TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Database

Big-Data Solutions for Manufacturing Health Monitoring and Log Analytics

Big-Data Lösungen für die Überwachung und die Log-Analyse von Fertigungsanlagen

David Tiede

Day of Birth: 2nd July 1995 in Annaberg-Buchholz, Germany
Course: Master Informatik
Matriculation number: 4606589
Matriculation year: 2019

Master Thesis

to achieve the academic degree of

Master of Science (M.Sc.)

Supervisor
René Dienel

Supervising Professor
Prof. Dr.-Ing. habil. Dirk Habich

Submitted on: 23rd August 2022
Defended on: 5th September 2022

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Database

Task for the Preparation of a Master Thesis

Course:	Master Informatik
Name:	David Tiede
Matriculation number:	4606589
Matriculation year:	2019
Title:	Big-Data Solutions for Manufacturing Health Monitoring and Log Analytics

Modern semiconductor manufacturing is a complex interplay of machines, material, men, and methods. A multitude of software applications is necessary to keep this complex machinery operational 24/7. Monitoring this application landscape as well as communication and access patterns can provide important insights into the health status of the shop floor. In particular, equipment integration log data streams need to be constantly monitored to maximize production. Because of the high event rates in modern factories, big-data tools are required for scalable state and history analytics. However, the actual choice of suitable big-data solutions and their technical realization remains a challenging task.

In his thesis, Mr. Tiede shall explore big-data solutions for log-data ingest, enrichment and analytics. As an analytical use-case, he shall pursue distributed tracing of equipment integration logs in a semiconductor frontend. In particular, he shall study methods that consider a transactional context when analyzing equipment state and history.

First, Mr. Tiede will thoroughly mine the literature and assess big-data solutions (such as Datadog, Splunk, ELK, Loki) for scalable log ingest and analytics. He will define requirements for such a solution to cope with the scale, heterogeneity, and dynamics of complex discrete manufacturing processes. He will familiarize himself with the structure of equipment log data for selected process steps in wafer production. By working out an exploratory data analysis, he will shed some light on the transaction structure of the equipment log-streams.

Next, Mr. Tiede will set up a demonstrator to ingest logs at scale. The solution shall include a suitable enrichment workflow to reflect an application log model. To assess the applicability, he will evaluate the scalability of the solution. SYSTEMA provides a solution to generate log data and can provide large-scale log data of a manufacturing system.

Finally, Mr. Tiede will study how the solution copes with common industrial analytics use cases. This may include the search for alarm events, semantic search, time ranges, operation-context, or distributed tracing patterns. In particular of interest are interactive workflows and challenges when mining distributed tracing data using a big-data health analytics solution. Mr. Tiede shall then carefully evaluate and discuss the capabilities and challenges of the developed solution. If possible, he will operationalize the developed model in a demonstrator to study equipment states and history in a realistic production setting.

SYSTEMA will provide the necessary log data sources, formats, technical infrastructure, licenses, and details under a confidentiality agreement.

Supervisor:	René Dienel
Issued on:	27th January 2022
Due date for submission:	2nd September 2022

Prof. Dr.-Ing. habil. Dirk Habich
Supervising Professor

Acknowledgement

I would first like to thank my thesis advisor, Mr. René Dienel, and my supervising professor, Prof. Dr.-Ing. habil. Dirk Habich. Without their assistance and dedicated involvement in every step throughout the process, this thesis would not have been written. I would like to thank you very much for your support and understanding over these past seven months. They consistently allowed this paper to be my own work, but steered me in the right direction whenever they thought I needed it.

I would also like to thank the experts of SYSTEMA GmbH who were involved during the compilation of the requirements and use cases: Dr. Holger Brandl, Herbert Helmstreit, Viktor Bilousov and Olaf Schmiedgen. Without their passionate participation and input, this thesis could not have been successfully conducted.

At this point, I would also like to thank the management of SYSTEMA GmbH for the support through the Deutschlandstipendium in recent years, and for facilitating interesting activities like the visit to IT conferences. I enjoyed working here during the internships and writing this thesis.

A sincere thank you to Timothy J. Airaudi and Michael Peuß for their diligent proofreading of this thesis.

Finally, I must express my very profound gratitude to my parents and my fiancé for providing me with continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you.

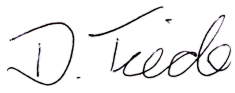
Statement of Authorship

I hereby certify that I have authored this document entitled *Big-Data Solutions for Manufacturing Health Monitoring and Log Analytics* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. During the preparation of this document I was only supported by the following persons:

René Dienel
Timothy J. Airaudi
Michael Peuß

Additional persons were not involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 23rd August 2022



David Tiede

Abstract

Modern semiconductor manufacturing is a complex process with a multitude of software applications. This application landscape has to be constantly monitored, since the communication and access patterns provide important insights. Because of the high event rates of the equipment log data stream in modern factories, big-data tools are required for scalable state and history analytics. The choice of suitable big-data solutions and their technical realization remains a challenging task.

