, and option buttons such as new search, save search, load saved search, and settings. Furthermore, search query results highlight the matching documents.

[46]http://www.elastic.co/products/shield

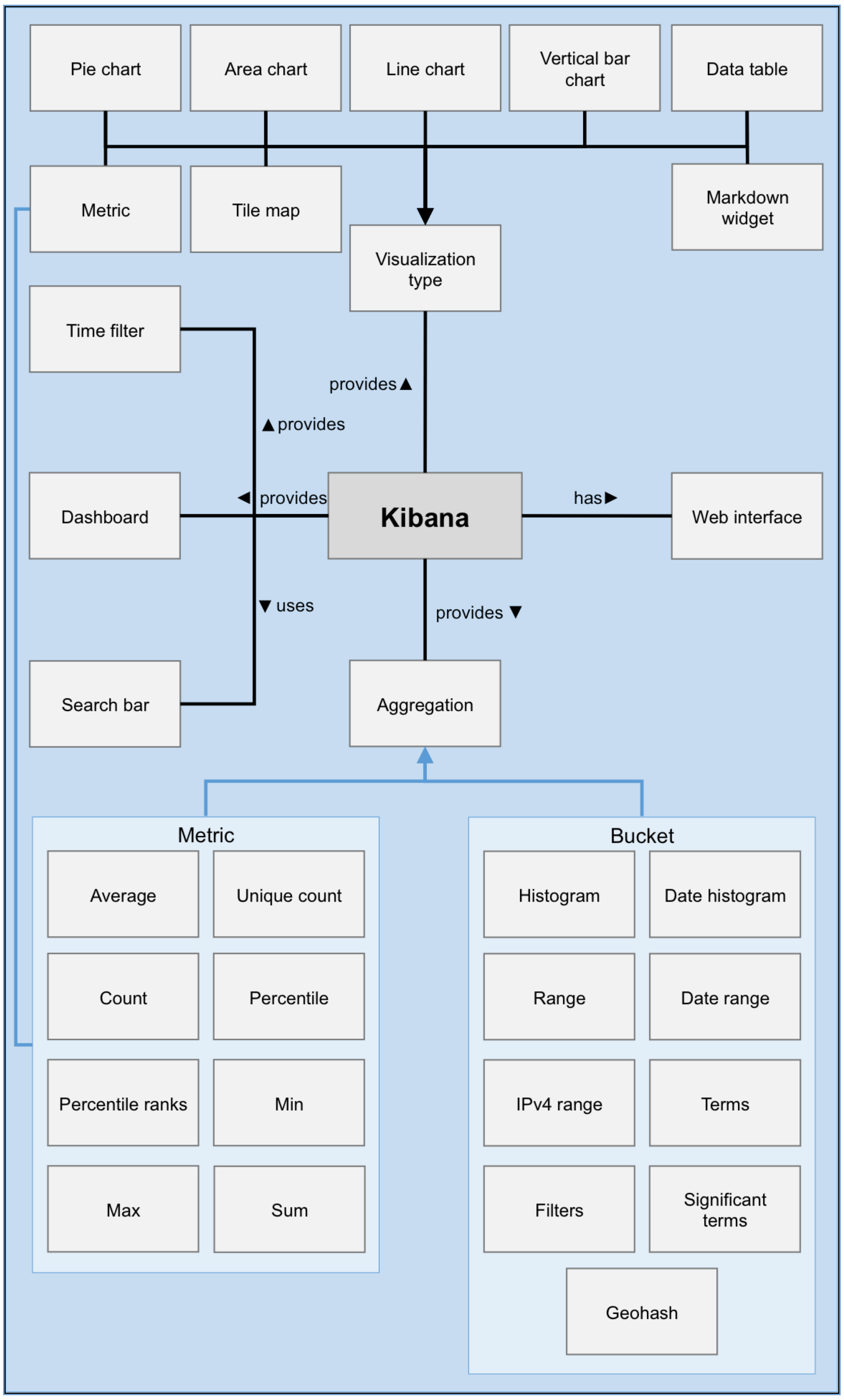
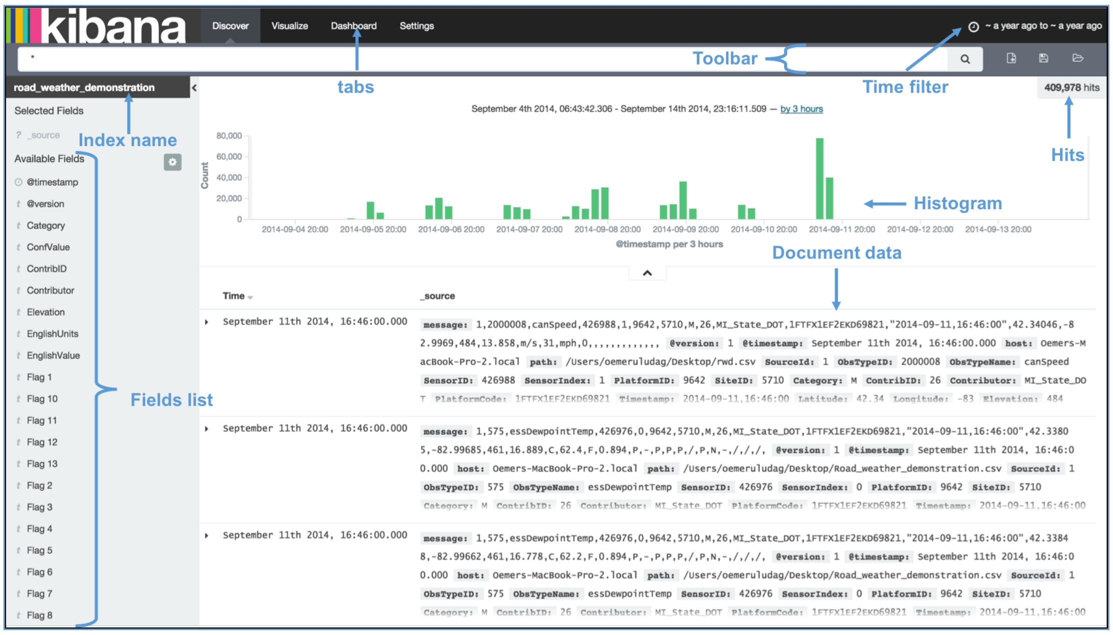


Figure 5.9.: Conceptualization of Kibana

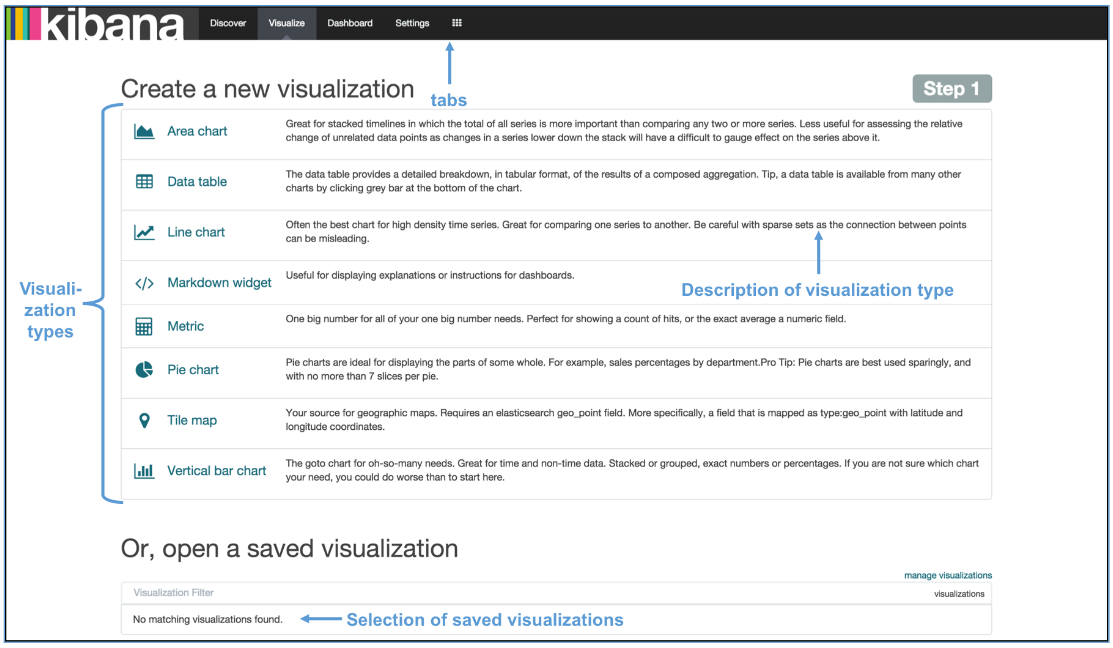
The time filter specifies which data of a particular time interval is contained. The discover page also contains a histogram, which shows the distribution of all documents matching the time filter in the selected index. The index name shows the name of the selected index. The fields list presents all fields within the selected index. The hits counter display the number of matching documents in the selected time interval. Last but not least, the document data displays all the documents along with the date in the entire field as selected [49].



```
September 11th 2014, 16:46:00.000   message: 1,2000008,canSpeed,426988,1,9642,5710,M,26,MI_State_DOT,1FTFX1EF2EKD69821,"2014-09-11,16:46:00",42.34046,-8
                                    2.9969,484,13.858,m/s,31,mph,0,,,,,,,,,,,,,  @version: 1  @timestamp: September 11th 2014, 16:46:00.000  host: Oemers-M
                                    acBook-Pro-2.local  path: /Users/oemeruludag/Desktop/rwd.csv  SourceId: 1  ObsTypeID: 2000008  ObsTypeName: canSpeed
                                    SensorID: 426988  SensorIndex: 1  PlatformID: 9642  SiteID: 5710  Category: M  ContribID: 26  Contributor: MI_State_DO
                                    T  PlatformCode: 1FTFX1EF2EKD69821  Timestamp: 2014-09-11,16:46:00  Latitude: 42.34  Longitude: -83  Elevation: 484

September 11th 2014, 16:46:00.000   message: 1,575,essDewpointTemp,426976,0,9642,5710,M,26,MI_State_DOT,1FTFX1EF2EKD69821,"2014-09-11,16:46:00",42.3380
                                    5,-82.99685,461,16.889,C,62.4,F,0.894,P,-,P,P,P,/,P,N,-,/,/,/,  @version: 1  @timestamp: September 11th 2014, 16:46:0
                                    0.000  host: Oemers-MacBook-Pro-2.local  path: /Users/oemeruludag/Desktop/Road_weather_demonstration.csv  SourceId: 1
                                    ObsTypeID: 575  ObsTypeName: essDewpointTemp  SensorID: 426976  SensorIndex: 0  PlatformID: 9642  SiteID: 5710
                                    Category: M  ContribID: 26  Contributor: MI_State_DOT  PlatformCode: 1FTFX1EF2EKD69821  Timestamp: 2014-09-11,16:46:00

September 11th 2014, 16:46:00.000   message: 1,575,essDewpointTemp,426976,0,9642,5710,M,26,MI_State_DOT,1FTFX1EF2EKD69821,"2014-09-11,16:46:00",42.3384
                                    8,-82.99662,461,16.778,C,62.2,F,0.894,P,-,P,P,P,/,P,N,-,/,/,/,  @version: 1  @timestamp: September 11th 2014, 16:46:0
                                    0.000  host: Oemers-MacBook-Pro-2.local  path: /Users/oemeruludag/Desktop/Road_weather_demonstration.csv  SourceId: 1
                                    ObsTypeID: 575  ObsTypeName: essDewpointTemp  SensorID: 426976  SensorIndex: 0  PlatformID: 9642  SiteID: 5710
                                    Category: M  ContribID: 26  Contributor: MI_State_DOT  PlatformCode: 1FTFX1EF2EKD69821  Timestamp: 2014-09-11,16:46:00
```

Figure 5.10.: Kibana's discover page

- **Visualize**: The visualize page is the most important page in Kibana 4. This page supports the creation of a visualization based on the selection of different visualization types or to load a saved visualization. The *Visualize* page provides an overview of different types of visualization provided and how to create a new visualization from a new search or saved search. Visualizations can also be shared with other users who have access to the specific Kibana instance. There are several different *Visualization types*, including area chart, data table, line chart, markdown widget, metric, pie chart, tile map, and vertical bar chart. The aforementioned visualizations can be saved and viewed individually or can be utilized in multiple dashboards. All visualizations in Kibana are using the aggregation features of Elasticsearch [17, 49]. Figure 5.11 provides an insight into the initial *Visualize* page which consists of two parts. The first part deals with the creation of new visualizations. Herein, all visualization types with their descriptions are listed. The second part



Create a new visualization

Area chart	Great for stacked timelines in which the total of all series is more important than comparing any two or more series. Less useful for assessing the relative change of unrelated data points as changes in a series lower down the stack will have a difficult to gauge effect on the series above it.
Data table	The data table provides a detailed breakdown, in tabular format, of the results of a composed aggregation. Tip, a data table is available from many other charts by clicking grey bar at the bottom of the chart.
Line chart	Often the best chart for high density time series. Great for comparing one series to another. Be careful with sparse sets as the connection between points can be misleading.
Markdown widget	Useful for displaying explanations or instructions for dashboards.
Metric	One big number for all of your one big number needs. Perfect for showing a count of hits, or the exact average a numeric field.
Pie chart	Pie charts are ideal for displaying the parts of some whole. For example, sales percentages by department.Pro Tip: Pie charts are best used sparingly, and with no more than 7 slices per pie.
Tile map	Your source for geographic maps. Requires an elasticsearch geo_point field. More specifically, a field that is mapped as type:geo_point with latitude and longitude coordinates.
Vertical bar chart	The goto chart for oh-so-many needs. Great for time and non-time data. Stacked or grouped, exact numbers or percentages. If you are not sure which chart your need, you could do worse than to start here.

Or, open a saved visualization

No matching visualizations found.

Figure 5.11.: Kibana's initial visualize page

enables opening already created and saved visualizations. Furthermore, it provides an index tab for switching between the different main tabs. Figure 5.12 illustrates the visualize page after the selection and creation of a visualization type. The main part of this page is the aggregation designer and visualization canvas. The aggregation designer is displayed on the left-hand side of the visualize page. The aggregation designer is utilized for configuring metric and bucket aggregations. The builder consists of two tabs, namely data and options. The first tab is used for specifying the metric and bucket aggregations. The second part is used to display the various types of view options associated with each visualization type. The preview canvas is utilized for displaying a preview of the visualization which is created by using the aggregation designer. Changes in the aggregation designer are automatically propagated to the preview canvas.

- **Dashboard**: The dashboard page represents collections of saved visualizations and searches that can be arranged in any order. Visualizations can be used on multiple dashboards and changes on the visualization are reflected to all of them automatically. A dashboard is used to combine different types of created visualizations and display them on a single page. The visualizations added to the dashboard can be arranged in any way as per the user's requirements. The visualizations can easily be moved, resized, edited, and removed. A dashboard can be saved and shared easily [17, 49]. Figure 5.13 shows that the dashboard page contains a dashboard canvas which consists

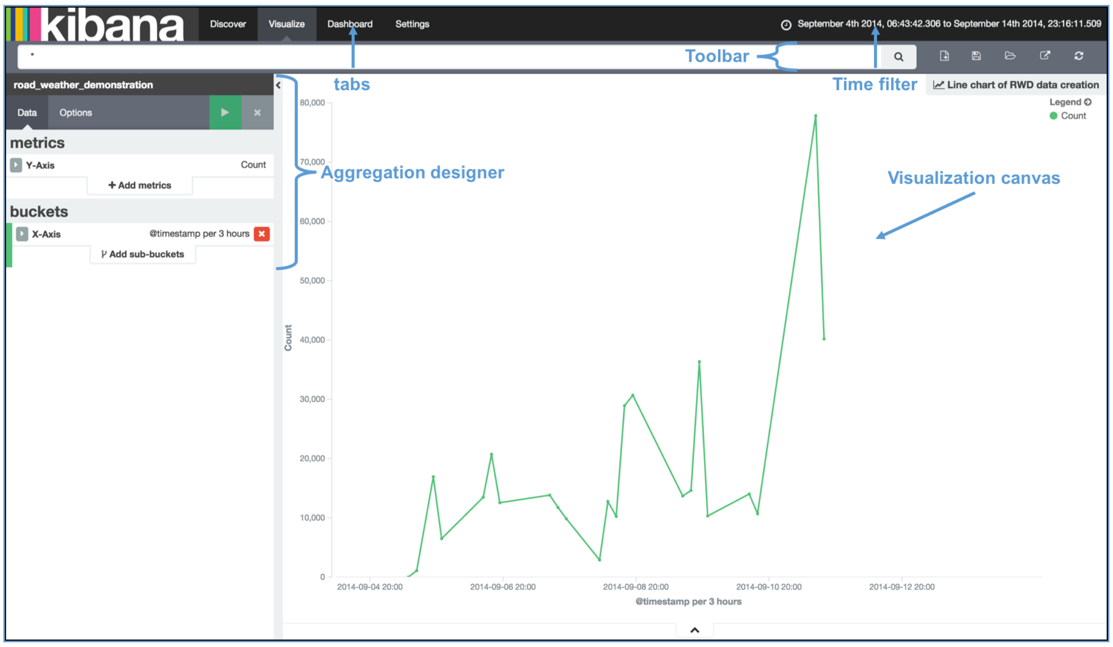


Figure 5.12.: Kibana's visualize page after creation of a visualization

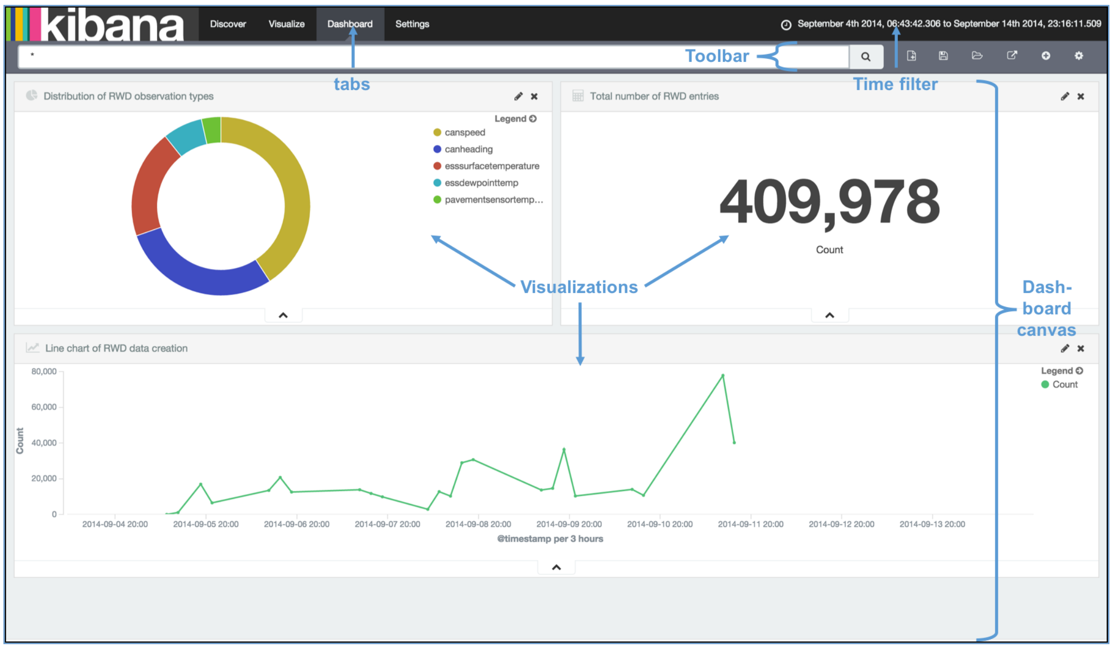
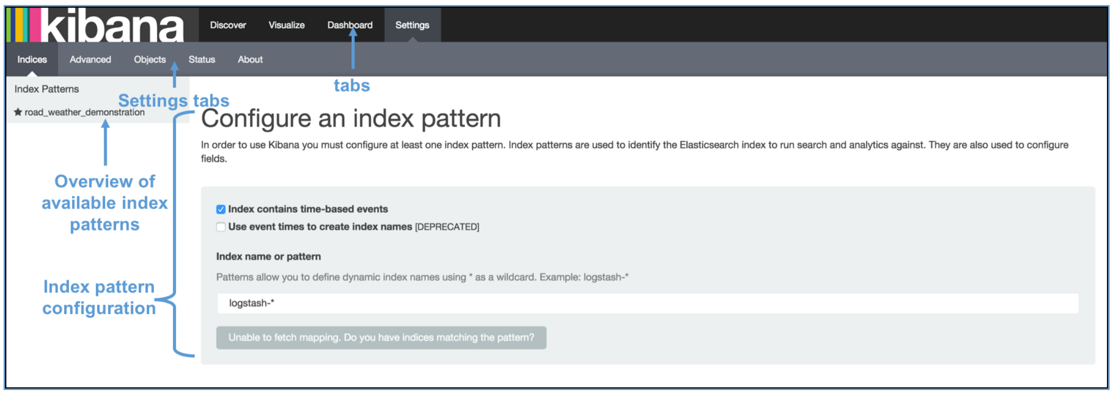


Figure 5.13.: Kibana's dashboard page

of multiple visualizations.

- **Settings**: The settings page provides various functionalities of configuring index patterns, scripted fields, and saved objects followed by information about the current Elasticsearch server status, installed plugins and the Kibana version. The settings page contains in total four tabs, namely indices, advanced, objects, and about.
  The indices tab is the default tab that opens whenever the user starts Kibana or clicks on the settings page. As Elasticsearch uses an index to process data, it represents the most important component, without which the user cannot analyze data, create visualizations, or build dashboards [49]. Figure 5.14 presents the initial indices tab when opened. On the left-hand side, it provides an overview of available index patterns. The center of the indices tab allows configuring an index pattern. By default, an index contains time-based events. If the data does not contain any time-based event, then the checkbox can be unchecked in order to configure an index. While configuring indices, the user can use patterns such as $*$. The user can also use the date format pattern to add indices that have the event timestamp attached to it [49]. For displaying additional properties related to an index, the index can be clicked. Figure 5.15 illustrates the indices tab after an index is clicked. This page lists every field in the index and the field's associated core type as recorded by Elasticsearch. This page allows changing the core types of fields and also enables setting the index as default index, refreshing the field list of the index and removing the index pattern. The advanced tab provides the option of editing the settings that directly control Kibana. The settings which are listed on this page can be either undocumented, unsupported, or experimental. Tweaking this settings can cause unexpected behavior [49]. Figure 5.16 provides a list of advanced settings. For instance, the `discover:sampleSize` can be changed in order to adjust the number of rows to show in a table. Figure 5.17 shows the objects tab that is used to view, edit, export, and import saved objects, such as saved searches, saved



Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events
[ ] Use event times to create index names [DEPRECATED]

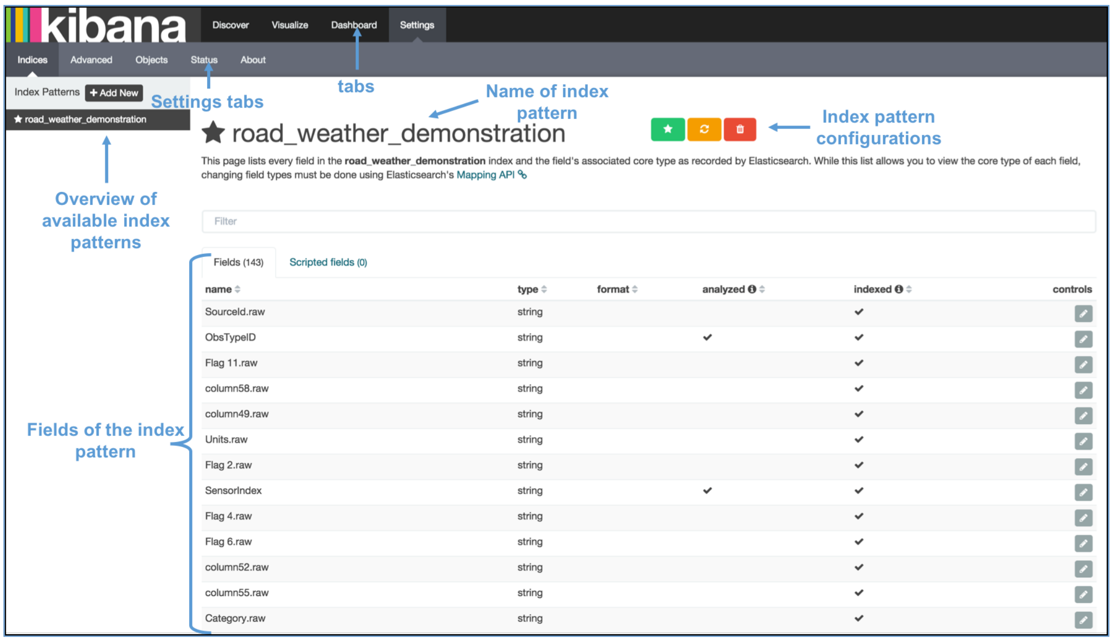
Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

logstash-*

Unable to fetch mapping. Do you have indices matching the pattern?

