Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Biraj Dhungel

Zentralisiertes Logging und Visualisierung in Echtzeit mit Elastic Stack

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Biraj Dhungel

Zentralisiertes Logging und Visualisierung in Echtzeit mit Elastic Stack

Abschlussarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Stefan Sarstedt
Zweitgutachter : Prof. Dr. Olaf Zukunft

Eingereicht am: 24.05.2018

Biraj Dhungel

Thema der Arbeit
Zentralisiertes Logging und Visualisierung in Echtzeit mit Elastic Stack

Stichworte
Elastic Stack, Zentralisierte Logging, Elasticsearch, Kibana, Dashboard, Logstash, Beats, Echtzeit Datenvisualisierung

Kurzzusammenfassung
In dieser Arbeit wurde unterschiedliche Tools von Elastic Stack näher untersucht, die Anforderungen für ein zentralisiertes Logging-System gesammelt um die Metriken und Lasttestergebnisse aus unterschiedliche Quellen zentral zu sammeln und in nähe Echtzeit visualisieren und analysieren zu können. Anschließend wurde eine Lösung entworfen und implementiert, die möglichst vielen Anforderungen entspricht. Am Schluss dieser Arbeit wurden alle Erkenntnisse bewertet. Das Ergebnis dieser Arbeit zeigt, dass man mit Hilfe von Elastic Stack möglichst schnell und Effizienz ein zentralisiertes Logging- System implementieren kann.

Biraj Dhungel

Title of the paper
Centralised Logging and realtime visualisation with Elastic Stack

Keywords
Elastic Stack, Centralised Logging, Elasticsearch, Kibana, Dashboard, Logstash, Beats, realtime visualisation

Abstract
In this work, Elastic Stack's various tools were examined in detail, gathering the requirements for a centralized logging system to collect the metrics and load test results centrally from different sources and visualize and analyze them in real-time. Then, a solution was designed and implemented that could meet many requirements. At the end of this work, all findings were evaluated. The result of this work shows that,with the help of Elastic Stack a efficient centralized logging system can be implemented.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungverzeichnis

1 Einleitung

The most powerful tool we have as developers is automation." – Scott Hanselman

Durch die zunehmende Beliebtheit und die Fortschritte des Internets steigt auch die Zahl der Webanwendungen. Webanwendungen beinhalten oft sehr kritische Daten, auf die jederzeit zugegriffen werden muss. Ein minimaler Systemausfall kann schwerwiegende wirtschaftliche Folgen nach sich ziehen, sodass diese Systeme eine hohe Verfügbarkeit und eine kurze Ausfallzeit gewährleisten müssen. Die stetig steigenden Anforderungen an die Geschwindigkeit, Flexibilität und Qualität sowie der große Wettbewerb haben die Art der Softwareentwicklung stark verändert. Ein modernes und ausfallsicheres Softwaresystem mit kurzer Release Zeit ist notwendig, um im Wettbewerb mithalten zu können. Traditionelle Entwicklung hat eine längere Release Zeit und die Software wird nur einmal vor dem Ausrollen getestet. Der Bedarf an schnellerer Vermarktung, höherer Anpassungsfähigkeit, geringerem Risiko und höherer Sichtbarkeit hat Agile[1] Softwareentwicklung hervorgebracht (Vgl. Vro14). Agile Softwareentwicklung mit kurzer Release Zeit fördert den Bedarf an kurzer und häufiger Testausführung, die eine hohe Testabdeckung besitzt. Um den aktuellen Softwareanforderungen zu genügen, werden heutzutage Lasttests automatisiert durchgeführt.
Automatisierte Lasttests erzeugen viele Logdateien, die in unterschiedlichsten Formaten und an unterschiedlichsten Ort gespeichert werden. Die erzeugten Testergebnisse müssen letztendlich analysiert werden. Manuelle Tests sind in Continuous Delivery[2] aufwändig und häufig noch fehlerhaft. Lasttests gehören zum wichtigsten Teil

[1] Agile ist eine Art der Softwareentwicklung, die möglichst unbürokratisch, einfach und iterativ abläuft

[2] Continuous Delivery ist eine moderne Methode der Softwareentwicklung zur Verbesserung des Softwareauslieferungprozesses.

der Softwareimplementierung. Mit Hilfe von Lasttests kann man Grenzen und Schwachstellen bei Softwaresystemen feststellen. Diese werden bei System- und Integrationstests nicht gefunden. Weiterhin können Informationen über Stabilität, Performanz und Skalierbarkeit der Anwendung unter Last ermittelt werden (Vgl. Qme). Ein Lasttest simuliert das Systemverhalten möglichst realitäts- und grenznah, um die funktionalen Fehler in einer Software zu finden. Mit Hilfe von Lasttests kann man Änderungen in der Antwortzeit in Abhängigkeit von der Last ermitteln, ferner unter welcher Last das System akzeptabel arbeiten kann, ab wann das System undefiniertes Verhalten zeigt oder sogar komplett abstürzt. Das ständige Wachsen von IT-Infrastruktur und Anwendungen erhöht die Komplexität von IT- Systemen. Die Anwendungen laufen nicht mehr auf realen Servern, sondern vielmehr in virtuellen Maschinen und Containern, die parallel arbeiten und verteilt sind (Vgl. Kla15). Die neuen Ressourcen müssen immer in bestehende Monitoring-Systeme hinzugefügt werden, um die Sicherheit der Softwaresysteme gewährleisten zu können. Es ist sehr zeitaufwändig und nicht skalierbar, sich an jedem Rechner anzumelden und die Log-Meldungen zu durchsuchen. Wie die neunte jährliche SANS[3] Log Management Studie klarstellt, ist die Aufzeichnung allein nicht ausreichend, um Informationen daraus zu gewinnen. 97 % der Unternehmen, die in der Studie befragt worden sind, sammeln regelmäßig Log-Protokolle, um Fehler zu finden. Aber nur 11 % haben ein automatisiertes Log-Management mit Analyseprozess (Vgl. She14). Logs sind früher in relationalen Datenbanken gespeichert worden. Aber heutzutage sind Anwendungen sehr komplex und verteilt. Die Wandlung vom gängigen monolithischen Programmierstil hin zu sog. Microservices[4] erhöht die Komplexität der Sammlung von Log-Meldungen, da es viele Quellen gibt, die Log-Meldungen erzeugen und die dann gesammelt werden müssen. In einem verteilten System ist es schwer, einzelne Logdateien zu durchsuchen, wenn ein Problem auftritt. Log-Meldungen sind meistens unstrukturiert oder semi-strukturiert, daher ist die typische relationale SQL-Datenbank eher ungeeignet für die Speicherung solcher Daten. NoSQL-Datenbanken erfüllen genau diesen Zweck und können mit semi- und unstrukturierten Daten umgehen. Die Log-Sammlung und Speicherung sind wichtige Prozesse in Log-Managementsystem.
Um die Performanz zu verbessern, muss man auch den Ressourcenverbrauch (CPU, Speicher, IO) analysieren. Die Simulation mit paralleler oder intensiver Nutzung des Systems ist dabei wichtig. Eine Ressourcenmetrik definiert, wie viele Ressourcen verbraucht werden, um die gegebene Leistung zu erreichen (Vgl. Lmc). Hierfür sind die Informationen aus Ressourcen wie CPU, Netzwerk, Datenbank, Platten etc. wichtig.

[3] https://www.sans.org/

[4] Microservices ist ein Software-Architekturstil, bei dem komplexe Anwendungen aus unabhängigen Teilprozessen zusammengesetzt sind. Siehe http://microservices.io/

Dabei werden beispielsweise Antwortzeiten sowie das Ressourcenverbrauchsverhalten des getesteten Systems unter einer gegebenen Last ermittelt. In einem normalen Softwareentwicklungsszenario, wo jeden Tag automatisierte Lasttests durchgeführt werden, ist es mit naiver Analyse schwierig, die Testergebnisse zu analysieren und davon Informationen abzuleiten. Es ist dabei wichtig, dass die Daten an einem zentralen Ort gespeichert werden, zeitlich protokolliert sind und ein einheitliches Format einhalten. Um sich einen Überblick über die Testergebnisse zu verschaffen, muss das Prozesslog vom Erzeugen, Speichern bis hin zum Analysieren und Visualisieren automatisiert werden. Log-Managementsysteme müssen zentralisiert werden, um alle Log- Informationen an einer Stelle zu haben. Zentralisierte Logging-Systeme mit Visualisierung und Reporting sind für jedes IT-Unternehmen wünschenswert.

In den letzten Jahren ist eine Reihe von Technologien für zentralisiertes Logging und Analyse entstanden. Elastic Stack ist eine beliebte Technologie dafür. Elastic Stack (früher ELK Stack) ist eine Kombination aus Softwarekomponenten, hauptsächlich aus 4 Komponenten: Elasticsearch[5], Logstash[6], Kibana[7] und Beats[8]. Die Softwarekomponenten zusammen bieten eine einfache Möglichkeit, zentralisiertes Logging und Visualisierung zu ermöglichen.

1.1 Firmenprofil der PPI AG

Die Grundlage für die vorliegende Arbeit wurde im Hause der Firma PPI AG[9] in Hamburg erarbeitet und entstand im Rahmen der Entwicklung eines Softwaresystems, um zentralisiertes Logging zu ermöglichen.

Die PPI AG ist ein Beratungs- und Softwarehaus, das hauptsächlich für Banken, Finanzierung und Finanzdienstleister tätig ist. Sie entstand im Jahre 2000 aus der 1984 gegründeten Pape und Partner Informationssysteme GmbH. Zurzeit beschäftigt PPI über 500 Mitarbeiter und zählt zu den stabilen, mittelständischen Unternehmen. Es wird hauptsächlich in Java entwickelt und Technologien wie Spring MVC und Hibernate kommen zum Einsatz.

---

[5] https://www.elastic.co/products/elasticsearch
[6] https://www.elastic.co/products/logstash
[7] https://www.elastic.co/products/kibana
[8] https://www.elastic.co/products/beats
[9] https://www.ppi.de

PPI ist in den zwei Geschäftsfeldern Consulting und Softwareentwicklung tätig.
• Consulting: PPI unterstützt bei Konzeption und Realisierung komplexer Softwaresysteme und bietet Fachberatung, IT-Konzeption, Integration komplexer IT-Systeme und Testmanagement.

• Softwareentwicklung: PPI entwickelt individuelle Software für Banken, Versicherung und Finanzdienstleister in enger Zusammenarbeit mit dem Kunden und seinen Anforderungen und unterstützt in jeder Phase des Softwarelebenszykluses.

1.2 Zielsetzung

Das Ziel dieser Arbeit besteht darin, herauszufinden, inwieweit Elastic Stack das Problem von zentralisiertem Logging löst. Hierfür ist es notwendig, sich mit den Grundlagen und Tools von Elasticsearch zu beschäftigen. Der Hauptpunkt ist dabei die Sammlung und zentralisierte Speicherung von Log-Meldungen wie Testergebnisse und Metriken aus unterschiedlichen Umgebungen, da die Log-Meldungen aus unterschiedlichen Rechnern stammen, auf denen die Last- und Performanztests laufen.

Das zweite wichtige Ziel ist die Automatisierung des Loggings, der Analyse und der Visualisierung der Log-Meldungen in Echtzeit. Dabei ist es notwendig, die Anforderungen für solche Systeme zu sammeln, passende Technologien und Entwürfe zu designen und so zu implementieren, dass möglichst alle Anforderungen erfüllt werden. Die Analyse und Visualisierung von Antwortverhalten, Performanz und Stabilität der Applikation unter Last sind dabei wichtig.

1.3 Aufbau der Arbeit

Kapitel 1: Im ersten Kapitel findet man die Motivation für diese Arbeit. Außerdem wird die Zielsetzung dieser Arbeit beschrieben und PPI vorgestellt.

Kapitel 2: Im Grundlagenkapitel werden die wichtigsten Grundbegriffe von zentralisiertem Logging und von Technologien, die für die Bachelorarbeit relevant sind, eingeführt.

Kapitel 3: Das Kapitel startet mit einem Interview des Testleiters, um die Anforderungen zu sammeln. Mit Hilfe der Anforderungsanalyse werden dann funktionale und nichtfunktionale Anforderungen an das zu implementierende System festgelegt.

Kapitel 4: Im vierten Kapitel geht es um das Design der Software. Hier wird die Kommunikationsmethode, die Log-Trennung pro System und ähnliches festgelegt.

Kapitel 5: Das darauf folgende Kapitel erläutert die Realisierung der Software: Parsen und Transformieren von Logdateien, Speichern und Visualisieren. Es wird beschrieben, wie die Software mit Hilfe des Elastic Stack implementiert wurde.

Kapitel 6: Anschließend wird im sechsten Kapitel evaluiert. Dafür werden die Ergebnisse der verschiedenen implementierten Systeme ausgewertet und mit den vorab aufgestellten Anforderungen verglichen.

Kapitel 7: Im letzten Kapitel wird die Arbeit zusammengefasst und die Erkenntnisse werden bewertet.

2 Grundlagen

Dieses Kapital enthält die Grundlagen jener Technologien, die im zentralisiertem Logging vorkommen. Zudem soll es dem Leser helfen, den praktischen Teil dieser Arbeit in diesem Anwendungskontext betrachten zu können.

2.1 TRAVIC-Corporate

TRAVIC-Corporate[10] ist ein Inhouse entwickeltes Softwareprodukt der PPI AG. Es handelt sich um einen mandantenfähigeren Bankrechner, der für den Zahlungsverkehr zwischen Kreditinstituten und Banken zuständig ist. Er nimmt Zahlungsverkehrsaufträge über verschiedene Kanäle wie EBICS, PeSIT und FTP an und verarbeitet diese. Die Verarbeitung umfasst Aufgaben wie Limit- und Kontenprüfung, Formatprüfung und -konvertierung.

TRAVIC-Corporate ist der Marktführer mit einem Marktanteil von über 90 %. Daher sind Skalierbarkeit und Lastbarkeit sehr wichtig. Darüber hinaus können über die Kanäle auch Dateien z. B. auch Protokolle oder Kontoauszüge heruntergeladen werden. Die Corporate Engine ist für folgende bankfachliche Aufgaben zuständig (Vlg. Tfs):

- Verifikation von elektronischen Unterschriften
- Formatprüfungen und -konvertierungen (SEPA, SWIFT, DTAZV, ISO20022, CGI etc.)
- Limit- und Kontenprüfungen
- Administration für die verteilte elektronische Unterschrift
- Weiterleitung von Kundenaufträgen an bankfachliche Systeme

Systemvoraussetzungen für die Installation sind:

[10] https://www.ppi.de/de/payments/ebics-zahlungsverkehr-fuer-kreditinstitute/ebics-bankrechner-travic-corporate/

Betriebssysteme

- AIX[11]
- Solaris[12]
- Linux

Datenbankmanagementsysteme

- DB2[13]
- Oracle[14]

Servlet Container

- z.B. Tomcat[15]

2.2 Lasttest in Travic-Corporate

Hohe Performanz und geringer Speicherplatzbedarf sind in Travic-Corporate sehr wichtig. Das Ziel der Lasttests besteht darin, Performanceschwächen möglichst früh herauszufinden, um einen hohen Durchsatz und eine gute Skalierbar gewährleisten zu können. Es wird die Antwortzeit unter gegebener Last getestet. Beim Lasttest durchlauf, wird auch NMON auf den Rechnern ausgeführt, und die Resourcen-Metriken in einer .nmon Datei protokolliert.

Für die Performance-Messung werden während der Durchführung von Lasttests folgende Werte protokolliert und als JSON Datei gespeichert:

11 https://www.ibm.com/power/operating-systems/aix
12 https://www.oracle.com/solaris/solaris11/index.html
13 https://www.ibm.com/analytics/us/en/db2/
14 http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html
15 http://tomcat.apache.org/

Anzahl der Jobs, Anzahl der Threads, die parallel laufen, Job Durchlauf pro Sekunde, Ausführungsdauer eines Jobs, Anzahl Proxy-Verbindungen, Anzahl nicht erfolgreicher Proxy-Verbindungen, fehlgeschlagene Anfragen wegen interner Fehler, Anzahl Anfragen seit letzter Zurücksetzung, Anzahl Anfragen pro Minute, Anzahl Threads pro Anfrage, Anzahl Idle Threads, die auf Anfrage warten etc.

2.3 Loggen

Loggen bedeutet Aufzeichnung der Ereignisse des Softwareprozesses in eine Datei. Diese Datei ist die erste Stelle, die geprüft wird, wenn ein Fehler in einem System eintritt. Meistens werden die Daten während der Laufzeit von einer Anwendung protokolliert, um die Laufzeit eines Systems bzw. einer Anwendung zu analysieren (Vgl. CSP13). Eine Log-Nachricht besteht normalerweise aus drei Teilen: Zeitstempel, wann das Ereignis geschehen ist, welches System hat den Log erzeugt und aktuelle Informationen (Vgl. Ruf). NIST[16] hat die Log Events in drei Typen kategorisiert: Security-Appliance, Betriebssystem und Anwendungsprogramme. Aber es gibt keine Standardisierung für Inhalt, Format und Wichtigkeit, sodass jedes System und jede Anwendung unterschiedliche Logdateien erzeugen. Daher sind die Logdateien schwer bearbeitbar und SQL-Anfragen sind auch nicht gut geeignet, da die Logdateien keine festen Schemata besitzen.

2.4 Zentralisiertes Logging

In einem verteilten System verliert man schnell den Überblick über Logdateien, wenn sie nicht zentralisiert gespeichert werden. Zentralisiertes Logging ist sinnvoll, wenn man über einen zentralen Ort auf alle Log-Dateien zugreifen möchte, um Server- oder Applikationsprobleme zu identifizieren. In größeren Unternehmen oder Entwicklungsprojekten ist das zentralisierte Logging von Serverumgebungen mit mehreren Maschinen bereits Alltag (Vgl. Mei16). Zentralisiertes Logging ist eine Logging-Lösung, bei der Logdateien aus unterschiedlichen Quellen verschiedenster Komponenten wie Netzwerkkomponenten, Betriebssystemen und Applikationen an einem zentralen Ort gespeichert werden und so ein zentralisierter Zugriff auf die Daten ermöglicht wird, anstatt auf jede Quelle individuell zugreifen zu müssen. Der Zugriff auf Daten erfolgt über eine API, womit die Suche, Visualisierung und Analyse einfacher

[16] https://www.nist.gov/

wird. Es kann auch dabei helfen, die IT besser zu steuern, vorausschauend zu kontrollieren und ein kontinuierliches Qualitätsmanagement zu etablieren (Vgl. Mor16). Beim zentralisierten Logging muss sich ein Unternehmen mit unterschiedlichen Geräten auseinandersetzen, da es unterschiedliche Betriebssysteme und Log-Verfahren durchläuft. Die Log-Meldungen besitzen unterschiedliche Zeitstempel und Nachrichtenformate, die einheitlich gehalten werden müssen. (Vgl. Vus10).