This thesis compares big-data architectures and discovers solutions for log-data ingest, enrichment, analytics and visualization. Based on the use cases and requirements of developers working in this field, a comparison of a custom assembled stack and a complete solution is made. Since the complete stack is a preferable solution, Datadog, Grafana Loki and the Elastic 8 Stack are selected for a more detailed study. These three systems are implemented and compared based on the requirements. All three systems are well suited for big-data logging and fulfill most of the requirements, but show different capabilities when implemented and used.

Zusammenfassung

Die moderne Halbleiterfertigung ist ein komplexer Prozess mit einer Vielzahl von Softwareanwendungen. Diese Anwendungslandschaft muss ständig überwacht werden, da die Kommunikations- und Zugriffsmuster wichtige Erkenntnisse liefern. Aufgrund der hohen Ereignisraten des Logdatenstroms der Maschinen in modernen Fabriken werden Big-Data-Tools für skalierbare Zustands- und Verlaufsanalysen benötigt. Die Auswahl geeigneter Big-Data-Lösungen und deren technische Umsetzung ist eine anspruchsvolle Aufgabe.

Diese Arbeit vergleicht Big-Data-Architekturen und untersucht Lösungen für das Sammeln, Anreicherung, Analyse und Visualisierung von Log-Daten. Basierend auf den Use Cases und den Anforderungen von Entwicklern, die in diesem Bereich arbeiten, wird ein Vergleich zwischen einem individuell zusammengestellten Stack und einer Komplettlösung vorgenommen. Da die Komplettlösung vorteilhafter ist, werden Datadog, Grafana Loki und der Elastic 8 Stack für eine genauere Untersuchung ausgewählt. Diese drei Systeme werden auf der Grundlage der Anforderungen implementiert und verglichen. Alle drei Systeme eignen sich gut für Big-Data-Logging und erfüllen die meisten Anforderungen, zeigen aber unterschiedliche Fähigkeiten bei der Implementierung und Nutzung.

List of Abbreviations

AGPLv3	GNU Affero General Public License Version 3
API	Application Programming Interface
CA	Certificate authority
CEF	Common Event Format
CSFW	SYSTEMA Client Server Framework
CSV	Comma Separated Values
EI	Enterprise integration
ELK	Elasticsearch, Logstash und Kibana
GB	Gigabyte
GEM	Generic Model for Communications & Control of Manufacturing Equipment
GHz	Gigahertz
HDFS	Hadoop Distributed File System
HSMS	High-Speed SECS Message Services
ICALEPCS	International Conference on Accelerator and Large Experimental Physics Control Systems
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
KQL	Kibana Query Language
LogQL	Logging Query Language
MB	Megabyte
MCS	Material Control System

MSG	Message
NCSA	National Center for Supercomputing Applications
NVMe	Non-Volatile Memory Express
PID	Process identifier
PRI	Priorität
PromQL	Prometheus Query Language
RAM	Random-access memory
regex	Regular expression
REST	Representational state transfer
RF	Functional requirement
RS	System requirement
RQ	Quality requirement
SaaS	Software as a service
SECS	SEMI Equipment Communication Standard
SEMI	Semiconductor Equipment and Materials International
SLA	Service Level Agreement
SQL	Structured Query Language
SSD	Solid State Drive
Syslog	System Logging Protocol
SYSTEMA	Systementwicklung Dipl.-Inf. Manfred Austen GmbH
TCP	Transmission Control Protocol
TICK	Telegraf- InfluxDB- Chronograf- Kapacitor
TLS	Transport Layer Security
UC	Use Case
URL	Uniform Resource Locator
XML	Extensible Markup Language

Contents

Chapter 1

Introduction

"Without big-data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway."
– *Geoffrey Moore, management consultant and author of "Crossing the Chasm"* [50]

1.1 Motivation

Production companies these days are becoming more and more automated by machines and robots [129]. With the increasing use of machines and robots, increases the amount of sensors and sensing elements required. And with a growing amount of sensors, more log data is generated in a company. In the field of factory automation, it is advantageous to have log data from a production plant available [103].

But this log data is not of any use, if it cannot be processed, visualized, evaluated and used to optimize the production process. A simple log viewer does not have the option to filter, visualize or even combine logs from multiple sources. Problem-solving with such an environment can be hard and time-consuming. Developers, which need to analyze the logs to find the cause of an anomaly, tend to write their own program for their specific use case. This is the reason for narrow software solutions, which only cover a few use cases.

To avoid spending time in the development of narrow software systems and investing additional time in optimizing production plants, a system that covers many use cases arises. This thesis has the goal to find a suitable solution for a wide range of use cases in big-data logging.

1.2 Structure

The following chapter goes into some basics of logging, like log level and well-known log structures in semiconductor production. It continues with the origin, scope and size of log data in this industry. Software developers used examples to explain how they use logging tools in their daily work. Based on these use cases, the requirements for a big-data logging tool were worked out.

Chapter 3 looks deeper into five papers presenting a logging or monitoring architecture for IoT devices or other big-data solutions. Since the architecture can be split up into multiple components with different functions, *Components for one layer* collects possible tools, which cover a part of a potential software stack. The *Software solutions for the stack* present solutions, which cover the whole process: collecting, analyzing, storing and visualizing.