Figure 5.14.: Kibana's settings page with index pattern configuration



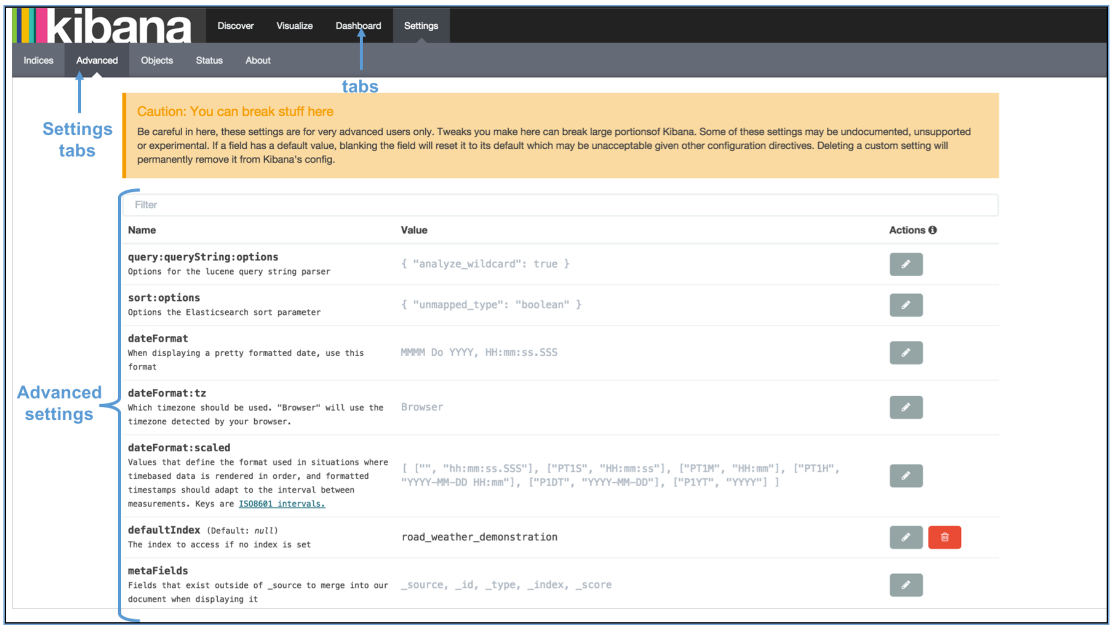
★ road_weather_demonstration

This page lists every field in the **road_weather_demonstration** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's Mapping API

Fields (143) | Scripted fields (0)

name	type	format	analyzed	indexed	controls
SourceId.raw	string			✓	
ObsTypeID	string		✓	✓	
Flag 11.raw	string			✓	
column58.raw	string			✓	
column49.raw	string			✓	
Units.raw	string			✓	
Flag 2.raw	string			✓	
SensorIndex	string		✓	✓	
Flag 4.raw	string			✓	
Flag 6.raw	string			✓	
column52.raw	string			✓	
column55.raw	string			✓	
Category.raw	string			✓	

Figure 5.15.: Kibana's settings page with index pattern settings



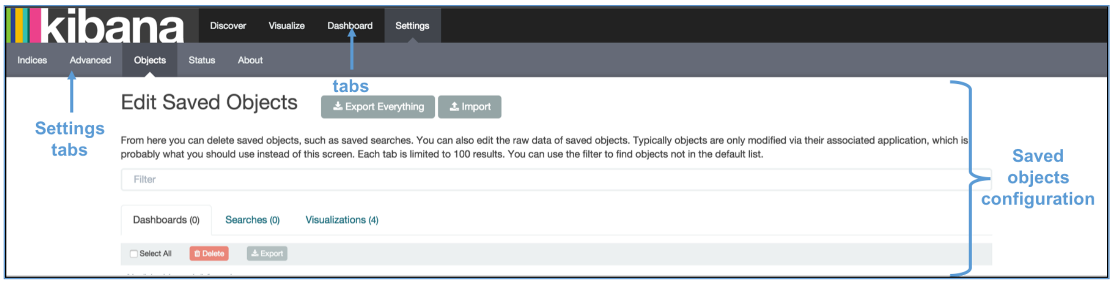
Caution: You can break stuff here

Be careful in here, these settings are for very advanced users only. Tweaks you make here can break large portionsof Kibana. Some of these settings may be undocumented, unsupported or experimental. If a field has a default value, blanking the field will reset it to its default which may be unacceptable given other configuration directives. Deleting a custom setting will permanently remove it from Kibana's config.

Name	Value	Actions
query:queryString:options Options for the lucene query string parser	{ "analyze_wildcard": true }	
sort:options Options the Elasticsearch sort parameter	{ "unmapped_type": "boolean" }	
dateFormat When displaying a pretty formatted date, use this format	MMMM Do YYYY, HH:mm:ss.SSS	
dateFormat:tz Which timezone should be used. "Browser" will use the timezone detected by your browser.	Browser	
dateFormat:scaled Values that define the format used in situations where timebased data is rendered in order, and formatted timestamps should adapt to the interval between measurements. Keys are ISO8601 intervals.	[["", "hh:mm:ss.SSS"], ["PT1S", "HH:mm:ss"], ["PT1M", "HH:mm"], ["PT1H", "YYYY-MM-DD HH:mm"], ["P1DT", "YYYY-MM-DD"], ["P1YT", "YYYY"]]	
defaultIndex (Default: *null*) The index to access if no index is set	road_weather_demonstration	
metaFields Fields that exist outside of _source to merge into our document when displaying it	_source, _id, _type, _index, _score	

Figure 5.16.: Kibana's settings page with advanced settings

visualizations, and saved dashboards. Figure 5.18 illustrates the Status tab. This tab provides information about the current status of the Elasticsearch



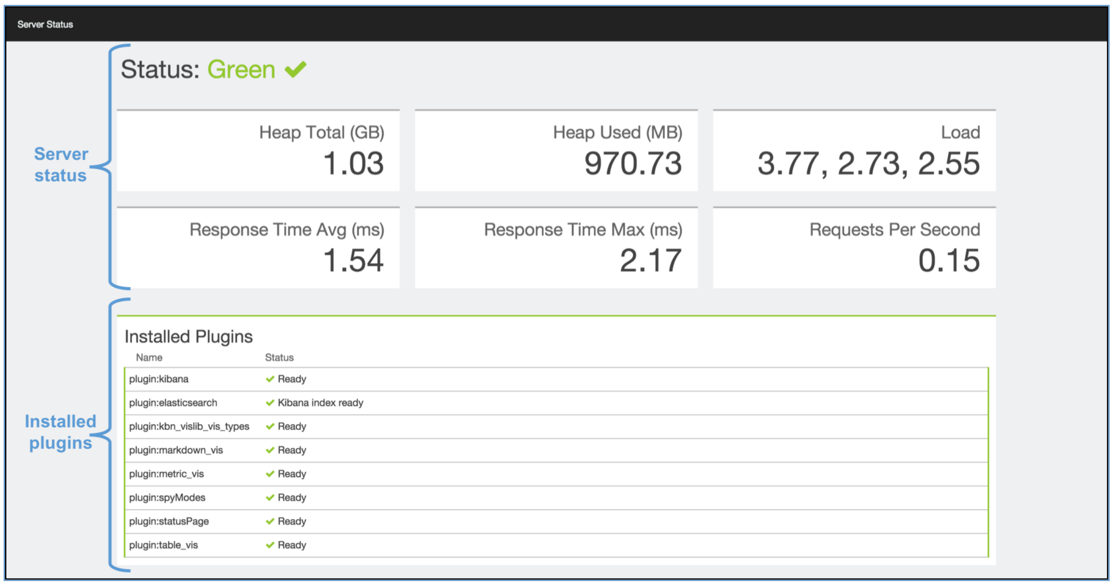
Edit Saved Objects

From here you can delete saved objects, such as saved searches. You can also edit the raw data of saved objects. Typically objects are only modified via their associated application, which is probably what you should use instead of this screen. Each tab is limited to 100 results. You can use the filter to find objects not in the default list.

Dashboards (0) Searches (0) Visualizations (4)

Figure 5.17.: Kibana's settings page with saved objects settings

server, including the total heap size, used heap size, average response time, max response time, and requests per seconds, to name a few. Furthermore, this page shows installed Kibana plugins. The about tab provides details of



Status: Green ✓

Heap Total (GB)	Heap Used (MB)	Load
1.03	970.73	3.77, 2.73, 2.55

Response Time Avg (ms)	Response Time Max (ms)	Requests Per Second
1.54	2.17	0.15

Installed Plugins

Name	Status
plugin:kibana	✓ Ready
plugin:elasticsearch	✓ Kibana index ready
plugin:kbn_vislib_vis_types	✓ Ready
plugin:markdown_vis	✓ Ready
plugin:metric_vis	✓ Ready
plugin:spyModes	✓ Ready
plugin:statusPage	✓ Ready
plugin:table_vis	✓ Ready

Figure 5.18.: Kibana's settings page with Elasticsearch's server status

Kibana, such as the running version, the build number, and the commit hash (see Figure 5.19).

- **Time filter**: The time filter is a powerful component that helps to drill down on data on a per time basis. The time filter supports seeing data of a specified range. By default, it displays data of the last 15 minutes. The time range can be changed by using the time picker. After clicking on the time picker, three options for selecting a time filter are displayed: quick, relative, and absolute. Figure 5.20 provides an insight into the various time filter options. The quick time filter helps to filter data quickly on some already available time ranges like today, this week, this

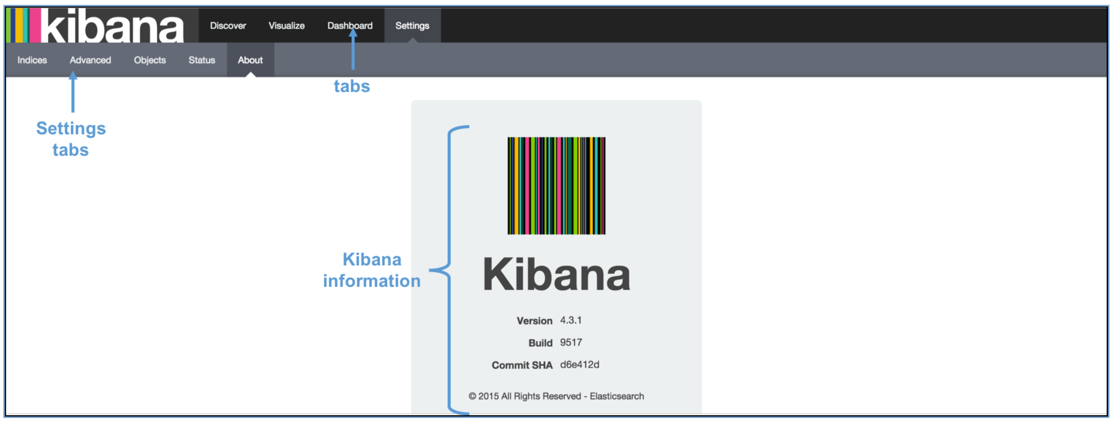


Figure 5.19.: Kibana's settings page with information about Kibana version

year, or last 5 years, and so on. The relative time filter supports to filter data based on relative time from the current time. By default, the this filter is set to relative with 15 minutes ago from now. The absolute time filter selects data based on a range of dates selected for from and to a date and time. The auto-refresh setting is used to set a refresh interval. However, the timer filter can also specified by using click and drag on an area of a histogram or other charts [17].

- **Search bar**: The user can query Elasticsearch data by using normal text queries or the Query DSL syntax in the search bar [88]. The Query DSL syntax can be found in [70]. The search bar provides the following types of searches among others:
  - **Free-text searches**: This search is aimed at filtering documents containing the search term. It searches in all the documents for all the fields containing the searched term. For instance, by specifying the search term `'ELK'` in the search bar, all documents are filtered which contain the term `ELK`.
  - **Boolean searches**: Kibana provides Boolean searches, including `AND`, `OR`, and `NOT` Boolean operators. Moreover, this operators can be combined together in order to perform more sophisticated searches. For example, the Boolean search `"Learning" AND "ELK"` will search for all documents that contain both terms: `"Learning"` and `"ELK"`.
  - **Wildcard searches**: Kibana also provides single and multiple character wildcard searches within single terms. For example, using the term `"plan*"` will search for all documents that have terms, such as `"plans"`, `"plant"`, `"planet"`, or `"planting"`, and so on.
  - **Field searches**: Field searches allow searching for specific values or ranges of values for fields in the indexed document. Field searches are performed by using the field name and the `:` character, followed by a value for the field which should be filtered on. For instance, the search term: `title :`

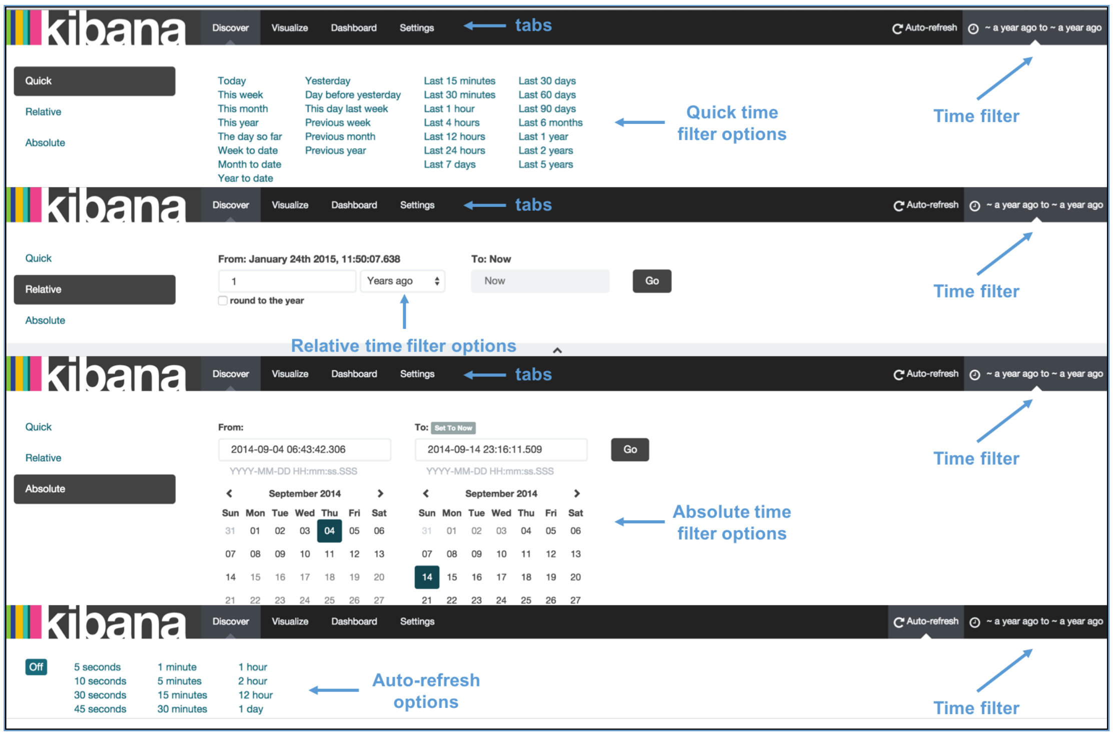


Figure 5.20.: Kibana's time filter settings

`"Learning ELK" AND category : "technology"` will search for all documents that have the title `Learning ELK` and the category `technology`.

- **Range searches**: Range searches aim to search for a range of values for a field. For instance, `volume : [ 100000 TO 200000]` will filter all documents which have a volume range from 100,000 to 200,000.

Additionally, the performed searches can be saved or already created searches can be loaded [17].

- **Aggregation**: In Kibana, aggregations are collections of data that are stored in buckets. Aggregations have grown from Elasticsearch's facets module. Aggregations are used for generating analytical information over stored documents. Aggregations are used for near real-time data analytics. There are different types of aggregations, namely metric and bucket aggregations [49].

- **Bucket aggregation**: Inside the bucket aggregations, buckets are created to store various documents and to group the stored documents. The specification of which bucket contains a document is based on the value of a specific field. Buckets are very similar to the `GROUP BY` functionality in SQL. Bucket aggregations can be combined with other types of aggregations, creating sub-aggregations. Bucket aggregations comprise date histogram, histogram, range, date range, IPv4 range, terms, filters, significant terms, and geohash [17, 49].

- **Histogram**: The histogram bucket aggregation buckets documents for a particular numeric interval specified in a selected numeric value field. This aggregation is like a range aggregation with equal intervals [17, 49].
- **Date histogram**: The date histogram is similar to histogram aggregation except that date histogram is used for a date/time field, whereas histogram is used for a numeric value field [49].
- **Range**: The range bucket aggregation is used to specify an interval of range in which each interval represents a bucket and to aggregate numeric or date/time fields. It is like a histogram except that the range bucket aggregation allows configuring different ranges as per the requirements, manually [17, 49].
- **Date range**: The date range bucket aggregation is utilized to specify an interval of range in date format in which each interval represents a bucket and to aggregate date/time fields. The interval size has to be specified manually [49].
- **IPv4 range**: The IPv4 bucket aggregation is utilized to specify an interval of range in IP format in which each interval represents a bucket [49].
- **Terms**: The terms bucket aggregation is utilized to create buckets based on the values of a field. Hereby, the buckets are created dynamically. In this aggregation, a field has to be specified that creates a bucket for all values that exist in the field and puts in each document that has a value in this field. This aggregation is similar to the `GROUP BY` functionality in SQL [49].
- **Filters**: The filter aggregation helps to create visualizations based on search queries. Hereby, a filter is specified for each bucket on the basis of which of the documents match the filter that fits into that bucket [49].
- **Significant terms**: The significant terms aggregation is utilized in order to find uncommonly common terms in the existing data. For instance, if a term is present in 10 documents out of 10,000 indexed documents, but occurs in 8 documents out of 50 documents returned from the search query, then this term is significant. This aggregation is helpful for creating subsets of the data to detect outliers and find anomalies. Possible use cases include finding trending topics on Twitter or detecting credit card fraud [49].
- **Geohash**: The geohash aggregation is utilized to create buckets based on `geo_point` fields and categorizes those points into buckets [49].
- **Metric aggregation**: The metric aggregation represents computations performed over a number of documents. This aggregation is utilized after creating a bucket aggregation. Subsequently, the metric aggregation is specified to calculate the value of each bucket, so this aggregation runs on each bucket and returns a single

value result per bucket. In visualizations, bucket aggregation usually determines the first dimension of the chart, whereas the value calculated by the metric aggregation is referred as the second dimension. Types of available metric aggregations are average, sum, unique count, count, min, max, percentile, and percentile ranks [17, 49].

- **Average**: The average metric aggregation is utilized to compute the average value of a numeric field stored in each bucket. The result for each bucket is the average of all values in that field [49].
- **Unique count**: The unique count metric aggregation is used to count the number of unique values for a field stored in each bucket. The result for each bucket is the total number of unique values for that field [49].
- **Count**: The main purpose of the count metric aggregation is to calculate the count of the number of fields in each bucket in a bucket aggregation [17].
- **Percentile**: The percentile metric aggregation is utilized to compute percentiles over numeric fields stored in buckets. Its main difference to other metric aggregations is that it stores multiple values per bucket. This is the reason why it is included to the category of multivalue metric aggregations. When specifying the percentile metric aggregation, a numeric value field has to be specified along with multiple percentage values. The result of this aggregation is the value for which a certain percentage of documents is inside the value [49].
- **Percentile ranks**: The percentile ranks metric aggregation is utilized to compute single or multiple percentile ranks over a numeric field, which has been extracted from documents and stored in buckets. Besides the percentile metric aggregation, the percentile ranks metric aggregation is also categorized as multivalue metric aggregations. This aggregation is used to present the percentage of values occurring that are below a certain value [49].
- **Min**: The min metric aggregation is utilized to compute the minimum value of a numeric field stored in each bucket. The result for each bucket is the minimum value for that field found in documents stored [49].
- **Max**: The max metric aggregation is utilized to compute the maximum value of a numeric field stored in each bucket. The result for each bucket is the maximum value for that field found in documents stored [49].
- **Sum**: The sum metric aggregation is utilized to compute the sum of a numeric field stored in each bucket. The result for each bucket is the sum of all values in that field [49].
- **Visualization type**: There are several different visualization types, including area chart, data table, line chart, markdown widget, metric, pie chart, tile map and

vertical bar chart. All visualizations in Kibana are using the aggregation feature of Elasticsearch [17, 49].

- **Pie chart**: Occasionally, pie charts are used to show parts of a whole or a percentage relationship. The relationship represents the distribution of data over multiple slices in a pie chart. A slice of the pie chart is determined by metrics aggregations, which can have the values count, sum, or unique count. Bucket aggregation specifies the type of data that has to be represented in one chart. The pie chart can be visualized either as a pie or as a donut [17, 49].

- **Area chart**: The area chart displays a line chart with filled areas below the lines. It can be used to visualize the total contribution of several different series, which is why it is especially useful to create stacked timelines. The areas can be displayed as stacked, overlapped, or some other variations. It uses metrics as y-axis and buckets for x-axis. Moreover, it enables defining sub-aggregations in buckets. It provides the functionality of split charts[47] or split area[48] [17, 49, 68].

- **Line chart**: The line chart displays aggregated data in the form of lines. The lines can be visualized on a scale of linear, logarithmic, or square root scale. The line chart can be used to display data over a period of time like high density time series and is often useful when comparing one series with another [17, 49, 68].

- **Vertical bar chart**: The vertical bar chart can be used for a variety of purposes and is suited to both time-based data and non-time-based data. It can be displayed either as stacked, percentage, or grouped. The y-axis is used for metrics, whereas the x-axis is used for buckets [17, 49].

- **Data table**: The data table displays tables of aggregated data stored in Elasticsearch. It provides the results of raw data in tabular format and supports to identify Top-N kinds of aggregations [17, 49].

- **Markdown widget**: The markdown widget is a text entry field that is used to display free-form information or instructions related to the dashboard and can be utilized for any requirements for text on dashboard. The markdown widget acts as additional information and can be used easily. Kibana renders the entered text and visualized the result on the dashboard [17, 49].

- **Tile map**: The tile map displays a map for results which are based on a geohash aggregation, which groups multiple coordinates into one bucket. The tile map is utilized to locate geographic locations based on coordinates. It requires a `geo_point` type field with latitude and longitude inputs. By default, the tile map displays the data points in the form of circles, where the size depends upon the selected precision. The color of the data point is signified by the actual values calculated by a metric aggregation [17, 49, 68].

[47]Split charts are multiple charts based on different aggregations [17].
[48]Split area is a split based on different aggregations [17].

- **Metric**: The metric visualization displays a single number for various metric aggregations without bucket. For instance, it can be utilized to compute the total number of hits or the sum or average of a field [17, 49, 68].

5.5. Key Findings

Based on the aforementioned key features of the ELK stack, the following key findings are deduced:

- **Scalability**: The ELK stack provides several possibilities of scaling. First of all, Logstash and Elasticsearch can be scaled by adding computing resources (vertical scaling). Additionally, Logstash can be scaled by adding a message queue or by adding more Logstash indexing instances. Elasticsearch can easily scale out horizontally by adding additional Elasticsearch nodes to the existing Elasticsearch cluster. Thus, the ELK stack is capable of vanquishing increasing data volume and data velocity.

- **Support of various data sources and data formats**: Walking along Logstash's event processing pipeline, Logstash's various input plugins determine the ELK stack's support for various data sources. Currently, Logstash comprises 49 different input plugins which help Logstash to tap a huge amount of data sources. Logstash's filter plugins represent the ELK stack's capabilities for ingesting various data formats. At the moment, the number of Logstash's filter plugins amount 42. However, many more different data formats are supported by using the grok filter plugin. On the one hand, specific filter plugins, e.g., csv, xml, and elasticsearch, can be used for structured data whereas on the other hand grok can be utilized for reading unstructured data by making the use of regular expressions and custom/predefined patterns.

- **Capabilities for geospatial analyses**: Geospatial analyses can be performed in Elasticsearch by using various filters and aggregations specially geared for document fields which have the `geo_point` data type. These comprise the `geo_bounding_box`, `geo_distance`, `geo_distance_range`, and `geo_polygon` filters, and additionally the `geo_distance`, `geohash_grid`, and `geo_bounds` aggregations.

- **Lack of data protection mechanisms**: By default, the ELK stack does not ship with any security functions. This is dangerous when users have free access to Elasticsearch via Kibana's user interface without any authentication. Due to this fact, stored data in Elasticsearch would be accessible for anyone in that network. For that reason, Elastic provides the chargeable Shield plugin in order to protect the whole ELK stack. This plugin has a 30 day trial license. At the end of the trial period, one have to subscribe in order to access to the Elasticsearch Cluster Health, Cluster Stats, and Index Stats APIs, since they will be automatically blocked if the new license is not installed. The Shield plugin provides an authentication mechanism for Elasticsearch so that username and password have to be provided in order to communicate with Elasticsearch, a login and session management for Kibana, which provides user authentication and session support, and a field- and document-level security so that individual fields of documents are restricted for unintended users, to name a few.

- **Provision of basic visualization types**: The descriptive study of Kibana already illustrated that Kibana is only capable of basic visualization types, namely *pie chart*, *area chart*, *line chart*, *vertical bar chart*, *data table*, *metric*, *tile map*, and *markdown widget*. Each visualization provides customization functionalities which are constrained. [49] demonstrates all visualization types with their possible customized realizations. However, many other visualization types are not supported in Kibana, e.g., radar charts, time lines, reports, matrices, layer diagrams, cluster maps, or gauges. Also, statistical graphics are not provided by Kibana, e.g., box plots, control charts, scatter plots, or Q-Q plots.

- **Limited data analytics capabilities**: Based on the stated attributes of search-based data discovery tools in Section 2.1, one can realize that the focus of search-based data discovery tools lies on the conflation of structured and unstructured data from various sources, creation of aggregations, summaries, and pre-calculations, and provision of an intuitive user interface for data exploration. By contrasting the aforementioned attributes with the four types of data analytics capabilities in Section 2.2, the data analytics capabilities of search-based data discovery tools cover descriptive analytics. This is also true for the ELK stack. It helps to drill down, query, and aggregate data with the help of bucket/metric aggregations and Query DSL. All in all, the ELK stack facilitates exploratory and descriptive data analytics and is not capable of performing diagnostic, predictive and prescriptive analytics. This means that the ELK stack does not support functionalities for interactive visualization, machine learning, data mining, predictive modeling, decision modeling, simulation, and optimization.

Part IV.

Experiments

6. Experiments

The increasing digitalization of business processes and models of companies broadens the access to yet not accessible meaningful knowledge that enables intelligent services. As the raising digital footprint of customers emerges in an outstanding number and variety of Big Data use cases, numerous technologies and tools in the area of Big Data ecosystem accrue [73, 43]. For the reason that technologies are not ordinarily designed for providing all-inclusive capabilities, their applicability for the different Big Data use cases has to be verified. In order to assess the applicability of the ELK stack for Big Data use cases appropriately, two types of experiments are selected. Firstly, the ELK stack's applicability should be verified with real use cases in the form of implementation experiments. This type of experiment aims to unveil qualitative characteristics of the ELK stack and to demonstrate how it could be used for data analytics. In total, three distinct data analytics experiments should be performed. The selection of the data analytics experiments is based on the conducted survey of [93]. The majority of the survey respondents are analyzing structured data (92%), semi-structured data (54%), e.g., XML, complex data, e.g., hierarchical or legacy sources, event data (54%), unstructured data (45%), and social media data (35%). Since the implementation of structured data is not very challenging, this type of data is neglected. Unfortunately, within this thesis, there is no access to hierarchical or legacy sources so that complex data is also neglected. The remaining four data types are selected for the following experiments:

- **Textual data analysis**: for assessing the ELK stack's capability for semi-structured data (see Section 6.1).
- **Machine-generated data analysis**: for assessing the ELK stack's capability for both semi-structured data and event data (see Section 6.2).
- **Social media data analysis**: for assessing the ELK stack's capability for both unstructured data and social media data (see Section 6.3).