Obwohl Sammeln und Speichern von Daten das Wichtigste beim Log-Management ist, liegt die größte Herausforderung in der Normalisierung und Kategorisierung von Informationen. Viele Unternehmen sammeln nur ihre Log-Meldungen. Verarbeiten sie ihre Daten aber weiter, stoßen sie häufig auf eines der folgenden zentralen Probleme des Log-Managements:

- Es gibt viele Systeme, die verteilt sind, deren Log zentral gesammelt werden müsste. Die Formate der Protokolle sind unterschiedlich.
- Die Inhalte der Protokolle sind verschieden.
- Die Zeitpunkte (Zeitstempel) in den Protokollen stimmen nicht überein.
- Das Schreiben von Log ist einfach zu implementieren, aber die Auswertung ist schwierig.

Damit wird es schwierig, die Log-Meldungen in Korrelation zu bringen, zu normalisieren, zu kategorisieren und zu analysieren. Aus diesem Grund ist es eine Herausforderung, die Daten automatisch zu speichern sowie die Analyse schnell und übersichtlich zu ermöglichen.
Das Bundesamt für Sicherheit in der Informationstechnik hat allgemeine Grundlagen für ein zentralisiertes Loggingsystem mit folgenden inhaltlich abzubildenden Mindestanforderungen definiert: Normalisierung, Aggregation, Filterung, Visualisierung und Auswertung sowie Alerten und Archivierung (Vgl. Bsi13). Die besonders wichtigen Themen werden im folgenden Abschnitt diskutiert.

2.5 Logstash

Logstash[17] ist eine Opensource Engine zur Sammlung und Verarbeitung von Daten, die die Datenströme und Ereignisse aus unterschiedlichen Quellen liest, sie transformiert, manipuliert und weiterleitet. Es ist in Ruby[18] geschrieben und kann in einer JVM-Umgebung gestartet werden. Es ist eine Pipeline, die den eingehenden Datenfluss mit unterschiedlichem Filter Plug-Ins weiterbearbeitet und in verschiedene Systeme weitergibt. Logstash lässt sich mit Plug-Ins erweitern, daher unterschützt es viele Quell- und Zielsysteme (Vgl Hop16). Per Default erzeugt Logstash Tag-basierte Indices im Format *logstash-YYYY.MM.dd*. Der Index Pattern lässt sich auch selbst bestimmen (Vgl. ELR62).

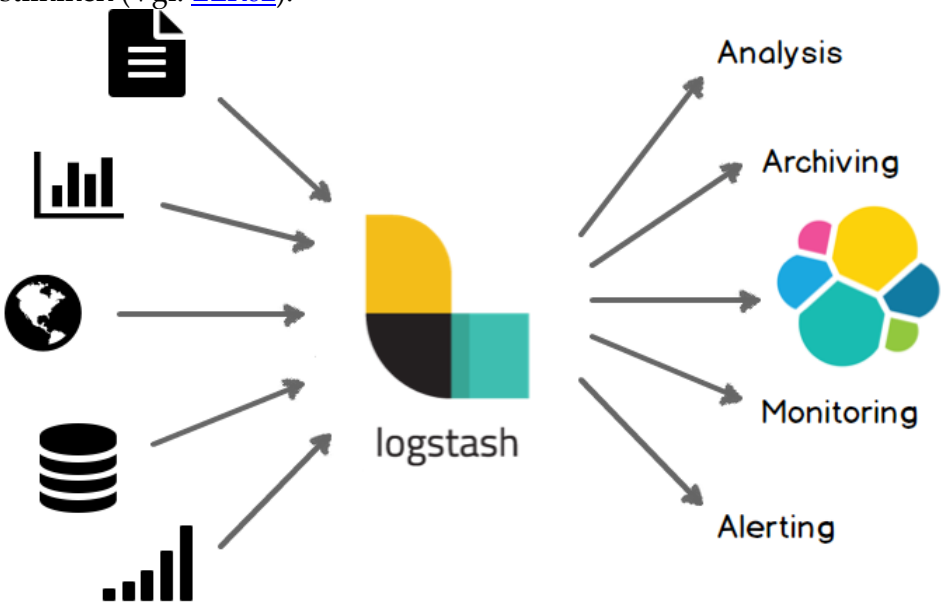

Abbildung 1: Logstash verschiedene Input und Output

Logstash benötigt zwingend eine Konfigurationsdatei, die mindestens aus Input und Output besteht.

2.5.1 Input

In Input werden die Daten aus unterschiedlichen Quellen wie Datenbank, Apache Server, File, Sysin, beats, http, jdbc, syslog gelesen.

[17] https://www.elastic.co/products/logstash
[18] https://www.ruby-lang.org/de/

```
input { stdin { } } output { stdout {} }
```

Listing 2.1 liest den Input von Kommando und gibt ihn wieder aus.

```
1   input { file{
2   path => "/test"} }
3   output { stdout {} }
```

Listing 2.2 liest die Daten von dem Pfad, die in *path* stehen und gibt es in Kommando aus

Beim Input merkt Logstash die letzte Position von der zu lesende Datei.

2.5.2 Filter

Nur mit Input und Output kann man nicht viel gewinnen. Die Macht von Logstash liegt im Filter, der mit vielen Plug-Ins erweiterbar ist. Wenn die Anforderungen nicht durch den bestehenden Plug-Ins erfüllt werden, kann man selbst Plug-Ins für Logstash schreiben. Die Daten und Ereignisse von der Quelle können mit Logstash-Filter parsen, Strukturen in Daten erkennen und umwandeln, womit die Daten schnell analysiert werden können und einheitliche Formate und Zeitstempel besitzen. Die wichtigen Filter für unseren Anwendungsfall sind:

MUTATE: Die bestehenden Felder werden bearbeitet, gelöscht oder auch neue Felder hinzugefügt. Es kann auch den Datentyp von einem Feld ändern.

CSV: Die CSV-Zeilen in Dateien werden in Felder geparst.

Date: Mit diesem Filter wird Datum aus unterschiedlichen Quellen formatiert, Zeitstempel mit Zeit, Datum oder andere Felder werden erzeugt.

GROK: Unstrukturierte Daten und Log-Meldungen werden mit Hilfe eines vordefinierten regulären Ausdrucks bestückt, und erhalten eine einheitliche Struktur (Vgl. ELR62).

2.5.3 Output

Die Daten werden in ein anderes System weitergegeben. Logstash bietet viele Outputs, wo die Daten gespeichert werden können. Die Daten können einfach mit *stdout* ausgegeben werden und auch in eine Datenbank wie Mongodb, Redis, Elasticsearch speichern. In Output kann man einen Index Pattern definieren, der in Elasticsearch erzeugt werden soll, und den Pfad von Template eingeben. Per Default werden die Daten im Tag-basierten Index "*logstash-YYYY-MM.dd*" gespeichert.

2.6 Elasticsearch

2.6.1 Einleitung

Elasticsearch[19] ist eine verteilte Datenbank mit Volltextsuche und Analytik, basierend auf der Java Bibliothek Apache Lucene. Der Kern von Elasticsearch ist Apache Lucene, aber es ist erweitert um Clustering, Monitoring, Aggregation etc. und versteckt die Komplexität, die bei Suche und Analyse kommt, und bietet Rest-Schnittstellen für den Datenaustausch. Neben der Rest API bietet Elasticsearch auch offizielle Klienten für Java, JavaScript, Python, .NET. Das Programm sucht und indexiert Dokumente verschiedener Formate, speichert die Suchergebnisse in einem NoSQL-Format (JSON), und wird mit Aggregation leicht gruppiert und ausgewertet (Vgl. GZ15). Wikipedia, Guardian, Stack Overflow, GitHub etc. benutzen Elasticsearch, um ihre Anforderungen wie hohe Verfügbarkeit, Skalierbarkeit, schnelle Ergebnisslieferung zu erfüllen (Vgl. Sto15).

Die Standardkonfiguration von Elasticsearch läuft in Port 9200, was leicht über die Datei elasticsearch.yml Datei zu ändern ist. Es bietet Java API in Port 9300 und Restful API in Port 9200.
Ein Request in Elasticsearch ist wie HTTP Request

```
1 <VERB> '<PROTOCOL>://<HOST>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

Listing 2.3 Elasticsearch Request Format

Es handelt sich bei Elasticsearch um eine NoSQL Datenbank, die flexibel horizontal erweiterbar ist. Es benutzt auch andere Begriffe, die mit SQL zu vergleichen sind.

Indizes(Index)	Datenbank(Database)

[19] https://www.elastic.co/products/elasticsearch

Typ(Type)	Tabelle(Table)
Feld(Field)	Zeile(Column)

Tabelle 1: Elasticsearch Begriffe in relationalen Datenbanken

Elasticsearch hat mehrere Bedeutungen von Index, was zur Verwirrung führen kann.

- Indizes(Index) als Nomen sind wie Datenbank in traditionellen relationalen Datenbanken.

- Index (Indizieren) als Verb ist Speicherung von Dokumenten in einen Index (Nomen). Es entspricht Insert in SQL.

- Invertierte Indizes (Inverted Index) ist die interne Datenstruktur von Elasticsearch, um die Daten suchbar zu machen. In Elasticsearch sind alle Felder eines Dokuments indiziert (haben einen Inverted Index) und daher suchbar; Felder ohne Index sind nicht suchbar. Es ist das Grundkonzept, das die Suche in Elasticsearch ermöglicht. Wie oben in der Tabelle zu sehen ist, besteht es aus einer eindeutigen Liste von Wörtern, die innerhalb eines Index vorkommt, und einer Referenz zu jedem Dokument, in welchem er vorkommt. (Vgl. Sto15).

Dokument 1: Elasticsearch ist die beste Suchmaschine
Dokument 2: Logstash ist keine Suchmaschine
Dokument 3: ELK ist Logstash, Elassticsearch und Kibana

Term	freq	Dokument
Elasticsearch	2	1,3
ist	3	1,2,3
der	1	1
beste	1	1
Suchmaschine	2	1,2
keine	1	2

ELK	3	1
Logstash	2	2,3
und	1	3
Kibana	1	3

Tabelle 2: Inverted Index in Elasticsearch

2.6.2 Clustering

Ein Cluster ist eine Sammlung von Knoten (engl. Nodes), die gemeinsam die Daten enthalten und eine Anfrage über mehrere Knoten bieten. Cluster können aus mehreren Nodes bestehen aber auch nur einem Knoten (engl. Node).
Ein Knoten ist ein einzelner Server, der Teil des Elasticsearch Clusters ist. Wenn neue Knoten hinzugefügt oder gelöscht werden, werden Cluster reorganisiert und die Daten automatisch verteilt. Ein Masterknoten ist ein Knoten, der dafür zuständig ist, das Hinzufügen und Löschen von Knoten zu managen. Wenn der Knoten gelöscht oder nicht erreichbar ist, wird ein neuer Masterknoten von dem Rest der Knoten gewählt. In Elasticsearch ist jeder Knoten vollständig und weiß, wo die Dokumente liegen, und kann den Request zu dem Knoten weiterleiten, der das Dokument enthält (Vgl. GZ15).

Ein Index ist ein logischer Namespace zur Sammlung von Dokumenten. Physikalisch werden die Daten in Shards gespeichert. Die Daten eines Elasticsearch Index werden in Shards aufgesplittet. Ein Shard kann ein primärer Shard oder Replikat sein. Jedes Dokument gehört zu einem primären Shard und Replikate sind einfach Kopien von primären Shards. Replikate werden verwendet, um Daten redundant zu speichern, damit auch wenn ein Knoten ausfällt, die Daten in einem anderen Knoten verfügbar sind. Wenn man einen Knoten mit den Default-Einstellungen startet und Daten speichert, erstellt er automatisch einen Index mit fünf Shards und verteilt die Daten in die Shards (Vgl. Hop16).

Status von Cluster

Rot: nicht alle primären Shards laufen, womit wir nur bedingt die Ergebnisse bekommen.
Gelb: primäre Shards laufen, aber nicht alle Replikate sind aktiv.
Grün: alle primären Shards und Replikate laufen.

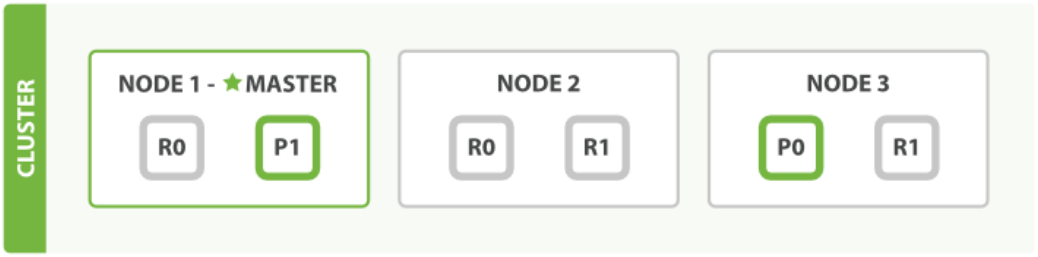

Abbildung 2: Cluster mit 3 Knoten

Es sind 2 Primäre Shards und jeweils 2 Replikate automatisch verteilt über 3 Knoten.

Routing Dokument in Shards

```
1 shards = hash(routing) % Number_of_Primary_shards
```

Listing 2.4: Hashfunktion um Shards zu bestimmen

Elasticsearch benutzt eine einfache Hashfunktion, um das neue Dokument zu speichern. Der Shard, in dem das Dokument gespeichert werden soll, wird mit der Funktion berechnet, wobei Routing die eindeutige ID des Dokuments ist, oder auch ein Custom Wert zugewiesen werden kann. Der Custom Routing Wert ist sinnvoll, wenn man logisch zusammengehörende Daten in einem Cluster speichern möchte, z. B. alle Dokument, die zu einem Benutzer gehören, in einem Shard (Vgl. EDG20).

Dokument in Elasticsearch
Ein Dokument in Elasticsearch ist ein JSON Objekt und hat eine eindeutige ID. Wenn die ID beim Speichern nicht mitgegeben wird, erzeugt Elasticsearch automatisch einen URL-safe, Base64-kodierten String und weist das Dokument zu. Ein Dokument hat Metadaten wie Index, Type und ID, Version, neben seinen eigenen Daten. Index, Type und ID machen das Dokument eindeutig, was bei der Suche mitgegeben werden muss (Vgl. GZ15).

```
1    "_index": "tcu_test",
2    "_type": "testType",
3    "_id": "AV8khgXRdhblsTTZHo8r",
4    "_version": 1,
5    "found": true,
6    "_source": {
7      "firstname": "biraj",
8      "lastname": " dhungel" }}
```

Listing 2. 5 Dokument in Elasticsearch

_index: der **Index**, wo das Dokument gespeichert ist
_type: **Type** von Dokument
_id: automatisch generierte eindeutige ID
_version: jedes Dokument in Elasticsearch ist versioniert. Wenn das Dokument aktualisiert oder gelöscht wird, wird die Versionsnummer erhöht, was für die Verwaltung von Konflikten dient.
_source: enthält das aktuelle Dokument.

2.6.3 CRUD in Elasticsearch

Um Dokumente in Elasticsearch anzulegen, zu ändern und zu löschen, werden HTTP Verben benutzt.

```
1   /[index}/{type}\{id}
2   {“feld” :”wert”,
3   …….
4   }
```

Listing 2.6 Format einer Request mit Daten in Elasticsearch

Ein Post Request mit

```
1  POST /tcu/user{
2  "firstname":"Biraj",
3  "lastname":"Dhungel",
4  "email":"biraj.dhungel@gmail.com",
5  "dateOfBirth":"13.11.1994"
6  }
```

Listing 2.7 Post Request über Rest Api

hat die Response

```
1  {
2    "_index": "tcu_test",
3    "_type": "testType",
4    "_id": "AV8klIDLdhblsTTZHo8s",
5    "_version": 1,
6    "result": "created",
7    "_shards": {
8      "total": 2,
9      "successful": 2,
10     "failed": 0
11   },
12   "created": true}
```

Listing 2.8: Respone von Post-Request

Da die Datei neu angelegt ist, bekommt die Datei die Versions-Nummer 1, und der Wert "*created*" sagt, dass das neue Dokument erfolgreich angelegt ist.

Ein Get Request liefert das gespeicherte Dokument als JSON Objekt mit Metadatei wie Indexname, Typ, Version des Dokuments, Boolean, ob es gefunden wurde etc. GET /TCU/user/123 liefert z.b

```
1
2  {
3    "_index": "tcu_test",
4    "_type": "testType",
5    "_id": "AV8khgXRdhblsTTZHo8r",
6    "_version": 1,
7    "found": true,
8    "_source": {
9      "firstname": "biraj",
10     "lastname":"dhungel"
11 }}
```

Listing 2.9 Response von Get Request in Elasticsearch

Man kann auch nur Teile von Dokumenten liefern mit

```
1 GET /tcu/user/123?_source=firstname, lastname
```

Listing 2.10 Get request mit Teil-Daten

oder nur das Dokument ohne Metadaten mit

```
1  GET /tcu/user/123/_source
```

Listing 2.11 Get Request ohne Metadaten

Ein Put Request ist von der Syntax hier genau wie ein Post Request, aber die ID ist hier zwingend notwendig, damit Elasticsearch weiß, in welchem Dokument es die Änderung vornehmen soll. Wenn das Dokument gefunden wird, aktualisiert es das Dokument, und erhöht die Versionsnummer um eins.

```
1  PUT /tcu/user/AV8khgXRdhblsTTZHo8r {
2  "firstname“:"biraj"
3  "lastname“:"dhungel"
4  "date":"13.12.1994"
5  }
```

Listing 2.12 Put Request in Elasticsearch

Mögliche Response für die Put Request.

```
 1  {
 2    "_index": "tcu_test",
 3    "_type": "testType",
 4    "_id": "AV8khgXRdhblsTTZHo8r",
 5    "_version": 2,
 6    "result": "updated",
 7    "_shards": {
 8      "total": 2,
 9      "successful": 2,
10      "failed": 0
11    },
12    "created": false
13  }
```

Listing 2.13 Response von einem Put Request

Ein Delete Request löscht das Dokument mit gegebener ID, und erhöht die Version wieder um 1. Eine Response zu der Anfrage

```
1  DELETE /tcu/user/AV8khgXRdhblsTTZHo8r
```

Listing 2.14

sieht folgendermaßen aus.