Analysis and Implementation discusses the benefits and drawbacks of a full stack in comparison to a custom assembled stack. Based on the presented stacks in chapter 3 Datadog, the Grafana Loki stack and the Elastic 8 stack are chosen to be implemented and evaluated in more detail.

In chapter 5 a comparison of the implemented solution is done. The first part is the comparison based on the levels in the horizontal structure, and the second part is based on the requirements. Finally, this work summarizes the findings and gives an outlook of further possible areas to be examined.

Chapter 2

Fundamentals and Prerequisites

The chapter starts with logging fundamentals, followed by the description of two custom log structures. Afterward, requirements and use cases for a big-data logging system are collected, to lay the foundation for the comparison done in this work.

2.1 Logging

Software engineers often use logs to keep track of important run-time data about their systems. The logging lines are included in the source code to generate the log data [49]. In manufacturing health monitoring, logging messages could be generated by sensors that observe the internal states in the system during production. The sensor log statements are then written into a log file, collecting the sensor information.

A logging statement typically includes a log level, a static text, and one or more variables [130, 71], and it is often presented in a format, easily readable for humans [126]. But the textual format for humans is often difficult to process by machines. Especially when it comes to further automated processing, indexing and analyzing, the log line needs to be structured.

Structured logging is the technique of enforcing a consistent, predetermined message format, which is machine-readable and can be easily parsed [126]. XML, JSON, the Common Event Format or the NCSA Common Log Format are a few formats used for structured logging [74, 125].

2.1.1 Log level

Structured logging helps to identify the log level of a log statement. Debug, info, warn, error, alert or fatal/emergency are the six log levels supported by most logging libraries. The log levels are sorted by the level of verbosity of a logged event, with “trace” being the most verbose and “fatal/emergency” being the least verbose [71].

Value	Severity	Keyword	Description
0	Emergency	`emerg`	System is unusable
1	Alert	`alert`	Action must be taken immediately
2	Critical	`crit`	Critical conditions
3	Error	`err`	Error conditions
4	Warning	`warning`	Warning conditions
5	Notice	`notice`	Normal but significant conditions
6	Informational	`info`	Informational messages
7	Debug	`debug`	Debug-level messages

Table 2.1: Syslog severity levels [51]

Syslog stands for System Logging Protocol and is a standard protocol used to send system log or event messages to a server. It is primarily used to collect disparate device logs from multiple different computers in a central location for monitoring and control purposes [111]. Syslog is defined in RFC 5424 [51], the Syslog protocol, which supersedes the earlier version, RFC 3164 [84]. A Syslog message consists of three parts: PRI (a calculated Priority value), HEADER (containing identifying information), and MSG (the actual message). The Priority value is an integer whose binary representation can be broken down into two parts: the Facility field and the Severity field. The Facility field describes the type of system the message was created

on. The Severity field, comprising the last three bits of the Priority, contains a numeric value between 0 and 7, with 0 being the most critical or urgent level as shown in Table 2.1 [111].

Eric Allman created Syslog as part of the Sendmail project in the 1980s [40]. Other applications quickly adopted it, and it has since become the standard logging option for Unix-like systems [2]. Currently, implementations exist for other operating systems as well and the log level scale is used by various devices and software solutions [94]. Nevertheless, you can find log scales using Syslog as a basis, with not all severities used or changed to their needs. The CSFW log is one example of a strong adaptation of the log level.

2.1.2 CSFW log

The Client Server Framework Log was developed by SYSTEMA and offers a way of structuring log lines. The individual parts are separated from each other by hyphens. At the beginning of the log line is the timestamp of the entry. This is followed by the corresponding process identifier (PID). The next part, the log level, has level six with the Syslog scale in common, but the rest was modified as shown in Table 2.2. The next two parts indicate the module name and the class name, which created the entry. These parts are followed by the content of the log message. The content could be plain text or another structured format like JSON or XML. Listing 2.1 shows how CSFW log entries look like in production.

Value	Keyword	Description
0	FATAL	Fatal
1	ERROR	Error
2	WARNING	Warning
3	ALWAYS	Always
4	COM	Communication
5	STEP	Step
6	INFO	Informational
7	METHOD	Method

Table 2.2: CSFW log levels

```
1 2022.04.06 13:20:04.895  - PID: 10407       - LEVEL: 3 ALWAYS    -
     database              - DSysDBRequester            -
     reInitConnection(): Try to initialize JDBC connection...
2 2022.04.06 13:20:23.666  - PID: 10407       - LEVEL: 7 METHOD    - bus
                      - DSysBus                    - <<
     readName(ESysConfigItem)
3 2022.04.06 13:41:23.932  - PID: 10407       - LEVEL: 7 METHOD    - bus
                      - DSysMsgToDataFieldConve  - >> toWireFormat
4 2022.04.06 13:41:23.933  - PID: 10407       - LEVEL: 4 COM       - bus
                      - DSysMsgToDataFieldConve  - Sending message on
     subject "CalculationProcess.Base.SystemControl"
5 2022.04.06 13:41:23.933  - PID: 10407       - LEVEL: 7 METHOD    - bus
                      - DSysMsgToDataFieldConve  - << toWireFormat
```

Listing 2.1: Example of CSFW log entries

2.1.3 SECS log

The SEMI Equipment Communication Standard (SECS) is a communication standard for computers and devices in the manufacturing industry published by Semiconductor Equipment and Materials International (SEMI). When speaking of SECS, GEM (Generic Model For Communications and Control Of Manufacturing Equipment) is often used in the same vein [68][106][107]. As the name SECS/GEM already suggests, it is actually a combination of several standards. The levels build on each other and provide functions for the next higher level [106]. With SECS/GEM, three levels are usually used. Each level is defined in its own SEMI standard, as seen in Figure 2.1.