The second type of experiment targets to assess the ELK stack's capability for Big Data use cases quantitatively. This includes the assessment of ELK stack's data ingestion, data indexing, and data querying performances. Thus, three benchmarks are performed within the performance benchmark analysis (see Section 6.4).

6.1. Experiment 1: Textual Data Analysis

This experiment aims to assess the ELK stack's applicability for semi-structured textual data. As a pioneer of this data type, one of the most prominent data file formats in the field of Big Data, namely CSV, is selected [124]. One major goal of this experiment is to find out whether the ELK stack requires time-based data in order to be able to perform analytics or not.

6.1.1. Data Basis and Data Overview

The Research Data Exchange (RDE) is a transportation data sharing system provided by the U.S. Department of Transportation Intelligent Transportation Systems (ITS). It promotes sharing of both archived and near real-time data from multiple sources and multiple modes [82]. The RDE presently has connected vehicle and ITS data from 14 locations to support the analysis and development of connected vehicle applications. The road weather demonstration data environment was created during the Integrated Mobile Observations project demonstration during the 2014 ITS World Congress. For public demonstration, participants were driven in a specially instrumented demo van in a short loop on Belle Isle, Detroit, Michigan, USA. Vehicle sensor data like GPS, wiper status, temperature, humidity, etc., were gathered for the simulated road weather conditions. This data triggered advisories to be issued to the passengers within the van [81]. Data collected by various sensors onboard the van during the demonstration was compiled into a CSV file. The CSV file contains in total 572,030 sensor data entries. The CSV file consists of 33 attributes. The most important attributes are presented below.

- **ObsTypeName**: This field contains a text description for the type of observation made in that record. For instance, observations of air temperature, air pressure, wiper status, and traction control state.
- **Timestamp**: This field contains the UTC date and time at which the observation was recorded at the source sensor.
- **Latitude**: This field is the latitude value of the location at which the recording sensor recorded the observation.
- **Longitude**: This field is the longitude value of the location at which the recording sensor recorded the observation.
- **Observation**: This field contains the recorded data value for that observation in metric units.
- **Units**: This field contains the name of the metric unit for the observation.

6.1.2. Implementation of the ELK Stack

Logstash

Listing 6.1 presents the configuration file of the road weather demonstration data pipeline.

Listing 6.1: Configuration file for textual data analysis

```
input {
    file {
        path => "/Users/oemeruludag/Desktop/rwd.csv"
    }
}
filter{
    csv {
        columns => ["SourceID", "ObsTypeID", "ObsTypeName",
                    "SensorID", "SensorIndex","SiteID",
                    "PlatformID", "Category", "ContribID",
                    "Contributor", "PlatformCode", "Timestamp",
                    "Latitude", "Longitude", "Elevation",
                    "Observation", "Units", "EnglishValue",
                    "EnglishUnits", "ConfValue", "Flag 1",
                    "Flag 2", "Flag 3", "Flag 4", "Flag 5",
                    "Flag 6", "Flag 7", "Flag 8", "Flag 9",
                    "Flag 10", "Flag 11", "Flag 12", "Flag 13"]
            separator => ","
    }
    if ([SourceId] == "SourceId") {
        drop { }
    }
    ruby {
        code =>"
            if !event['Latitude'].nil?
                event['Latitude'] =
                    (event['Latitude'].to_f* 100).round / 100.00
            end
            if !event['Longitude'].nil?
                event['Longitude'] =
                    (event['Longitude'].to_f* 100).round / 100.00
            end"
    }
    mutate {
        add_field => [ "[Location]", "%{Longitude}" ]
        add_field => [ "[Location]", "%{Latitude}" ]
    }
    date {
        match => ["Timestamp" , "YYYY-MM-dd,HH:mm:ss"]
        target => "@timestamp"
    }
}
output{
    elasticsearch {
        hosts => "localhost:9200"
        index => "road_weather_demonstration"
        document_type => "rwd"
        template => "/Users/oemeruludag/Desktop/rwd_temp.json"
        template_name => "road_weather_demonstration"
    }
    stdout{
        codec => rubydebug
    }
}
```

Within the `input` block, the input plugin `file` is used. It enables streaming of events from files. However, the input plugin `file` requires specification of the `path` of the to-be-selected file.
Within the `filter` block, various filter plugins are utilized, namely `csv`, `ruby`, `mutate`, and `date`. Since the road weather demonstration data is stored in the CSV format, the `csv` filter plugin is very helpful due to its capability to take an event field in the CSV format, and subsequently to parse and store it as individual fields. The `columns` field enables definition of a list of column names, in the order they appear in the CSV, which are afterwards transformed to event fields. The `separator` is self-explanatory, which defines the column separator value. The `drop` function is required in order to delete the header line of the data, otherwise Elasticsearch will include them into the data. For executing Ruby code in Logstash's configuration file, the `ruby` filter plugin is utilized. It requires to specify the `code` parameter, which contains the Ruby code. Within this section, the `Latitude` and `Longitude` event fields are adjusted. Particularly, in a first step the `if` clause checks whether `Latitude` and `Longitude` are `nil`[49] or not. In case that both event fields are not `nil`, they are parsed with the function `.to_f` to the data type *float*. In addition, their decimal places are rounded to two decimal places. The reason for this conversion is that Elasticsearch requires a certain format for geolocation fields, namely the fields have to belong to the field data type *float* and they have to possess maximal two decimal places. The `mutate` filter is utilized in order to create a new field `Location` with the `add_field` parameter. The `Location` field consists of the fields `Longitude` and `Latitude`. The order of the specification is nontrivial because Elasticsearch uses the first parameter as the longitude, whereas the second parameter is used as the latitude. A wrong specification results in an erroneous geolocation. As mentioned before, the `date` filter is utilized for parsing dates from fields and then using that date or timestamp as the Logstash timestamp for the event. By default, Logstash chooses a timestamp based on the first time it sees the event, particularly at input time. For that reason, the originally created `Timestamp` event field replaces Logstash's default setting and is utilized as the event timestamp.
The `output` block contains the `elasticsearch` and `stdout` output plugins. The `elasticsearch` plugin is used in order to forward parsed road weather information data to Elasticsearch, where they are stored. The `hosts` parameter sets the host of the remote instance, in this case *localhost* with Elasticsearch's standard communication port number *9200*. The `index` parameter defines to which index[50] the events are written.

[49] *nil* is a special Ruby data type that means "nothing". It's equivalent to null or none in other programming languages [92].
[50] An Elasticsearch *index* is like a database in a relational database where data is stored [39].

The `document_type` parameter specifies to which Elasticsearch document type[51] the events are written. The `template`[52] and `template_name` parameters are utilized for setting the path of the custom template and defining how the template is named inside Elasticsearch. The used template can be found in Appendix A.1. The second output plugin `stdout` with the `codec => rubydebug` specification outputs event data using the ruby *awesome_print* library[53]. Figure 6.1 illustrates an example output of an event in the command line. The `"message"` field contains the raw event data followed by parsed event fields.

Elasticsearch

The application of the ELK stack and the road weather demonstration data does not require any specific configurations in Elasticsearch. By default, it is directly ready to store the incoming data by Logstash. Its REST API can be used by Kibana for data analytics and discovery. Within the ELK stack, Elasticsearch's index is specified by Logstash's `elasticsearch` output plugin. Herein, the index should be called `road_weather_demonstration`. By executing the command in Listing 6.2, all available Elasticsearch indices can be displayed.

Listing 6.2: Command for listing all Elasticsearch indices

```
curl 'localhost:9200/_cat/indices?v'
```

Listing 6.3 provides an excerpt of Elasticsearch's response to the aforementioned command:

Listing 6.3: List of available Elasticsearch indices

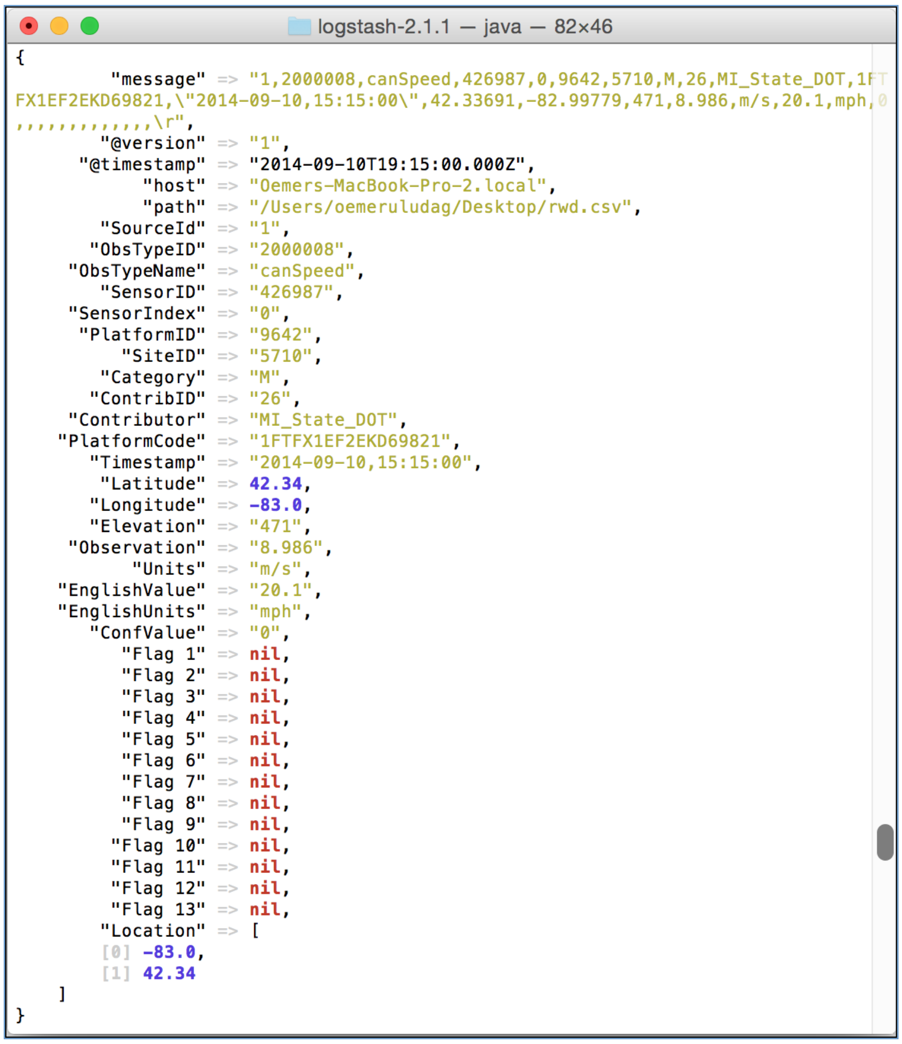
```
health  index                       pri rep docs.count store.size
yellow  .kibana                       1   1          6     35.3kb
yellow  road_weather_demonstration    1   1     572030    174.6mb
```

It illustrates that currently Elasticsearch has two indices: `.kibana` and `road_weather_demonstration`. The first index is created automatically by Kibana which stores saved searches, visualizations, and dashboards. The second index is created within the data processing process. This index currently contains 572,030 documents which matches with the number of sensor entries of the road weather demonstration data. Additionally, Listing 6.3 shows that the `road_weather_demonstration` index has one primary and one replica shard, and has a yellow health, means that some replica shards are not yet allocated. Listing 6.4 demonstrates an example usage of Elasticsearch's Count API. Within this example, it used with the `term` auery in order to count the number of the `ObsTypeName` document field with the value `canspeed`.

[51]A *document type* in Elasticsearch represents a class of similar documents [36].

[52]An *index template* allows to define templates that will automatically be applied to new created indices. Templates include both settings and mappings, and a simple pattern template [29].

[53]http://github.com/michaeldv/awesome_print/



```
{
         "message" => "1,2000008,canSpeed,426987,0,9642,5710,M,26,MI_State_DOT,1FT
FX1EF2EKD69821,\"2014-09-10,15:15:00\",42.33691,-82.99779,471,8.986,m/s,20.1,mph,0
,,,,,,,,,,,,,\r",
        "@version" => "1",
      "@timestamp" => "2014-09-10T19:15:00.000Z",
            "host" => "Oemers-MacBook-Pro-2.local",
            "path" => "/Users/oemeruludag/Desktop/rwd.csv",
        "SourceId" => "1",
       "ObsTypeID" => "2000008",
     "ObsTypeName" => "canSpeed",
        "SensorID" => "426987",
     "SensorIndex" => "0",
      "PlatformID" => "9642",
          "SiteID" => "5710",
        "Category" => "M",
       "ContribID" => "26",
     "Contributor" => "MI_State_DOT",
    "PlatformCode" => "1FTFX1EF2EKD69821",
       "Timestamp" => "2014-09-10,15:15:00",
        "Latitude" => 42.34,
       "Longitude" => -83.0,
       "Elevation" => "471",
     "Observation" => "8.986",
           "Units" => "m/s",
    "EnglishValue" => "20.1",
    "EnglishUnits" => "mph",
       "ConfValue" => "0",
          "Flag 1" => nil,
          "Flag 2" => nil,
          "Flag 3" => nil,
          "Flag 4" => nil,
          "Flag 5" => nil,
          "Flag 6" => nil,
          "Flag 7" => nil,
          "Flag 8" => nil,
          "Flag 9" => nil,
         "Flag 10" => nil,
         "Flag 11" => nil,
         "Flag 12" => nil,
         "Flag 13" => nil,
        "Location" => [
        [0] -83.0,
        [1] 42.34
    ]
}
```

Figure 6.1.: Example of an event output

Listing 6.4: Example of Elasticsearch's Count API

```
curl -XGET 'localhost:9200/road_weather_demonstration/rwd/_count' -d '
{
    "query" : {
        "term" : { "ObsTypeName" : "canspeed" }
    }
}'
```

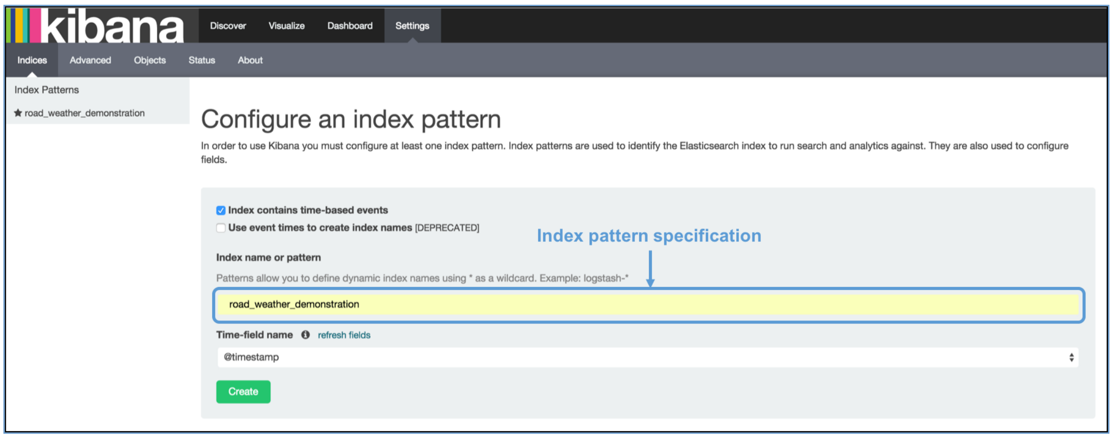
Listing 6.5 presents Elasticsearch's count response. It shows that the `road_weather_demonstration` index contains in total 572,030 documents which have the `canspeed` value for the `ObsTypeName` document field.

Listing 6.5: Elasticsearch's Count API response

```
{
  "count" : 572030,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  }
}
```

Kibana

Firstly, an index pattern has to be configured in order to access to the road weather demonstration data. Listing 6.1 shows that the `index` parameter of the `elasticsearch` output plugin is specified with the value `road_weather_demonstration`. This means that all of the processed data is stored in the `road_weather_demonstration` index. Consequently, the index pattern `road_weather_demonstration` can be specified in Kibana's settings page (see Figure 6.2). Additionally, the `@timestamp` field is specified as the time-field name in order to filter events with the global time filter. Fig-



Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

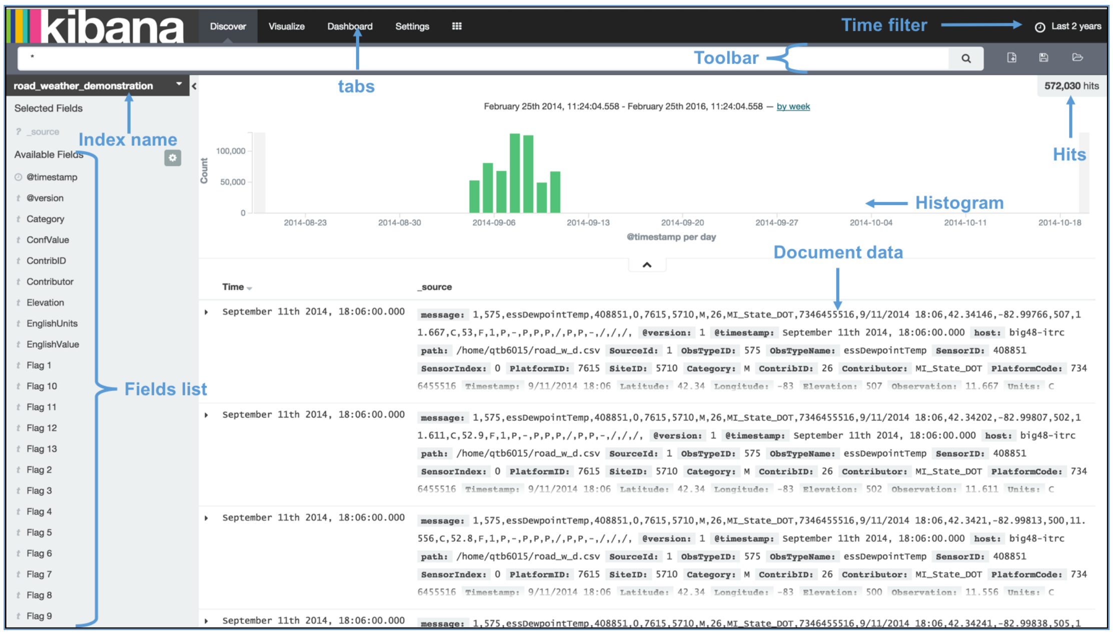
[✓] Index contains time-based events
[ ] Use event times to create index names [DEPRECATED]

Index name or pattern
Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*
road_weather_demonstration

Time-field name
@timestamp

Figure 6.2.: Index pattern specification of road weather demonstration data

ure 6.3 presents the discover page for the road weather demonstration data. The relative time filter is specified as 2 years ago in order to see all processes sensor data appropriately. The hits counter for the search character `*` shows that 572,030 documents are in the actual filter. This indicates that all raw sensor data from the CSV file are successfully stored in Elasticsearch. On the left-hand side the available fields of the road weather demonstration data is present. Figure 6.4 shows that the field search: `ObsTypeName : "canSpeed"` is applied. In total 218,460 documents have the `"canSpeed"` value for the `ObsTypeName` field. In addition, the results of the field search are highlighted in yellow in the document data. For this example, also the click and



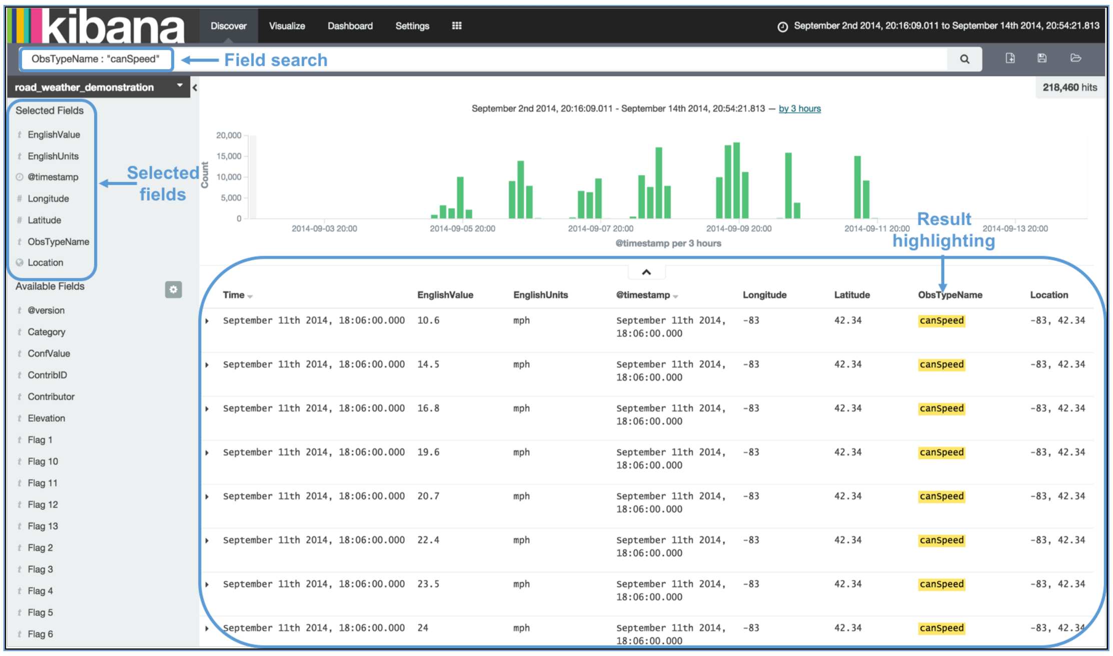
```
September 11th 2014, 18:06:00.000   message: 1,575,essDewpointTemp,408851,0,7615,5710,M,26,MI_State_DOT,7346455516,9/11/2014 18:06,42.34146,-82.99766,507,1
                                    1.667,C,53,F,1,P,-,P,P,P,/,P,P,-,/,/,/,  @version: 1  @timestamp: September 11th 2014, 18:06:00.000  host: big48-itrc
                                    path: /home/qtb6015/road_w_d.csv  SourceId: 1  ObsTypeID: 575  ObsTypeName: essDewpointTemp  SensorID: 408851
                                    SensorIndex: 0  PlatformID: 7615  SiteID: 5710  Category: M  ContribID: 26  Contributor: MI_State_DOT  PlatformCode: 734
                                    6455516  Timestamp: 9/11/2014 18:06  Latitude: 42.34  Longitude: -83  Elevation: 507  Observation: 11.667  Units: C

September 11th 2014, 18:06:00.000   message: 1,575,essDewpointTemp,408851,0,7615,5710,M,26,MI_State_DOT,7346455516,9/11/2014 18:06,42.34202,-82.99807,502,1
                                    1.611,C,52.9,F,1,P,-,P,P,P,/,P,P,-,/,/,/,  @version: 1  @timestamp: September 11th 2014, 18:06:00.000  host: big48-itrc
                                    path: /home/qtb6015/road_w_d.csv  SourceId: 1  ObsTypeID: 575  ObsTypeName: essDewpointTemp  SensorID: 408851
                                    SensorIndex: 0  PlatformID: 7615  SiteID: 5710  Category: M  ContribID: 26  Contributor: MI_State_DOT  PlatformCode: 734
                                    6455516  Timestamp: 9/11/2014 18:06  Latitude: 42.34  Longitude: -83  Elevation: 502  Observation: 11.611  Units: C

September 11th 2014, 18:06:00.000   message: 1,575,essDewpointTemp,408851,0,7615,5710,M,26,MI_State_DOT,7346455516,9/11/2014 18:06,42.3421,-82.99813,500,11.
                                    556,C,52.8,F,1,P,-,P,P,P,/,P,P,-,/,/,/,  @version: 1  @timestamp: September 11th 2014, 18:06:00.000  host: big48-itrc
                                    path: /home/qtb6015/road_w_d.csv  SourceId: 1  ObsTypeID: 575  ObsTypeName: essDewpointTemp  SensorID: 408851
                                    SensorIndex: 0  PlatformID: 7615  SiteID: 5710  Category: M  ContribID: 26  Contributor: MI_State_DOT  PlatformCode: 734
                                    6455516  Timestamp: 9/11/2014 18:06  Latitude: 42.34  Longitude: -83  Elevation: 500  Observation: 11.556  Units: C

September 11th 2014, 18:06:00.000   message: 1,575,essDewpointTemp,408851,0,7615,5710,M,26,MI_State_DOT,7346455516,9/11/2014 18:06,42.34241,-82.99838,505,1
```

Figure 6.3.: Index pattern specification of road weather demonstration data

drag function is applied on the area of the histogram. It illustrates the distribution of the matching documents over a time period of 12 days. On the left-hand side, important fields, including `EnglishValue`, `EnglishUnits`, `@timestamp`, `Longitude`, `Latitude`, `ObsTypeName`, and `Location` are selected. This results in the creation of a data table which provides a compact view on important fields of the data. Based on the `Location` field, a tile map is created by using the visualize page. Figure 6.5 presents the tile map configuration after the initial visualize page. Within the aggregation designer, the count metric aggregation is selected with the geohash bucket aggregation on the `Location` field. This configuration allows viewing of the geographical distribution of sensor data. As expected, Figure 6.6 demonstrates that the demonstration configuration was on Belle Isle, Detroit, Michigan, USA. After the creation of various visualizations, a dashboard is created. Figure 6.7 shows that this dashboard uses the dark theme configuration of Kibana. The dashboard consists of a pie chart, metric, line chart, and tile map. The pie chart presents the distribution of different sensor observation types with the donut setting. The metric visualization displays the actual number of documents for the given time frame. The line chart provides an overview of the temporal distribution of the road weather demonstration data creation. The tile map shows the geographical distribution of the road weather demonstration data.