```
1    { "found": true,
2     "_index": "tcu_test",
3     "_type": "testType",
4     "_id": "AV8khgXRdhblsTTZHo8r",
5     "_version": 3,
6     "result": "deleted",
7     "_shards": {
8       "total": 2,
9       "successful": 2,
10      "failed": 0
11    }}
```

Listing 2.15 Response von einer Delete Request

Bei der Delete Operation wird auch die Versionsnummer um eins erhöht. Auch wenn das zu löschende Dokument nicht gefunden wird, wird eine Response mit erhöhter Versionsnummer zurückgegeben, womit Elasticsearch intern die Änderungen in die richtige Reihenfolge über mehrere Knoten durchführen kann.

2.6.4 Suche in Elasticsearch

Elasticsearch bietet für einfache Suchanfragen eine „Lite Suche“, wo man die Suchparameter direkt in der URL anhängt, für komplexe Anfragen bietet Elasticsearch eine „Full request body“ Version, wo JSON als Request Body mitgegeben wird. Für die komplexe Suche wird DSL benutzt.

Beispiel Query Litesuche

```
1 GET tcu_test/testType/_search?q=firstname:biraj
```

Listing 2.16 Query Lite Suche in Elasticsearch

Liefert alle Dokumente, die zur Anfrage passen und die Score zu jedem Dokument.

Query DSL
Da jeder Term in Elasticsearch indexiert wird, bietet Elasticsearch viele Möglichkeiten zur Anfrage und zum Filtern von Suchergebnissen. Query DSL ist sehr flexibel, lesbar und geeignet, mächtige Queries zu erstellen.

```
GET tcu_test/testType/_search{"query": {"term" : {"firstname": "biraj"}}
}
```

Listing 2.17 Query DSL Suche in Elasticsearch

2.6.5 Mapping

Jedes Datenfeld hat ein Mapping, das den einzelnen Typen vom Feld definiert, damit Elasticsearch weiß, wie es mit den Daten umgehen soll. Jedes Dokument in Elasticsearch ist ein JSON Objekt und einzelne Datenfelder haben Typen, auch wenn er nicht explizit mitgegeben wird. Die Daten können in Elasticsearch auch schemafrei gespeichert werden. Wenn Dokumente unbekannten Typen indexiert werden, wird basierend auf den Inhalten ein neuer Dokumenttyp angelegt. Bei bestehenden Typen können auch dynamisch neue Attribute hinzugefügt werden. Elasticsearch verwendet per Default dynamisches Mapping. Um die Suche zu verbessern wird ein Mapping zu einem Typ definiert. Die einzelnen Felder können als Datum, Nummer, String als Volltext oder exact-value definiert werden. Elasticsearch definiert viele Datentypen wie String, Byte, Short, Integer, Long, Boolean, Date, Objekt, Array etc.
Wenn kein explizites Mapping für ein Dokument eingegeben ist, erzeugt Elasticsearch automatisch je nach Wert ein Mapping als boolean, Long, double, Date oder String (Vgl. ERM62).

```
1 POST tcu_test/user{
2   "mappings": {
3     "testType": {
4       "properties": {
5         "firstname" : {"type" :"text"},
6         "lastname" :{"type": "text"},
7         "birthdate":{"type": "date"},
8         "age": { "type": "integer"}}}
9 }
```

Listing 2.18: Beispiel Mapping für einen Index definieren

Das Mapping in Listing 2.18 definiert, dass sich der String als „exact-value“ verhalten soll. Per Default ist String als Volltext evaluiert. Bei Volltext Evaluation kann man Analyzer definieren. Standard Analyzer ist per Default eingestellt, aber man kann whitespace, simple, English oder selbst definierte Analyzer wählen.

2.6.6 Analyzer

Die Analyse macht die Volltextsuche möglich und gibt die relevanten Daten aus, auch wenn die Anfrage nicht exakt mit den vorhandenen Daten übereinstimmt. Ein Analyzer kombiniert drei Funktionalitäten: Character-Filter, Tokenizer und Token-Filter.

Die Zeichenkette geht zunächst durch den Character-Filter, der die Aufgabe übernimmt, die gegebene Zeichenkette zu filtern, wie z.B. HTML-Tags entfernen, Leerzeichen entfernen, Sonderzeichen zu Text umwandeln etc. Es gibt unterschiedliche Analyzer wie Whitespace Analyzer, Language Analyzer, Pattern Analyzer etc. Man kann auch eigendefinierte Analyzer benutzen.

Tokenizer: der Tokenizer nimmt die Zeichenkette entgegen und zerlegt sie in kleine Teile. Es gibt viele Tokenizer wie Standard, Letter, Whitespace, Classic Tokenizer. Der Standard-Tokenizer teilt den Text, wenn ein Leerzeichen oder Zeichensetzung kommt.

Token-Filter: der Token-Filter nimmt die Folge der Terme vom Tokenizer entgegen. Die Terme können gefiltert und geändert werden wie Umwandlung von Groß- und Kleinbuchstaben, Synonyme hinzufügen oder nicht relevante Terme löschen wie eine, keine usw.

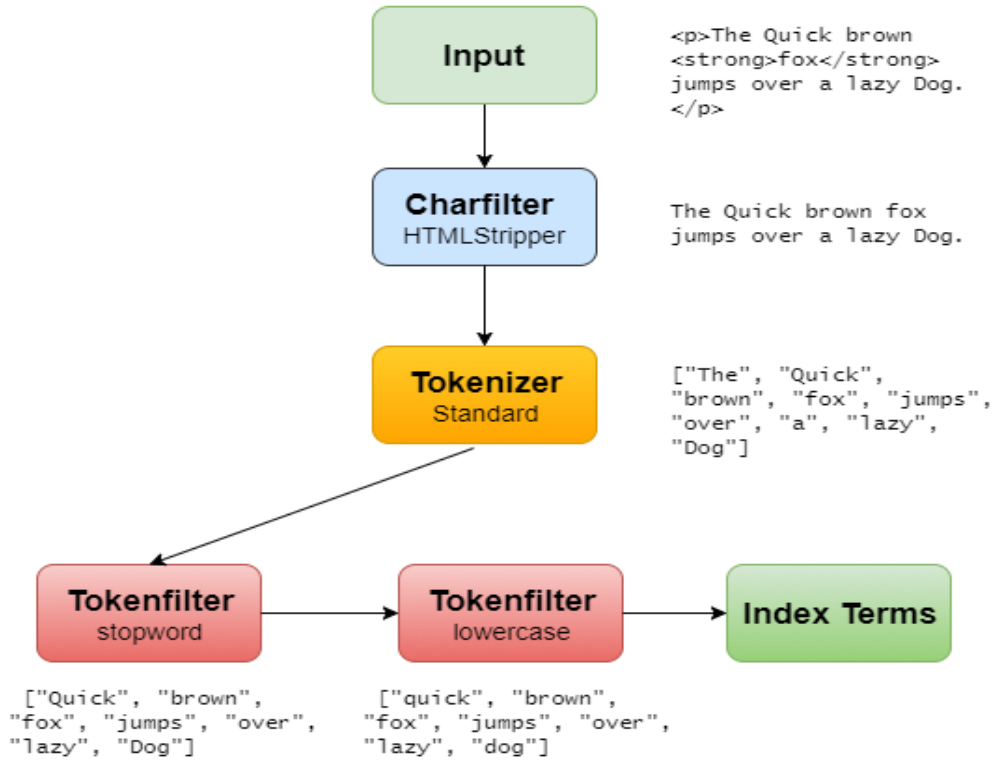

Abbildung 3: Standard Analyzer in Elasticsearch

2.6.7 Aggregation

Eines der wichtigsten Module von Elasticsearch ist die Aggregation, die es ermöglicht, die Datensätze zu berechnen, Anfragen zu stellen, zu sortieren, zu kombinieren. Aggregation als Verb (Zusammenschluss von Teilen um ein Ganzes zu formulieren). Mit Aggegration in Elasticsearch kann man die Datensätze analysieren und auswerten. Die Aggregation ist sehr mächtig für Reporting und Dashboards. In Elasticsearch kann man Aggregation in Bucket-Aggregation und Metrik-Aggregation unterscheiden (Vgl. GZ15)

Bucket-Aggregation ist ein Mechanismus, um die Daten zu gruppieren. Ein Bucket enthält eine Sammlung von Dokumenten, die ein bestimmtes Kriterium erfüllen. In

der Aggregation wird überprüft, ob bestimmte Werte von Dokumenten Bucket-Kriterien erfüllen. Konzeptuell ist das ähnlich wie *„Group by“* in SQL. Range Aggregation, Date Aggregation, Histogramm-Bucket liegen in dieser Kategorie.

Range Aggregation

```
POST tcu_test/user/_search
{
 "aggs" : {
  "age_ranges" : {
    "range" : {
      "field" : "age",
       "ranges" : [
          { "from" : 18, "to" : 20 },
          { "from" : 20, "to" : 25 },
          { "from" : 25, "to" : 30 }
       ]
    }
  }
}
}
```

Listing 2.19 Post Request mit Range Aggregation

```
 "aggregations": {
    "age_ranges": {
      "buckets": [
        {
          "key": "18.0-20.0",
          "from": 18,
          "to": 20,
          "doc_count": 3
        },
        {
          "key": "20.0-25.0",
          "from": 20,
          "to": 25,
          "doc_count": 2
        },
        {
          "key": "25.0-30.0",
          "from": 25,
          "to": 30,
          "doc_count": 5
```

```
21           }
22         ]
23       }
24
```

Listing 2.20 Range Aggregation Response in Elasticsearch

Hier sind die Dokumente über Age-Group eingeteilt. Genauso kann man mit Datenaggregation Dokumente nach Datum in Buckets sortieren.

Histogramm Bucket: Histogramm Bucket braucht zwei Parameter: Numerisches Feld und Intervall, die die Größe des Bucket definieren.

Metrik-Aggregation
Hier werden statistische Informationen über Dokumente in einem Bucket berechnet. Metriken sind einfache mathematische Operationen wie min, median, max, min, value count, Perzentile usw. über bestimmte Datensätze. Es gibt single-value Metriken und multi-value Metriken. Die Max-Metrik liefert beispielsweise die maximalen Werte von einem Feld in bestimmten Datensätzen.

```
1  POST tcu_test/user/_search
2  {
3    "aggs" : {
4       "avg_age" :{
5          "max" : { "field" : "age"}
6          }
7       }
8  }
```

Listing 2.21: Post Request mit Metrik Aggregation

```
 1    {
 2      "took": 3,
 3      "timed_out": false,
 4      "_shards": {
 5        "total": 5,
 6        "successful": 5,
 7        "skipped": 0,
 8        "failed": 0
 9      },
10      "aggregations": {
11        "avg_age": {
12          "value": 30 } } }
```

Listing 2.22: Codebeispiel für Metrik Aggregation Query und Response um durchschnittliches Alter zu finden

Perzentile Metrik

Perzentile sind sehr wichtig, wenn man etwas im Grenzwert analysieren muss, z.B. Antwortzeiten, die über 90% liegen. Durchschnitt und Median werden öfters benutzt, aber Perzentil sind aussagekräftig, weis sie alle Messergebnisse (außer den größten Ausreißern) mit einbeziehen. Wenn die 90% Perzentil bei 60ms liegt, bedeutet das, dass 90% der Anfragen Antwortzeiten von maximal 60ms haben.

```
1 { "latenz" : 60,  "zeitstempel" : "03.10.2017:14:00" }
2 { "latenz" : 100, "zeitstempel" : "03.10.2017:14:30" }
3 { "latenz" : 80,  "zeitstempel" : "03.10.2017:15:00" }
4 { "latenz" : 99,  "zeitstempel" : "03.10.2017:16:20" }
5 { "latenz" : 102, "zeitstempel" : "03.10.2017:16:45" }
6 { "latenz" : 75,  "zeitstempel" : "03.10.2017:17:00" }
7 { "latenz" : 82 , "zeitstempel" : "03.10.2017:17:30" }
8 { "latenz" : 140, "zeitstempel" : "03.10.2017:18:00" }
```

Listing 2.23: Beispieldatensatz um Perzentile zu finden