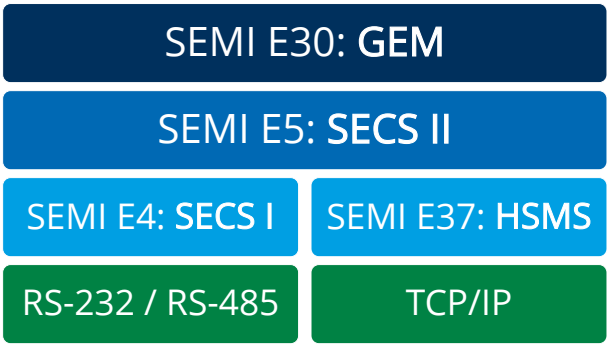


Figure 2.1: SECS/GEM communication and protocol stack

SEMI E30 GEM

The GEM standard defines the machine states and business rules. It also specifies guidelines such as how and when SECS II messages should be used, as well as what the resulting activity should be. It also specifies features such as Status Data Collections, Trace Data Collections, Alarms Management, Spooling, Remote Command, and so on [68].

SEMI E5 SECS II

This is the most significant standard for this work, because it describes how a SECS message, which is also written to the log, is structured. The API describes the interface between the host and the equipment, and it depicts the structure of the messages. Each message is named by "Function" and "Stream" (category) [68].

The SECS II message is split up into header and body, which can be seen in Figure 2.2. The header contains the function name and metadata about the transaction. The body could be empty for some functions, but normally it includes a description, a structure with the function's message body and exceptions. The function's message body is structured in lists and items. Each item is numbered on a new line and can be grouped in nested lists as well [68].

SEMI E37 HSMS

The High-Speed SECS Message Services (HSMS) is currently the primary transport protocol standard in the SECS/GEM stack and is built on top of the TCP/IP protocol. It supersedes the "SEMI E4 SECS I" standard, which was built for serial communication and did not allow higher speed or a variety of platforms [108].