6.1.3. Key Findings

Based on the first experiment, the following key finding is elicited:



September 2nd 2014, 20:16:09.011 to September 14th 2014, 20:54:21.813

ObsTypeName : "canSpeed"

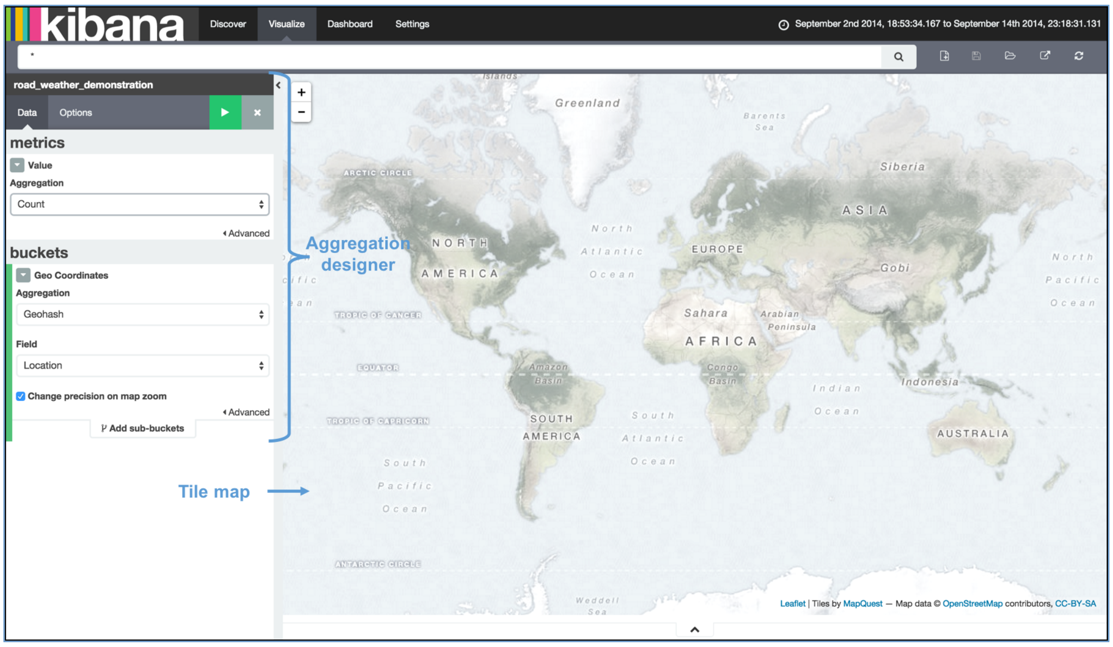
road_weather_demonstration — 218,460 hits

Selected Fields: EnglishValue, EnglishUnits, @timestamp, Longitude, Latitude, ObsTypeName, Location

Available Fields: @version, Category, ConfValue, ContribID, Contributor, Elevation, Flag 1, Flag 10, Flag 11, Flag 12, Flag 13, Flag 2, Flag 3, Flag 4, Flag 5, Flag 6

Time	EnglishValue	EnglishUnits	@timestamp	Longitude	Latitude	ObsTypeName	Location
September 11th 2014, 18:06:00.000	10.6	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	14.5	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	16.8	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	19.6	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	20.7	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	22.4	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	23.5	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34
September 11th 2014, 18:06:00.000	24	mph	September 11th 2014, 18:06:00.000	-83	42.34	canSpeed	-83, 42.34

Figure 6.4.: Index pattern specification of road weather demonstration data



September 2nd 2014, 18:53:34.167 to September 14th 2014, 23:18:31.131

road_weather_demonstration

metrics
Value
Aggregation: Count

buckets
Geo Coordinates
Aggregation: Geohash
Field: Location
☑ Change precision on map zoom

Leaflet | Tiles by MapQuest — Map data © OpenStreetMap contributors, CC-BY-SA

Figure 6.5.: Tile map creation

- **Timestamping of events**: Logstash automatically creates a timestamp for each

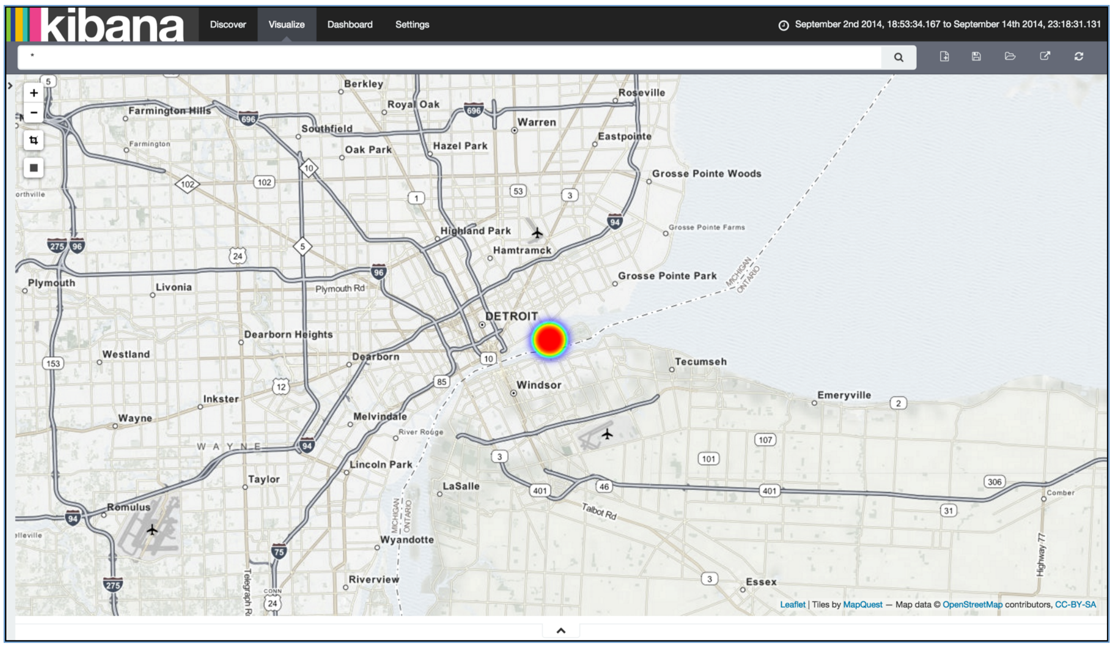

kibana
Discover
Visualize
Dashboard
Settings
September 2nd 2014, 18:53:34.167 to September 14th 2014, 23:18:31.131
DETROIT
Windsor
Leaflet | Tiles by MapQuest — Map data © OpenStreetMap contributors, CC-BY-SA

Figure 6.6.: Geographical location of road weather demonstration data

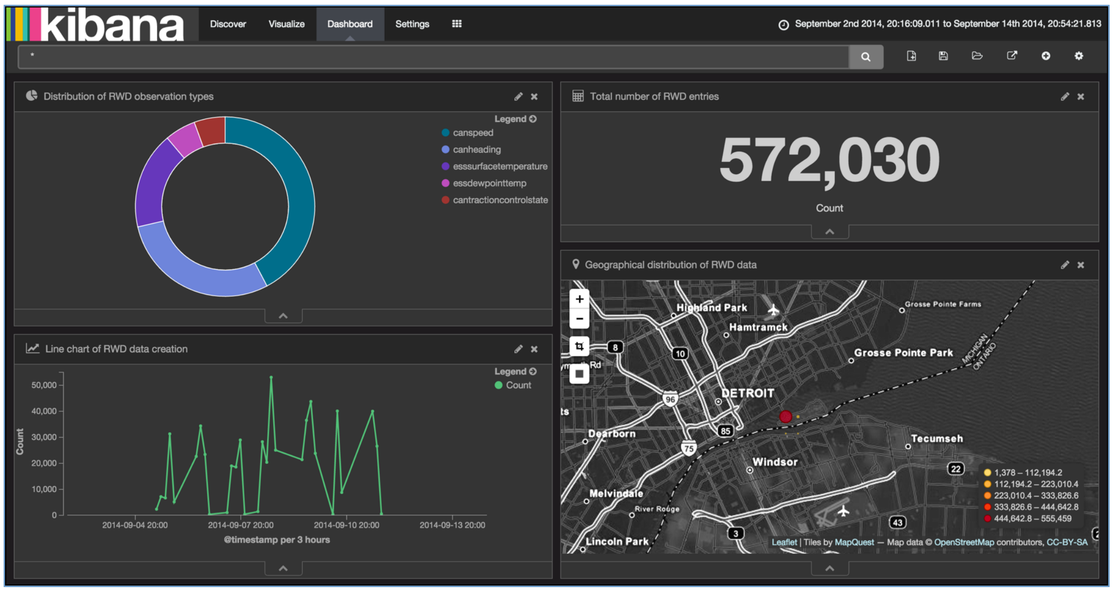


Figure 6.7.: Dashboard of textual data analysis

event at input time. However, Logstash also allows use of custom date fields as the timestamp of events by providing the `date` filter plugin. This indicates that Logstash is capable of assigning all events with timestamps so that Elasticsearch and Kibana can perform time-based analyses.

6.2. Experiment 2: Machine-Generated Data Analysis

This experiment deals with analyzing semi-structured machine-generated data from vehicles with the ELK stack. In 2014, machine-generated data was one of the most fundamental data sources for Big Data environments. Machine-generated data within vehicles is the basis for real-time data analytics [43]. This data may also contain sensitive information, e.g., vehicle identification number, which has to be protected. In order to assess the capability of the ELK stack for performing real-time data analytics and able to protect sensitive information, semi-structured machine-generated data in the XML format is synthesized.

6.2.1. Data Basis and Data Overview

As a sample data source, randomly-generated 2.16 million events are used. Messages each are 525 bytes in size and the events contain string values. The sample data source is representing the following attribute fields:

- **level**: This attribute is comprised of an unvarying string value.
- **time**: This attribute has a timestamp of the event.
- **timel**: This attribute represents the UNIX timestamp of the event
- **id**: This attribute contains the unique identifier of the event.
- **cat**: This attribute includes an unvarying string value.
- **host_org**: This attribute provides information about the hostname.
- **vin**: This attribute contains the vehicle identification number.
- **clat**: This attribute represents the current latitude value of event.
- **clon**: This attribute represents the current longitude value of event.
- **msg_txt**: This attribute includes event-related information.

Some of the fields, namely *timel*, *id*, *vin*, *clat*, and *clon* are randomly generated as realistically as possible, e.g., the range of *clat* and *clon* values are within the bounding box of the USA, and other fields are created fixed using Python version 2.7.11.

6.2.2. Implementation of the ELK Stack

Logstash

Listing 6.6: Configuration file for machine-generated data analysis

```
input {
  file {
    path => "/Users/oemeruludag/Desktop/xml_data/data_*.xml"
    start_position => "beginning"
  }
}
filter {
  xml {
    store_xml => false
    source => "message"
    xpath => [
             "/log/@level", "level",
             "/log/@time", "time",
             "/log/@timel", "timel",
             "/log/@id", "id",
             "/log/@cat", "cat",
             "/log/@host", "host_org",
             "/log/@vin", "vin",
             "/log/@msg", "msg",
             "/log/@clat", "clat",
             "/log/@clon", "clon",
             "/log/msg/@msg_txt","msg_txt"
             ]
  }
  anonymize {
    algorithm => "MD5"
    fields => ["vin"]
    key => "longencryptionkey"
  }
  mutate {
    replace => [ "[clat]", "%{[clat]}" ]
    replace => [ "[clon]", "%{[clon]}" ]
  }
  if [clon] != '' {
    ruby {
      code => "
        if !event['clon'].nil?
          event['clon'] = (event['clon'].to_f * 100).round / 100.00
        end
      "
    }
  }
  if [clon] != '' {
    mutate {
      add_field => [ "[location]", "%{clon}" ]
    }
  }
  if [clat] != '' {
    ruby {
      code => "
        if !event['clat'].nil?
          event['clat'] = (event['clat'].to_f * 100).round / 100.00
        end
      "
    }
  }
  if [clat] != '' {
    mutate {
      add_field => [ "[location]", "%{clat}" ]
    }
  }
}
output {
  elasticsearch {
    hosts => "localhost:9200"
    index => "machine_generated-%{+YYYY.MM.dd}"
    document_type => "mach_gen"
    template => "/Users/oemeruludag/Desktop/mach_gen_template.json"
    template_name => "machine_generated-*"
  }
  stdout{
    codec => rubydebug
  }
}
```

In contrast to the first experiment, the configuration of file of Logstash in Listing 6.6 uses a wildcard for the file name within the `file` input plugin so that all generated files in the specified path can be ingested by Logstash. The `filter` block has several important filter plugins. Firstly, the `xml` filter plugin uses XPath expressions in order to retrieve the values of the specified fields. However, relying only on XPath can be critical for log files which are highly variable since the XPath specifications have to be predefined. The sole usage of the `xml` filter plugin is suitable for flat XML log files, but deficient for multiple varying hierarchical log files. For that reason, it should be used in combination with the `multiline` filter plugin in order to collapse multiline events and merge them into a single event and the `ruby` filter plugin for using loops and conditions. The `anonymize` filter plugin helps to hash the sensitive information of the `vin` field with MD5 algorithm. Logstash provides further algorithms such as SHA1, SHA256, SHA384, SHA512, and MURMUR3 for protecting sensitive information. If necessary, custom algorithms can be developed which can be used within the `anonymize` filter plugin. As in the first experiment, the longitude, `clon`, and latitude, `clat`, have to be adjusted so that Elasticsearch is able to use them as geolocation parameters. In opposition to the first experiment, the `index` specification within the `elasticsearch` output plugin uses an index pattern with a timestamp pattern. With this setting, for each day a new Elasticsearch index can be created. This is especially important for daily machine-generated data, since in Elasticsearch unnecessary data in a particular daily index can be deleted without deleting the whole Elasticsearch index. In other words, older data can be deleted without deleting the whole data. Logstash also supports the creation of index on yearly, monthly, and hourly basis, to name a few. Lastly, the `template_name` specification also uses a pattern, since the configured Elasticsearch template should not be only relevant for one index but also for all similar indices. The utilized template can be found in Appendix A.2.

Elasticsearch

As in the first experiment, the application of the ELK stack does not require any specific configurations in Elasticsearch. With the specifications in the configuration file of Logstash (see Listing 6.6), Elasticsearch stores the incoming events of Logstash in the JSON format. By executing the `indices` command, the same as in Listing 6.2, all available present Elasticsearch indices can be displayed. Listing 6.7 provides an excerpt of Elasticsearch's response to the `indices` command:

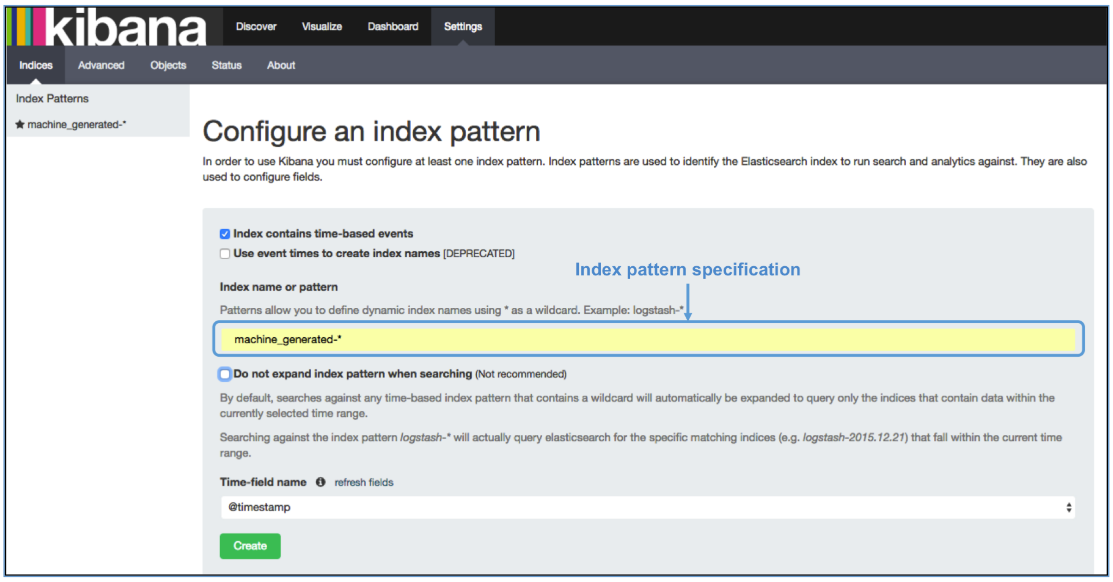
Listing 6.7: List of available Elasticsearch indices

```
health  index                       pri rep docs.count store.size
yellow  .kibana                       1   1         13   58.3kb
yellow  machine_generated-2016.04.09 1   1    2160000    1.6gb
```

It illustrates that currently Elasticsearch has two indices: `.kibana` and `machine_generated-2016.04.09`. The first index is created automatically by Kibana which stores saved searches, visualizations, and dashboards. The second index is created by Logstash. This index currently contains 2,160,000 documents which matches with the number of events generated by the Python script. The latter index name also shows that the index was created on the 9th of April, 2016.

Kibana

In order to be able to analyze the machine-generated in Kibana, an index pattern has to be defined. Figure 6.8 presents the definition of the index pattern `machine_generated-*`. The asterisk within the index pattern implies that all indices which are created on the daily basis (see Listing 6.6) should be selected. This enables aggregation and analysis of data across multiple indices. Figure 6.9 illustrates Kibana's click and drag functionality which can be done easily by hovering the cursor anywhere over the histogram. In this figure, the cursor was dragged to select a specific time interval, namely from 5 minutes to 30 seconds. Thus, the time interval specified has been changed in the time filter, which is in the top-right corner, and the data in the data table is changed as per the time range interval specified. Within Kibana, machine-generated data can be queried while data is collected by Logstash and indexed by Elasticsearch.



Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events
[ ] Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

machine_generated-*

[ ] Do not expand index pattern when searching (Not recommended)

By default, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range.

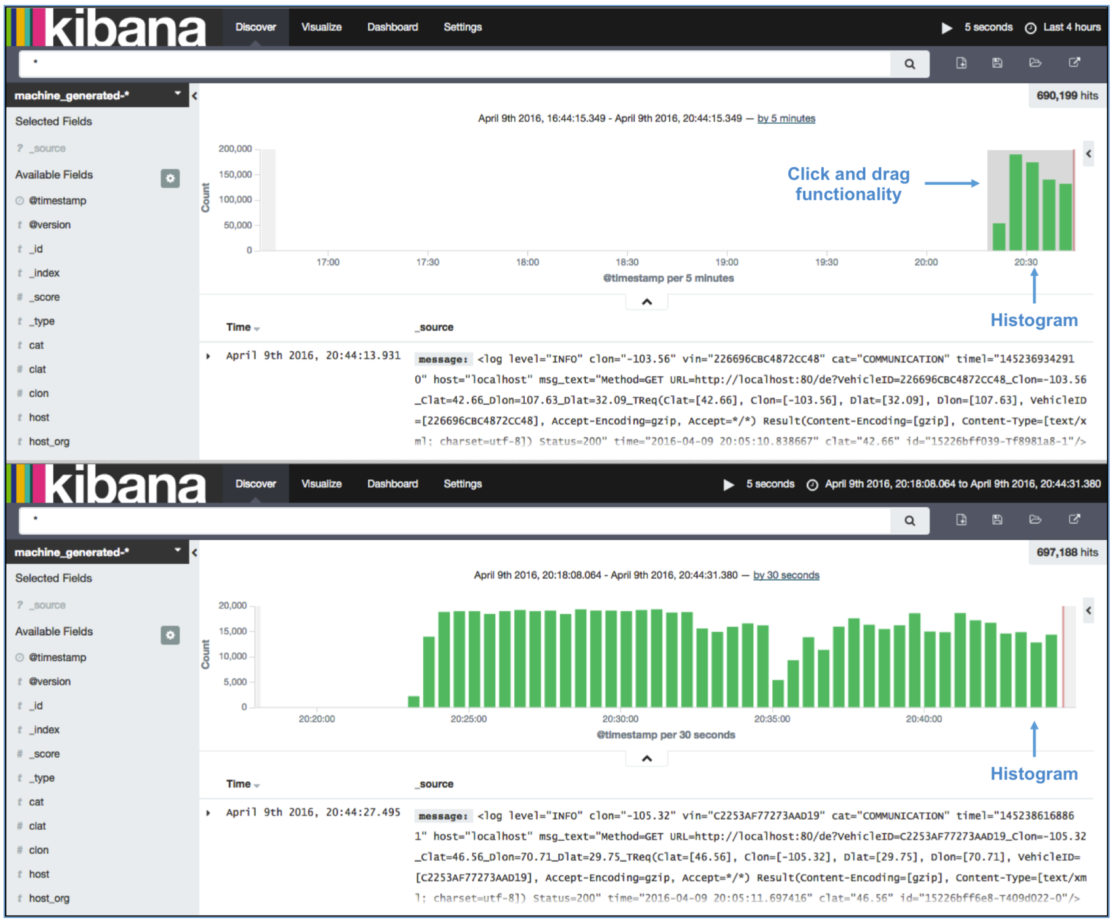
Searching against the index pattern *logstash-** will actually query elasticsearch for the specific matching indices (e.g. *logstash-2015.12.21*) that fall within the current time range.

Time-field name

@timestamp

Figure 6.8.: Index pattern specification of machine-generated data

Figure 6.10 shows that the field search `vin: F*` is applied. This is a wildcard search which results in the selection of all documents, containing vins beginning with the "F" expression. Automatically, the corresponding field in the data table is highlighted in yellow. The two dots on the right upper corner in Figure 6.10 indicates that Kibana's discover page is currently reloading new indexed data. Besides the dots, the 5 seconds implies Kibana's selected refresh interval. In other words, every 5 seconds, Kibana is rendered with new indexed data. By comparing the first screenshot with the second one in Figure 6.10, one can see that Kibana automatically applies the previously specified wildcard search and displays the resulting data. Moreover, the hits number is reloaded which represents the matching documents for the specified search in the selected time interval. In Listing 6.6, the MD5 algorithm was used in the `anonymize` filter plugin for the `vin` field. Figure 6.11 illustrates that Kibana displays the hashed values for this field successfully. The ELK stack also supports data traceability by unique persistent identifiers (see Figure 6.12). The `_id` field in Figure 6.12 represents the document's unique id which is automatically created by Elasticsearch whereas `id` field conforms with the primarily unique identifier of the ingested event by Logstash. Besides the creation of visualizations, Kibana also allows exporting underlying aggregated data of visualizations. Kibana provides the export functionality that allows to download the data as a raw or formatted CSV file. Moreover, Figure 6.13 demonstrates Kibana's visualization-related information. These include the data table, the aggregation request from Kibana to Elasticsearch, the corresponding response provided by Elasticsearch, and some statistics related to the execution of the request. Figure 6.14 shows the created dashboard for the machine-generated data. The dashboard consists of a vertical bar chart, metric, and an area chart. The vertical bar chart presents the time distribution


kibana
Discover
Visualize
Dashboard
Settings
5 seconds
Last 4 hours
machine_generated-*
Selected Fields
_source
Available Fields
@timestamp
@version
_id
_index
_score
_type
cat
clat
clon
host
host_org
690,199 hits
April 9th 2016, 16:44:15.349 - April 9th 2016, 20:44:15.349 — by 5 minutes
Count
@timestamp per 5 minutes
Click and drag functionality
Histogram
Time
_source
April 9th 2016, 20:18:08.064 to April 9th 2016, 20:44:31.380
697,188 hits
April 9th 2016, 20:18:08.064 - April 9th 2016, 20:44:31.380 — by 30 seconds
@timestamp per 30 seconds

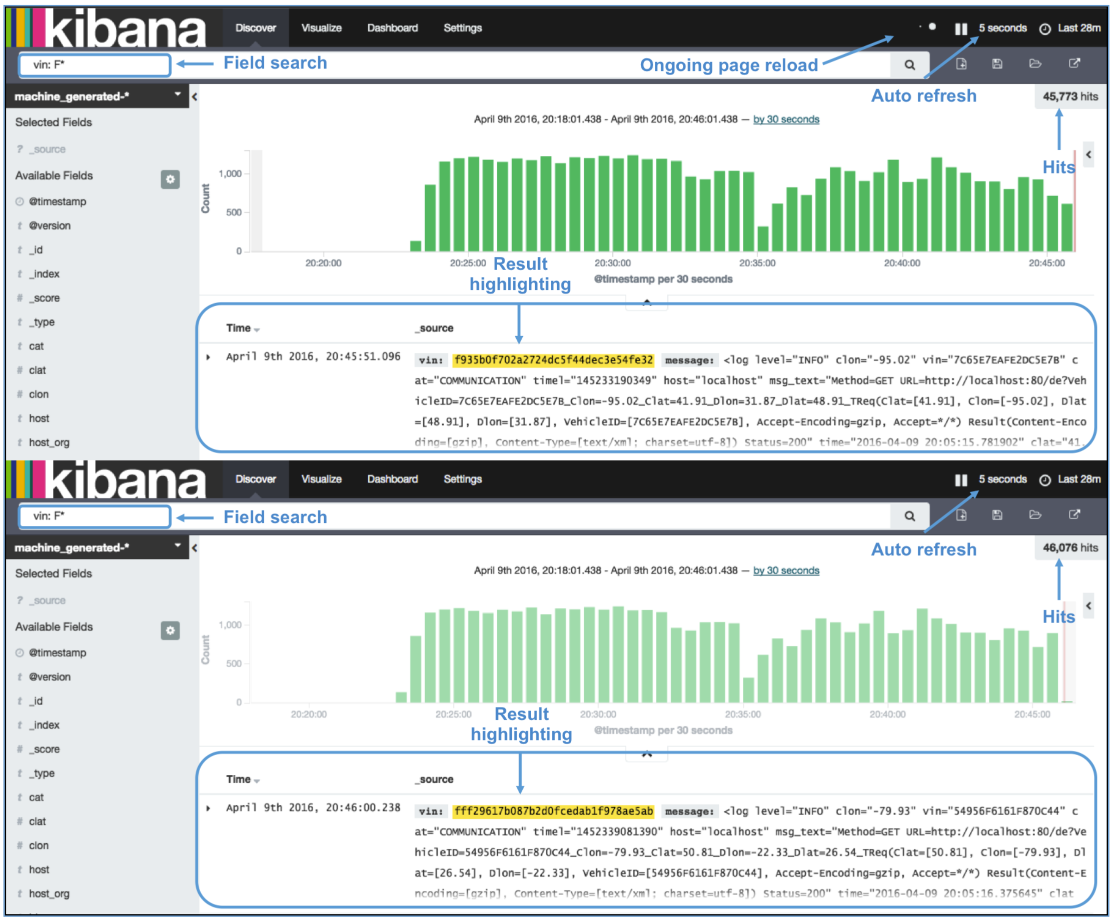
```
April 9th 2016, 20:44:13.931   message: <log level="INFO" clon="-103.56" vin="226696CBC4872CC48" cat="COMMUNICATION" time1="145236934291
                               0" host="localhost" msg_text="Method=GET URL=http://localhost:80/de?VehicleID=226696CBC4872CC48_Clon=-103.56
                               _Clat=42.66_Dlon=107.63_Dlat=32.09_TReq(Clat=[42.66], Clon=[-103.56], Dlat=[32.09], Dlon=[107.63], VehicleID
                               =[226696CBC4872CC48], Accept-Encoding=gzip, Accept=*/*) Result(Content-Encoding=[gzip], Content-Type=[text/x
                               ml; charset=utf-8]) Status=200" time="2016-04-09 20:05:10.838667" clat="42.66" id="15226bff039-Tf8981a8-1"/>
```

```
April 9th 2016, 20:44:27.495   message: <log level="INFO" clon="-105.32" vin="C2253AF77273AAD19" cat="COMMUNICATION" time1="145238616886
                               1" host="localhost" msg_text="Method=GET URL=http://localhost:80/de?VehicleID=C2253AF77273AAD19_Clon=-105.32
                               _Clat=46.56_Dlon=70.71_Dlat=29.75_TReq(Clat=[46.56], Clon=[-105.32], Dlat=[29.75], Dlon=[70.71], VehicleID=
                               [C2253AF77273AAD19], Accept-Encoding=gzip, Accept=*/*) Result(Content-Encoding=[gzip], Content-Type=[text/xm
                               l; charset=utf-8]) Status=200" time="2016-04-09 20:05:11.697416" clat="46.56" id="15226bff6e8-T409d022-0"/>
```