```
GET  tcu_test/latenz/_search
{
 "aggs" : {
   "load_times" :{
      "percentiles" : {
      "field" : "latenz",
      "percents": [
        80,
        90,
        99
    ]}}}}
 "aggregations": {
    "antwortzeit": {
    "values": {
    "50.0": 99.5,
    "80.0": 124.80000000000001,
    "90.0": 140,
    "99.0": 140
    }
    }
 }
```

Listing 2.24: Beispiel Query und Response um Perzentile zu finden

Der Mittelwert der Datensätze liefert den Wert 92,25. Der Median der Datensätze liefert den Wert 90,5. Das 90% Perzentil der Datensätze liefert den Wert 113,4. Die Ergebnisse sehen ganz normal aus. Aber was ist mit der Antwortzeit um 18:00. Das bedeutet die obersten 10 % haben eine Antwortzeit mehr als 113ms.
Aggregation in Elasticsearch kann man mit SQL Query mit Count und Groupby vergleichen. SELECT COUNT (Color) FROM table GROUP BY Color, wobei COUNT (Color) Metrik ist und Group By (Color) Bucket.

2.7 Kibana

Kibana[20] ist ein Open-Source Analysetool, das zur Visualisierung von Daten aus Elasticsearch entwickelt wurde. Es bietet beinahe eine Echtzeitanalyse und flexible Darstellungsmöglichkeiten für die gespeicherten Informationen in Elasticsearch. Kibana greift über die REST-API auf die Daten von Elasticsearch zu und bietet dem Nutzer

[20] https://www.elastic.co/products/kibana

die Möglichkeit, die Ergebnisse beliebig zu filtern (Vgl. BL15). Kibana ist so implementiert, dass es optimal mit Elasticsearch arbeitet. Durch verschiedene Darstellungs- sowie Aggregationsmöglichkeiten können viele verschiedene Sichten auf die Informationen erzeugt werden (Vgl. EKR62). Auf sogenannten Dashboards werden Darstellungen angeordnet und organisiert. Anhand der verschiedenen Konfigurationsmöglichkeiten kann man sich die Darstellung und Views so wie benötigt selbst definieren. Kibana ist sehr mächtig und leicht zu benutzen. Kibana bietet viele Visualisierungstools um Überblick über Rechner, Applikation und Benutzer zu gewinnen. Es bietet viele Visualisierungen wie Histogramm, Liniendiagramm, Kreisdiagramm, Metrikdiagramm etc. Das Frontend nimmt die Daten von Elasticsearch entgegen und bietet dem Nutzer die Möglichkeit, die Ergebnisse beliebig zu filtern. Das Ergebnis sind dynamische, interaktive und ansprechende Darstellungen der Daten in realtime. Die Daten, die in der dokument-basierten Struktur von Elasticsearch gespeichert sind, können sowohl explorativ untersucht werden, als auch in Visualisierungen wie Torten- und Balkendiagrammen zu Dashboards zusammengestellt werden.

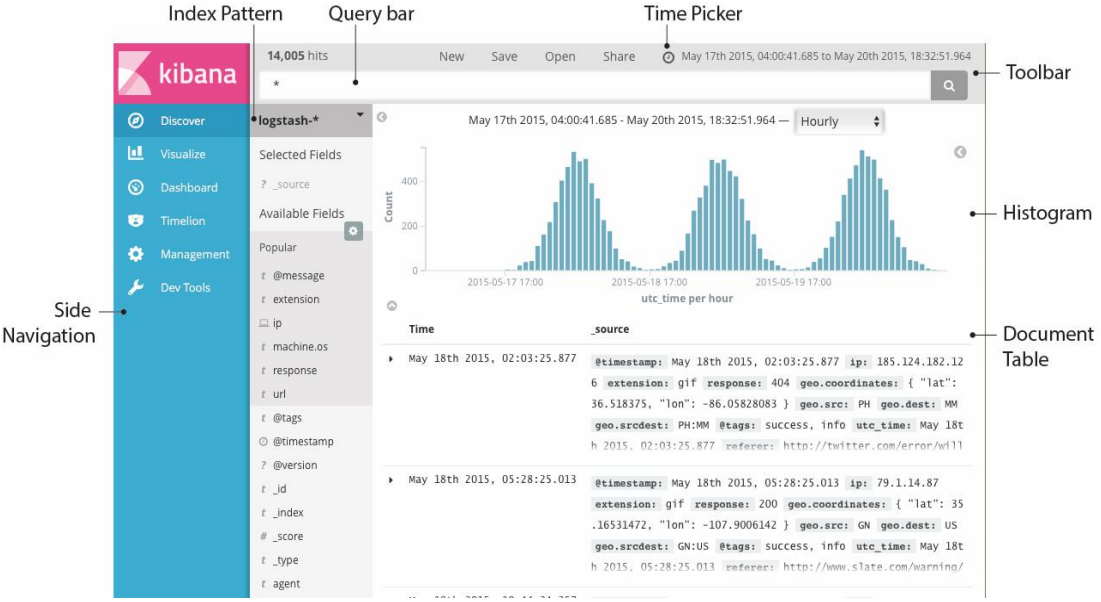

Abbildung 4: Aufbau von Kibana

2.7.1 Suche

Ein vordefinierten Index Pattern über den Management-Reiter, mit Übereinstimmung einer oder mehreren Indizes von Elasticsearch, ist für den Aufruf der Daten von Elasticsearch notwendig. Man kann den vollständigen Namen des Index eingeben oder auch Wildcards benutzen, wenn man über mehrere Indizes die Daten abrufen möchte. Z. B. metricbeat-.* stimmt mit allen Indizes von Elasticsearch überein, die Metricbeat als Präfix haben (Vgl. EKR62).

Über den Reiter Discover kann man die Query erstellen, um die Daten in den jeweiligen Indices zu suchen. Kibana bietet zwei Sucharten. Für die einfache Suche kann man Lucene Query Syntax verwenden. Für die komplexe Suche bietet Kibana DSL-Anfragen. Neben der einfachen Eingabe eines Suchstrings können auch Teile einer Abfrage in die Elasticsearch Query- DSL eingegeben werden, die dann automatisch als Anfrage in einer konstruierten Filtered-Query verwendet wird. Auch für die komplexe Suche kann man voll JSON basierte DSL Anfragen benutzen. Die Suche kann man auch zeitlich begrenzen. Filter mit Feld ist auch möglich. Die Suche kann auch gespeichert werden, um sie später beim Visualisieren zu benutzen.

2.7.2 Visualize

Über den Reiter Visualize kann man die Daten von Elasticsearch Indizes visualisieren und später im Dashboard anzeigen. Die Visualisierung basiert auf Elasticsearch Anfragen und Aggregationen. Besonders wird Elasticsearch Aggregation benutzt, um die relevanten Daten zu extrahieren und sinnvoll darzustellen. Für das Visualisieren kann man die gespeicherten Anfragen verwenden oder auch neue Anfragen schreiben.

Kibana bietet viele verschiedene Arten von Visualisierung. Hier wird nur die für diese Arbeit relevante Art von Visualisierung erläutert. Grob kann man 4 Arten unterscheiden: Basic Charts, Data, Maps und Time Series
Basic Chart enthält Linien-, Flächen- und Balkendiagramm und Daten können auf der X- und Y-Achse visualisiert werden. Die Y-Achse enthält die Metrik-Aggregation und die Y-Achse die Bucket-Aggregation. Data enthält Datatabellen, Metrik, Goal und Gauge. Die Metrik-Visualisierung visualisiert eine einzige Zahl für die gewählte Aggregation. Count Average, Sum, Min, Max, Perzentiles sind z.B. einzelne Metriken, die sich darstellen lassen. Sowie Goal visualisiert Metrik den Abstand zu einem definierten Ziel. Maps stellt Koordinaten und Region Map dar, um die geographischen

Daten zu visualisieren. Time Series kombiniert unterschiedliche Daten aus dem multiple time Series Datensatz.
Kreisdiagramm: Das einzelne Stück eines Kreisdiagramms ist von der Metrik- Aggregation festgelegt. Buckets-Aggregation legt fest, welche Information aus dem Dataset entnommen werden soll.

2.7.3 Dashboard

Das Dashboard zeigt die gespeicherte Visualisierung an. Man kann die Visualisierung ordnen, deren Größe ändern, speichern und mit anderen teilen. Über Management/Kibana/Saved Objects/Dashboards kann man auch das gespeicherte Dashboard exportieren, importieren.

2.7.4 Dev Tools

Ist eine Konsole als Plugin in Kibana UI vorhanden, kann man damit Anfragen an Elasticsearch über Rest API machen. Dies ist nützlich, da es API Suggestion vorschlägt und die Elasticsearch Anfrage leichter macht. Die Anfragen können auch als cURL kopiert werden und über normale Konsolen durchgeführt werden.

Abbildung 5: Dev Console in Kibana

2.7.5 Monitoring und Alerten

Diese werden von kostenpflichtigem Plug-Ins angeboten. Sie helfen, einen Überblick über die Komponente Elasticsearch Cluster zu schaffen. Man kann auch das Dashboard und einzelne Visualisierungen in unterschiedlichem Format exportieren. Alarme stellen automatische Push-Nachrichten über unterschiedliche Kanäle bereit, wenn eine vorkonfigurierte Regel zutrifft.

2.7.6 Management

Hier kann man den Index Pattern definieren, wo Kibana im Reiter Discover suchen soll. Er enthält auch alle gespeicherten Suchvorgänge, Visualisierungen und Dashboards. Diese können editiert, gelöscht, exportiert, und neu importiert werden.

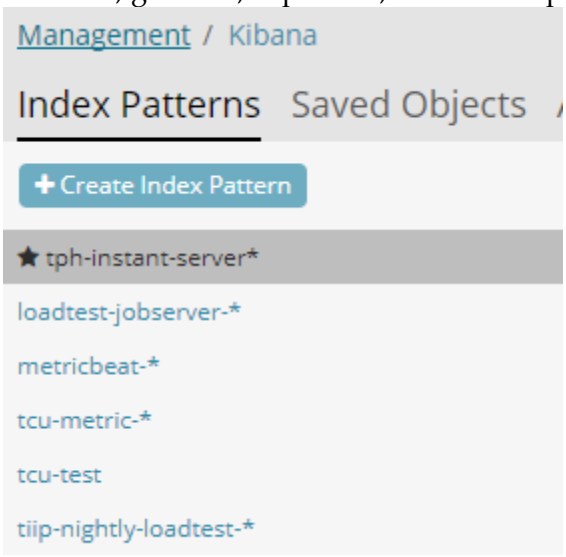

Abbildung 6: Index Pattern in Kibana

2.8 Beats

Daten wie CPU-Auslastung oder, Netzwerkstatus des Servers sind auch interessant für die Analyse. Dafür bietet Elastic Stack sogenannten Beats[21]. Beats sind leichtgewichtige Plattformen, die als Daten-Shipper die Logdateien aus unterschiedlichen Quellen an Logstash oder Elasticsearch übermitteln. Es gibt viele Beats, die von Elastic stack unterschützt werden wie Metricbeat, Filebeat, Packetbeat, Heartbeat, Winlogbeat, Libbeat etc(Vgl. EBR62).
Metricbeat sammelt die Metriken über das System und Dienste. Mit Hilfe von Metricbeat erhält man leicht den Überblick über CPU-Auslastung auf Systemebene, Speicher und Dateisysteme, den IO von Festplatten, dem Netzwerk etc. Metrik Beat ist ein Bestandteil von Elastic Stack und ist leicht mit Logstash, Elasticsearch und Kibana

[21] https://www.elastic.co/products/beats

integrierbar. Mit der Default Einstellung schickt Metricbeat die Daten zu Elasticsearch in *localhost:9200,* aber es lässt sich leicht über die metricbeat.yml Datei konfigurieren, welche Metriken wir brauchen und wohin die Metriken geschickt werden sollen.
Packetbeat sammelt die Netzwerkdaten und Datenbankanalyse. Mit Filebeat kann man Logdateien an einen zentralen Server verschicken. Libbeat ist ein Opensource Framework, das die Erstellung eines eigenen Beats erleichtert.

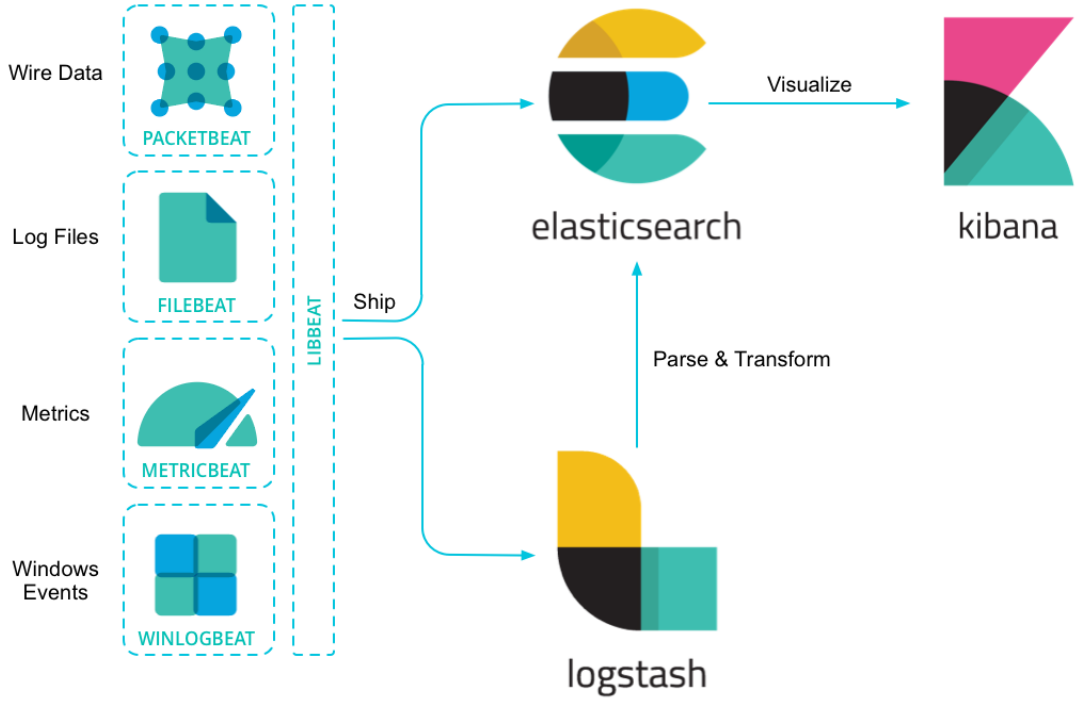

Abbildung 7: Verschiedene Beats

3 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen beschrieben. Anhand dieser Anforderungen wird in Kapitel 4 eine Lösung entworfen.

3.1 Interview

Zur Sammlung der Information wird die Interview-Technik verwendet, da der betroffene Testleiter aktiv beteiligt ist. Das Ziel der Interviews ist es, genau herauszufinden, was beim zentralisierten Logging und der Datenanalyse beachtet werden soll und wie es möglichst effizient implementiert werden kann.

1. Welchen Umfang haben die Log-Meldungen, die während des Lasttests erzeugt werden?
   Die einzelne Logdatei pro Lastest kann ca. 1 GB sein. Die. nmon Datei ist dagegen nur in KB Größe.

2. Welche Quellen sind vorhanden, um Logs zu erzeugen. Erzeugen sie semistrukturierte Daten, unstrukturierte Daten oder streng strukturierte Daten?
   Die Testsysteme laufen auf den Betriebssystemen Linux, Solaris und AIX. TRAVIC-Corporate und die Datenbank laufen auf jeweils einer virtuellen Maschine. Die Messergebnisse werden vom Lasttreiber, JMX, NMON, Trace-Dateien und über Datenabfragen bezogen. Lastestergebnisse sind strukturiert und haben ein JSON Format, das direkt in eine NoSQL Datenbank gespeichert werden können. Die Metrikdaten von NMON sind eher unstrukturiert.

3. Visualisierung und Auswertung sind sehr wichtig bei der Analyse der Logdatei. Was soll dabei besonders beachtet werden?
   Beim Lasttest sind die Perzentile sehr wichtig. 90 % Perzentile sollen gemessen werden. Die Testergebnisse sollten mit vorherigen Tests verglichen werden können. Die Metriken wie CPU-Auslastung, Speicherplatzbedarf, Netzwerkzugriff bezogen auf Zeitstempel, wann der Lasttest durchgeführt wird, sollen visualisiert werden.

4. Welche Dateien sollen gesichert werden?
   Die Lasttestergebnisse vom Job-Server, EBICS-Server, Metriken wie CPU-Auslastung, Netzwerk IO, File Zugriff, Memory etc. sollen gespeichert werden. Die Dokumente sollen im JSON Format gespeichert werden.

5. Es gibt viele Technologien für die Stack Log-Daten Analyse. Soll die Software Opensource sein oder Lizenz-abhängig?
   Die Technologie sollte möglichst kein Geld kosten und die kommerzielle Nutzung muss erlaubt sein.

6. Die Logdateien werden letztendlich von unterschiedlichen Quellen gesammelt. Wie wichtig ist die Trennung von Daten von einem System zu anderen Systemen?
   Es sollen die Testergebnisse von unterschiedlichen Systemen unterschieden, getrennt gespeichert und gelöscht werden können. Auch die Metriken der Server sollen pro Testumgebung unterschieden werden können. Die Daten sollen auch später pro Test-Prozess aufrufbar sein.

7. Die naive Analyse pro Testdurchführung ist sehr zeitaufwendig. Welche Prozesse sollten dabei automatisiert werden müssen?
   Das Programm soll in der Lage sein, nach der Durchführung von Tests die Daten automatisch zu speichern und die Visualisierung zu aktualisieren.

8. Ausfallsicherheit, Skalierbarkeit?
   Die Daten sollten bis 2 Jahre rückwärts auffindbar sein. Die Testergebnisse erzeugen mehrere JSON-Dateien, aber die Parameter können dynamisch sein. Das Programm soll auch die Daten speichern können, wenn Parameter fehlen oder zu viel sind.

Die folgenden Anforderungen werden durch das Interview mit dem Testexperten und durch eigene Recherche abgeleitet.

3.2 Funktionale Anforderungen

FA 1 Normalisierung und Indexen der Testergebnisse

Viele Systeme schreiben ihre Log-Daten in unterschiedlichem Format. Es muss in einheitlichem Format gespeichert werden und mit einheitlichem Zeitstempel protokolliert werden, wobei sichergestellt werden muss, dass die wichtigste Log-Information nicht verloren geht.

FA 2 Zentrales Logging und Datenspeicherung

Es gibt viele Testserver, die Testergebnisse in Logdateien schreiben. Die Ergebnisse sollen in eine zentrale Stelle gespeichert werden. Metriken aus Hostrechner wie CPU, Netzwerk Daten, File Zugriff sollen auch gesammelt werden und an einem zentralen Ort gespeichert werden.

FA 3 Logs-Trennung

Logs von einem System sollen unterschieden werden von solchen, die zu einem anderen System gehören. Logs, die zusammengehören, sollen zusammen gespeichert werden.

FA 4 Automatisierung von Prozessen

Die Tests laufen reglmäßig und erzeugen Messergebnisse in unterschiedlichen Logdateien. Die Logdateien sollen in regelmäßigen Zeitintervallen möglichst automatisiert in einer Datenbank gespeichert und visualisiert werden.

FA 5 Suchen und Korrelieren der Logdatei über ein Benutzerinterface

Es soll die Möglichkeit geben, einfach nach den gespeicherten Daten suchen, um die Daten zu korrelieren und visualisieren. Eine Analyse über alle Logdateien aller Systeme mit Hilfe einer Query und Filter-basierte Visualisierung im Webbrowser soll möglich sein.

FA 6 Visualisierung und Auswertung von Logdatei

Die Daten sollen einfach ausgewertet werden, ein Diagramm soll erstellt werden. Die Visualisierung soll gespeichert und geteilt werden können, damit sie zu einem späteren Zeitpunkt mit anderen Visualisierungen verglichen werden kann.

FA 7 Monitoring und Reporting

Automatisierte Push-Nachrichten, wenn die Logdateien ein bestimmtes Pattern haben. Die erstellten Diagramme und Dashboards sollen exportiert werden können.

FA 8 Open Source, möglichst keine weiteren Lizenzkosten

3.3 Nicht-funktionale Anforderungen

NFA 1 Fehlertoleranz

Log-Meldungen haben kein festes Schema und die Parameter können sich dynamisch ändern. Die Datenbank soll keinen Fehler ausgeben, wenn es mehr oder weniger Parameter sind.

NFA 2 Ausfallsicherheit

Die Daten sollen verfügbar sein, auch wenn manche Server ausfallen.

NFA 3 Leicht erweiterbar

Neue Datenquelle sollen in das bestehende Monitoring-System hinzugefügt werden können.

4 Entwurf

Im oberen Kapitel wurden Anforderungen analysiert und in funktional und nicht funktional aufgeteilt. Im nächsten Schritt wird eine Lösung entworfen, die möglichst viele Anforderungen abdeckt.

4.1 Zentrale Sammlung

Die Daten sollen von verschiedenen Testservern zu einem zentralen Server geschickt werden, wo die Daten geparst und gefiltert werden und in Elasticsearch weitergeleitet werden.
Um Metriken aus Linux zu sammeln wird Metricbeat benutzt, da es in regelmäßigen Zeitintervallen die Log-Datei direkt in Elasticsearch schickt, was einen wichtigen Schritt für die automatisierte Log-Analyse ist. Da Solaris und AIX Beats nicht unterschützen, werden die Metriken von diesem Rechner mit Hilfe von NMON gesammelt. Die. nmon Dateien sollen aus unterschiedlichen Rechnern in einen zentralen Server gesendet werden. Die Ergebnisse, die in der. nmon Datei gespeichert sind, sind unstrukturiert. Daher sollen sie geparst werden, bevor man sie speichern und die Information visualisieren kann. Das ist die Stelle, wo Elastic Stack nicht reicht. Daher soll ein Parser für NMON geschrieben werden, der die unstrukturierte. nmon Datei in CSV/JSON umwandelt.

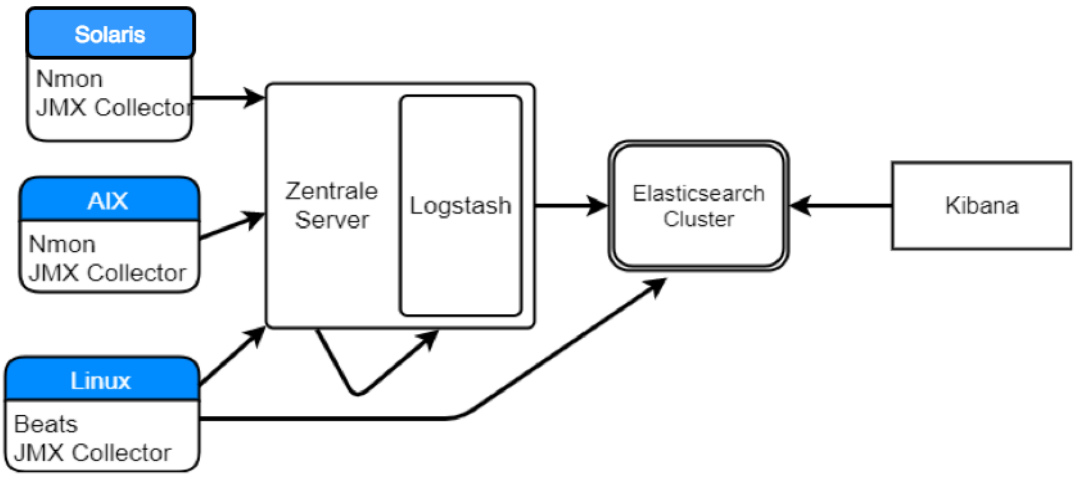

Abbildung 4.1 Ablauf Zentralisiertes Logging und Visualisierung

4.2 Schnittstelle zu andere System

Die Kommunikation zwischen den einzelnen Komponenten erfolgt über HTTP und die Daten und Anfrage werden als JSON geschickt.

4.3 Trennung von Daten

Wie in FA3 beschrieben ist die Trennung des Logs sehr wichtig, um die Logs aus unterschiedlichen Systemen unterschieden zu können. Da die Log-Meldungen auch anhand des Zeitraums abgegrenzt werden können, sollen die Daten in einem nach Wochen geordneten Index gespeichert werden. Die Daten sollen mindestens 2 Jahre gespeichert werden. Wenn alte Daten nicht gebraucht werden, kann man sie über den Indexnamen eindeutig identifizieren und gezielt löschen.
Hierfür soll der Indexname folgendermaßen aussehen.
testname_systemname_%{+YYYY.ww}: eindeutiger Key für den Indexnamen (z.B. *tcumetric_ziro_2017_39*)

4.4 Zeitstempel

Jeder Logeintrag soll mit Zeitstempel protokollieren, wann das Ereignis geschehen ist.
Die NMON schreibt Logs mit Zeitstempel nur bis Sekundenbereich. Und der JMX-Kollektor protokolliert die Ereignisse im Millisekundenbereich. Daher wird das Format "*yyyy-mm-dd hh:mm:ss*" für nmon Logs und für alle anderen Logs "*yyyy-mm-ddThh:mm:ss.ZZZ*" benutzt.

Beispiel Zeitstempel für NMON Log: *2017-11-22 12.23:12*
Beispiel Zeitstempel für Lasttestergebnisse: *2017-11-22T12:23:22:111*

5 Implementierung

Dieses Kapital beinhaltet die Implementierung von NMON-Json Parser, NMON-CSV Parser, und die Konfiguration der Teile der Elasticstack, womit zentralisiertes Logging und Visualisierung möglich werden.

5.1 Parser

5.1.1 NMON CSV Parser

Ein Parser ist in Java geschrieben, der die .nmon Datei in CSV parst.

```
java -jar nmonCsvParser "servername" "path to the .nmon file"
```

Listing 5.1: NMON-Csv Parser Aufruf

Der Aufruf sieht beispielsweise so aus.:

```
java -jar nmonCsvParser ziro C:\Users\bdh\Desktop\nmonresult\ziro.nmon
```

Listing 5.2: Beispiel NMON-Csv Parser Aufruf

Es werden zwei Parameter erwartet: der erste Parameter ist der Servername, dessen Log-Datei geparst werden muss. Der Name ist hier wichtig, da es eine CSV Datei mit demselben Namen erzeugt, die später in Logstash als Input kommt. Der zweite Parameter ist der Pfad von der .nmon Datei. Die Implementierung des Parsers ist im Anhang A2 zu finden.