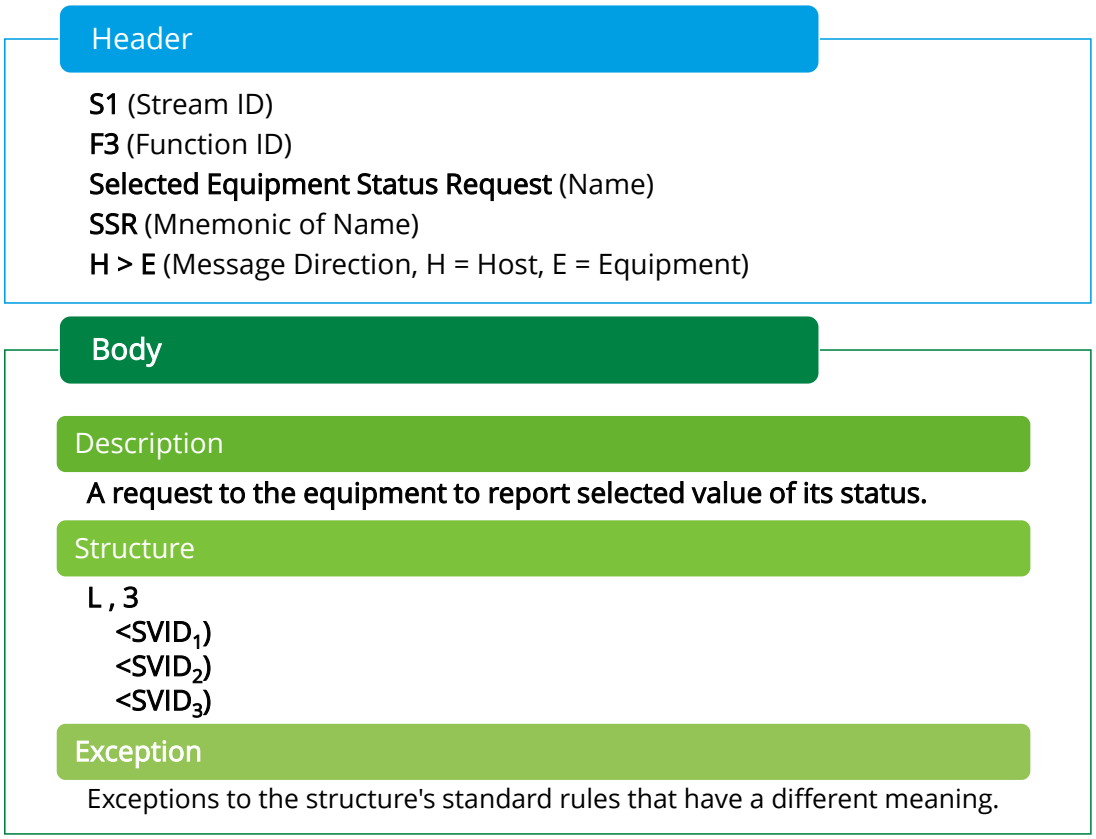


Figure 2.2: SEMI E5 SECS II message structure and example

Chosen parts of the SECS message are written with the timestamp into the log file. There appears to be no standard on how this is done, since you can find differently structured log entries, as shown in Listing 2.2 *Example of two different SECS log entries*.

```
16.04.2022 02:19:00 : H <-- E [0x3467ADAD]
S1F4   :  Selected Equipment Status Request
<L [4]
   <U1 73 >
   <B 5 >
   <A >
   <A >
>.

04Apr22 01:09:33.531  S02,F35 (OUT) [DEV 0x0000] [SYS 0x130059B8] [W 1]
<L 2
  <U1 1, 0>
  <L 0>
>
```

Listing 2.2: Example of two different SECS log entries

Besides the two log format examples listed above, there are many other formats that can be used, because of personal needs and requirements.

2.2 Existing system and data

This work uses logs from the production environment of semiconductor manufacturing plants. The next section gives a brief overview of the manufacturing process, followed by a few numbers on the log data generated in these systems.

2.2.1 Production process

The production process of computer chips can be split up into three phases, which are often done in different companies:

Poly silicon to wafers Today, computer chips are produced mainly from semiconductor materials, such as doped silicon. Semiconductor material blocks, also called ingots, can be made using a variety of techniques. The goal is to get as pure an ingot as possible by removing impurities. After the production of the ingot, a wafer is created by cutting the ingot into slices.

Wafer to chips Usually, several wafers are bundled into one lot. The wafers in a lot are then processed simultaneously or directly one after the other. Step by step, the wafer is then processed into up to ten thousand computer chips [117]. Normally, the manufacturing process from a wafer to computer chips takes around two months, but it could take up to six months as well. The time of this process depends on the complexity of the chips, and additional quality checks and touch-ups increase the production time. The log data for this thesis is produced in this production phase.

Chips to device In the last step, the chips are combined with devices or to embedded systems, which is a computer hardware device with software that is designed to execute a specific function, either standalone or as part of a larger system [123].

2.2.2 Log data in numbers

To get an overview of the amount of log data that must ultimately be processed by a system, the numbers of a semiconductor fabrication company with a monthly capacity of around 70,000 wafers are presented. To systematically control materials over the stages of procurement and storage, and to facilitate the uninterrupted and continuous flow of materials through the production pipeline, a Material Control System (MCS) is used [113]. Eight application servers are used to run the MCS and the system produces around 15 to 17 million CSFW log lines using five to seven Gigabytes of storage per machine per day, and a factory can contain several machines.

Depending on the size and the equipment in another semiconductor company, the log amount varies. Statistics of their production line show around 100 SECS log entries per minute, which can add up to ca. 150,000 entries a day using around 50 Megabytes. Especially for smaller companies and production lines lacking sensors on the equipment, the amount could be just a tenth as well.[1]

[1] The statistics are from various partners of SYSTEMA GmbH.

2.3 Requirements

The requirements for a possible system to handle this amount of data are categorized into functional, system and quality requirements. For a better evaluation of the various solutions, the requirements are evaluated using the MoSCoW method [122], one of the most cited and utilized methods of software requirements prioritization according to [3]. The MoSCoW method classifies the requirements into M must have, S should have, C could have and W won't have.

M — Must have These requirements are conditions that must be met in the final product. They are a non-negotiable necessity, and the project will fail without them.

S — Should have These features are important but not vital, and they do not need to be released right away. However, it is regarded as significant and valuable to users, and such demands are ranked second on the priority list.

C — Could have These requirements are desired but not an obligatory necessity. If the project's timelines are threatened, this point will be eliminated first from scope, according to the approach.

W — Won't have A need that will not be implemented in the current version, but could be added in the future. Typically, such requirements have no bearing on the project's success [20].

The following requirements were classified according to my suggestion and subsequent discussion in the SYSTEMA Enterprise Integration Analytics project team.

2.3.1 Functional requirements

The functional requirements are categorized by the major steps of a log system, because the software solutions are often split up into decoupled components for each step, and sometimes it is possible to exchange one component in the software stack. Normally a software stack begins with the ingestion of the log lines, continues with the analysis and the indexing, and finally visualizes the log data.

Requirements for collection

RF11 Extendable file parser M

It should be possible to handle other logging file formats and extract attributes out of the data.

RF12 SECS parser and CSFW parser C

There are multiple different logging formats. Two of them should be supported by the system. The first format is called SEMI Equipment Communication Standard (SECS) and the second format is called SYSTEMA Client Server Framework Log Format. This format needs a parser for a complex log with attached structured data in the XML format.

RF13 Enrichment S

Parsed attributes should be enriched with additional data from a database system or another external system.

RF14 Combine data from multiple sources S

A solution should be able to join the data from multiple logs and multiple hosts. This includes that data needs to be pushed to a central point, since the location of the clients is not known every time.