Figure 6.9.: Click and drag functionality for drilling down data

of current latitudes. It categorizes the latitude values in buckets with the size of 2. It shows that the distribution of the generated latitude values are almost constant over time. The metric visualization displays the total number of documents, the average and the upper/lower standard deviation values for the current latitudes and longitudes, and the unique count of the unique vehicles for the given time frame. The area chart provides an overview of the temporal distribution of the machine-generated data. As mentioned in Section 6.2, one of the main goals of this experiment is to assess the capability of the ELK stack for performing real-time data analytics. In order to evaluate this, Logstash's timestamp for events at input time and Kibana's timestamp[54] for events visibility are juxtaposed in opposition. Table 6.1 displays the observed timestamps of the comparison. By calculating the difference between those, one can determine how long the whole ELK stack needs for data filtering, indexing, and visualization. For this

[54]Since Kibana does not provide an automatically created timestamp when it sees for the first time a new event, a manually generated timestamp is used.



```
April 9th 2016, 20:45:51.096   vin: f935b0f702a2724dc5f44dec3e54fe32  message: <log level="INFO" clon="-95.02" vin="7C65E7EAFE2DC5E7B" c
                               at="COMMUNICATION" time1="145233190349" host="localhost" msg_text="Method=GET URL=http://localhost:80/de?Veh
                               icleID=7C65E7EAFE2DC5E7B_Clon=-95.02_Clat=41.91_Dlon=31.87_Dlat=48.91_TReq(Clat=[41.91], Clon=[-95.02], Dlat
                               =[48.91], Dlon=[31.87], VehicleID=[7C65E7EAFE2DC5E7B], Accept-Encoding=gzip, Accept=*/*) Result(Content-Enco
                               ding=[gzip], Content-Type=[text/xml; charset=utf-8]) Status=200" time="2016-04-09 20:05:15.781902" clat="41.
```

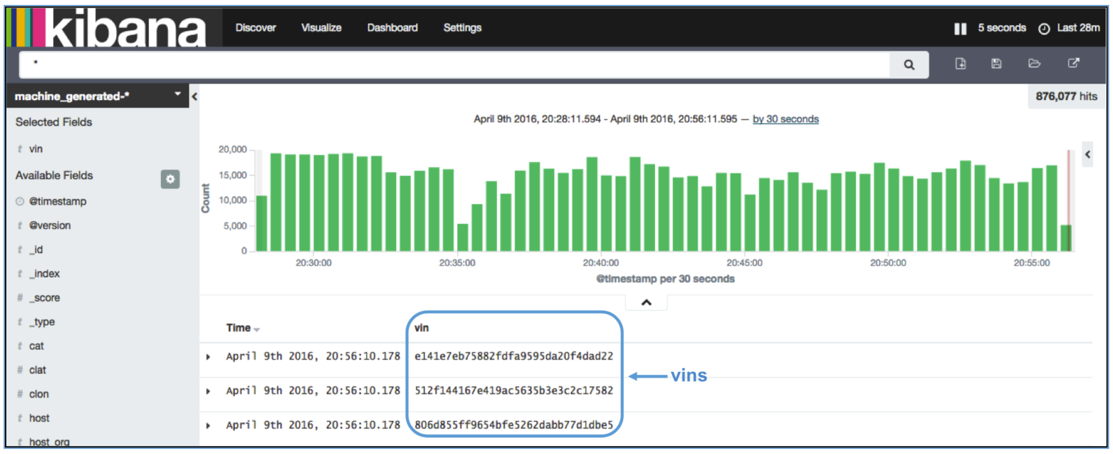
```
April 9th 2016, 20:46:00.238   vin: fff29617b087b2d0fcedab1f978ae5ab  message: <log level="INFO" clon="-79.93" vin="54956F6161F870C44" c
                               at="COMMUNICATION" time1="1452339081390" host="localhost" msg_text="Method=GET URL=http://localhost:80/de?Ve
                               hicleID=54956F6161F870C44_Clon=-79.93_Clat=50.81_Dlon=-22.33_Dlat=26.54_TReq(Clat=[50.81], Clon=[-79.93], Dl
                               at=[26.54], Dlon=[-22.33], VehicleID=[54956F6161F870C44], Accept-Encoding=gzip, Accept=*/*) Result(Content-E
                               ncoding=[gzip], Content-Type=[text/xml; charset=utf-8]) Status=200" time="2016-04-09 20:05:16.375645" clat
```

Figure 6.10.: Querying data while data is collected and indexed

setting, the ELK stack needs on average 6.607 seconds in order to make an processed event accessible. This delay is for the most part affected by Kibana's refresh interval which makes up at least 5 seconds. However, this delay increases when data filtering operations in Logstash become more complex or when the events are located on remote servers and therefore e.g., the Logstash-forwarder has to be used. Independently from Logstash's and Elasticsearch data ingest and indexing performance, indexed data is never be directly accessible for search and analytics since the Elasticsearch index has to be refreshed in order to make data accessible. Due to this reason, the denotation "near real-time data analytics" seems to be more appropriate than "real-time data analytics" since indexed data is not instantly accessible in the ELK stack.

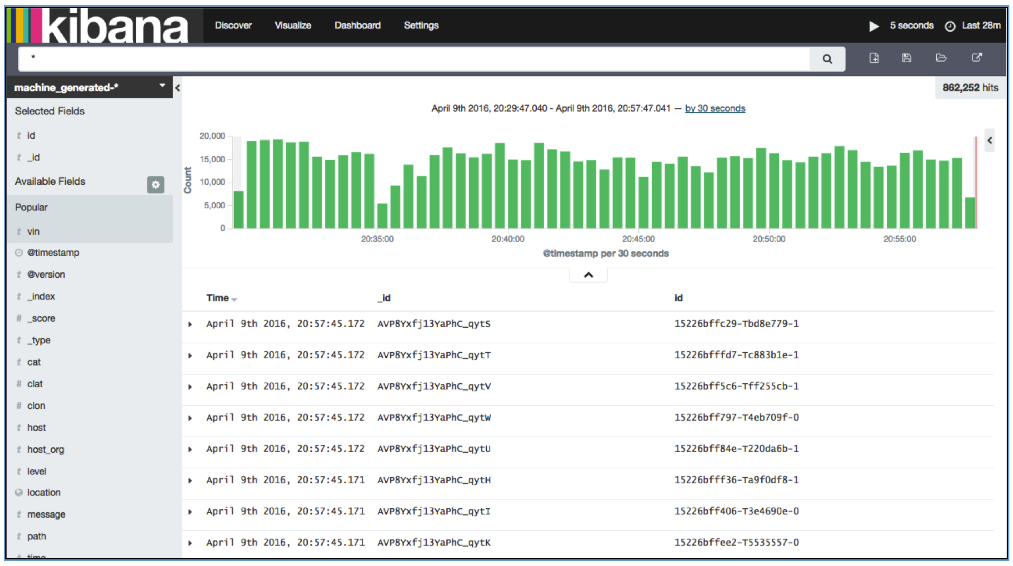
6.2.3. Key Findings

Based on the machine-generated data analysis experiment, the following key findings are deduced:



Time	vin
April 9th 2016, 20:56:10.178	e141e7eb75882fdfa9595da20f4dad22
April 9th 2016, 20:56:10.178	512f144167e419ac5635b3e3c2c17582
April 9th 2016, 20:56:10.178	806d855ff9654bfe5262dabb77d1dbe5

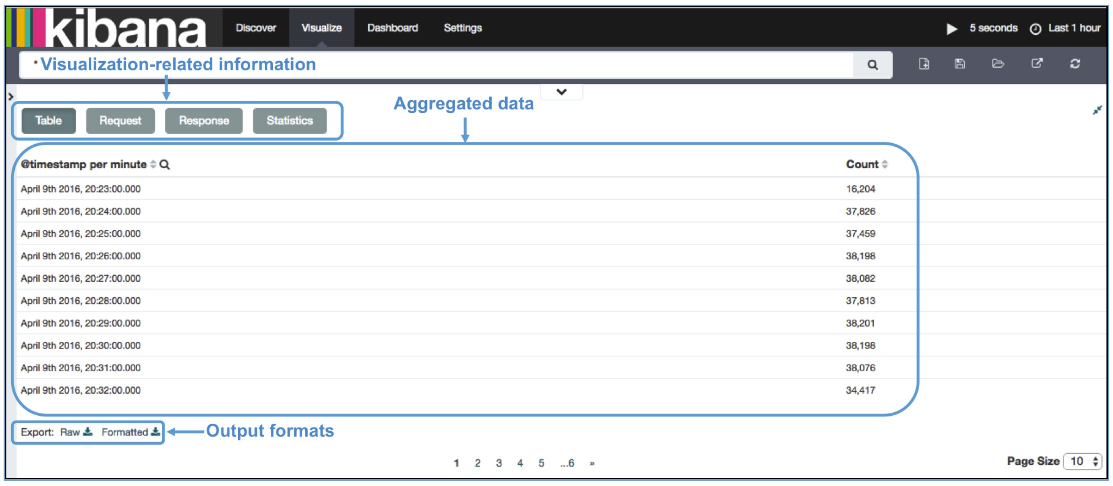
Figure 6.11.: Anonymization of vins



Time	_id	id
April 9th 2016, 20:57:45.172	AVP8Yxfj13YaPhC_qytS	15226bffc29-Tbd8e779-1
April 9th 2016, 20:57:45.172	AVP8Yxfj13YaPhC_qytT	15226bfffd7-Tc883b1e-1
April 9th 2016, 20:57:45.172	AVP8Yxfj13YaPhC_qytV	15226bff5c6-Tff255cb-1
April 9th 2016, 20:57:45.172	AVP8Yxfj13YaPhC_qytW	15226bff797-T4eb709f-0
April 9th 2016, 20:57:45.172	AVP8Yxfj13YaPhC_qytU	15226bff84e-T220da6b-1
April 9th 2016, 20:57:45.171	AVP8Yxfj13YaPhC_qytH	15226bfff36-Ta9f0df8-1
April 9th 2016, 20:57:45.171	AVP8Yxfj13YaPhC_qytI	15226bff406-T3e4690e-0
April 9th 2016, 20:57:45.171	AVP8Yxfj13YaPhC_qytK	15226bffee2-T5535557-0

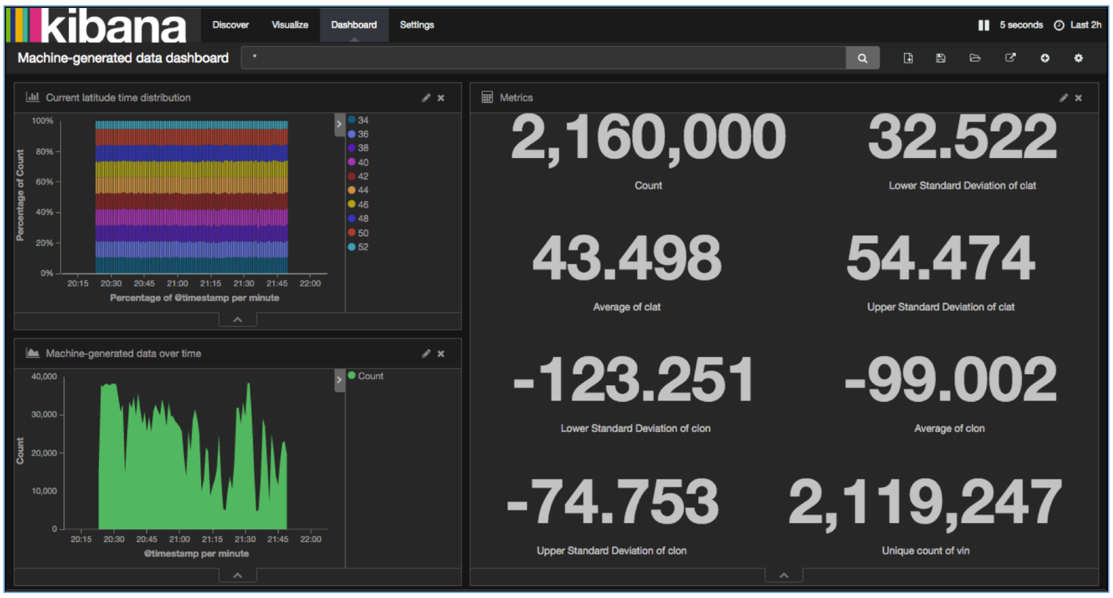
Figure 6.12.: Data traceability

- **High data variability may cause data parse failures**: The solely usage of Logstash's `xml` filter plugin seems to be insufficient for highly-variable and multiple hierarchical XML data since the usage of XPath alone is not capable of retrieving fields which have inconstant positions. With the utilization of the `multiline` and `ruby` filter plugins, Logstash becomes capable of handling this type of data in some degree. However, programming languages such as Python provide more appropriate library functions for parsing this type of XML data.
- **Data anonymization capabilities**: Logstash provides an `anonymize` filter plugin in order to protect and hash sensitive information of fields by providing various



@timestamp per minute	Count
April 9th 2016, 20:23:00.000	16,204
April 9th 2016, 20:24:00.000	37,826
April 9th 2016, 20:25:00.000	37,459
April 9th 2016, 20:26:00.000	38,198
April 9th 2016, 20:27:00.000	38,082
April 9th 2016, 20:28:00.000	37,813
April 9th 2016, 20:29:00.000	38,201
April 9th 2016, 20:30:00.000	38,198
April 9th 2016, 20:31:00.000	38,076
April 9th 2016, 20:32:00.000	34,417

Figure 6.13.: Supported output formats


kibana
Discover
Visualize
Dashboard
Settings
5 seconds
Last 2h
Machine-generated data dashboard
Current latitude time distribution
Percentage of Count
Percentage of @timestamp per minute
34
36
38
40
42
44
46
48
50
52
Machine-generated data over time
Count
@timestamp per minute
Metrics

Value	Metric
2,160,000	Count
32.522	Lower Standard Deviation of clat
43.498	Average of clat
54.474	Upper Standard Deviation of clat
-123.251	Lower Standard Deviation of clon
-99.002	Average of clon
-74.753	Upper Standard Deviation of clon
2,119,247	Unique count of vin

Figure 6.14.: Dashboard of machine-generated data analysis

selection of hashing algorithms. Kibana automatically displays the hashed sensitive information.

- **Data traceability capabilities**: The ELK stack supports data traceability by using originally created unique identifiers of events or by generating unique identifiers for newly created Elasticsearch documents.
- **Simultaneous indexing and querying of events**: While data is ingested by

Data accessibility delay			
Observation	**Logstash timestamp**	**Kibana timestamp**	**Difference**
1	00:15:54.230	00:16:04.000	00:00:09.770
2	00:16:04.999	00:16:12.000	00:00:07.001
3	00:16:13.757	00:16:20.000	00:00:06.243
4	00:16:22.832	00:16:28.000	00:00:05.168
5	00:16:31.145	00:16:36.000	00:00:04.855
Average	-	-	00:00:06:607

Table 6.1.: Illustration of data accessibility delay

Logstash and is indexed by Elasticsearch, various searches can be executed in Kibana which directly provide adjusted results for new indexed data.

- **Automatic update of web interface**: While machine-generated data is ingested by Logstash and indexed by Elasticsearch, Kibana automatically refreshes search results and visualizations with the help of the Auto-refresh functionality by selecting a refresh interval option.

- **Fractional amount of supported output formats**: Kibana provides only raw and formatted CSV files as output formats for underlying aggregated data of visualization which can be used as data input for further analyses.

- **Near real-time data analytics**: The ELK stack is capable of near real-time data analytics since it needs time in order to filter, index, and visualize data. Regardless of this data accessibility delay, data is never directly accessible in Kibana since Elasticsearch's index has to be refreshed in order to make data accessible for search and analysis.

6.3. Experiment 3: Social Media Data Analysis

This experiment deals with analyzing unstructured social media data from Twitter with the ELK stack. The value of social media can be used for many cases which has a great impact on the effectiveness of the business and its strategies. This analysis can be performed for a wide range of activities, such as marketing, brand building and management, and customer focus, to name a few. For instance, automotive companies can track the heartbeat of the automobile market, e.g., whenever they launch a new model or a new design. By making use of this power, companies can make changes on their marketing strategies or help the customers for a better relationship management. Also, this would be useful for a company to monitor their marketing campaigns [43, 49]. In order to assess the capability of the ELK stack for analyzing social media data, unstructured Twitter data is utilized. In order to create a meaningful context, tweets containing the following key words: ″Big Data″, ″Elasticsearch″, ″Logstash″, ″Kibana″, ″Elastic″, and

"ELK stack" should be analyzed.
In this experiment, the ELK stack should answer the following questions:

- How many times have the predefined key words been tweeted about in a time interval?
- Which are the top languages in which people tweet about the predefined key words?
- Which are the different geographical locations from where people are tweeting about the predefined key words?
- From which devices are people mostly tweeting about the predefined key words?
- From which countries, using different devices, are people tweeting about the predefined key words?
- What are the top retweeted user screen names related to the predefined key words tweeting from different devices?
- What are the top user screen names tweeting about the predefined key words?
- What are the most popular hashtags related to the predefined key words?

6.3.1. Data Basis and Data Overview

Unlike the previous experiments, the total size of the social media data and the number of fields are not predetermined. The first experiment has a total number of 33 fields whereas this experiment has more than 800 distinct fields which are continuously growing. This increase is affected by Elasticsearch. In other words, whenever Elasticsearch detects an unknown field which is not yet in Elasticsearch's data schema, it adds the new field into it. Some important attribute fields include:

- **@timestamp**: This attribute represents the formatted creation date of the tweet.
- **coordinates**: This attribute contains the latitude and longitudes values of the tweet location.
- **lang**: This attribute provides information about the used language for the tweet.
- **user.location**: This attribute contains information about the user's specified location.
- **user.name**: This attribute represents the Twitter user name.
- **text**: This attribute equals the tweet message.
- **place.country_code**: This attribute represents the double-digit country code.

- **retweeted_status.user.screen_name**: This attribute indicates that the tweet was retweeted and included the user screen name in the tweet.
- **entities.hashtags.text**: This attribute corresponds to the keywords entered in the Logstash configuration file.

6.3.2. Implementation of the ELK Stack

Logstash

Listing 6.8: Configuration file for social media data analysis

```
input {
  twitter {
      consumer_key => "XXXXXXXXXXXXXXXX"
      consumer_secret => "XXXXXXXXXXXXXXXX"
      oauth_token => "XXXXXXXXXXXXXXXX"
      oauth_token_secret => "XXXXXXXXXXXXXXXX"
      keywords => [ "Big Data", "Elasticsearch", "Logstash",
                    "Kibana", "Elastic", "ELK stack" ]
      full_tweet => true
  }
}
output {
  elasticsearch {
    hosts => "localhost:9200"
    index => "social_media-%{+YYYY.MM.dd}"
    document_type => "sm"
    template => "/Users/oemeruludag/Desktop/twitter_template.json"
    template_name => "social_media-*"
  }
  stdout{
    codec => rubydebug
  }
}
```

In order to be able to fetch tweets from Twitter, a Twitter developer account has to be created. During the creation process, Twitter provides the required values for the `consumer_key`, `consumer_secret`, `oauth_token`, and `oauth_token_secret` specifications within the `twitter` input plugin. Due to reasons of security, the original values are hidden here. Following these, the `keywords` field accepts the aforementioned keywords within an array. For storing whole tweets, the `full_tweet` field has to be set to `true`. Setting it to false would specify Logstash to fetch tweets with limited fields. In this experiment, `filter` plugins are not specified since Logstash parses the incoming Twitter into JSON. Since the incoming data should be stored in Elasticsearch, the `elasticsearch` output plugin is utilized. Within this output plugin, the `social_media-*` template is used, which can be found in Appendix A.3.

Elasticsearch

This experiment does also not require any specific configurations in Elasticsearch as in the previous experiments. By executing the `indices` command, the same as in Listing 6.2, all available present Elasticsearch indices can be displayed. Listing 6.9 provides an excerpt of Elasticsearch's response to the `indices` command:

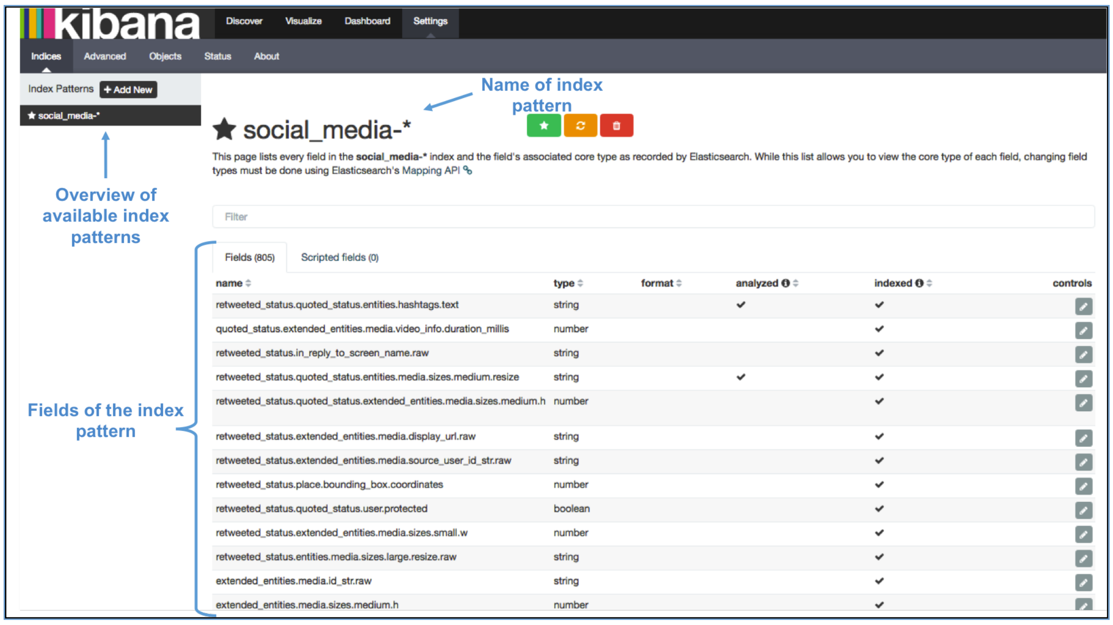
Listing 6.9: List of available Elasticsearch indices

```
health  index                        pri rep docs.count store.size
yellow  social_media-2016.04.11        1   1       2684   13.6mb
yellow  .kibana                        1   1         15    143kb
yellow  social_media-2016.04.10        1   1        872    3.7mb
```

It illustrates that currently Elasticsearch has three indices: `social_media-2016.04.11`, `.kibana`, and `social_media-2016.04.10`. Both social media indices are created during the Twitter analysis. Since in Logstash, the index pattern on a daily basis is selected, Elasticsearch automatically created for both days new indices. In total, 3,556 tweets are stored in Elasticsearch with a total size of 17.3 MB. As Logstash continues to run, the number of documents is increasing since new Tweets are fetched continuously by Logstash and stored in Elasticsearch.