```
1 ZZZZ,T0001,15:23:04,03-OCT-2017
2 CPU_ALL,T0001,0.4,0.4,0.0,99.1,0.9,4
3 MEM,T0001,29.9,99.4,2446.6,2035.9,8192.0,2048.0
4 MEMNEW,T0001,8.7,43.1,18.4,29.9,22.7,48.4
5 NET,T0001,2.9,0.0,1.4,0.0
6 NETPACKET,T0001,28.7,0.1,11.6,0.1
7 NETSIZE,T0001,103.8,74.0,121.2,74.0
8 NETERROR,T0001,0.0,0.0,0.0,0.0,0.0,0.0
```

```
9 JFSFILE,T0001,59.9,82.7,15.4,1.1,2.3,80.3,0.1,54.3
```

Listing 5.3: Code Abschnitt einer .nmon Datei

Der Nmon Output für einen bestimmten Zeitstempel sieht wie folgt aus. Jeder Abschnitt beginnt mit einem String "ZZZZ" und beinhaltet wichtige Information in mehreren Zeilen. Mit Hilfe des regulären Ausdrucks wird der Abschnitt gelesen und in CSV geschrieben. Der Parser liest die Datei bis zum Ende und erzeugt eine Zeile für einen Zeitstempel, mit allen wichtigen Werten im CSV Format.

```
15:23:04, 03-OCT-2016, CPU_ALL, T0001, 0.4,  0.4, 0.0, 99.1, 0.9,
4, MEM, T0001, 29.9,  99.4, 2446.6, 2035.9, 8192.0, 2048.0, PROC,
T0001, 0.13, 0.00, 309, 2249, 140, 4, 2, 3, 0, 0, 0, 0, 0, NET,
T0001, 2.9, 0.0, 1.4, 0.0, ........
JFSFILE, T0001, 59.9, 82.7, 15.4, 1.1, 2.3, 80.3, 0.1, 54.3.
```

Listing 5.4: Output von Parser in CSV Format

5.1.2 NMON JSON Parser

Der NMON JSON Parser nimmt die Daten aus NMON entgegen und parst sie zu einer validen JSON Datei, die direkt in Elasticsearch geschickt werden kann. Der Aufruf erwartet auch 2 Parameter wie beim CSV Parser. Jahr und Kalenderwoche werden von Java dynamisch festgelegt und Zeilen mit validem Index-Format geschrieben, was beim Bulk Import[22] zur Elasticsearch sehr wichtig ist. Der JSON Output sieht folgendermaßen aus.

```
"index":{"_index":"tcu-nmon-tank-2017-48","_type": "nmon"}}
{"jfs-
file_/usr":86.2,"jfsinode_/":39.5,"PCPU_ALL":"PCPU_ALL","jfs-
file_/var/adm/ras/livedump":0.1,"NETPACKET":"NETPA-
CKET","page_Paging             R1610031508_03-tank":"T0001","jfs-
file_/opt/oracle":59.2,
"DISKBSIZE":"DISKBSIZE","page_faults":10900.3,"diskbsize_hdisk1"
:10.0,"pcpu01_Idle     ":0.24,"diskbsize_hdisk0":4.0,"cpu02_Idle%
":100.0,..., netpacket_en0-writes/s":40.3}
```

Listing 5.5 JSON Output von NMON-Json Parser

[22] Der Bulk API von Elasticsearch bietet die Möglichkeit, mehrere Requests abzuschicken und legt das Format der JSON Datei fest.

Dabei definiert die erste Zeile den Index und Type von Elasticsearch, wohin die Daten geschickt werden sollen, und die zweite Zeile enthält die aktuelle Log Nachricht.

5.2 Logstash

5.2.1 Konfiguration

Konfiguration von Logstash für NMON Logs

```
bin/logstash.bat -f ziro-logstash.conf
```

Listing 5.6 Aufruf Logstash mit eigene Konfigdatei

Um den Logstash zu startet, muss man eine Konfigdatei mitgeben, wo sich die notwendige Konfiguration befindet.

```
1 input {
2    file{
3      path           =>           "C:\Users\bdh\workspacetcp\Nmon-
4 CsvToJson\src\ziro.csv"
     start_position => beginning}}
```

Listing 5.7 : Input Abschnitt von Konfigdatei

In Input ist der Pfad der Datei definiert, die Logstash weiterverarbeiten soll. *start_position* als `beginning` zwingt Logstash, die CSV Datei der ersten Zeile zu lesen.

```
filter{
  csv {
   columns => [
    "time",
    "date",
    "cpu_all",
    "cpu_all_timeinterval",
    "cpu_all_user",
    "cpu_all_sys",
    "cpu_all_wait",
    "cpu_all_idle",
    "cpu_all_busy",
    "cpu_all_cpu",
[.....]
    "filespace_export_data",
```

```
    "filespace_export_home",
    "opt_oracle"
    ]
    separator => ","
    remove_field => ["message" ,"mytimestamp"]
   }
 mutate {
   add_field => {
     "mytimestamp" => "%{date} %{time}"
   }
     }
 date {
   match => ["mytimestamp", "dd-MMM-yyyy HH:mm:ss"]
   target => "@timestamp }}
```

Listing 5.8 Filter in Logstash

Als Filter benutzen wir den CSV Plugin von Logstash. Das Plugin liest Zeilenweise die CSV Datei, und erzeugt eine JSON, die als Attribute die *columns*-Namen des Arrays und und als Werte die durch Komma getrennten Werte von der CSV Datei hat.

Logstash fügt zusätzliche Felder wie *source*, *type*, *tags*, *timestamp*, *source_host*, *source_path* etc. Die ganze Lognachricht als JSON wird zusätzlich in *@message* Feld gespeichert. Felder können ignoriert werden in dem sie in *removefield* geschrieben werden. Dann werden die Felder nicht an Elasticsearch weitergegeben.
Logstash definiert seine eigenen Zeitstempel, und enthält den Zeitpunkt, wann die Daten an Elasticsearch gesendet werden. Aber der Zeitpunkt wo die Ereignisse geschehen ist, ist wichtig, um später nach dem Datum zu suchen und zu analysieren.
In mutate werden zwei Felder *Date* und *Time* Feld gemergt und ein neues Feld "*mytimestamp*"erzeugt, der *dd-MMM-yyyy HH:mm:ss* Format entspricht. Der Default *@timestamp* hat die Zeitstempel, wann die Daten in Elasticsearch geschickt wurden. Aber uns interessiert die Zeitstempel, wann die Ereignisse passiert sind. In Date wird der Wert von *@timestamp* durch die wert in *mytimestamp* ersetzt, damit die *@timestamp* die Ereignisse geschehener Zeitstempel hat.
Hierbei existiert zwei Felder *@timestamp* und *mytimestamp* mit dem gleichen Datum. "*mytimestamp*" wird auch in *ignorefeld* Liste gesetzt, um es nicht in Elasticsearch zu importieren.

```
output{
  elasticsearch {
```

```
    manage_template => true
    template_overwrite => true
    hosts => ["http://bigdata01:9200"]
    index => "tcu-nmon-tank-%{+YYYY.ww}"
    document_type => "logsnmon"
    template=>
 "C:\Users\bdh\Downloads\logstashnew\logstash-5.6.3\tank-nmon-temp-
late.json"

   }
}
```

Listing 5.9 Output CSV zu Elasticsearch

Nachdem die Daten mit CSV Plug-Ins zu JSON gemappt sind, können sie zu Elasticsearch geschickt werden. Den Wert *manage_template* und *template_overwrite* auf "true" zu setzen erlaubt uns, eigene Templates für Mapping zu definieren. *hosts* enthält die URL von Elasticsearch, wohin es geschickt werden soll. *index* definiert das Indexpattern und *document_type* definiert den Typ des Elasticsearch Index. In Template ist der Pfad zu unserer eigenen definierten Mapping enthalten.

```
{
 "template" : "tcu-nmon-tank-*",
 "version" : 1,
 "settings" : {
   "index.refresh_interval" : "5s
 },
 "mappings" : {
   "_default_" : {
     "dynamic": "true",
     "_all" : {"enabled" : true, "norms" : false},
     "properties" : {
       "@version": { "type": "keyword", "include_in_all": false
          "mytimestamp" : { "type" : "date" ,
           "format" : "dd-MMM-yyyy HH:mm:ss"
         },
          "cpu_all":    { "type": "text"  },
          "cpu_all_timeinterval":{ "type": "text"  },
{...
```

Listing 5.10 selbst definierte Mapping für Elasticsearch

Template Maps sind eine JSON Datei, die definiert, wie die einzelnen Felder gemappt werden sollen. Template enthält einen regulären Ausdruck, und alle Indexe benutzen das Template, dessen Name mit dem Regulären Ausdruck übereinstimmt. In *properties* werden die Typen von Feldern definiert.

Konfiguration von Logstash für Lasttest Ergebnisse

Die Lasttestergebnisse sind mit dem JMX Kollektor im JSON Format geschrieben, womit es direkt an Elasticsearch gesendet werden kann. Die erste Zeile enthält den Indexnamen und Type und die zweite Zeile die Log-Meldung. Die Logdateien sind je nach System unterschiedlich groß. Der JMX Kollektor erzeugt auch Dateien mit einer Größe von über 1 GB.

```
{"index":{"_index":"tcu-test","_type":"gxpmvf01_50631"}}
{"host":"gxpmvf01","jmxport":50631,"object-
name":"java.lang:type=ClassLoading",
 "objectname_domain":"java.lang","objectname_type":"ClassLoa-
ding",
"attr.LoadedClassCount":17999,
"attr.TotalLoadedClassCount":18355,                "attr.Unloade-
dClassCount":356,"@version":1,"system":"SRV001","ti-
mestamp":1498514456478,"@timestamp":"2017-06-26T22:00:56.478Z"}
{"index":{"_index":"tcu-test","_type":"gxpmvf01_50631"}}
{"host":"gxpmvf01","jmxport":50631,"object-
name":"java.lang:type=ClassLoading",              "objectname_do-
main":"java.lang","objectname_type":"ClassLoading",  "attr.Loade-
dClassCount":17999,
"attr.TotalLoadedClassCount":18355,
"attr.UnloadedClassCount":356,"@version":1,                ","ti-
mestamp":1498514456478,"@timestamp":"2017-06-26T22:00:56.478Z"}
```

Listing 5.11 Format einer Logdatei aus dem JMX Kollektor

```
1 filter{
2     json {
3    source => "message"
4  }
5 if([message]             =~             /{"index":{"_index":"tcu-
6 test","_type":"gxpmvf01_50631"}}/ ){
7         drop{}
8    }
}
```

Listing 5.12 Logstash Filter Konfiguration

```
output{
  elasticsearch {
    hosts => ["http://bigdata01:9200"]
    index => "tcu-lasttest-%{+YYYY.ww}"
    document_type => "tculogs"
   }
   stdout { codec => rubydebug }
}
```

Listing 5.13 Logstash Output Konfiguration

Hier ist kein Mapping explizit angegeben. In diesem Fall greift Elasticsearch auf das dynamische Feld Mapping zu. Wenn ein neues, noch nicht vorhandenes Feld, kommt, definiert Elasticsearch den Typ von diesem Feld automatisch. Hier macht das Dynamische Mapping Sinn, weil die Logdatei jederzeit erweitert werden kann und neue Felder hinzugefügt werden können, und Elasticsearch passt das Mapping automatisch für die neu hinzugekommenen Felder anpasst. Die Tabelle 3 beschreibt, wie das neue nicht definierte Feld ein Typ zugewiesen bekommt.

null	die Feld wird nicht hinzugefügt
true oder false	boolean Feld
fließkomma zahl	float Feld
integer	Long Feld
object	Objekt Feld
String	Date, falls der Wert in Date Erfassung (Date Detection) eintrifft, double oder Long, falls er in numerischer Erfassung eintrifft sonst als Text Feld

Tabelle 3: Default Mapping von Werten in Elasticsearch

5.3 Elasticsearch

Die Einstellung von Elasticsearch kann man in der Datei Elasticsearch.yml ändern.

```
1  cluster.name: tcu
2  node.name: node1
3  network.host: bigdata01
4  http.port: 9200
5  path.data: /var/lib/elasticsearch
6  path.logs: /var/log/elasticsearch
```

Listing 5.14 Elasticsearch Konfiguration

Travic-Corporate besitzt einen Elasticsearch Cluster mit 2 Nodes für den ein Replikat pro Index definiert ist.

```
{
  "name" : "bigdata01",
  "cluster_name" : "tph-elasticsearch",
  "cluster_uuid" : "iXMxc8qgR5eFxYtnn-e2TA",
  "version" : {
    "number" : "5.6.1",
    "build_hash" : "667b497",
    "build_date" : "2017-09-14T19:22:05.189Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

Listing 5.15 Elasticsearch Status prüfen

Wenn Elasticsearch gestartet wird, kann man über hostname:port überprüfen, ob Elasticsearch läuft und welche Version Elasticsearch besitzt. In unserer Anwendung ist es bigdata01:9200.

```
   GET /cluster/health
```

Listing 5.16 Health Aufruf von Elasticsearch

Elasticsearch bietet den Endpoint /cluster/health, mit demman Status und Info vom Cluster aufrufen kann. Hier ist wichtig, dass der Status auf Grün ist. In Abbildung 5.1

ist zu sehen, dass unser Elasticsearch-Cluster aus 2 Nodes besteht. Da der Index nur eine logische Aufteilung ist und physikalisch die Daten in Shards gespeichert werden, gibt es auch nur eine Anzahl von Shards und keinen Index. Der Wert von *active_primary_shard* ist 211 und *active_shards* ist 422. Das heißt, unser Cluster besitzt pro Shard ein Replikat.


```
{
  "cluster_name": "tph-elasticsearch",
  "status": "green",
  "timed_out": false,
  "number_of_nodes": 2,
  "number_of_data_nodes": 2,
  "active_primary_shards": 211,
  "active_shards": 422,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 0,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks": 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 100
}
```


Abbildung 5.1 Elasticsearch Health

```
GET /_cat/indices
```

Listing 5.17 Listet alle Indices unseres Clusters

Post mit cURL

Für kleine Daten kann man einfach über cURL die JSON Dateien in Elasticsearch speichern.

Die JSON Datei muss folgendes Format einhalten.

```
{ "index" : { "_index" : "test", "_type" : "type1", "_id" : "1"
}}
{ "field1" : "value1" }
{ " index " : { "_index" : "test", "_type" : "type1", "_id" :
"2"}}
{ "field2" : "value2" }
{ "index" : { "_index" : "test", "_type" : "type1", "_id" : "3"
}}
{ "field1" : "value3" }
```

Listing 5:18 Format JSON Datei für Bulk Import

Die erste Zeile definiert Index und Typ. Die folgende Zeile enthält die Logdatei, die in Index und Typ gespeichert wird. Die optimale Größe von Daten, die übertragen werden sollen, ist ca. 15 MB.

```
curl -XPOST 'http://bigdata01:9200/_bulk' --data-binary @bulk-
import.json
```

Listing 5.19 cURL Befehl um JSON Datei zu Indexen

Aber da meistens die Logfile größer ist, wird ein Shell Skript geschrieben, das die Datei in viele kleine Dateien zerlegt und mehrmals Post Request durchführt.

```
1 rm -rf temp
2 mkdir temp
3 split -l 20000 test1.json temp/chunk
4 cd temp
5 for f in *;
6 do
7     curl  -XPOST  'http://bigdata01:9200/_bulk'  --data-binary
8 @$f
  done
```

Listing 5.20 Shell Skript für Bulkimport

Beim Speichern wird die Daten erstmal im Index Puffer (Buffer) gespeichert. Es kann zu EsRejectedExecutionException kommen, wenn auf einmal viele Index Requests gemacht werden. Daher wird hier die gesamte Logdatei in viele kleine Dateien mit maximal 20000 Zeilen aufgespalten und in einer Schleife mehrmals per Postrequest an Elasticsearch gesendet.

5.4 Metricbeat

Einstellung

In die metricbeat.yml Datei wird konfiguriert, wohin die von Metricbeat gesammelte Logdatei geschickt werden soll, und man definiert den Index. Host ist hier die Adresse, wo das Elasticsearch Cluster installiert ist. Der Parameter *index* definiert den

Namen des Indexes, der in Elasticsearch erzeugt wird. Es wird ein neuer Index pro Woche generiert, der beispielsweise *tcu-metric-2017-42* als Indexnamen hat.

```
output.elasticsearch:
 # Array of hosts to connect to.
 hosts: ["bigdata01:9200"]
 index: "tcu-metric-%{+yyyy.ww}"
```

Listing 5.21 Ausschnitt von metricbeat.yml für Index in Elasticsearch

Die metric.modules definieren alle Schnittstellen, von denen der Metricbeat die Logdatei sammeln soll. In Metrik Sets sind die Komponenten definiert, deren Logs Metricbeat sammeln soll.

```
  metricbeat.modules:
  - module: system
   metricsets:
     # CPU stats
     - cpu
     # System Load stats
     - load
     # Per filesystem stats
     - filesystem
     # File system summary stats
     - fsstat
     # Memory stats
     - memory
     # Network stats
     - network
     # Per process stats
     - process
```

Listing 5.22 Ausschnitt von Metricbeat.yml

Die JSON Datei *Metricbeat.template* definiert das Mapping von allen Feldern, die Metricbeat sammeln soll. In *settings* kann man die maximale Anzahl von Feldern festlegen. Template definiert das Pattern des Indexnamens in Elasticsearch, wo die Daten indexiert werden sollen. Alle Indexe, die der Pattern Matching, benutzen das gleiche Mapping.

```
"settings": {
    "index.mapping.total_fields.limit": 10000,
    "index.refresh_interval": "5s"
  },
  "template": "tcu-metric-*"
}
```

Listing 5.23 Metricbeat.template Ausschnitt

5.5 Kibana

5.5.1 Konfiguration

Kibana ist einfach über die Datei kibana.yml konfigurierbar. Beim Starten liest Kibana die Konfigurationsdatei kibana.yml und läuft in der Default Konfiguration in localhost:5601. Um den Host und Port Nummer zu ändern oder mit Elasticsearch zu verbinden, die einem anderen Rechner übergeben werden, muss man die Datei kibana.yml aktualisieren. Den Status von Kibana kann man über host:port/status aufrufen. In unserem Anwendungsfall ist er über bigdata01/8080/status aufrufbar.