RF15 Ingest log data later in bulk S

In some cases, there should be no direct connection of the logging equipment to a logging software solution. In such cases, the log data is transferred as an archive and processed and analyzed at a later time.

RF16 Pseudonymization S

The parser should have an option to replace personally identifiable information within the log data by artificial identifiers or pseudonyms.

RF17 Load of log collection client S

The client for the log collection runs normally in a production environment. Therefore, the client should not put a heavy load on the host system.

Requirements for analysis

RF21 Filter by attributes M

The software should allow filtering of the logs by defined attributes like host, log level, and content.

RF22 Full-text search M

A full-text search over the ingested log data should be possible.

RF23 Sorting by time M

The system should be able to sort the log based on a timestamp.

RF24 Auto-completion S

When writing queries, suitable suggestions should be shown and auto-completion should be possible.

RF25 API for Visualization M

A solution should provide an interface, which does not need to be public, to visualize the data in a dashboard.

RF26 Query API C

An interface like SQL would be great to query the indexed logs.

RF27 Anomaly detection C

Anomaly detection or outlier detection should help to identify rare events or observations which deviate significantly from the majority of the log data and do not conform to the normal behavior.

Requirements for visualization

RF31 Table view M

Logs should be able to be viewed in a table sorted by selected attributes like time.

RF32 Histogram S

A solution should be able to show the quantified data in a histogram for a few minutes and up to one month.

RF33 Scatter plots S

The system should be able to show the logs in scatter plots like production time based on the day of the week.

RF34 Grouping sequences C

Repeating sequences in the log files should be able to be grouped.

RF35 Real-time log tail C

The dashboard should have a view showing the produced logs in real-time. A time less than one hour fulfills this requirement.

RF36 Mark log lines C

The visual interface could provide an option to mark log lines with a specific pattern. It would be ideal if the marks could be shown in different background colors of the log line.

RF37 Comment log lines C

The visual interface could provide an option for multiple users to comment on a log line and to show the comment history linked to it.

RF38 Export selected original log data S

An option to export selected log data in various formats would be useful. Export should include the export of the original log data and the parsed log data in formats like JSON.

RF39 Sequence diagram W

Grouped sequences should be shown in a sequence diagram.

Commercial requirements

RF41 Usage of existing solution M

Prefer existing or commercial tools and avoid the development of a new solution.

RF42 User Management M

Different user roles with different rights should be possible. It is necessary to differentiate between users for managing the system and creating visualizations and users, who can only view the dashboards.

RF43 Minimal IT support S

Operation and Maintenance of the logging system should be possible with minimal support by IT.

RF44 No publishing of source code M

The license of any system used should not include the release of the source code as a condition.

2.3.2 System requirements

The system requirements are criteria for an appropriate set of resources to satisfy a system [53]. Every software system needs a different set of resources, and no two systems have identical requirements [13]. At this point, a specific system is not selected. Since the various software stacks have different system requirements, there are only a few essentials a possible software stack should fulfill.

RS01 Hard disk space M

The system should be able to store the indexed log data, which size may depend on the used system.

RS02 Secure transmission M

The transport of log data has to be transferred via a secure channel like TLS.

2.3.3 Quality requirements

Advertisers would like us to believe that all businesses make high-quality goods. The term "quality" has become so overused that it has lost its meaning for many people. And quite often the customer defines what product has a high-quality [59]. The following requirements are based on the potential expectation of customers. They are split into availability requirements, which include the storing time, and performance requirements, concentrating on scalability.

Availability

RQ11 High availability S

The service downtime should be less than 9 hours per year, which equals an availability percentage of 99.9%.

RQ12 Storing of full data S

The visualization of the log data should be able to show 14 days, including all log levels.

RQ13 Storing of filtered data S

After 14 days, log data should be able to get filtered to save storage. The filter should keep log lines with a specific log level or content.

RQ14 Archived data S

The aggregated logs should be saved for the make span of the product. This is typically two to six months in the semiconductor industry.

RQ15 On-premises and Cloud S

A solution that can be deployed on-premises and in the Cloud is preferred.

Performance

RQ21 Scalability M

The system should be able to handle an increase in workload without a significant performance degradation or the ability to quickly enlarge the system.

RQ22 Volume of indexed information S

The indexed information should not take more than double the original log data.

RQ23 Real-time update interval C

The view of the log tail should show incoming log lines within 60 seconds.

2.4 Use Cases

The following use cases are intended to illustrate a specific system behavior the stakeholder wants to use the system for. To concentrate on the most significant features when picking the system, only the most important use cases were summarized, but many other Uses Cases do exist.

2.4.1 Finding specific communication sequence

Operators of production plants notice irregularities or errors in production and want to find out the cause. A few parameters like time or a product identification number for the anomaly are known. Based on very little information, analysts try to find a specific communication sequence in the log. To understand and analyze the log, the filtering of unnecessary log lines and grouping of similar events is essential. A visualization could speed up the process of analyzing, because reoccurring patterns could be seen quicker.