Kibana

Figure 6.15 provides an excerpt of the previously mentioned fields. Since Elasticsearch is schema-free, it automatically detects appropriate data types for fields and analyzes them in order to make them available for full-text searches. In order to answer all of the aforementioned questions in Section 6.3, two dashboards with appropriate visualizations are created. Figure 6.16 demonstrates the first dashboard. The line chart on the top left corner helps to find out how many tweets are being tweeted over the last 12 hours for the key words "Big Data", "Elasticsearch", "Logstash", "Kibana", "Elastic", and "ELK stack". The vertical bar chart on the top right corner provides the necessary information in order to find out how many people are tweeting in different languages. With the help of the area chart on the left, one can find out the number of people who are tweeting in various languages using any Android, Web, iPhone, or iPad devices over the last 12 hours. The vertical bar chart on the right visualizes the number of people who are tweeting from various countries using any of Android, Web, iPhone, and iPad devices. The line chart at the bottom helps to find out what the most popular retweeted user names are for the predefined key words. Figure 6.17 illustrates the second dashboard. The vertical bar chart on the top left corner helps to find out the most popular hashtags related to the predefined key words over the last 12 hours on an hourly basis. With the help of the vertical bar chart on the top right corner, one can compute the number of people who are tweeting from different geographical locations. The pie chart on the left computes the top tweeting user's screen name, who has tweeted the most tweets related to the predefined key words. The area chart on the



★ social_media-*

This page lists every field in the **social_media-*** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's Mapping API

name	type	format	analyzed	indexed	controls
retweeted_status.quoted_status.entities.hashtags.text	string		✓	✓	
quoted_status.extended_entities.media.video_info.duration_millis	number			✓	
retweeted_status.in_reply_to_screen_name.raw	string			✓	
retweeted_status.quoted_status.entities.media.sizes.medium.resize	string		✓	✓	
retweeted_status.quoted_status.extended_entities.media.sizes.medium.h	number			✓	
retweeted_status.extended_entities.media.display_url.raw	string			✓	
retweeted_status.extended_entities.media.source_user_id_str.raw	string			✓	
retweeted_status.place.bounding_box.coordinates	number			✓	
retweeted_status.quoted_status.user.protected	boolean			✓	
retweeted_status.extended_entities.media.sizes.small.w	number			✓	
retweeted_status.entities.media.sizes.large.resize.raw	string			✓	
extended_entities.media.id_str.raw	string			✓	
extended_entities.media.sizes.medium.h	number			✓	

Figure 6.15.: settings page with an excerpt of available fields

right helps to find out how many tweets are coming from any of Android, Web, iPhone, and iPad devices over the last 12 hours. The metrics at the bottom provide some useful statistical information, including the total count of documents in the index, the unique count of used hashtags, languages, retweeted languages, screen names of people who have tweeted, retweeted status user screen names, and time zones from which people have tweeted, percentiles of user favorites count, and percentile rank of 5,000 for the user status count.

6.3.3. Key Findings

With the help of social media data analysis experiments, the subsequent key finding is derived:

- **Schema-free data schema**: Since Elasticsearch is schema-free, it is able to detect appropriate data types for the document fields. Also, whenever Elasticsearch identifies an unseen document field, it automatically integrates it into a document type. Due to this functionality, the present data schema is continuously extended without any manual configuration. This is also called implicit mapping. However, Elasticsearch has problems by detecting `geo_point` data types. It automatically assigns numeric documents fields with two-digits to the `float` data type erroneously. Therefore, in Logstash an index template has to be provided so that Elasticsearch assigns to a certain document field the specified data type. This functionality is called explicit mapping.

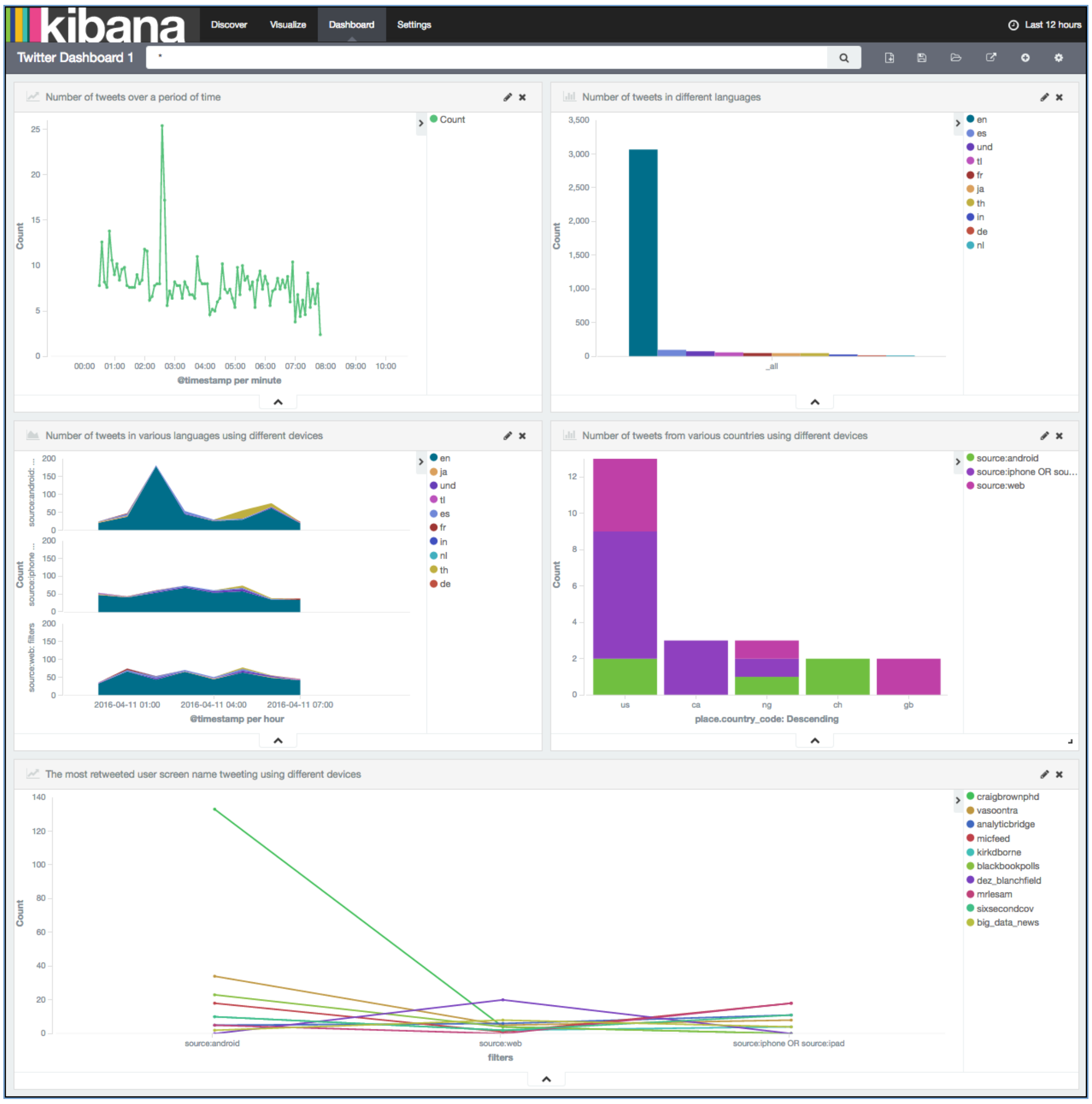
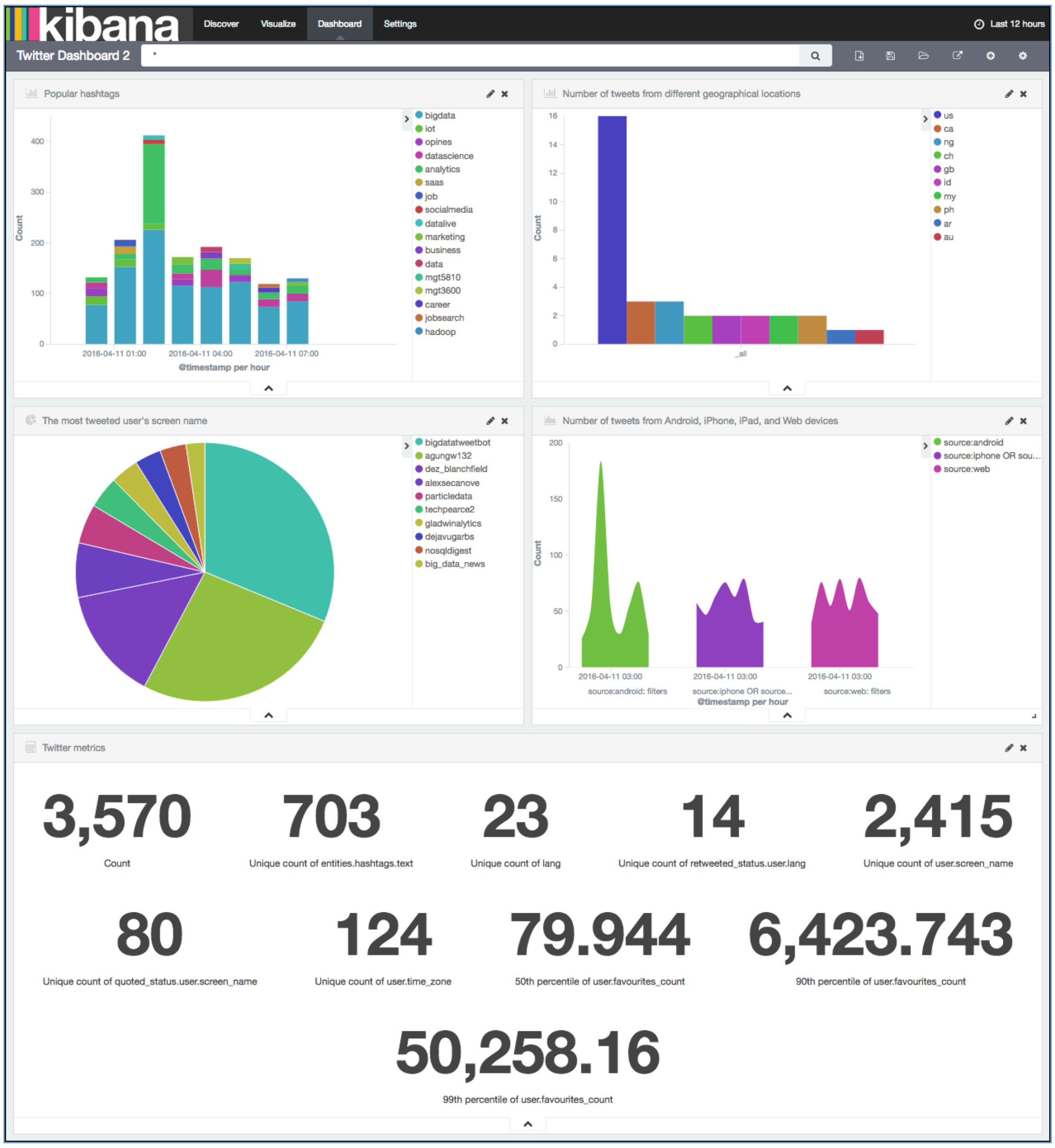


Figure 6.16.: First dashboard of social media data analysis

6.4. Experiment 4: Performance Benchmark Analysis

The previously presented experiments were of qualitative nature. However, this experiment addresses the quantitative assessment of the ELK stack. This is especially relevant for Big Data use cases which exhibit high data velocity. Figure 6.18 already indicates which parts of the ELK stack are benchmarked. These are benchmarks for Logstash's data ingestion performance, Elasticsearch data indexing performance, and Elasticsearch data querying performance. Kibana's performance is neglected since it only renders Elasticsearch's search responses without providing any calculation or com-



3,570
Count

703
Unique count of entities.hashtags.text

23
Unique count of lang

14
Unique count of retweeted_status.user.lang

2,415
Unique count of user.screen_name

80
Unique count of quoted_status.user.screen_name

124
Unique count of user.time_zone

79.944
50th percentile of user.favourites_count

6,423.743
90th percentile of user.favourites_count

50,258.16
99th percentile of user.favourites_count

Figure 6.17.: Second dashboard of social media data analysis

putation logic.

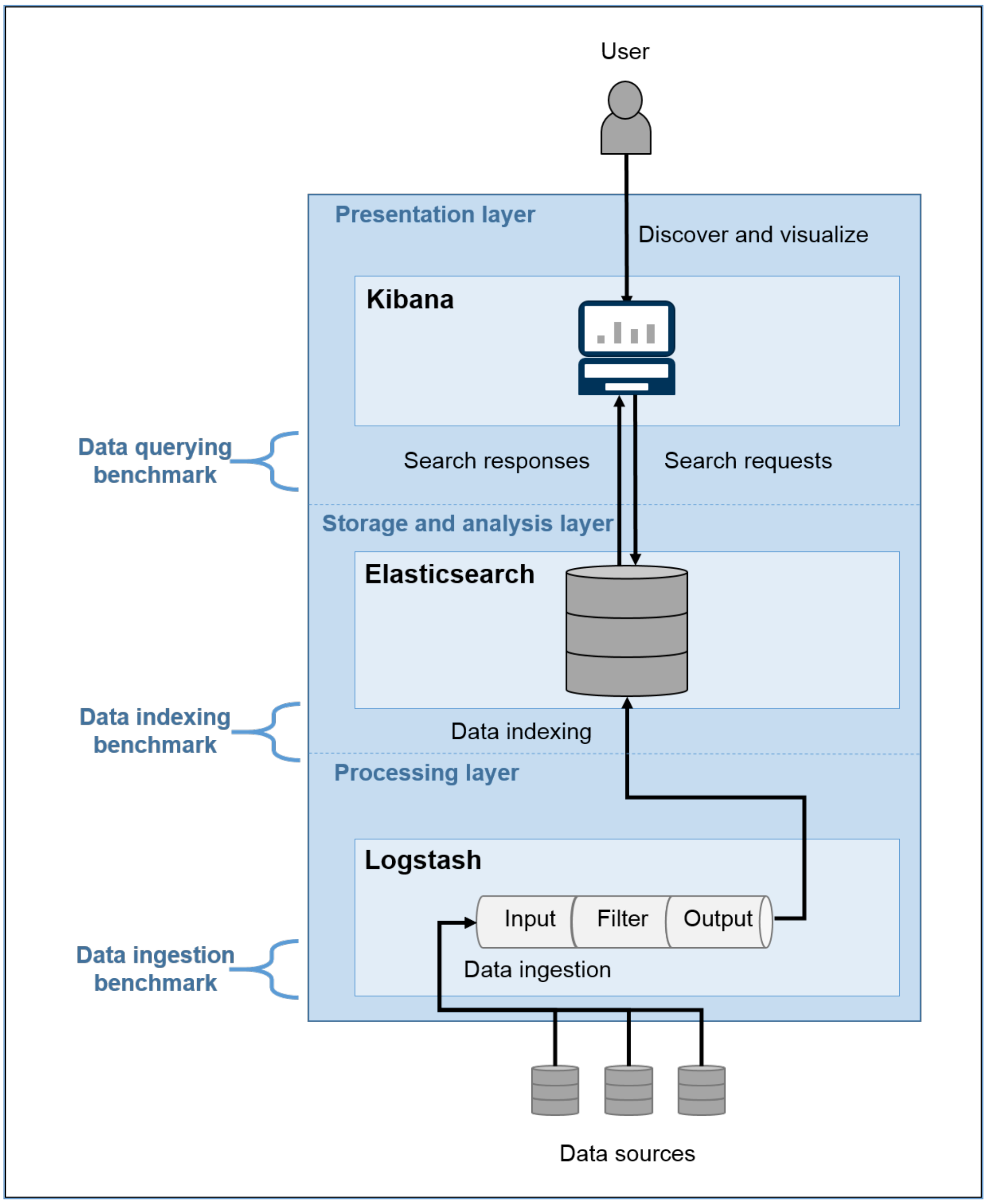


Figure 6.18.: Mapping between ELK stack architecture and performance benchmarking

6.4.1. Data Basis and Data Overview

Benchmark 1: Data ingestion performance of Logstash

For Logstash's data ingestion performance, similar semi-structured machine-generated data in the XML format as in the second experiment is used. Instead of synthesizing data with a Python script, real machine-generated data is utilized for this benchmark which only consists of string values.

Benchmark 2: Data indexing performance of Elasticsearch

The data basis for Elasticsearch's data indexing performance constitutes of synthesized machine-generated data in the JSON format. The key-value pairs of the JSON files resemble the attributes and values of the XML files from the second experiment. One difference is that the JSON files additionally have header parts which consist of corresponding Elasticsearch index and document type key-value pairs. These header parts are important, since they allow to use Elasticsearch's Bulk API[55].

Benchmark 3: Data querying performance of Elasticsearch

The benchmark for assessing Elasticsearch's data querying performance uses the same JSON files as in the second benchmark.

6.4.2. Benchmark Results

Benchmark 1: Data ingestion performance of Logstash

This benchmark analyzes Logstash's ingestion performance. The evaluation time frame amounts 30 minutes. The selection of the time frame equals the time frame in which the data velocity is very high. Within this benchmark, the Logstash-forwarder with the corresponding `lumberjack` input plugin is used. In this context, the Logstash-forwarder listens to a predefined directory on a remote server and directly forwards the newly created XML files to the central Logstash server. Logstash applies some filtering operations on the incoming events, namely the `metrics` filter plugin for providing data ingestion rates (documents per second), the `xml` filter plugin with the `xpath` field in order to extract attributes and their values from the XML files, the `date` filter plugin for transforming the date format of a retrieved date field into Logstash's predefined timestamp format, the `grok` filter plugin for extracting two more attributes with their values by using regular expressions, and some conditions for recalculating some numeric values of fields. Within the `output` block, the `elasticsearch` output plugin is defined in order to ship the filtered events to Elasticsearch. Furthermore, it also contains the `stdout` filter plugin with a `codec` so that the data ingestion rates with corresponding timestamps can be printed in the command line. This benchmark is based on the following settings:

- Logstash's heap size amounts 10 GB,
- Logstash runs on a single thread,
- the benchmark runs on a single node,

[55]http://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html

- the OS of the underlying node is Ubuntu version 14.04.3, and
- the node has 8 CPUs with 2.40 GHz each.

Figure 6.19 delineates Logstash's data ingestion performance. The lightly grey columns represent the amount of newly created events at a specific minute. The green line illustrates Logstash's ingestion performance retrieved by the `metrics` filter plugin. The error bars indicate the lower and upper standard deviations of Logstash's performance. For each data point of the green line, 12 ingestion rates at a specific minute are observed. The mean and standard deviation are calculated based on observations. The blue line represents the Logstash-forwarder's performance for sending events from the remote server to the central Logstash server. The Logstash-forwarder's shipping performance varies highly in the range of 600 to 800 documents per second. Logstash's data ingestion performance seems to be more stable — it is in the range of 400 to 500 documents per second. The number of created machine-generated events lies just over 400 events per second. In total, 28 of 30 data points show that Logstash is significantly able to ingest more data events than created. Also, the Logstash-forwarder is able to ship a lot of more events than are created. The creation of XML files and Logstash-forwarder's shipping performance reveal a positive correlation. Between Logstash and other data points, there are no noteworthy correlations. In general, one can say that Logstash is able to handle a high data velocity of events as well as in time frames in which the amount of created machine-generated data is very high for a specific minute. During some test benchmarks with synthesized data, Logstash was able to ingest 40,000 to 45,000 when no filtering operations were performed by Logstash and the data velocity was higher than in this benchmark. This indicates that Logstash's performance is highly dependent on both the incoming events and complexity of filtering operations.

Benchmark 2: Data indexing performance of Elasticsearch

In this benchmark, Elasticsearch's data indexing performance is analyzed. It aims to derive the indexing rate and the total indexing time for a given size of machine-generated JSON files. Additionally, it targets to unveil performance differences when fields should be analyzed (make them available for full-text search) or not. In order to able to compare analyzed vs. not analyzed fields, explicit mappings have to be provided for Elasticsearch so that it does not automatically decide whether a field should be full-text searchable or not. Particularly, in this benchmark 12 different number of documents are created, namely: 100 K, 200 K, 400 K, 600 K, 800 K, 1 M, 2 M, 3 M, 4 M, 5 M, 6 M, and 7 M documents. Elasticsearch's Bulk API allows to send these JSON documents very fast to the responsible Elasticsearch node. However, an appropriate Bulk size of a request has to be identified. The Elasticsearch community recommends to use a Bulk size between 5 MB to 11 MB. For this benchmark, the Bulk size of 5.5 MB is selected since it helps to evenly distribute the documents per Bulk request and has demonstrated the best performance among other Bulk sizes in test benchmarks. This benchmark is based on optimized settings for data indexing, which are stated in the following:

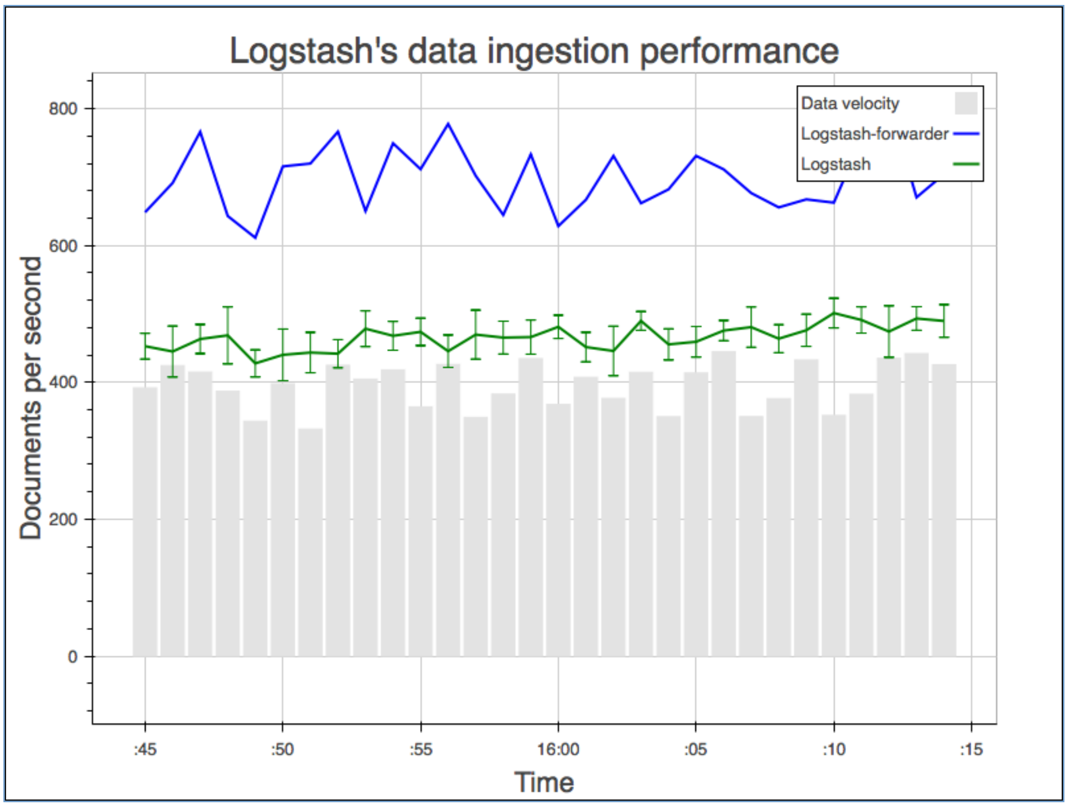


Figure 6.19.: Overview of Logstash's data ingestion performance

- Elasticsearch's heap size amounts 31 GB,
- Elasticsearch runs on a single thread,
- Elasticsearch's replication factor amounts 1,
- the used Elasticsearch index consists of two shards,
- the throttle type of the index is disabled,
- the refresh interval of the index is disabled,
- the benchmark runs on a single node,
- the OS of the underlying node is Ubuntu version 14.04.3, and
- the node has 8 CPUs with 3.50 GHz each.

Figure 6.20 illustrates Elasticsearch's data indexing performance in terms of indexed documents per second. The red line represents Elasticsearch's indexing performance when fields are not analyzed whereas the green line shows Elasticsearch's indexing performance when fields are analyzed. The error bars for both lines indicate the lower and upper standard deviations of Elasticsearch's performance. For each data point, 5 indexing rates are observed. The mean and standard deviation per data point are calculated based on these observations. This figure already indicates that Elasticsearch's performance becomes more stable after 2 M documents while increasing number of indexed documents. This can be recognized by the size of the error bars. After becoming more stable, Elasticsearch performs significantly better when fields are not analyzed. It is able to index 9,560 to 10,800 events per second after becoming stable. When Elasticsearch should analyze fields, it is able to index 9,160 to 9,520 documents per second. Furthermore, one can see that Elasticsearch's performance fluctuates greater when Elasticsearch does not analyze document fields. Figure 6.21 comes to similar conclusions

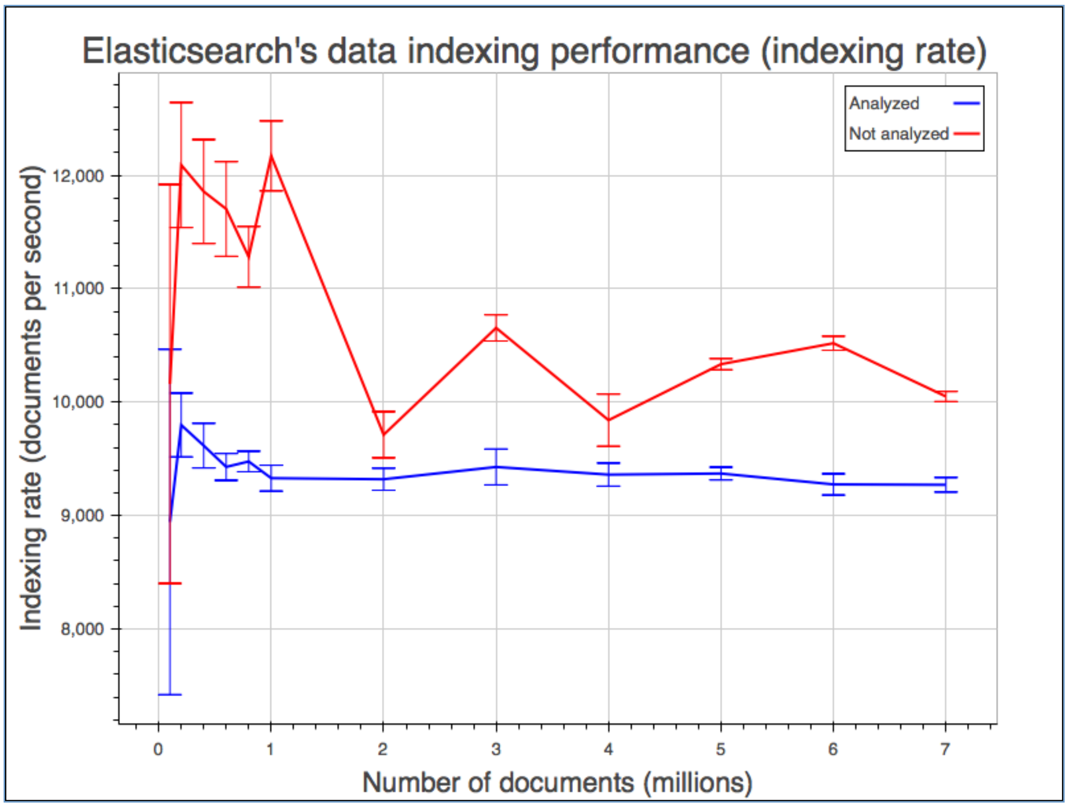


Figure 6.20.: Overview of Elasticsearch's data indexing performance (indexing rate)

since it uses the same results but a different point of view, namely total indexing time instead of indexing rate. In general, one can say that Elasticsearch will need less indexing time when it does not analyze fields. However, within the range between 100 K and 2 M documents, some overlaps of the error bars are present which indicate that at these points the indexing time does not significantly differ. When Elasticsearch has to analyze fields, it will need on average 755 seconds or 12.5 minutes in order to index 7 M documents with a total size of about 3.8 GB. In contrast, it will need on average 696 seconds or 11.6 minutes in order to index 7 M documents when fields do not have to be analyzed.