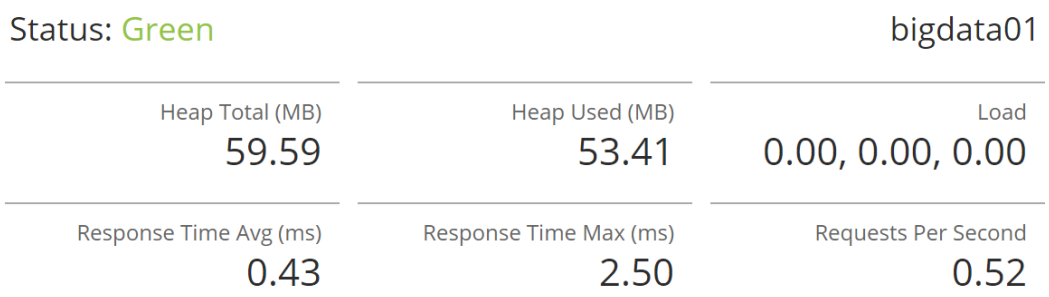

Abbildung 8: Kibana Status

5.5.2 Index Pattern

Index	Daten des Indexes

Tcu-lasttest-*	Alle Ergebnisse von Lasttest, deren Index mit Tcu-last-test- beginnt
Tcu-metric-*	Daten von Metricbeat, der in Linux Server läuft
Tcu-nmon-lock-* Tcu-nmon-sanchez-* Tcu-nmon-tank-*	Nmon Metriken von AIX, Solaris, und Linux Servern, wo die Lasttest durchgeführt wird

Tabelle 4: Erstellte Indexpattern in Kibana

In Kibana sind verschiedene Indexpattern angelegt, womit die Daten in Elasticsearch gesucht werden können. *tcu-lasttest-** beinhalt Daten aus alle Indexes, die ein Suffix tcu-lasttest haben. So wie *tcu-metric-** beinhaltet alle Index für Metricbeat. Für NMON Daten sind Index Pattern pro Server angelegt wie Z.B. *tcu-nmon.cain*.

5.5.3 Suchanfragen

Kibana lässt sich die Suche speichern und später bei der Visualisierung wird sie benutzt. Bei der Visualisierung werden in Bezug auf den Daten, die in Suche enthalten sind, Diagramme gestellt. Unterschiedliche Suchen für CPU, Netzwerk, Memory, Filesystem etc. werden gespeichert, damit sie beim Visualisieren benutzt werden können.

```
POST tcu-metricbeat-ziro-2017.44
"query": {
    "query_string": {
      "query":  "metricset.module:  system  AND  metricset.name:
cpu",
    }
}
```

Listing 5.24 Suche in Kibana

Listet alle JSON Dokumente mit dem Modul „system“ und metricset Name „cpu“. Die Suche wird benutzt um die CPU Auslastung pro Prozess zu analysieren.

```
"query": {
    "query_string": {
      "query": "metricset.module: system AND metricset.name: net-
work",
    }
```

```
}
```

Listing 5.25: Suche in Kibana

Listet alle JSON Dokumente von Network. Die Ergebnisse der Suche sind in Anhang A.1.

5.5.4 Visualisieren

Anhand der Suche und gespeicherten Indexpatterns werden Kreis-, Linien-, Metrik- und Flächendiagramme visualisiert. Die Indexe in Elasticsearch haben Daten aus Metricbeat, NMON, und Lasttest. Kibana bietet viele Diagramme zur Visualisierung von Daten. Die Daten werden aggregiert und visualisiert. Im Folgenden werden Teile der verwendeten Diagramme für Metricbeat, Lasttest Ergebnisse und NMON Daten vorgestellt.

Beats
Das Indexpattern *tcu-metric-.** wird benutzt um die Daten von Metricbeat in Kibana zu holen.

Metrikdiagramm: Die Suche aus Listing 5.25 wird benutzt um die Netzwerkdaten als Metriken darzustellen. Dafür wird Visualizer Typ Metrik gewählt und die Felder *system.network.in.bytes* und *system.network.out.bytes* werden benutzt.

Liniendiagramm: Die CPU Nutzung von User und Kernel ist als Liniendiagramm dargestellt. Auf der X-Achse sind die Attribute *@timestampe* und auf der Y-Achse sind die Felder *system.cpu.user.pct* und *system.cpu.system.pct* dargestellt.

Flächendiagramm: Hauptspeicher Nutzung pro Prozess wird als Flächendiagramm dargestellt. Die Y-Achse enthält das Feld *system.process.cpu.total.pct* und die X-Achse hat einen Zeitstempel, der pro Stunde die durchschnittliche CPU Nutzung pro Prozess zeigt.

Kreisdiagramm: Die einzelnen Sektoren des Kreisdiagramms beziehen sich auf die Metrik “total size” im Feld system.filesystem.mount_point. Die folgende Abbildung 9 zeigt die Summe von File Size und die einzelnen Teile, die es verwenden.

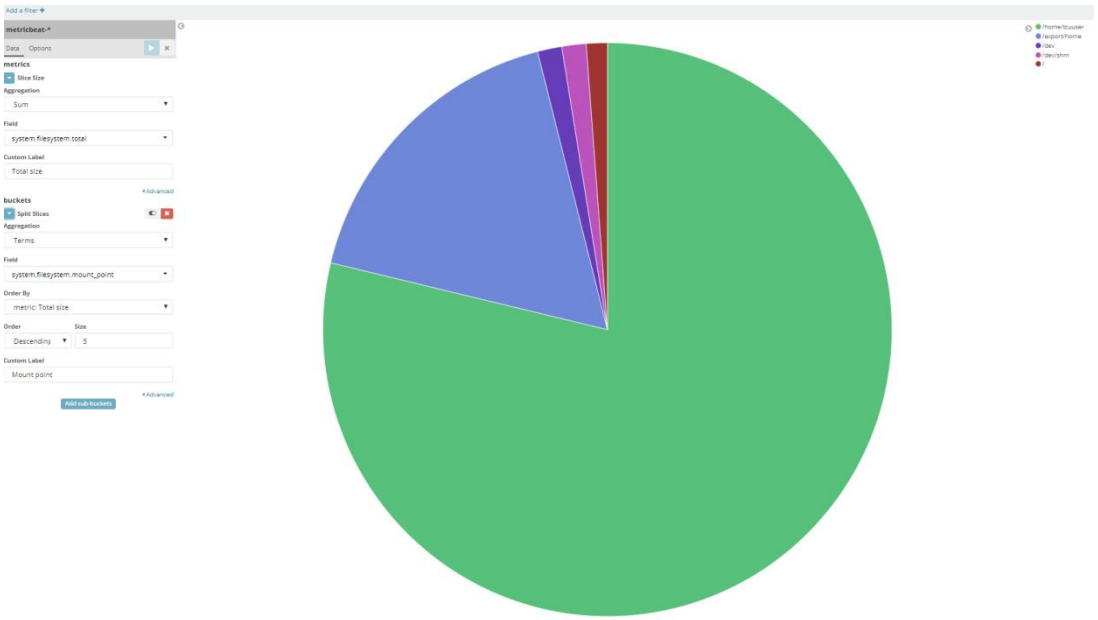

Abbildung 9: Kreisdiagramm Filesystem

Das Dashboard im Anhang gibt einen Überblick über die Diagramme von Metric Beat.

NMON Daten

Die NMON Daten werden in Elasticsearch indexiert und in Kibana Index Pattern angelegt, wo danach gesucht werden kann. Daten wie CPU Load pro usr/sys/wait, Diskread KB/s, DiskwriteKb/s, Disktransfer Kb/s, Filesystem used%, Memory usage, Netzwerk kb/s Networkpacket/s, Diskbusy % etc. werden als Liniendiagramm, Datentabelle und Flächendiagramm dargestellt.

Netzwerk Transfer kB/s als Liniendiagramm

Die Y-Achse enthält die Netzwerk read/s und write/s.
Die X-Achse trägt die Zeitstempel, wann das Ereignis geschehen ist.
Die Abbildung 10 zeigt die Anzahl der Pakete die von Ethernet pro Sekunde gesendet und empfangen wurden.

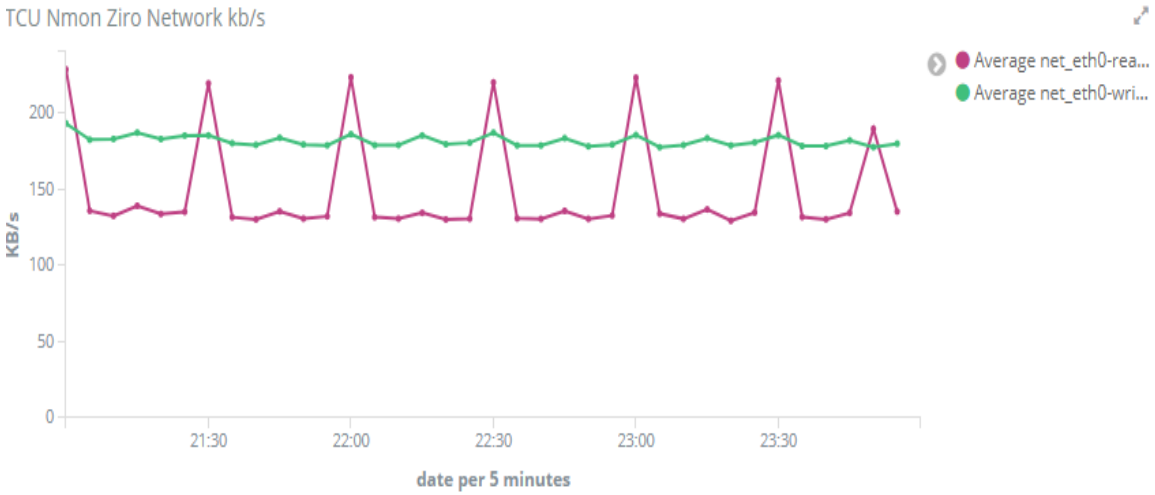

Abbildung 10: NMON Netzwerk kb/s

Memory Usage als Flächendiagramm

Die Y-Achse enthält das Memory Cache, aktive und inaktive Werte.
Die X-Achse hat die Zeitstempel, wann das Ereignis geschehen ist.
Die folgende Abbildung zeigt Cache, aktives und inaktive Memory im Laufe der Zeit.

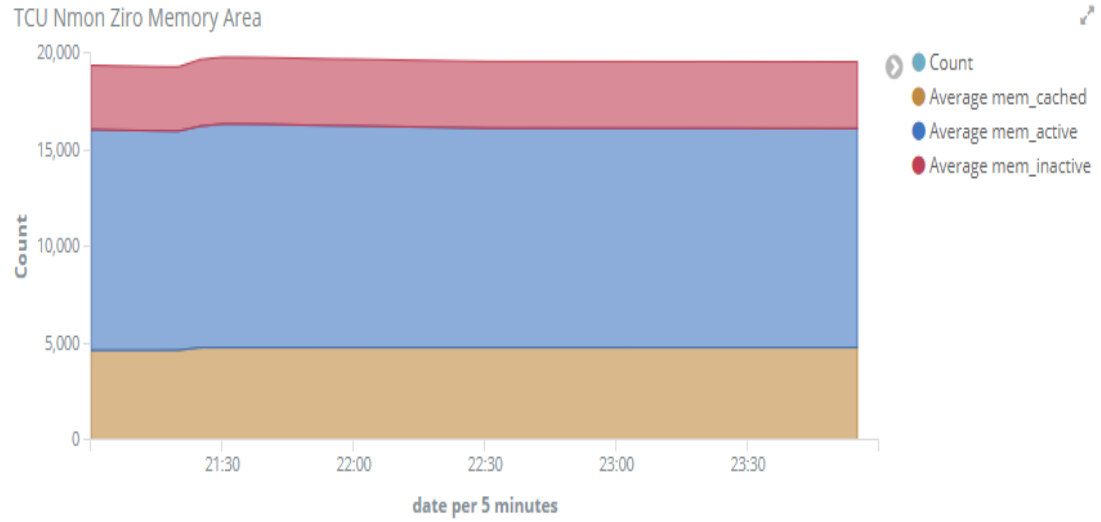

Abbildung 11: NMON CPU Usage

TCU Lasttest
Um die Daten von TCU Lasttest zu filtern und visualisieren werden Indexe mit dem Indexpattern tcu-lasttest-* benutzt. Die Lasttest-Ergebnisse stammen von Job-Server und EBICS-Server. Die Diagramme werden generisch für alle Server erstellt. In Kibana kann man aber dynamisch den Index und Type filtern, und die Diagramme werden automatisch in Bezug auf diese Daten aktualisiert. Der Status über die Verbindung, Threads, Perzentile in Antwortzeit, Heap Memory, Info über Garbage Kollektor, Anzahl Connection pro Benutzer etc. werden in unterschiedlichen Diagramme dargestellt.

5.5.5 Dashboard

Es werden unterschiedliche Dashboards für den Lasttest und die Metriken erstellt und die einzelnen Diagramme hinzugefügt, so dass Daten und Diagramme, die zusammengehören in ein Dashboard gestellt werden und, damit ein Überblick über das jeweilige System oder den jeweiligen Test hergestellt wird.

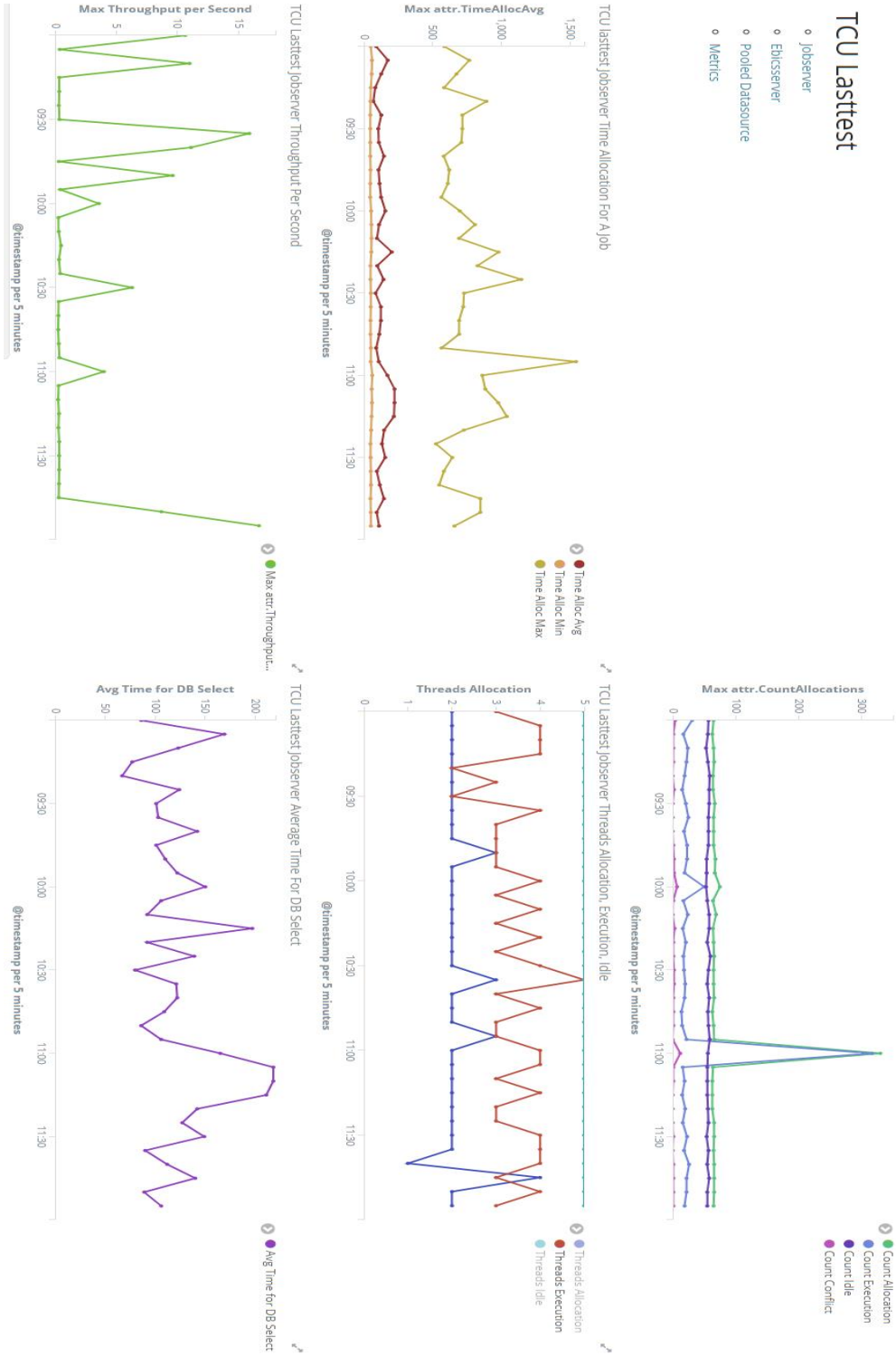

Abbildung 12: Lasttest Dashboard

6 Evaluation

6.1 NMON vs. Beats

Indexing zu Elasticsearch über Logstash und über cURL.

Um die Metriken aus Ressourcen zu sammeln, werden folgende Varianten implementiert.

1. NMON – CSV Parser – Logstash – Elasticsearch
2. NMON – JSON Parser – cURL – Elasticsearch
3. Beats – Elasticsearch

Logstash hat den Vorteil, dass die Daten beliebig manipuliert werden, bevor sie nach Elasticsearch gesendet wird. Hier können wichtige Werte gesetzt oder nicht notwendige Werte gefiltert werden. Logstash eignet sich für Parsen und Formatieren komplexer Log-Meldungen wie z.B. von Webservern, die rund um die Uhr laufen und Logs erzeugen, die viel geparst werden müssen, um einen Überblick über die Daten zu gewinnen. Aber für unseren Anwendungsfall, wo die Log-Meldungen nur während der Durchführung des Lasttests gesammelt werden, ist Logstash nicht zwingend notwendig. Daher wird die zweite Variante um Metriken aus Ressourcen zu sammeln, bevorzugt, da die .nmon Datei nur in KB-Größe vorhanden ist und sich einfach mit einem Parser in eine JSON Datei konvertieren lässt.

Für Betriebssysteme, die Beats unterstützen lassen sich sehr einfach unterschiedliche Metriken zu sammeln und nach Elasticsearch zu senden. Außerdem muss man keinen Parser bauen, um irgendwelche Daten zu formatieren. NMON zeichnet auch nur die Daten wie CPU, Filesystem, Speicherplatz, Festplatte etc. Metricbeat bietet viele Module, die in der Konfigurationsdatei hinzugefügt werden können, um die Metriken aus unterschiedlichen Quellen wie Apache, Docker, Mysql, Redis, Mongodb, Host Rechner etc. zu sammeln. Aber da es nicht möglich ist, Metrik Beat in AIX und Solaris zu installieren, brauchen wir einen Parser, um die NMON Daten zu parsen. Beats bieten auch Initial Import, womit viele Diagramme importiert werden können. Metric Beat deckt schon viele Anfragen und Auswertungen ab, und alle notwendigen

Visualisierungen. NMON generiert unterschiedliche Attributnamen für unterschiedliche Systeme, daher wird es schwer, in Kibana ein generisches Diagramm für alle Systeme zu erstellen und später gezielt nach System zu filtern. Daher ist Beat besser geeignet für zentralisiertes Logging für die Systeme, die es unterschützen.

6.2 Performanz Messung

Der JMX Kollektor schreibt die Testergebnisse in valide JSON Dateien, die unterschiedliche Größen haben. Um die Ergebnisse in Elasticsearch zu senden, werden zwei Varianten implementiert.