Use Case	Finding specific communication sequence
ID	UC01
Actor	Software Developer
Trigger	Anomaly in the production process
Pre-condition	Log data and few known parameters of anomaly like time and product
Normal flow	Filtering, grouping and visualizing the log data to find specific communication sequence
Post-condition	Known cause of the anomaly

Table 2.3: UC01 Finding specific communication sequence

2.4.2 Watching system changes

After system updates it should be possible to determine whether the system continues to run as before, whether new services are called, new conflicts arise or the utilization of production lines changes. This requires the selection of specific periods of time to compare the period before and after the system change.

Use Case	Watching system changes
ID	UC02
Actor	Integration Software Engineer
Trigger	Software updates or software changes
Pre-condition	Time for system change and log data before and after the system change
Normal flow	Selection of the time periods before and after the software change to view chosen parameters and compare them
Post-condition	Discovery of differences in the system before and after the change like new called services, new conflicts or changed performance

Table 2.4: UC02 Watching system changes

2.4.3 Comparison with expected production path

In complex production lines, it is useful to see the production path of individual products. This would allow comparing the used production path to the expected production path. This method helps to validate, whether the software is working as planned and expected.

Use Case	Comparison with expected production path
ID	UC03
Actor	Integration Tester
Trigger	Fault in produced product
Pre-condition	Log data of production process
Normal flow	Filtering of the log data according to the specific product and ordering according to time
Post-condition	Finding any deviations from the normal production process

Table 2.5: UC03 Comparison with expected production path

2.4.4 Enrichment with metadata

Log data of production lines consists of products with unique identifiers. To find solutions to occurring issues, it is helpful to have additional data shown to the product identifier, such as metadata like carrier or intended production process. This additional metadata is stored in external databases or can be queried via SQL.

Use Case	Enrichment with metadata
ID	UC04
Actor	Production Manager
Trigger	Need for additional data for a specific product
Pre-condition	Log data and system with additional product data
Normal flow	Log line is enriched with metadata
Post-condition	Showing the additional metadata in the dashboard

Table 2.6: UC04 Enrichment with metadata

2.4.5 Decoupled log analysis

In some cases, the log data is available as an archive or excerpt from a file and is not integrated into a running system. In this case, the log data must be able to be analyzed separately and decoupled from a productive system. The use cases mentioned before could all occur as decoupled log analysis.

Use Case	Decoupled log analysis
ID	UC05
Actor	Integration Software Engineer
Trigger	Submission of log data for analysis
Pre-condition	File with log data
Normal flow	Import of the log data into the decoupled system for the desired analysis
Post-condition	Finished analysis with the gained insight

Table 2.7: UC05 Decoupled log analysis

Chapter 3

State of the Art and Potential Software Stacks

The following chapter is about the resources available regarding big-data logging. This includes various papers describing software stacks used in big-data and IoT systems. The second section gives an overview of potential software solutions for big-data logging.

3.1 State of the art software stacks

A specific search for papers on semiconductor logging architectures yields only a few results. But when searching for IoT logging architectures, literature and comparable works can be found in numbers. This section examines some of these papers to find commonalities in the architecture of big-data logging systems.

3.1.1 IoT flow monitoring system

The paper [19] describes a flow monitoring system for IoT (Internet of Things) networks, as illustrated in Figure 3.1. The author describes the typical architecture of flow monitoring in four steps. It starts with the *Packet Observation* to capture the packets from the sensors and pre-process them for further use. The second step is the *Flow Metering & Export*, "where packets are aggregated into flows and flow records are exported." This is followed by *Data Collection*, which "receive, store and pre-process flow data from one or more flow exporters." The final step is *Data Analysis*. The author uses Logstash [83] of the Elastic Stack for Data Collection and Elasticsearch [38] and Kibana [70] for the Data Analysis.

Figure 3.1: Workflow for flow monitoring on IoT networks [19]

3.1.2 Big-Data IoT monitoring system

The authors of [105] describe a "parallel data processing system for IoT security monitoring" which is intended for large amounts of security events, that can be classified as big-data, from terminal devices and network infrastructure. The security events are stored in log files and the system is implemented in an Apache™ Hadoop® environment, which is a framework for distributed processing of large data sets across a computer cluster [8].

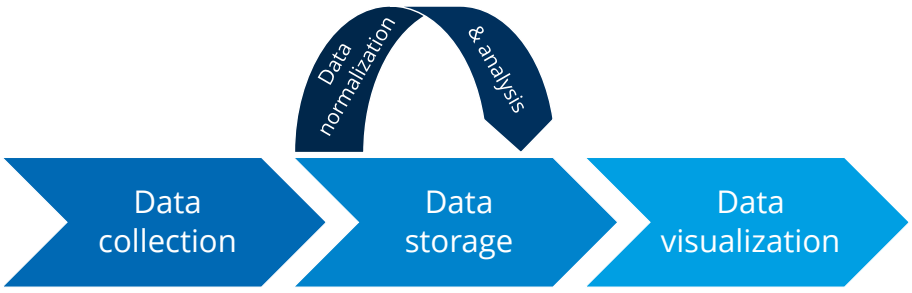


Figure 3.2: System architecture for big-data security monitoring of IoT Networks [105]

It consists of a *data collection* component, which collects the security events from different sources and forwards them to secure *data storage*. The data storage is implemented as a Hadoop Distributed File System (HDSF). The component for *data normalization & analysis* converts the data of the storage into the unified format CSV (Comma Separated Values) and performs preprocessing and filtering. During the analysis, it runs functions of aggregation, like extremes or means values, and correlation, like abnormal effects, and transmits the results

back to the HDSF. The *data visualization* component allows the visual analysis and presents it in histograms, pie charts and linear graphics. This whole process is shown in Figure 3.2.