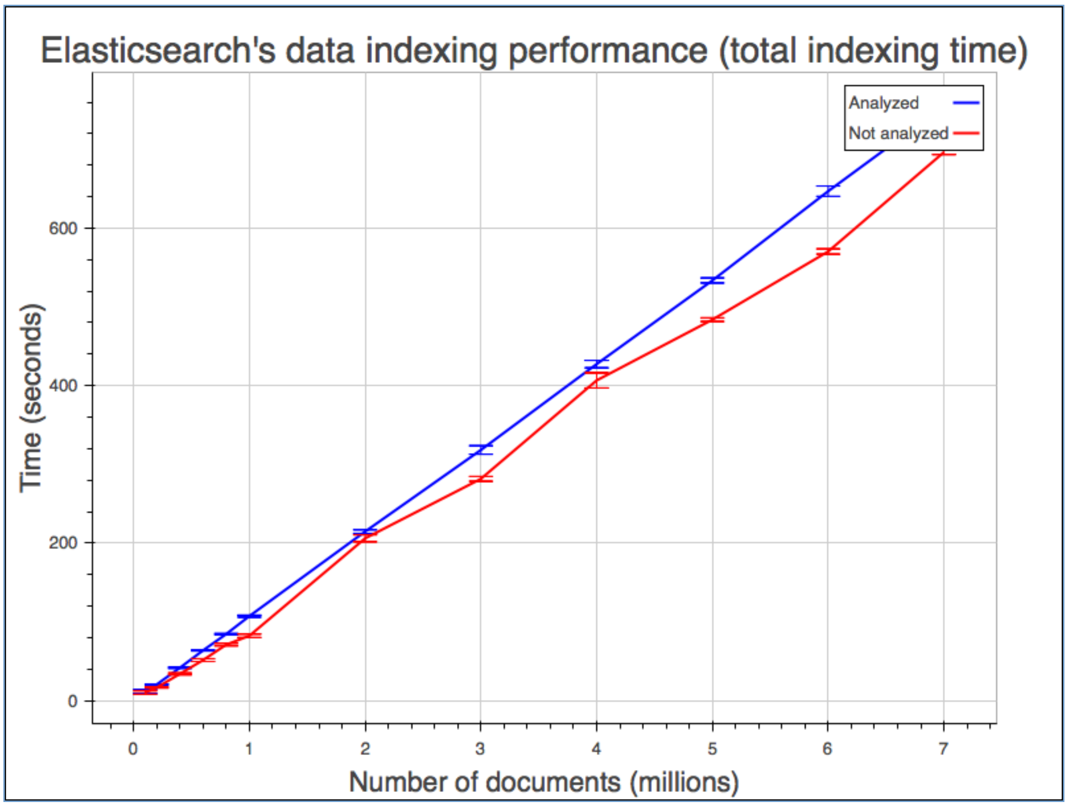


Figure 6.21.: Overview of Elasticsearch's data indexing performance (total indexing time)

Benchmark 3: Data querying performance of Elasticsearch

Last but not least, Elasticsearch's data querying performance should be analyzed within the last benchmark. In total, two analytical queries are performed and their execution times are ascertained. The analytical queries are explained below:

- **Filtered geo distance query**: Find all documents which their distances to a specified geographical location are less than 6,000 kilometers.
- **Filtered range query with average aggregation**: Find all documents within the geographical location of the USA and calculate their average longitude and latitude values.

Again, this benchmark also uses 12 different number of machine-generated JSON files (see 6.4.2) and measures the query performance of Elasticsearch. All document fields

are analyzed in this benchmark. This benchmark uses nearly the same settings as the benchmark for Elasticsearch's data indexing performance. However, in this benchmark the throttle type of the index is not disabled and the Optimize API[56] is used before performing the benchmarks.
Figure 6.22 delineates the results of the filtered geo distance query. Again, for all data points, 5 execution times are observed. The mean and standard deviation per data point are calculated based on the observations. Section 5.3.2 has already highlighted that a major difference between Elasticsearch's queries and filters lies in the fact that filters can be cached and therefore subsequent execution of the filters are faster than the first one. The blue line represents the filtered query with including the first observation with the high first execution time whereas the red line does not include the first observation. One can see that the execution time is reduced when the first observation is neglected. Furthermore, the error bars are smaller for the filtered query without the first observation. Elasticsearch needs on average 706.8 milliseconds for 7 M documents with a total size of about 3.8 GB in order to find all documents which their distances to a specified geographical location are less than 6,000 kilometers, when the first observation should be considered. Otherwise, the average execution time for this query for 7 M documents amounts 273.5 milliseconds. Although the number of documents is increased sharply, the corresponding execution times do not increase in the same extent. The results for second query, namely the filtered range query with average aggregation can be seen in Figure 6.23. It shows in some degree the similar aspect of filtered queries but not with the same extent. The reason for this is that the filtered range query is combined with an aggregation which cannot be filtered. Therefore, the total execution times do not significantly differ from each other. Elasticsearch needs on average 79.4 milliseconds for 7 M documents with a total size of about 3.8 GB in order to find all documents within the geographical location of the USA and to calculate their average longitude and latitude values, when the first observation should be considered. Otherwise, the average execution time for this query for 7 M documents amounts 75.25 milliseconds.

6.4.3. Key Findings

Based on the performed benchmark analysis, the following key findings are deduced:

- **High throughput of events**: By considering Logstash's data ingestion performance and Elasticsearch's data indexing performance, one can see that the ELK stack is able to handle high data velocity. 28 of 30 observations show that Logstash is able to ingest more events than generated on time frame which exhibits high data velocity. Also, Elasticsearch is capable of indexing a vast amount of events per second, in particular 9,160 to 10,800 events per second.

- **Improvement of Elasticsearch's indexing performance when document fields are not analyzed**: The second benchmark has demonstrated that Elasticsearch's indexing performance can be improved when document fields are not analyzed.

[56]http://www.elastic.co/guide/en/elasticsearch/reference/current/indices-optimize.html

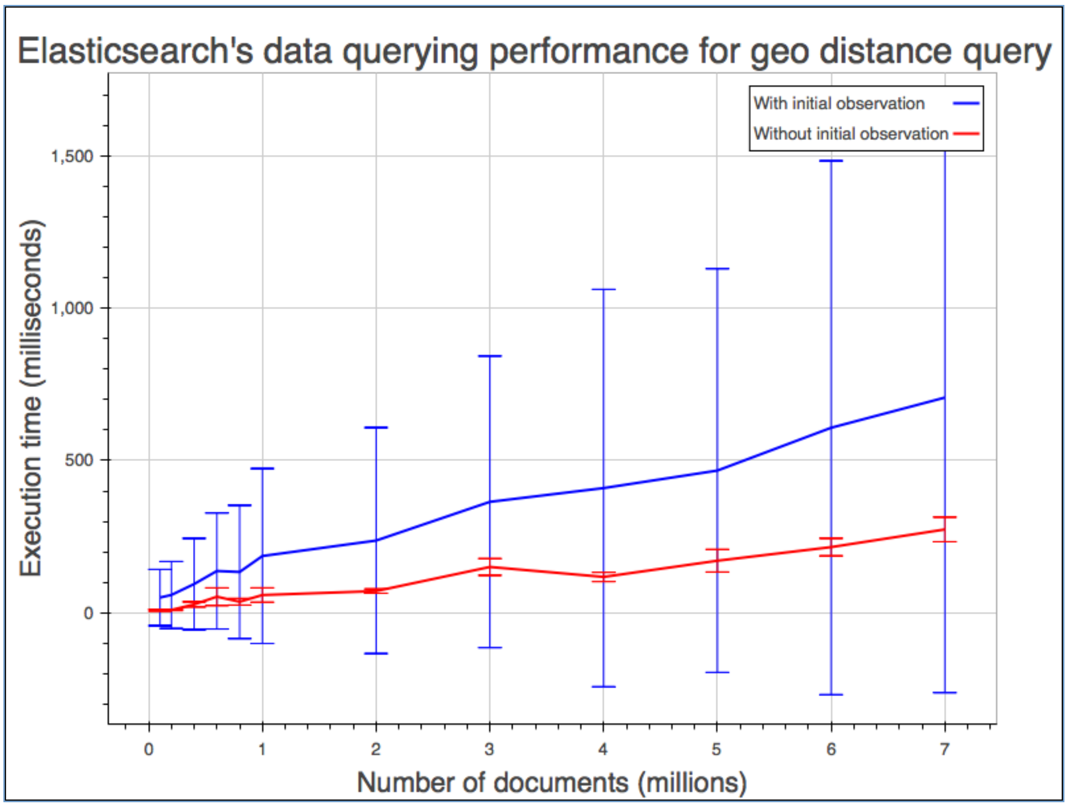


Figure 6.22.: Overview of Elasticsearch's data querying performance for geo distance query

This is specifically important when Elasticsearch's indexing performance should be optimized and certain document fields should be not analyzed since a full-text search would make no sense, e.g., numeric document fields.

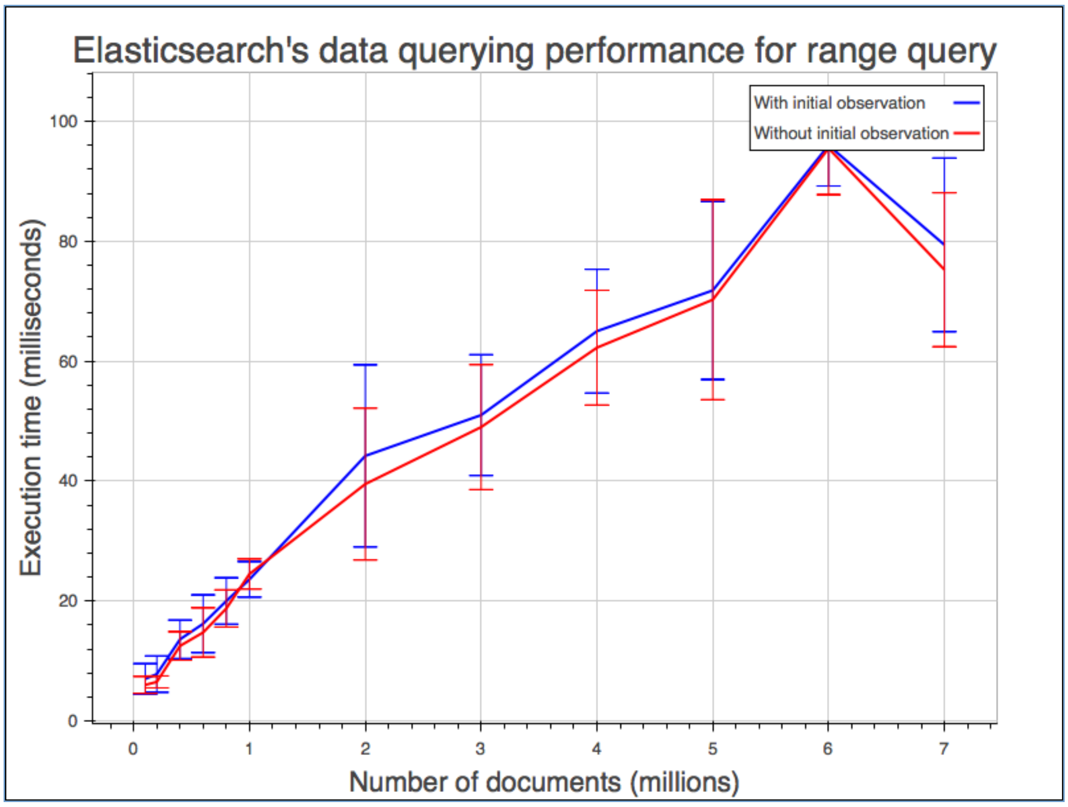


Figure 6.23.: Overview of Elasticsearch's data querying performance for range query

Part V.

Conclusion

7. Conclusion and Outlook

This chapter summarizes the thesis in Section 7.1, highlights the key findings in Section 7.2, reveals the limitations of the thesis in Section 7.3, and provides a brief outlook of possible future investigations in Section 7.4.

7.1. Summary

The aim of the thesis was to assess the applicability of the ELK stack for Big Data use cases. At the beginning of this thesis, the motivation for the aforementioned problem was described, followed by the deduced objectives and underlying research approach in order to ensure rigor and relevance. Search-based data discovery tools and the four types of data analytics capability were presented in order to establish a common comprehension for the subsequent analyses. Within the descriptive study, Solr with its respective SiLK stack and Splunk were identified as similar technologies, which were briefly described with their key features and potential use cases. Subsequently, the search engines and search-based data discovery tools were juxtaposed in opposition so that their characteristics could be differentiated from each other. Based on personal experience within this thesis, additional criteria for further comparison were presented for search-based data discovery tools. As the gist of the descriptive study, the individual technologies of the ELK stack were analyzed by elaborating their key features. With the help of four experiments, the ELK stack was implemented and benchmarked in order to assess its applicability for Big Data use cases qualitatively and quantitatively. Based on the performed descriptive study and experiments, key findings were deduced. The remainder of this thesis summarizes the key findings about the ELK stack, describes the limitation of the conducted analyses, and touches upon possible future work on the topic.

7.2. Results

The ELK stack is a serious competitor to commercial search-based data discovery tools, e.g., Attivio, IBM, Oracle, or Splunk. It is actively developed by a strong team of developers and an actively engaged community. Although it is relatively young, it is able to master the 5 Vs of Big Data very well due to its scalable architecture, data traceability functionalities, support of various data formats and data sources, and its user-friendly web interface for descriptive data analytics and full-text searches. Also by considering the implementations and performance benchmarks, the ELK stack's eligibility for Big

Data use cases can be quickly noticed.
The following key findings are deduced based on the conducted descriptive study in Chapter 5 and performed experiments in Chapter 6:

- **Scalability**: The ELK stack can be scaled by adding computing sources (vertical scaling). Logstash can be scaled by adding a message queue or by adding more Logstash indexing instances (see Section 5.2.2). Elasticsearch can easily scaled out by adding additional Elasticsearch nodes to an existing cluster (horizontal scaling) (see Section 5.3.2).

- **Support of various data sources and data formats**: Logstash provides 49 different input plugins to tap a huge amount of data sources. Additionally, it offers 42 filter plugins for ingesting various data formats. With the help of its grok filter plugin, unstructured data can be ingested by making the use of regular expressions and custom/predefined patterns (see Section 5.2.2).

- **Capabilities for geospatial analyses**: Elasticsearch offers several geo filters and aggregations in order to perform various geospatial analyses (see Section 5.3.2).

- **Lack of data protection mechanisms**: By default, the ELK stack does not include any authentication mechanisms. By using the chargeable Shield product of Elastic, the ELK stack can be secured. It provides functionalities for encrypting communications and a role-based access control (see Section 5.3.2).

- **Provision of basic visualization types**: Kibana provides a set of basic visualization types, including pie chart, area chart, line chart, vertical bar chart, data table, metric, tile map, and markdown widget. However, more sophisticated or statistical graphics are not offered by Kibana (see Section 5.4.2).

- **Limited data analytics capabilities**: The ELK stack's data analytics capability is determined by Elasticsearch. Elasticsearch is primarily designed as a search engine which is why it provides only descriptive analytics capabilities, namely a set of bucket and metric aggregations. Elasticsearch does not support functionalities for interactive visualization, machine learning, data mining, predictive/decision modeling, simulation, and optimization (see Section 5.3.2).

- **Timestamping of events**: Logstash is able to add timestamps to incoming data or use custom date fields as timestamps so that Elasticsearch and Kibana can perform time-based analyses (see Section 6.1.2).

- **High data variability may cause data parse failures**: Logstash's capabilities for highly-variable and multiple hierarchical XML data seem to be insufficient. Although it provides some appropriate filter plugins, programming languages such as Python provide more appropriate library functions for parsing this type of XML data (see Section 6.2.2).

- **Data anonymization capabilities**: Logstash provides an anonymize filter plugin in order to protect and hash sensitive information of fields (see Section 6.2.2).
- **Data traceability capabilities**: The ELK stack supports data traceability by using originally created unique identifiers of events or by generating unique identifiers in Elasticsearch (see Section 6.2.2).
- **Simultaneous indexing and querying of events**: Kibana can perform searches and aggregations on continuously ingested and indexed data (see Section 6.2.2).
- **Automatic update of web interface**: While data is indexed by Elasticsearch, Kibana automatically refreshes search results and visualizations with the help of its auto-refresh functionality (see Section 6.2.2).
- **Fractional amount of supported of output formats**: Kibana only provides raw and formatted CSV files as output formats which can be used as data input for further analyses (see Section 6.2.2).
- **Near real-time data analytics**: The ELK stack is capable of near real-time data analytics. It needs time for filtering, indexing, and visualizing data. Regardless of this data accessibility delay, data is never directly accessible in Kibana since Elasticsearch's index has to be refreshed (see Section 6.2.2).
- **Schema-free data schema**: Elasticsearch is schema-free wherefore it is able to detect appropriate data types for the document fields (see Section 6.3.2).
- **High throughput of events**: By considering Logstash's data ingestion performance and Elasticsearch's data indexing performance, one can see that the ELK stack is able to handle high data velocity (see Section 6.4.2).
- **Improvement of Elasticsearch's indexing performance when document fields are not analyzed**: Elasticsearch's indexing performance can be improved when document fields are not analyzed (see Section 6.4.2).

7.3. Limitations

After the extensive descriptive study in Chapter 5 and performed experiments in Chapter 6, the following limitations can be crystallized out:

- **Neglected quantitative evaluation**: The majority of the applicability assessments of the ELK stack are based on qualitative evaluation, e.g., support of various data sources and input data types. Adding more quantitative assessments of the stack would improve the expressiveness of the conducted analyses. For instance, comparing the ingesting and indexing times in Logstash and Elasticsearch for different experiments would provide insights into the ELK stack's performance of handling different data structures and data formats. Also, monitoring the CPU and disk space usage would point out the strengths and weaknesses of the ELK stack during data ingestion and storing. This would help to examine Logstash's limitations for different filter operations. Additionally, this would highlight how Elasticsearch manages and stores different documents in its indices.

- **Neglected inspection of security aspects**: The descriptive study has touched the security weaknesses of the ELK stack lightly. It has mentioned that Elasticsearch does not provide by default security mechanisms. Moreover, it also proposed the usage of the chargeable Shield plugin for Elasticsearch in order to protect data. However, it did not demonstrate the various security features of the Shield plugin in detail.
- **Fractional amount of performed experiments**: In this thesis, only four experiments were carried out. Although they addressed various Big Data use cases adequately, there are still use cases which are not examined. For instance, Logstash's capability for handling various data types was assessed with structured machine-generated and textual data. An extensive assessment of its capability for unstructured, highly variable, and hierarchical log files is missing.
- **Limited expressiveness of conducted analyses for other technology versions**: The descriptive study has stated that the ELK stack is actively developed and strongly promoted by a large community. This indicates that each new release may cause big changes in the capabilities, features, and performance of the stack. Due to this reason, some assessments may be irrelevant or invalid for newer versions. For instance, the Logstash-forwarder is replaced by Elastic's new Beats[57] product. This replacement entails major performance and capability changes of forwarding event data from remote servers to a central Logstash servers, e.g., provision of different types of beats which are optimized for sending different types of operational data to Logstash. As a consequence, the performed benchmark for Logstash's performance may lose expressiveness.
- **Constrained level of detail**: This thesis focused on providing an holistic view on the ELK stacks key features and capabilities. For that reason, in depth analyses of the underlying technologies were not possible. This is especially important for Elasticsearch, because it provides many more querying capabilities than were explored. For example, Elasticsearch provides a Validate API in order to allow a user to validate a potentially expensive query without executing it [38]. Another example is the Multi Search API which allows to execute several search requests within the same API [32].
- **Missing evaluation of Big Data ecosystem integration**: The evaluation of the ELK stack was mostly focused on the holistic assessment of the ELK stack without considering its integration within a Big Data software architecture. However, this evaluation seems indispensable, since the data analytics capabilities of the ELK

[57]http://www.elastic.co/products/beats

stack are limited on descriptive analytics and near real-time search. The usage of additional Big Data technologies are necessary for subsequent data analytics tasks.

7.4. Future Work

A sincere effort has been made in this thesis to delineate the key features and capabilities of the ELK stack and to assess its applicability for various Big Data use cases. This thesis has clearly demonstrated the strengths and weaknesses of the ELK stack, with its main focus lying on descriptive analytics and full-text search. However, the limitations of this thesis predesignate the direction for future research. The ELK stack should not be considered in isolation without its integration with a Big Data ecosystem. This is especially of interest, if a Big Data use case requires more advanced data analytics capabilities than descriptive analytics. This limitation results in two future research directions. Firstly, Elastic provides Elasticsearch for Hadoop (ES-Hadoop)[58] that is a two-way connector which provides a near real-time search for Hadoop while the Hadoop ecosystem offers a multitude of analytics capabilities. It is designed for leveraging Hadoop's Big Data analytics capabilities and the near real-time search of Elasticsearch. Figure 7.1 provides an overview of the ES-Hadoop integration. Although, Logstash is not provided in this figure, it can be used for collecting and parsing log files which are sent to Elasticsearch and analyzed by different Hadoop technologies. According to that, a future research direction may be analyzing the ELK stack's integration with the Hadoop ecosystem. For instance, different Big Data use cases can be implemented which involve the data transfer between the two ecosystems. Herein, different benchmarks can be performed in order to assess the reliability and performance of the data transfer between those and uncovering potential risk points, e.g., scalability of the connector. Secondly, a Big Data software architecture usually consists of multiple

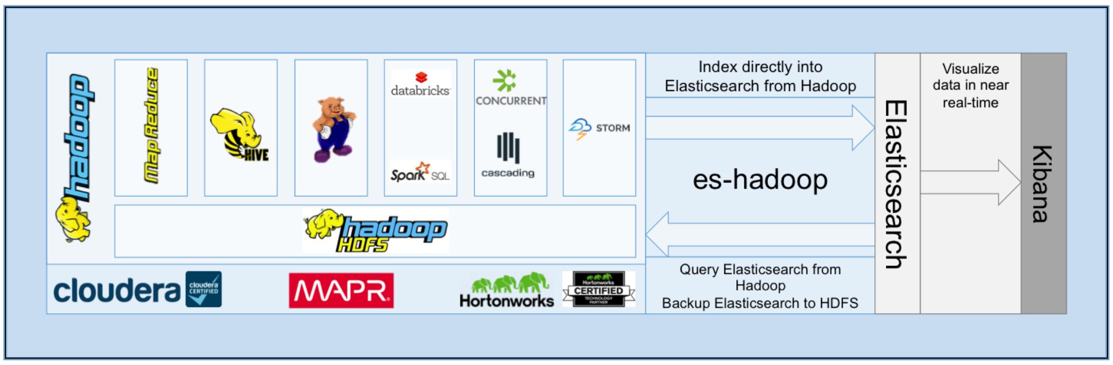

hadoop
MapReduce
HIVE
databricks
Spark SQL
CONCURRENT
cascading
STORM
hadoop HDFS
cloudera
MAPR
Hortonworks
CERTIFIED
Index directly into Elasticsearch from Hadoop
es-hadoop
Query Elasticsearch from Hadoop
Backup Elasticsearch to HDFS
Elasticsearch
Visualize data in near real-time
Kibana

Figure 7.1.: Elasticsearch for Hadoop [14]

infrastructure layers and components in order to support a specific application domain.

[58]http://www.elastic.co/products/hadoop

Figure 7.2 provides an overview of the infrastructure layers and components in order to support connected transportation systems. In this architecture, each Big Data technology is responsible for a certain task. For instance, Kibana can be used for data science, Elasticsearch for search, and Logstash for data ingestion, loading, and integration. Following this, describing the ELK stack's integration within this architecture may be a future research direction. During this description, the ELK stack can be separated into its three individual technologies in order to uncover meaningful combinations with other technologies.