1. JSON Datei – Logstash – Elasticsearch
2. JSON Datei – Shell Skript – Elasticsearch

Die Lasttests müssen manchmal wiederholt durchgeführt werden, aber es ist nur ein Ergebnis davon in Elasticsearch notwendig. Aber wenn der Logstash immer läuft, sendet er jedes Mal die Log-Meldungen zu Elasticsearch, womit wir auch unnötige Ergebnisse in Elasticsearch haben, die nicht so gut analysiert werden können. Der Vorteil von Logstash ist, dass er mit sehr großen Dateien umgehen kann und diese verlustfrei nach Elasticsearch schicken kann. In der Performanz-Messung werden die Lasttest-Ergebnisse einmal mit Shell Skript und einmal über Logstash nach Elasticsearch gesendet.
Um die Jsondatei von ca. 1.2 GB mit Logstash nach Elasticsearch zu senden, hat es ca. 33 m 27 s gedauert, mit Shell Skript erfolgt die Übermittlung in ca. 6m 38s. Um die Zeit zu verringern wird auch versucht, Logstash mit erhöhter Heapgröße abzuschicken. .

```
LS_HEAP_SIZE=4g bin/logstash.bat –f
```

Listing 6.1 Logstash starten mit erhöhter Heapgröße

Auch mit erhöhter Heapgröße hat es nicht viel gebracht. Die kürzeste Zeit bei mehreren Durchläufen lag bei 27 m 19s.

Die Ergebnisse zeigen, dass über Shell Skript die Indexierung deutlich schneller ist als über Logstash.

Zweitens hat man über cURL mehr Kontrolle darüber, welche Logdatei nach Elasticsearch geschickt werden soll und welche nicht. Bei der Bulk-Indexierung werden

zunächst die ganzen Dateien in Memory geladen und danach wird weitergearbeitet. Beim Versuch der Bulk-Indexierung werden kleine bis große JSON Dateien nach in Elasticsearch geschickt. Der cURL hat in einem Bulkimport eine Indexierung von bis zu 167 MB geschafft. Jede weitere große Datei wird gecrashed und nicht in Elasticsearch gesendet. Die Offizielle Seite von Elasticsearch schlägt vor, eine 5-15 MB Bulk-Datei in eine Request zu schicken. In unserem Fall wird die Logdatei in 20000 Zeilen geteilt, die ca. 15 MB entsprechen und wiederholt in Elasticsearch geschickt. (Vgl. ERB62). Die Version NMON- JSON Parser und Shell Skript sind für unseren Anwendungsfall besser geeignet um die Logdatei zu indizieren als NMON-CSV Parser und Logstash Version.

Performanzeinstellung in Elasticsearch
Heapsize ist per Default auf 2 GB. Es kann für Zwecke der Effizienz bis zur Hälfte des Memory zugewiesen werden, aber nicht mehr als 32 GB (Vgl. ERH62).
Beim Starten der Elasticsearch kann die Heapgröße geändert werden.

```
1 ES_JAVA_OPTS="-Xms8g -Xmx8g" ./bin/elasticsearch
```

Listing 6.2 Elasticsearch Memory Zuweisung

6.3 Anforderungserfüllung

Ist die definierte Anforderung mit der Implementierung abgedeckt?

Anforderung 1: Normalisierung und Indexen der Testergebnisse
Die Daten können in Logstash gefiltert und strukturiert werden. Das CSV Plug-In ermöglicht, CSV zu JSON umzuwandeln, sowie die anderen Filter Plug-Ins ermöglichen, neue Felder hinzufügen, zusammenzuführen oder zu löschen. Einheitliche Zeitstempel werden auch mit Date Plug-Ins erreicht. Der NMON- JSON Plugin wandelt die .nmon Datei in valide JSON Datei, die direkt in Elasticsearch abgeschickt werden können. Der NMON- JSON Parser wandelt direkt die .nmon Daten in ein JSON, das direkt mit Shell Skript in Elasticsearch gesendet wird. Der Indexname wird auch dynamisch in Java erzeugt.

Anforderung 2: Zentrales Logging und Datenspeicherung
Die Testergebnisse und Metrikdaten aus unterschiedlichen Servern werden im zentralen Elasticsearch Cluster gespeichert. Logstash und Shell Skript leiten die Daten aus unterschiedlichen Quellen in Elasticsearch. Metrik Beat läuft auch im Hostsystem

und schickt ebenfalls die Metrikdaten in Elasticsearch, womit zentralisiertes Logging gewährleistet ist.

Anforderung 3: Logs-Trennung
Der Index Name hat das Format *system_test_JAHR_KW* womit die Logs von einem System zu einem anderen System klar getrennt sind. Die Metrikdaten haben auch das Index Pattern *TCU-nmon-hostname-JAHR-KW*. Es wird pro Woche ein neuer Index automatisch angelegt.

Anforderung 4: Automatisierung von Prozessen
Da die Performanz Messung gezeigt hat, dass cURL für große Datenmengen schneller ist als Logstash, wird die Variante cURL bevorzugt. Dadurch wird die Logdatei nicht automatisch nach Elasticsearch gesendet, sondern über Shell Skript. In der Lösung schreibt Logstash einen neuen Eintrag in eine Datei, und leitet ihn in Elasticsearch weiter. Die Visualisierungen in Kibana werden auch automatisch an die neuen Einträge angepasst.

Anforderung 5: Suchen und Korrelieren der Logdatei über ein Benutzerinterface
In Kibana ist ein Indexpattern definiert, nach dem die Daten gesucht werden können. Daten können über konkrete Indexnamen suchen oder auch den Indexnamen als regulären Ausdruck definieren. Es kann auch nach Feld oder Typ gesucht werden. Die Suche kann in Kibana gespeichert werden und später bei der Visualisierung benutzt werden.

Anforderung 6: Visualisierung und Auswertung von Logdatei
Wir haben unterschiedliche Visualisierungen für die gespeicherten Daten für Indexe erstellt und im Dashboard zusammengestellt, damit man schnell einen Überblick über die Daten bekommen kann. Die Visualisierung lässt sich auch als Link teilen.

Anforderung 7: Monitoring und Reporting
Monitoring ist nicht direkt in der Opensource Version von Elastic Stack enthalten. Es gibt kostenpflichtige Plug-Ins von Fremdfirmen, was eine künftige Aufgabe ist.

Anforderung 8: Open Source, möglichst keine weiteren Lizenzkosten
ELK Stack sind Opensource und lassen die Probleme mit zentralisiertem Logging und Echtzeit-Visualisierung ohne kostenpflichtige Plug-Ins darstellen.

Nicht funktionale Anforderungen 1: Skalierbarkeit

Mit dynamischem Mapping von Elasticsearch können neue Werte automatisch gemappt werden.

Nicht funktionale Anforderungen 2: Ausfallsicherheit
Die Daten werden in Elasticsearch Nodes redundant gespeichert. Unser Elasticsearch-Cluster besteht aus 2 Nodes. Wenn eine Node nicht verfügbar ist, hat die andere Node vollständige Daten und ist in der Lage Funktionsfähig weiterzuarbeiten.

Nicht funktionale Anforderungen 3: Erweiterbar
Logstash ist eine auf Plug-Ins basierende Software, die unterschiedliche Quell- und Zielsysteme unterstützt. Man kann auch selbst eine Plugin schreiben, wenn es keine Plug-Ins für bestehende Anforderungen gibt. Auch wenn eine neue Log Source kommt, kann man es sehr leicht erweitern.

Folgende Punkte zeigen, dass Elastic Stack eine gute Wahl für zentralisiertes Logging ist.

- Elasticsearch, Kibana, und Logstash passen zueinander.
- Opensource ist aktuell sehr beliebt
- Große aktive Community
- Logstash für Manipulation und Verarbeitung von semistrukturierten Daten gut geeignet
- leichte Umsetzung von Technologie
- Einfache API von Elasticsearch um Daten zu speichern und Anfragen zu erstellen
- Aggregation, Visualisierung und Auswertung von Daten aus Elasticsearch über Kibana ist sehr einfach und beliebt
- hohe Skalierbarkeit von Elasticsearch

7 Zusammenfassung

7.1 Zusammenfassung

Diese Studie hat versucht, die Frage zu beantworten, inwiefern Elastic Stack geeignet ist, zentralisiertes Logging zu realisieren. Zu diesem Zweck wurde zunächst eine quantitative Studie zu den einzelnen Technologien hinter Elastic Stack durchgeführt, die das zentralisierte Logging ermöglichen. Während der Analyse wurden durch Interviewtechnik Anforderungen für ein zentralisiertes Logging-System gesammelt. Im Rahmen dieser Arbeit wurde ein zentrales Logging und Analysesystem mit Hilfe von ELK Stack entworfen und realisiert. Es wurde nach einer Möglichkeit gesucht, die wichtigsten Logdateien aus unterschiedlichen Servern an einem zentralen Ort zu speichern und in Echtzeit zu visualisieren. Von den gesammelten Anforderungen wurden möglichst viele Anforderungen abgedeckt, außer Monitoring und Reporting, da es die Anforderung FA 7 nicht erfüllt. Die Alternative dafür wird im Ausblick vorgestellt. Während der Implementierung gab es verschiedene Herausforderungen zu bewältigen, einige waren zu erwarten, einige traten unerwartet und plötzlich auf. Besonders herausfordernd war es, sehr große Dateien in Elasticsearch ohne Datenverlust zu speichern. Vergrößerte Heap Size für Elasticsearch und Logstash war möglich. Es konnte auch eine Alternative gefunden werden, wenn Logstash nicht benutzt wird: ein Shell Skript, das die große Datei in kleine Dateien zerteilt und ein Request pro Datei macht. Ein zeitlicher Index in Elasticsearch ist auch in der Implementierung zu finden. Die wichtigsten Konzepte von Elasticsearch wie Dynamisches Mapping werden während der Implementierung mit Beispielen erklärt. Zum Abschluss der Arbeit wurden die Anforderungen, die aus der Analyse abgeleitet worden sind, evaluiert, um herauszufinden, inwiefern die Anforderungen durch die implementierte

Lösung erfüllt sind. Die Evaluation hat deutlich gemacht, dass Bulk Import von Daten in Elasticsearch viel schneller über cURL möglich ist als über Logstash.

7.2 Fazit

Dieses Kapitel beschreibt das Fazit, das sich aus der Bearbeitung dieser Bachelorarbeit ergab.
Unter den Aspekt der technischen Bearbeitungen fallen mehrere Punkte. Die Ergebnisse zeigen, dass man mit Hilfe von Elastic Stack mit wenig Aufwand ein zentralisiertes Logging- System implementieren kann. Die Installation der Elastic Komponente ist auch recht einfach unter der Voraussetzung, dass die JVM in der Maschine installiert ist. Durch zentralisiertes Logging hat man einen besseren Überblick über die Software- Komponenten und Ressourcen.
Neben den technischen Aspekten gab es einige fachliche Aspekte, die während der Bearbeitung als bemerkenswert festgehalten wurden. Das manuelle Parsen von NMON Datei zu CSV/JSON, damit es gespeichert wird, ist einen negativen Punkt im zentralisierte Logging. Beats sind besser geeignet als NMON, falls das System Beats unterstützt. Die Zusammenfassung der Daten aus den verschiedenen Systemen bringt einen Gewinn an Übersichtlichkeit. Die Konfiguration für Elastic Stack ist auch recht überschaubar und leicht veränderbar.
Der Aufruf zur Elasticsearch direkt über Kibana UI vereinfacht den Einstiegspunkt und erleichtert die Anfragen durch Auto-Vervollständigung. Die Daten können dynamisch gefiltert werden, die Diagramme werden auch jeweils mit neuen Daten automatisch angepasst. Damit müssen nicht viele Dashboards mit unterschiedlichem Index oder Typ erstellt werden, sondern ein regulärer Ausdruck für die Indexe kann definiert werden und Daten können dynamisch für den jeweiligen Index oder Typ der konkret gespeicherten Suche gefiltert werden. Die Daten können umfassender betrachtet und analysiert werden. Im Laufe der Bearbeitung und der Auseinandersetzung mit dem Themenbereich ist immer deutlicher geworden, welche Schlagkraft und Verantwortung darin steckt. Das hat zur Folge, dass ein System wie dieses sehr umsichtig gestaltet werden muss. Kibana ist sehr gut geeignet für Visualisierung und Anfragen über die Daten. Die Installation ist sehr einfach. Die Lernkurve ist sehr steil. Das Framework bietet sehr viel, womit man mit wenig Aufwand und Konfiguration einer zentralisierten Logging ermöglicht.

7.3 Ausblick

Manche Anwendungen können sehr kritisch sein und sollten ständig überwacht werden. Es ist sehr hilfreich, wenn die Überwachung automatisiert ist, damit das Risiko beim System-absturz oder bei Anwendungsfehlern früh erkannt wird und Gegenmaßnahmen ergriffen werden können. Es gibt X-PACK[23] Plug-Ins und auch viele Opensource Plug-Ins für das Monitoring von Daten in Elasticsearch. ElastAlert[24] ist ein Opensource Projekt veröffentlicht von Yelp, womit Monitoring in Elasticsearch möglich ist. Wenn die definierte Regel stoßen, kann mit Hilfe dieses Plugins nahezu in Echtzeit über verschiedene Kanäle alarmiert werden (Vgl. Ylp14). Damit kann Inkonsistenz in Daten erfasst werden. Diese entstandene Arbeit kann noch ausgebaut werden und in Zukunft für ein solches Monitoring umgesetzt werden.

[23] https://www.elastic.co/products/x-pack
[24] https://github.com/Yelp/elastalert

A. Anhang

A.1. Suchergebnisse von Elasticsearch in Kibana


```
{
  "took": 5,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 4,
    "max_score": 4.4961133,
    "hits": [
      {
        "_index": " tcu-metricbeat-ziro-2017.44",
        "_type": "doc",
        "_id": "AV8GWQzOdhblsTTZDmbk",
        "_score": 4.4961133,
```

```
19        "_source": {
20          "@timestamp": "2017-10-10T13:01:00.127Z",
21          "beat": {
22            "hostname": "ziro",
23            "name": "ziro",
24            "version": "5.6.2"
25          },
26          "metricset": {
27            "module": "system",
28            "name": "cpu",
29            "rtt": 242
30          },
31          "system": {
32            "cpu": {
33              "cores": 2,
34              "idle": {
35                "pct": 1.9258
36              },
37              "iowait": {
38                "pct": 0
39              },
40              "irq": {
41                "pct": 0
42              },
43              "nice": {
44                "pct": 0
45              },
46              "softirq": {
47                "pct": 0.002
48              },
49              "steal": {
50                "pct": 0
51              },
52              "system": {
53                "pct": 0.008
54              },
55              "user": {
56                "pct": 0.0642
57              }
58            }
```

```
59          },
60          "type": "metricsets"
61        }
62      }
```

A.2. NMON JSON Parser

```
 1 package parser;;
 2 import java.io.File;
 3 import java.io.FileNotFoundException;
 4 import java.io.IOException;
 5 import java.io.InputStream;
 6 import java.io.PrintWriter;
 7 import java.time.LocalDate;
 8 import java.time.Year;
 9 import java.time.temporal.WeekFields;
10 import java.util.ArrayList;
11 import java.util.Arrays;
12 import java.util.HashMap;
13 import java.util.List;
14 import java.util.Locale;
15 import java.util.Map;
16 import java.util.Scanner;
17
18 import com.fasterxml.jackson.core.JsonGenerationException;
19 import    com.fasterxml.jackson.databind.JsonMappingExcep-
20 tion;
21 import com.fasterxml.jackson.databind.ObjectMapper;
22
23 /**
```

```
24  * This Class parses nmon data to Json, which can be di-
25 rectly send to
26  * Elasticsearch.
27  *
28  * @author bdh
29  *
30  */
31 public class App {
32
33  private static final String SUFFIX = "tcu-nmon-";
34  private static final String DELIMETER = "ZZZZ";
35  private  static  final  String  INDEXPATTERN  =  "{\"in-
36 dex\":{\"_index\":\"%s%s-%d-%d\",\"_type\": \"nmon\"}}";
37
38  public static void main(String[] args) throws Exception {
39   if (args.length != 2) {
40    throw new IllegalArgumentException(
41      "\n Argument doesnot matches the format. \n args[0]
42 Servername,\n args[1] path to .nmon file");
43   } else {
44    String serverName = args[0].toLowerCase();
45    String path = args[1];
46    App app = new App();
47    try {
48     Scanner scanner = new Scanner(new File(path));
49     List<List<String>>   headerList   =   app.extractHea-
50 der(scanner);
51     app.parseValueToJson(scanner, headerList, serverName);
52    } catch (IOException e) {
53     throw new Exception("Could not read the file path");
54    }
55   }
56  }
57
58  private List<List<String>> extractHeader(Scanner scanner)
59 {
60   List<List<String>>    headerList    =    new    Array-
61 List<List<String>>();
62   scanner.useDelimiter(DELIMETER);
63   // fetches header which is used as JSON key
```

```
64   if (scanner.hasNext()) {
65    String chunk = scanner.delimiter() + scanner.next();
66    InputStream is = new ByteArrayInputStream(chunk.get-
67 Bytes());
68    Scanner chunkScanner = new Scanner(is);
69    List<String> firstLine = new ArrayList<String>();
70    firstLine.add(DELIMETER);
71    firstLine.add("stamp");
72    firstLine.add("time");
73    firstLine.add("date");
74    headerList.add(firstLine);
75    while (chunkScanner.hasNextLine()) {
76     String nextLine = chunkScanner.nextLine();
77     // Ignore lines from Header matching the below patterns
78     if (!nextLine.startsWith("AAA") && !nextLine.starts-
79 With("BBBP") && !nextLine.startsWith("TOP")
80       && !nextLine.startsWith("ZZZZAAA") && !next-
81 Line.startsWith("BBBL") && !nextLine.startsWith("BBBN")) {
82      List<String> items = Arrays.asList(next-
83 Line.split("\\s*,\\s*"));
84      for (int it = 1; it < items.size(); it++) {
85       String fieldName = items.get(0).toLowerCase() + "_"
86 + items.get(it);
87       items.set(it, fieldName);
88      }
89      headerList.add(items);
90     }
91    }
92    chunkScanner.close();
93   }
94   return headerList;
95  }
96
97  /*
98   * Fetch the result and save as JSON value.
99   */
100  private void parseValueToJson(Scanner scanner,
101 List<List<String>> headerList, String serverName) throws
102 IOException {
103   PrintWriter out = new PrintWriter(serverName + ".json");
```

```
104   LocalDate date = LocalDate.now();
105   int currentYear = Year.now().getValue();
106   WeekFields   weekFields   =   WeekFields.of(Locale.get-
107 Default());
108   int  currentWeek  =  date.get(weekFields.weekOfWeekBase-
109 dYear());
110   while (scanner.hasNext()) {
111    String chunk = scanner.delimiter() + scanner.next();
112    InputStream    inputStream    =    new    ByteArrayIn-
113 putStream(chunk.getBytes());
114    Scanner chunkScanner = new Scanner(inputStream);
115    Map<String, Object> map = new HashMap<String, Object>();
116    while (chunkScanner.hasNextLine()) {
117     String nextLine = chunkScanner.nextLine();
118     int i = 0;
119     while (i < headerList.size()) {
120      List<String> headers = headerList.get(i);
121      String header = headers.get(0);
122      List<String>     items     =     Arrays.asList(next-
123 Line.split("\\s*,\\s*"));
124      if (items.get(0).equals(header)) {
125       headers.size();
126       // Combine date and time together
127       if (items.get(0).equals(DELIMETER)) {
128        String dateTime = items.get(3) + " " + items.get(2);
129        items.set(3, dateTime);
130       }
131       List<Object> objectList = new ArrayList<Object>();
132       for (int a = 0; a < items.size(); a++) {
133        try {
134         Double d = Double.parseDouble(items.get(a));
135         objectList.add(a, d);
136        } catch (NumberFormatException nfe) {
137         objectList.add(a, items.get(a));
138        }
139       }
140       for (int j = 0; j < objectList.size(); j++) {
141        map.put(headers.get(j), objectList.get(j));
142       }
      }
```

```
        i++;
       }
      }
      ObjectMapper mapper = new ObjectMapper();
      String json = mapper.writeValueAsString(map);
      String indexPattern = String.format(INDEXPATTERN, SUF-
  FIX, serverName, currentYear, currentWeek);
      out.println(indexPattern);
      out.println(json);
      out.flush();
      chunkScanner.close();
     }
     out.close();
     scanner.close();
    }
   }
```