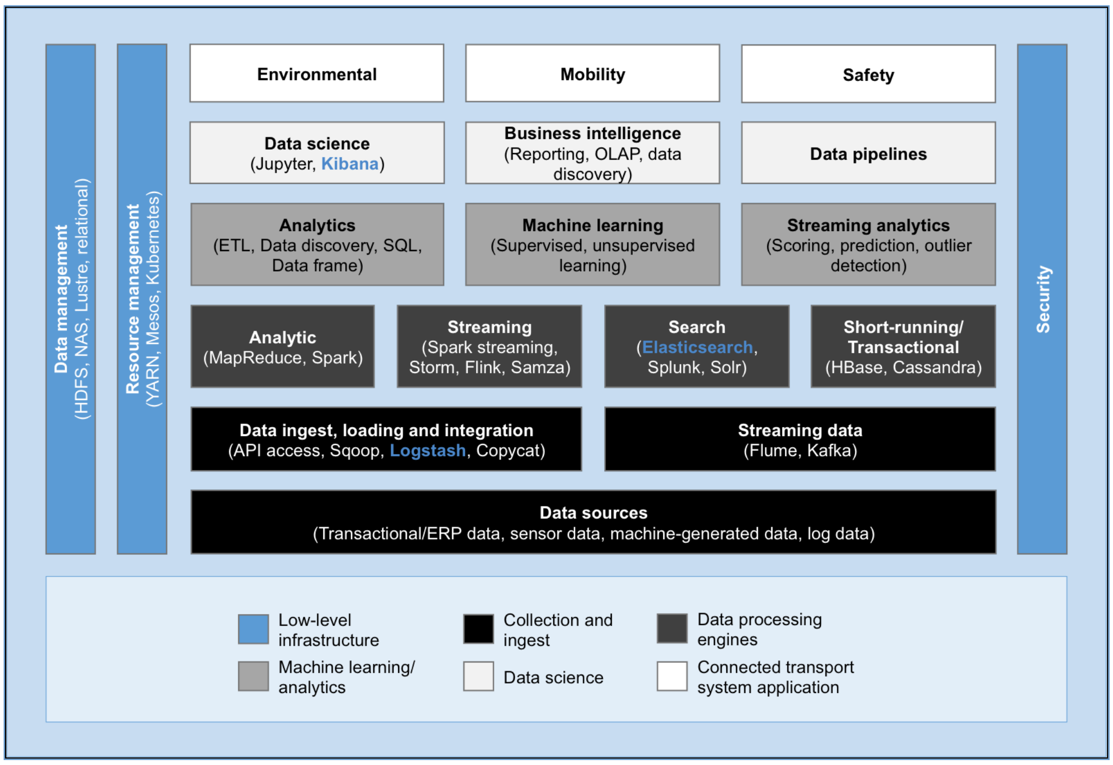

Data management (HDFS, NAS, Lustre, relational)
Resource management (YARN, Mesos, Kubernetes)
Environmental
Mobility
Safety
Data science (Jupyter, Kibana)
Business intelligence (Reporting, OLAP, data discovery)
Data pipelines
Analytics (ETL, Data discovery, SQL, Data frame)
Machine learning (Supervised, unsupervised learning)
Streaming analytics (Scoring, prediction, outlier detection)
Analytic (MapReduce, Spark)
Streaming (Spark streaming, Storm, Flink, Samza)
Search (Elasticsearch, Splunk, Solr)
Short-running/ Transactional (HBase, Cassandra)
Data ingest, loading and integration (API access, Sqoop, Logstash, Copycat)
Streaming data (Flume, Kafka)
Data sources (Transactional/ERP data, sensor data, machine-generated data, log data)
Security
Low-level infrastructure
Collection and ingest
Data processing engines
Machine learning/ analytics
Data science
Connected transport system application

Figure 7.2.: Big Data software architecture for connected transportation systems [18]

Appendix

A. Appendix

A.1. Index Template for Textual Data Analysis

Listing A.1 comprise the index template which was utilized for the textual data analysis in Section 6.1.2.

Listing A.1: Index template for textual data analysis in Section 6.1.2

```
{
  "template": "road_weather_demonstration",
  "order":    1,
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "rwd": {
      "dynamic_templates": [
         {
            "string_fields": {
               "mapping": {
                  "index": "analyzed",
                  "omit_norms": true,
                  "type": "string",
                  "fields": {
                     "raw": {
                        "index": "not_analyzed",
                        "ignore_above": 256,
                        "type": "string"
                     }
                  }
               },
               "match_mapping_type": "string",
               "match": "*"
            }
         }, {
            "boolean_fields": {
               "mapping": {
                  "type": "boolean"
               },
               "match": "*is_*"
            }
         }
      ],
      "_all": {
        "enabled": false
      },
      "properties": {
          "Location": {
          "type": "geo_point"
        }
      }
    }
  }
}
```

A.2. Index Template for Machine-Generated Data Analysis

Listing A.2 presents the index template which was utilized for the machine-generated data analysis in Section 6.2.2.

Listing A.2: Index template for machine-generated data analysis in Section 6.2.2

```
{
"template": "machine_generated-*",
  "order":    1,
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "mach_gen": {
      "dynamic_templates": [
         {
            "string_fields": {
               "mapping": {
                  "index": "analyzed",
                  "omit_norms": true,
                  "type": "string",
                  "fields": {
                     "raw": {
                        "index": "not_analyzed",
                        "ignore_above": 256,
                        "type": "string"
                     }
                  }
               },
               "match_mapping_type": "string",
               "match": "*"
            }
         }, {
            "boolean_fields": {
               "mapping": {
                  "type": "boolean"
               },
               "match": "*is_*"
            }
         }
      ],
      "_all": {
        "enabled": false
      },
      "properties": {
          "location": {
          "type": "geo_point"
        }
      }
    }
  }
}
```

A.3. Index Template for Social Media Data Analysis

Listing A.3 presents the index template which was used for the social media data analysis in Section 6.3.2.

Listing A.3: Index template for social media data analysis in Section 6.3.2

```
{
  "template": "social_media-*",
  "order":    1,
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "sm": {
      "_all": {
        "enabled": false
      },
      "dynamic_templates" : [ {
         "message_field" : {
           "match" : "message",
           "match_mapping_type" : "string",
           "mapping" : {
             "type" : "string",
             "index" : "analyzed",
             "omit_norms" : true
           }
         }
       }, {
         "string_fields" : {
           "match" : "*",
           "match_mapping_type" : "string",
           "mapping" : {
             "type" : "string",
             "index" : "analyzed",
             "omit_norms" : true,
             "fields" : {
                "raw" : {
                    "type": "string",
                    "index" : "not_analyzed",
                    "ignore_above" : 256
                }
              }
           }
         }
       } ],
      "properties": {
        "text": {
          "type": "string"
        },
          "coordinates": {
          "properties": {
             "coordinates": {
                "type": "geo_point"
             },
             "type": {
                "type": "string"
             }
          }
       }
      }
    }
  }
}
```

Bibliography

[1] Vahid Abbasi. Phonetic analysis and searching with google glass api. Master's thesis, Uppsala Universitet, 2015.

[2] Veronika Abramova, Jorge Bernardino, and Pedro Furtado. Experimental evaluation of nosql databases. *International Journal of Database Management Systems*, 6(3):1 – 16, 2014.

[3] Yusuf Abubakar, ThankGod S. Adeyi, and Ibrahim Gambo Auta. Performance evaluation of nosql systems using ycsb in a resource austere environment. *International Journal of Applied Information Systems (IJAIS)*, 7(8):23 – 27, 2014.

[4] J. Andreeva, A. Beche, S. Belov, I. Dzhunov, I. Kadochnikov, E. Karavakis, P. Saiz, J. Schovancova, and D. Tuckett. Processing of the wlcg monitoring data using nosql. In *Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics*, page 032048. IOP Publishing, 2014.

[5] Rob Appleyard and James Adams. Using the elk stack for castor application logging at ral. In *The International Symposium on Grids and Clouds (ISGC) 2015*, Taipei, Taiwan, 2015. Proceedings of Science.

[6] S. Bagnasco, D. Berzano, A. Guarise, S. Lusso, M. Masera, and S. Vallero. Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem. In *Journal of Physics: Conference Series*, page 012016. IOP Publishing, 2015.

[7] Jun Bai. Feasibility analysis of big log data real time search based on hbase and elasticsearch. In *Natural Computation (ICNC), 2013 Ninth International Conference on*, pages 1166 – 1170, Shenyang, China, 2013. IEEE.

[8] Michael J. Baker. Writing a literature review. *The Marketing Review*, 1(2):219 – 247, 2000.

[9] Radoslav Bodó, Daniel Kouril, Jirí Sitera, Miloš Mulac, Pavel Vondruška, and Michal Procházka. Crunching logs for fun and profit. Technical report, Západočeské univerzity, 2013.

[10] Radoslav Bodóa and Daniel Kouřil. Efficient management of system logs using a cloud. In *The International Symposium on Grids and Clouds (ISGC) 2013*, Taipei, Taiwan, 2013. Proceedings of Science.

[11] Frank Buytendijk. Hype cycle for big data, 2014. *Gartner*, 2014.

[12] Daniel Cea, Jordi Nin, Rubén Tous, Jordi Torres, and Eduard Ayguadé. Towards the cloudification of the social networks analytics. In *Modeling Decisions for Artificial Intelligence*, pages 192 – 203. Springer, 2014.

[13] Neil Chandler. Agenda overview for analytics, business intelligence and performance management, 2014. *Gartner*, 2014.

[14] Wo Chang. Nist big data interoperability framework: Volume 3, use cases and general requirements. `http://bigdatawg.nist.gov/_uploadfiles/NIST.SP.1500-3.pdf`, 2015. Accessed: 2016-02-27.

[15] Hong-Mei Chen, Rick Kazman, and Florian Matthes. Engineering big data in enterprises. *IEEE*, 2015.

[16] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171 – 209, 2014.

[17] Saurabh Chhajed. *Learning ELK Stack*. Packt Publishing, 2015.

[18] Mashrur Chowdhury, Amy Apon, Kakan Dey, Venkat Gudivada, Pradip Srimani, Beth Plale, Andre Luckow, Hongxin Hu, Chad Steed, John McGregor, Yuanchang Xie, Linh Ngo, Jason Hallstrom, and Jason Thatcher. *Data Analytics in Connected Transportation Systems*. SciTech Publishing, 2016.

[19] DB-Engines. Db-engines ranking of search engines. `http://db-engines.com/en/ranking/search+engine`. Accessed: 2016-02-27.

[20] DB-Engines. Method of calculating the scores of the db-engines ranking. `http://db-engines.com/en/ranking_definition`. Accessed: 2016-02-27.

[21] DB-Engines. Vergleich der systemeigenschaften elasticsearch vs. solr vs. splunk. `http://db-engines.com/de/system/Elasticsearch%3BSolr%3BSplunk`. Accessed: 2016-02-27.

[22] Yuri Demchenko, Canh Ngo, Cees De Laat, Peter Membrey, and Daniil Gordijenko. Big security for big data: Addressing security challenges for the big data infrastructure. In Willem Jonker and Milan Petković, editors, *Secure Data Management: Lecture Notes in Computer Science*, pages 76 – 94, Trento, Italy, 2014. Springer International Publishing.

[23] Manda Sai Divya and Shiv Kumar Goyal. Elasticsearch: An advanced and quick search technique to handle voluminous data. *COMPUSOFT, An international journal of advanced computer technology*, 2(6):171 – 175, 2013.

[24] Martijn T. Dwars, Rik Nijessen, and Rick Wieman. Tracking events at magnet.me. 2014.

[25] Elastic. Deploying and scaling logstash. https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html. Accessed: 2016-02-27.

[26] Elastic. elasticsearch. https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html. Accessed: 2016-02-27.

[27] Elastic. The elk stack in a devops environment. https://www.elastic.co/webinars/elk-stack-devops-environment. Accessed: 2016-02-27.

[28] Elastic. Filter plugins. https://www.elastic.co/guide/en/logstash/current/filter-plugins.html. Accessed: 2016-02-27.

[29] Elastic. Index templates. https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-templates.html. Accessed: 2016-02-27.

[30] Elastic. An introduction to the elk stack. https://www.elastic.co/webinars/introduction-elk-stack. Accessed: 2016-02-27.

[31] Elastic. Map/reduce integration. https://www.elastic.co/guide/en/elasticsearch/hadoop/current/mapreduce.html. Accessed: 2016-02-27.

[32] Elastic. Multi search api. https://www.elastic.co/guide/en/elasticsearch/reference/2.1/search-multi-search.html#search-multi-search. Accessed: 2016-02-27.

[33] Elastic. Output plugins. https://www.elastic.co/guide/en/logstash/current/output-plugins.html. Accessed: 2016-02-27.

[34] Elastic. Setting up and advanced logstash pipeline. https://www.elastic.co/guide/en/logstash/current/advanced-pipeline.html. Accessed: 2016-02-27.

[35] Elastic. Sizing elasticsearch. https://www.elastic.co/blog/found-sizing-elasticsearch. Accessed: 2016-02-27.

[36] Elastic. Types and mappings. https://www.elastic.co/guide/en/elasticsearch/guide/current/mapping.html. Accessed: 2016-02-27.

[37] Elastic. Using kibana with shield. https://www.elastic.co/guide/en/shield/current/kibana.html. Accessed: 2016-02-27.

[38] Elastic. Validate api. https://www.elastic.co/guide/en/elasticsearch/reference/2.1/search-validate.html. Accessed: 2016-02-27.

[39] Elastic. What is an elasticsearch index? `https://www.elastic.co/blog/what-is-an-elasticsearch-index`. Accessed: 2016-02-27.

[40] Per Fredelius. Faceted search with a large amount of properties. Master's thesis, Chalmers University of Technology, 2014.

[41] Xiaoming Gao, Vaibhav Nachankar, and Judy Qiu. Experimenting lucene index on hbase in an hpc environment. In *Proceedings of the first annual workshop on High performance computing meets databases*, pages 25 – 28. ACM, 2011.

[42] Gartner. Search-based data discovery tools. `http://www.gartner.com/it-glossary/search-based-data-discovery-tools`. Accessed: 2016-02-27.

[43] Ali Gencay. A hadoop based architecture for real time and batch workloads in the automotive industry. Master's thesis, Technische Universität München, 2014.

[44] Andrew S. Gordon, David S. Millman, Lisa Steiger, Karen E. Adolph, and Rick O. Gilmore. Researcher-library collaborations: Data repositories as a service for researchers. *Journal of Librarianship and Scholarly Communication*, 3(2), 2015.

[45] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015.

[46] David E. Gray. *Doing research in the real world*. SAGE Publications, London, UK, 2013.

[47] Isabell Grube. Enhancing invenio digital library with an external relevance ranking engine. Bachelor thesis, Hochschule Karlsruhe - Technik und Wirtschaft, 2012.

[48] Akash Gupta, Ankit Rokde, Gulshan Saluja, Rohit Kondekar, and Richa Maru. Content based video retrieval system. Bachelor thesis, Visvesvaraya National Institute of Technology, 2012.

[49] Yuvraj Gupta. *Kibana Essentials*. Packt Publishing, 2015.

[50] Radu Hambasan, Michael Kohlhase, and Corneliu Prodescu. Mathwebsearch at ntcir-11. In *Proceedings of the 11th NTCIR Conference*, Tokyo, Japan, 2014. NTCIR.

[51] Sachin Handiekar and Anshul Johri. *Apache Solr for Indexing Data*. Packt Publishing, 2015.

[52] Zirije Hasani, Boro Jakimovski, Margita Kon-Popovska, and Goran Velinov. Real time analytic of sql queries based on log analytic. In *ICT Innovations 2015 Web Proceedings*, pages 78 – 87, Ohrid, Macedonia, 2015.

[53] Terry Elizabeth Hedrick, Leonard Bickman, and Debra J. Rog. *Applied Research Design: A Practical Guide*. SAGE Publications, 1993.

[54] Kurt Hurtado and Tal Levy. Going beyond the needle in a haystack: Elasticsearch and the elk stack. https://speakerd.s3.amazonaws.com/presentations/70d4390b68504e02b4a40a1aa3754532/strata15-elk-stack-needle-haystack.pdf. Accessed: 2016-02-27.

[55] IHS. Big data in the driver's seat of connected car technological advances. http://press.ihs.com/press-release/country-industry-forecasting/big-data-drivers-seat-connected-car-technological-advance. Accessed: 2016-02-27.

[56] INFORMS. What is analytics? https://www.informs.org/About-INFORMS/What-is-Analytics. Accessed: 2016-02-27.

[57] Morten A. Iversen. When logs become big data. Master's thesis, University of Oslo, 2015.

[58] Antonio J. Jara, Pablo Lopez, David Fernandez, Jose F. Castillo, Miguel A. Zamora, and Antonio F. Skarmeta. Mobile digcovery: Discovering and interacting ith the world through the internet of things. *Personal and Ubiquitous Computing*, 18(2):323 – 338, 2014.

[59] Joohyoung Jeon, Minjeong An, and Hongchul Lee. Nosql database modeling for end-of-life vehicle monitoring system. *Journal of Software*, 10(10):1160 – 1169, 2015.

[60] Zahari Dimitrov Kassabov. Log analysis and classification of cern control systems. http://openlab.web.cern.ch/printpdf/2089, 2014. Accessed: 2016-02-27.

[61] Avita Katal, Mohammad Wazid, and R. H. Goudar. Big data: Issues, challenges, tools and good practices. In *Contemporary Computing (IC3), 2013 Sixth International Conference on*, pages 404 – 409, Noida, India, 2013. IEEE.

[62] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering version 2.3. Technical report, Keele University, Keele University and University of Durham, 2007.

[63] Markus Klose and Daniel Wrigley. *Einführung in Apache Solr*. O'Reilly Verlag, 2014.

[64] Mathias Knudsen, Jostein Løvbråten, and Anders Dalmo. Deploying a virtualised high-interaction honeynet. Bachelor thesis, Gjøvik University College, 2014.

[65] Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 328 – 331. ACM, 2014.

[66] Jan Korenň. Fulltext search in the database and in texts of social networks. Master's thesis, National College of Ireland, 2013.

[67] Rafał Kuć and Marek Rogoziński. *Elasticsearch Server Second Edition*. Packt Publishing, 2014.

[68] Abdelkader Lahmadi and Frédéric Beck. Powering monitoring analytics with elk stack. In *9th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2015)*, Ghent, Belgium, 2015.

[69] Pablo Lopez, David Fernandez, Rafael Marin-Perez, Antonio J. Jara, and Antonio F. Gomez-Skarmeta. Scalable oriented-service architecture for heterogeneous and ubiquitous iot domains. *Pervasive and Mobile Computing*, pages 1 – 23, 2013.

[70] Apache Lucene. Apache lucene - query parser syntax. `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`. Accessed: 2016-02-27.

[71] Lucidworks. Lucidworks silk analytics. `https://lucidworks.com/products/silk/`. Accessed: 2016-02-27.

[72] Lucidworks. Silk. `https://doc.lucidworks.com/assets/attachments/SiLK.pdf`. Accessed: 2016-02-27.

[73] Andre Luckow, Ken Kennedy, Fabian Manhardt, Emil Djerekarov, Bennie Vorster, and Amy Apon. Automotive big data: Applications, workloads and infrastructures. In *Proceedings of IEEE Conference on Big Data*, Santa Clara, CA, USA, 2015. IEEE.

[74] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

[75] Beniamino Di Martino, Rocco Aversa, Giuseppina Cretella, Antonio Esposito, and Joanna Kołodziej. Big data (lost) in the cloud. *International Journal of Big Data Intelligence*, 1(1/2):3 – 17, 2014.

[76] Kerim Meijer, Manos Tsagkias, and Magiel Bruntink. Efficiently storing and searching distributed data. Master's thesis, University of Amsterdam, 2013.

[77] Alexander Miller and Dominik Lekar. Evaluation of analysis and visualization tools for performance data, 2014.

[78] James Miller. *Mastering Splunk*. Packt Publishing, 2014.

[79] Bart De Muynck. How to derive value from big data in transportation. *Gartner*, 2015.

[80] Ravi Narasimhan and T. Bhuvaneshwari. Big data - a brief study. *International Journal of Scientific & Engineering Research*, 5(9):350 – 353, 2014.

[81] U. S. Department of Transportation Federal Highway Administration. Road weather demonstration. `https://www.its-rde.net/data/showds?dataEnvironmentNumber=10016`. Accessed: 2016-02-27.

[82] U. S. Department of Transportation Federal Highway Administration. Welcome to the research data exchange. `https://www.its-rde.net/home`. Accessed: 2016-02-27.

[83] Text Mining Online. Dive into nltk, part iv: Stemming and lemmatization. `http://textminingonline.com/dive-into-nltk-part-iv-stemming-and-lemmatization`. Accessed: 2016-02-27.

[84] Alberto Paro. *ElasticSearch Cookbook - Second Edition*. Packt Publishing, 2015.

[85] Juan Luis Pérez. Desing and evaluation of scalable iot event processing platform. Master's thesis, Universitat Politècnica de Catalunya, 2014.

[86] Bernhard Pflugfelder. Real-time data analytics mit elasticsearch. `https://inovex.de/fileadmin/files/Vortraege/real-time-data-analytics-mit-elasticsearch-bernhard-pflugfelder-jax2014.pdf`. Accessed: 2016-02-27.

[87] Jean François Puget. Proactive analytics. `https://www.ibm.com/developerworks/community/blogs/jfp/entry/proactive_analytics?lang=en`, 2013. Accessed: 2016-02-27.

[88] Marc Reilly. Is virtualisation the most secure way to provide shared resources and applications. Master's thesis, National College of Ireland, 2013.

[89] Timothy Riley. Detecting potentiallz compromised credentials in a large-scale production single-signon system. Master's thesis, Naval Postgraduate School, 2014.

[90] James Robertson and Suzanne Robertson. Volere: Requirements specification template edition 6.0. Technical report, Indiana University and Rutgers University, Bloomington, IN, USA and Piscataway, NJ, USA, 2015.

[91] Marek Rogoziński and Rafał Kuć. *Mastering Elasticsearch - Second Edition*. Packt Publishing, 2015.

[92] RailsBridge Ruby. nil. `http://docs.railsbridge.org/ruby/nil`. Accessed: 2016-02-27.

[93] Philip Russom. Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, pages 1 – 35, 2011.

[94] Chris Sanders and Jason Smith. *Applied Network Security Monitoring*. Syngress, 2013.

[95] Kurt Schlegel. Hype cycle for business intelligence and analytics, 2014. *Gartner*, 2014.

[96] Kurt Schlegel. Hype cycle for business intelligence and analytics, 2015. *Gartner*, 2015.

[97] Marc Seeger. Building blocks of a scalable web crawler. Master's thesis, Stuttgart Media University, 2010.

[98] Alfredo Serafini. *Apache Solr Beginner's Guide*. Packt Publishing, 2013.

[99] Sunaina Sharma and Veenu Mangat. Technology and trends to handle big data: Survey. In *Advanced Computing & Communication Technologies (ACCT), 2015 Fifth International Conference on*, pages 266 – 271, Haryana, India, 2015. IEEE.

[100] Betsy Page Sigman. *Splunk Essentials*. Packt Publishing, 2015.

[101] Ajitpal Singh. Cloudsearch, dell cloud service application. Master's thesis, National College of Ireland, 2013.

[102] Ajitpal Singh and Horacio González-Vélez. Hierarchical multi-log cloud-based search engine. In *Eighth International Conference on Complex, Intelligent and Software Intensive Systems*, pages 211 – 219, Birmingham, UK, 2014. IEEE.

[103] Gary Smith. Log analysis with the elk stack (elasticsearch, logstash and kibana). `https://www.linuxfestnorthwest.org/sites/default/files/slides/Log%20Analysis%20with%20the%20ELK%20Stack.pdf`. Accessed: 2016-02-27.

[104] Splunk. Embedded oem program. `https://www.splunk.com/it_it/view/oem-partner/SP-CAAAH3C`. Accessed: 2016-02-27.

[105] Splunk. Splunk enterprise overview. `http://docs.splunk.com/index.php?title=Documentation:Splunk:Overview:Whatsinthismanual:6.1beta&action=pdfbook`. Accessed: 2016-02-27.

[106] Johannes Stoll. Development of a distributed software architecture for the search and analysis of open data. Master's thesis, Hochschule Offenburg, 2014.

[107] Quiran Storey. The design and implementation of a security and containment platform for peer-to-peer media distribution. Master's thesis, Stellenbosch University, 2013.

[108] François Terrier. On elasticsearch performance. `https://blog.liip.ch/archive/2013/07/19/on-elasticsearch-performance.html`. Accessed: 2016-02-27.

[109] Jessica Thompson, Andrew Hankinson, and Ichiro Fujinaga. Searching the liber usualis: Using couchdb and elasticsearch to query graphical music documents. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, Miami, Florida, USA, 2011.

[110] Jr. Trevor Roberts, Josh Atwell, Egle Sigler, and Yvo van Doorn. *DevOps for VMware Administrators*. VMware Press, 2015.

[111] James Turnbull. *The Logstash Book Version 2.0*. 2015.

[112] Mattijs Ugen. Scalable performance for a forensic database application. Master's thesis, University of Twente, 2013.

[113] Risto Vaarandi and Mauno Pihelgas. Using security logs for collecting and reporting technical security metrics. In *Military Communications Conference (MILCOM), 2014 IEEE*, pages 294 – 299. IEEE, 2014.

[114] Risto Vaarandi Vaarandi and Paweł Niziński. Comparative analysis of open-source log management solutions for security monitoring and network forensics. In *Proceedings of the 2013 European Conference on Information Warfare and Security*, pages 278 – 287, Jyväskylä, Finland, 2013.

[115] Konstantinos Vandikas and Vlasios Tsiatsis. Performance evaluation of an iot platform. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 141 – 146, Oxford, UK, 2014. IEEE.

[116] Jan vom Brocke, Alexander Simons, Bjoern Niehaves, Kai Reimer, Ralf Plattfaut, and Anne Cleven. Reconstructing the giant: On the importance of rigour in documenting the literature search process. In *17th European Conference on Information Systems (ECIS)*, Verona, Italy, 2009. ECIS.

[117] Jakub Ševcech, Róbert Móro, Michal Holub, and Mária Bieliková. User annotations as a context for related document search on the web and digital libraries. *Informatica*, 38(1):21 – 30, 2014.

[118] Samuell Fosso Wamba, Shahriar Akter, Andrew James Edwards, and Denis Gnanzou. How 'big data' can make big impact: Findings from a systematic review and a longitudinal case study. *International Journal of Production Economics*, 165:234 – 246, 2015.

[119] Jonathan Stuart Ward. *Efficient Monitoring of Large Scale Infrastructure as a Service Clouds*. PhD thesis, University of St Andrews, 2015.

[120] Jonathan Stuart Ward and Adam Barker. Observing the clouds: A survey and taxonomy of cloud monitoring. *Journal of Cloud Computing: Advances, Systems and Applications*, 3(1):1 – 30, 2014.

[121] Chunlei Wu, Adam Mark, and Andrew I. Su. Mygene.info: gene annotation query as a service. *bioRxiv*, 2014.

[122] Mikio Yamamoto and Kenneth Church. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27(1):1 – 30, 2001.

[123] David Yates. A system for characterising internet background radiation. Bachelor thesis, Rhodes University, 2014.

[124] Leishi Zhang, Andreas Stoffel, Michael Behrisch, Sebastian Mittelstadt, Tobias Schreck, René Pompl, Simon Weber, Holger Last, and Daniel Keim. Visual analytics for the big data era-a comparative review of state-of-the-art commercial systems. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 173 – 182. IEEE, 2012.

[125] Qing Zou. A novel open source approach to monitor ezproxy users' activities. *Code4Lib Journal*, (29), 2015.