A.3. NMON CSV Parser

```
 1 package main;
 2
 3 import java.io.BufferedReader;
 4 import java.io.ByteArrayInputStream;
 5 import java.io.File;
 6 import java.io.IOException;
 7 import java.io.InputStream;
 8 import java.io.InputStreamReader;
 9 import java.io.PrintWriter;
10 import java.util.Scanner;
11 import java.util.regex.Matcher;
12 import java.util.regex.Pattern;
13
14 public class Application {
15
16  private static final String ZIRO = "ziro";
17  private static final String CAIN = "cain";
```

```
18  private static final String LOCK = "lock";
19  private static final String SANCHEZ = "sanchez";
20  private static final String SANCOR = "sancor";
21  private static final String SQUEALER = "squealer";
22  private static final String TANK = "tank";
23  private static final String CYPER = "cyper";
24  private static final String DELIMETER = "ZZZZ";
25
26  public static void main(String[] args) throws IOException
27 {
28   if (args.length != 2) {
29    throw new IllegalArgumentException(
30      "\n Argument doesnot matches the format. \n args[0]
31 Servername,\n args[1] path to .nmon file");
32   } else {
33    String serverName = args[0].toLowerCase();
34    String path = args[1];
35    Application app = new Application();
36    if (serverName.equals(ZIRO) || serverName.equals(CAIN)
37 || serverName.equals(LOCK) || serverName.equals(SANCHEZ)
38      ||      serverName.equals(SANCOR)      ||      server-
39 Name.equals(SQUEALER) || serverName.equals(TANK)
40      || serverName.equals(CYPER)) {
41     app.parseNmontoCSV(serverName, path);
42    } else {
43     System.err.println("Servername is not correct");
44    }
45   }
46  }
47
48  private  void  parseNmontoCSV(String  serverName,  String
49 path) throws IOException {
50
51   Scanner scanner = null;
52   try {
53    scanner = new Scanner(new File(path));
54    scanner.useDelimiter(DELIMETER);
55
56    PrintWriter out = new PrintWriter(serverName + ".csv");
57    while (scanner.hasNext()) {
```

```
58      String chunk = scanner.delimiter() + scanner.next();
59      InputStream is = new ByteArrayInputStream(chunk.get-
60  Bytes());
61      BufferedReader reader = new BufferedReader(new In-
62  putStreamReader(is));
63      String csvWithAllData;
64      String line = "";
65      String dateTime = "";
66      String cpu = "";
67      String mem = "";
68      String proc = "";
69      String net = "";
70      String netpacket = "";
71      String neterror = "";
72      String jsffile = "";
73      String diskbusy = "";
74      String diskread = "";
75      String diskwrite = "";
76      String diskxfer = "";
77      String diskbsize = "";
78
79      while ((line = reader.readLine()) != null) {
80       if (line.matches("ZZZZ.*")) {
81        Pattern pattern = Pattern.compile("ZZZZ,.*?,(.*)");
82        Matcher matcher = pattern.matcher(line);
83        if (matcher.find()) {
84         dateTime = matcher.group(1);
85        }
86       } else if (line.matches("CPU_ALL.*")) {
87        cpu = line;
88       } else if (line.matches("MEM,.*")) {
89        mem = line;
90       } else if (line.matches("PROC,.*")) {
91        proc = line;
92       } else if (line.matches("NET,.*")) {
93        net = line;
94       } else if (line.matches("NETPACKET,.*")) {
95        netpacket = line;
96       } else if (line.matches("NETERROR,.*")) {
97        neterror = line;
```

```
 98      } else if (line.matches("JFSFILE,.*")) {
 99       jsffile = line;
100      } else if (line.matches("DISKBUSY,.*")) {
101       diskbusy = line;
102      } else if (line.matches("DISKREAD,.*")) {
103       diskread = line;
104      } else if (line.matches("DISKWRITE,.*")) {
105       diskwrite = line;
106      } else if (line.matches("DISKXFER,.*")) {
107       diskxfer = line;
108      } else if (line.matches("DISKBSIZE,.*")) {
109       diskbsize = line;
110      }
111
112     }
113     if      (serverName.equals(SANCOR)      ||      server-
114 Name.equals(SANCHEZ)) {
115      csvWithAllData = dateTime + "," + cpu + "," + mem +
116 "," + proc + "," + net + "," + netpacket + "," + neterror
117 + ","
118        + diskbusy + "," + diskread + "," + diskwrite + ","
119 + diskxfer + "," + diskbsize;
120     } else if (serverName.equals(ZIRO)) {
121      csvWithAllData = dateTime + "," + cpu + "," + mem +
122 "," + proc + "," + net + netpacket + diskbusy + "," +
123 diskread
124        + "," + diskwrite + "," + diskxfer + "," + diskbsize
125 + "," + jsffile;
126     } else if (serverName.equals(SQUEALER)) {
127      csvWithAllData = dateTime + "," + cpu + "," + mem +
128 "," + proc + "," + net + "," + netpacket + "," + diskbusy
    + ","
         + diskread + "," + diskwrite + "," + diskxfer + ","
    + diskbsize + "," + jsffile;
     } else {
      csvWithAllData = dateTime + "," + cpu + "," + mem +
    "," + proc + "," + net + "," + netpacket + "," + neterror
    + ","
         + diskbusy + "," + diskread + "," + diskwrite + ","
    + diskxfer + "," + diskbsize + "," + jsffile;
```

```
        }
        out.println(csvWithAllData);
        out.flush();
       }
       out.close();
       scanner.close();
      } catch (IOException e) {
       throw new IOException("Could not read the file path");
      }
     }
    }
```

A.4. Logstash Konfigiguration um .CSV Daten in Elasticsearch zu indexen

```
# Logstash Configuration to convert from csv to json
input {
  file{
    path        =>        "C:\Users\bdh\workspacetcp\Nmon-CsvToJson\src\lock.csv"
    start_position => beginning
  }
}
filter{
  csv {
    columns => [
      "time",
      "date",

      "cpu_all",
      "cpu_all_timeinterval",
      "cpu_all_user",
      "cpu_all_sys",
      "cpu_all_wait",
      "cpu_all_idle",
```

```
21        "cpu_all_busy",
22        "cpu_all_cpu",
23
24        "memory",
25        "memory_timeinterval",
26        "memory_realfree_inperc",
27        "memory_virtualfree_inperc",
28        "memory_realfree_inmb",
29        "memory_virtualfree_inmb",
30        "memory_realtotal_inmb",
31        "memory_virtualtotal_inmb",
32
33
34        "processes",
35        "processes_timeintervalprocess",
36        "processes_runnable",
37        "processes_swapin",
38        "processes_pswitch",
39        "processes_syscall",
40        "processes_read",
41        "processes_write",
42        "processes_fork",
43        "processes_exec",
44        "processes_sem",
45        "processes_msg",
46        "processes_asleep_bufio",
47        "processes_asleep_rawio",
48        "processes_asleep_diocio",
49
50        "networkio",
51        "networkio_timeinterval",
52        "networkio_en0_read_kb/s",
53        "networkio_lo0read_kb/s",
54        "networkio_en0write_kb/s",
55        "networkio_lo0write_kb/s",
56
57        "networkpacket",
58        "networkpacket_timeinterval",
59        "networkpacket_en0_read/s",
60        "networkpacket_lo0read/s",
```

```
61        "networkpacket_en0write/s",
62        "networkpacket_lo0write/s",
63
64        "networkerror",
65        "networkerror_timeinterval",
66        "networkerror_en0_ierrs",
67        "networkerror_lo0-ierrs",
68        "networkerror_en0-oerrs",
69        "networkerror_lo0_oerrs",
70        "networkerror_en0_collisions",
71        "networkerror_lo0_collisions",
72
73        "diskbusy",
74        "diskbusy_timeinterval",
75        "diskbusy_hdisk3",
76        "diskbusy_hdisk0",
77
78        "diskreadkb/s",
79        "diskread_timeinterval",
80        "diskread_hdisk3_kb/s",
81        "diskread_hdisk0_kb/s",
82
83        "diskwritekb/s",
84        "diskwrite_timeinterval",
85        "diskwrite_hdisk3_kb/s",
86        "diskwrite_hdisk0_kb/s",
87
88        "diskxfer_disk/s",
89        "diskxfer_timeinterval",
90        "diskxfer_hdisk3",
91        "diskxfer_hdisk0",
92
93          "diskbsize" ,
94        "diskbsize_timeinterval",
95        "diskbsize_hdisk1",
96        "diskbsize_hdisk0",
97
98        "filespace_usedpercentage",
99        "filespace_timeinterval",
100       "filespace_/",
```

```
101       "filespace_usr",
102       "filespace_var",
103       "filespace_tmp",
104       "filespace_admin",
105       "filespace_opt",
106       "filespace_var_adm_ras_livedump",
107       "filespace_export"
108       ]
109   separator => ","
110   #remove_field => ["message"]
111
112     }
113   mutate {
114     add_field => {
115       "mytimestamp" => "%{date} %{time}"
116     }
117       }
118   date {
119     match => ["mytimestamp", "dd-MMM-yyyy HH:mm:ss"]
120     target => "@timestamp"
121   }
122 }
123 output{
124   elasticsearch {
125       manage_template => true
126         template_overwrite => true
127       hosts => ["http://bigdata01:9200"]
128       index => "tcu-nmon-lock-%{+YYYY.ww}"
129       document_type => "logsnmon"
130       template     =>     "C:\Users\bdh\Downloads\logstas-
131 hnew\logstash-5.6.3\lock-nmon-template.json"
132     }
      stdout { codec => rubydebug }
    }
```

A.5. Logstash Konfiguration um Lasttest Ergebnisse in Elasticsearch zu indizieren

```
   input {
1    file{
 2     path  =>  "C:\Users\bdh\Desktop\tcu-nmon-lasttest-job-
 3 server.json"
 4     start_position => beginning
 5 }
 6 filter{
 7     json {
 8    source => "message"
 9  }
10    if([message]        =~        /{"index":{"_index":"tcu-
11 test","_type":"gxpmvf01_50631"}}/ ){
12      drop{}
13    }
14 }
15
16 output{
17  elasticsearch {
18    hosts => ["http://bigdata01:9200"]
19    index => "tcu-lasttest-%{+YYYY.ww}"
20    document_type => "tculogs"
21    }
22  stdout { codec => rubydebug }
  }
```

A.6. Shellskript um große Json Dateien zu zerkleinen und von Elasticsearch zu indizieren

```
rm -rf temp
mkdir temp
split -l 20000 test1.json temp/chunk
cd temp
for f in *;
do
    curl -XPOST 'http://bigdata01:9200/_bulk' --data-binary
@$f
done
```

Glossar

C		
	cURL	Command Line Werkzeug um Request mit URL zu machen
D		
	DSL	Domain Specific Language
E		
	EBICS	Electronic Banking Internet Communication Standard
F		
	FTP	File Transfer Protocol
H		
	Hibernate	ORM Framework für Java, welches das JPA Interface implementiert.
	HTTP	Hypertext Transfer Protocol
J		
	JPA	Java Persistence API
	JSON	Javascript Object Notation ist ein Format für Nachrichten austausch, der für mensch und Maschine Lesbar ist.
	JVM	Java Virtual Machine
	JMX	Java Management Extensions
	JDBC	Java Database Connectivity
N		

	NMON	Ein Tool von IBM um Logs von einem System zu Sammeln
	NoSQL	Kurzform von "Not only SQL". NoSQL Datenbanken ist eine nicht relationale Datenbank, die mehr Flexibilität bieten als Relationale Datenbank, besitzen meistens keine festen Schemas und sind horizontal skalierbar.
P		
	PeSIT	Protocol d'Echanges pour un Systeme Interbancaire de Telecompensation
R		
	REST	REpresentational State Transfer ist ein Protokoll, die über HTTP läuft.
S		
	SYSLOG	Ein Protokoll um Event Nachrichten zwischen Rechner und Software Anwendungen auszutauschen
	Spring MVC	Request/Response basierte Framework für Java Web Entwicklung

Literatur

[GZ15] Clinton Gormley und Zachary Tong *Elasticsearch: The Definitive Guide O'Really Media, 2015*. ISBN 978-1-449-35854-9

[Hop16] Florian Hopf *Elasticsearch: Ein praktisches Einstieg dpunkt.verlag GmbH, 2016*. ISBN 978-3-86490-289-5

[CSP13] Anton Chuvakin, Kevin Schmidt und Chris Phillips *Logging and Log management: the authoritative guide to understanding the concepts surrounding logging and log management* Syngress, *2013*. ISBN 978-1-597-49635-3

[QSS] [Kle13] Stephan Kleuker *Qualitätssicherung durch Softwaretests: Vorgehensweisen und Werkzeuge zum Test von Java-Programmen, 2013*. ISBN 978-3-8348-2068-6

[Vro14] Remko Vroomans *Warum Agile-Projektmanagement? (2014)* URL: https://www.prowareness.de/wp-content/uploads/2014/10/Warum_Agile_Projektmanagement.pdf – [Online; Zugriff 11.01.2018]

[Mei16] Claudia Meindl *Logging mit Elasticsearch, Logstash und Kibana (2016)* URL: https://alphanodes.com/de/logging-elasticsearch-logstash-kibana – [Online; Zugriff 17.12.2017]

[Sto15] Andreas Stotzer *Einführung zu Elasticsearch (2015)* URL: https://www.diso.ch/fachartikel-und-einfuehrung-elasticsearch-andreas-stotzer/ – [Online; Zugriff 21.12.2017]

[LA] [Kor12]Robert Korherr Wie Sie Logdateien professionell auswerten (2012) URL: https://www.computerwoche.de/a/wie-sie-logdateien-professionell-auswerten,2509639,2 – [Online; Zugriff 21.12.2017]

[Tfs] *TRAVIC Corporate Features und Systemvoraussetzungen* URL: https://www.ppi.de/fileadmin/user_upload/Software-Produkte/Publikationen/en/A705e_TRAVIC-Corporate_web.pdf – [Online; Zugriff 03.01.2018]

[Bsi13] Bundesamt für Sicherheit in der Informationtechnik *IT-Grundschutz: M 2.500 Protokollierung von IT-Systemen (2013)* URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02500.html – [Online; Zugriff 15.12.2017]

[Qme] QMethods *Test Automation: Automation von Testfällen, Lasttests und synthetischem Monitoring* URL: https://www.qmethods.de/service-testautomation.html – [Online; Zugriff 12.12.2017]

[Kla15] Waldemar Kaus *Log Management: Die Intelligence in DevOps (2015)* URL: https://www.opitz-consulting.com/fileadmin/user_upload/Collaterals/Artikel/whitepaper-logmanagement_sicher.pdf). – [Online; Zugriff 23.12.2017]

[She14] Jerry Shenk *Ninth Log Management Survey Report (2014)* URL: https://www.sans.org/reading-room/whitepapers/analyst/ninth-log-management-survey-report-35497 – [Online; Zugriff 23.12.2017]

[Vus10] Enikö Vivien Visky *Log-Management: Wichtige gesetzliche Pflicht für Unternehmen(2010)* URL: https://www.tecchannel.de/a/log-management-wichtige-gesetzliche-pflicht-fuer-unternehmen,2021947,2 – [Online; Zugriff 18.01.2018]

[Lmc] Logmatic.io *Logging Best Practices: One Step beyond Application Monitoring* URL: https://logmatic.io/public/Logmatic.io_WhitePaper_Logging%20Best%20Practices.pdf – [Online; Zugriff 22.01.2018]

[Ruf] Urs Rufer Zentrales Logging zur Erkennung von Sicherheitsvorfällen URL: http://www.security-zone.info/newsletter/maerz05/redaktion/U_Rufer.pdf – [Online; Zugriff 19.02.2018]

[Mor16] Jamie Morgan *What is Centralized Log Management (CLM)? (2016)* URL: https://www.stratalux.com/blog/what-is-centralized-log-management-clm/ – [Online; Zugriff 19.02.2018]

[BL15] Mirjam Bornschein, Daniel Lienert *Datenvisualisierung mit Kibana(2015)* URL: https://punkt.de/de/blog/2015/datenvisualisierung-mit-kibana.html – [Online; Zugriff 23.02.2018]

[ERM6.2] Elastic. Co*Elasticsearch Reference [6.2] Mapping* URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html – [Online; Zugriff 23.02.2018]

[ERB6.2] Elastic. Co *Elasticsearch Reference 6.2 Bulk Api* URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/bulk.html – [Online; Zugriff 23.02.2018]

[ERH6.2] Elastic. Co *Elasticsearch Reference 6.2 Heap Size* URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html – [Online; Zugriff 23.02.2018]

[ELR6.2] Elastic. Co *Logstash Reference 6.2* URL: https://www.elastic.co/guide/en/logstash/current/index.html – [Online; Zugriff 23.02.2018]

[EKR6.2] Elastic. Co *Kibana Reference 6.2* URL: https://www.elastic.co/guide/en/kibana/current/introduction.html – [Online; Zugriff 25.02.2017]

[EBR6.2] Elastic. Co *Beats Platform Reference 6.2* URL: https://www.elastic.co/guide/en/beats/libbeat/6.2/getting-started.html – [Online; Zugriff 25.02.2017]

[EDG2.0] Elastic. Co *The Defination Guide 2.0* URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/routing-value.html – [Online; Zugriff 25.02.2017]

[Ylp14] Yelp ElastAlert - Easy & Flexible Alerting With Elasticsearch (2014) URL: https://elastalert.readthedocs.io/en/latest/ – [Online; Zugriff 27.02.2017]

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24.05.2018 Biraj Dhungel