3.1.3 IoT Cloud Computing Stack

The IoT Log Collection Stack is presented based on [12] in [95]. The stack is divided into six layers, where the lowest layer is the hardware, which is not shown in Figure 3.3 to be better comparable to the other stacks. The *Event layer* is the part responsible for generating events and formatting it into the key-value pair structure JSON. Different protocols are then being used in the *Protocol layer* to carry the data out of the embedded devices into the cloud.

The *Cloud layer* saves the incoming log data into the Cloud storage and checks the integrity of the data. The *Application layer* is a software stack that includes tools, programs, and scripts for extracting relevant data from the cloud storage and aligning it along a timeline. In addition, there are instruments to help investigators with forensic analysis. The *Presentation layer* is the system's human interface, giving capabilities for visualizing and displaying data.

Figure 3.3: IoT Cloud Log Collection Stack [95]

3.1.4 Big-Data Logging Architecture

The conference proceedings [24] talk about three layers in a logging and monitoring system software stack. The first layer is the *Collector*, which gathers the messages from the devices and sends them to an application for processing. *Analysis* application is the second layer, which is consuming and processing the data streams. The data processor then stores the data in a database. The last layer is *Visualization*, and this layer gets the data from the database and pushes it to a responsive web dashboard. This document presents the architecture, as seen in Figure 3.4, but does not mention a concrete application system supporting this architecture.

Figure 3.4: Big-Data Architectures for Logging and Monitoring [24]

3.1.5 IoT Energy Conservation System

Another concept for an Energy Conservation System based on IoT Technology splits up the stack similarly, and is illustrated in Figure 3.5. Their system design consists of *Recording*, *Computing* and *Visualizing*. The recording includes data perception and information transmission and storage. Computing concentrates on automatic control and energy use strategy recommendations. The last layer is for the visual result presentation. They used Autodesk Revit® [102] for storing the model, and they integrated their own implementation [128].

Figure 3.5: Concept of an Energy Conservation System [128]

3.1.6 Similarities of the architectures

The architectures in the last sections were built for different purposes, but they all have some aspects in common. In the beginning, they collect data from various sensors and logs to store them in a central place. The data is then processed by applications, which filter, enrich or analyze the data, so it can be presented and used to gain insight into the monitored system.

Table 3.1 shows the similarities of the described architectures. The architecture described in subsection 3.1.4 only has different wording for the various layers. In subsection 3.1.5 the first layer *Recording* includes the storage as well, but besides that, it is similar to subsection 3.1.4. The subsection 3.1.3 and subsection 3.1.2 divide the *Analysis & Storage* layer into the storage and the analysis or application part. *IoT flow monitoring system* sets different priorities for the architecture. The *Collection* process is split up into *Packet Observation* and *Flow Metering*. Even though the layer name *Data Collection* would fit better into *Collection*, the task of the layer is receiving, storing and pre-processing [19], which fits more into the *Analysis & Storage* layer. Since Kibana is used for presenting the statistics in their implementation, *Visualization* is part of their *Data Analysis* layer.

Stack	Collection	Analysis & Storage	Visualization
1 IoT flow monitoring system [19]	Packet Observation Flow Metering	Data Collection Data Analysis	Data Analysis
2 Big-Data IoT monitoring system [105]	Data collection	Data storage, normalization and analysis	Data visualization
3 IoT Cloud Computing Stack [95]	Event layer Protocol layer	Cloud layer Application layer	Presentation layer
4 Big-Data Logging Architecture [24]	Collector	Analysis	Visualization
5 IoT Energy Conservation System [128]	Recording	Recording Computing	Visualizing

Table 3.1: Similarities of the architectures

For the remainder of the work in this thesis, the stack *Collection*, *Analysis & Storage* and *Visualization* is used. *Collection* is the part, which collects the data at the client and sends the data to a central device. Sometimes pre-processing is done at this stage. The *Analysis & Storage* is the stage, where the processing is done, and the data is stored. This stage could most likely be split up, into the analysis part and the storage part. Since the processed data needs to be stored, these parts are combined in this thesis. The *Visualization* layer is for presenting the processed data in dashboards to the user.

3.2 Selection of software stack

Looking for a software system that covers the *Requirements*, you can find software systems, which cover only a part of the stack and systems which cover all three parts of the stack. To get a better comparison, this document is looking at both possibilities.

3.2.1 Components for one layer

The process starts with the collection of the log, followed by the analyses and ends with the visualization of the log data. This order is used to present possible components.

Collection

Fluentd One of the first results looking for log data collection is the open source tool *Fluentd* developed by Treasure Data. Fluentd attempts to format data as JSON, which allows it to unify log processing, including collecting, filtering, buffering, and exporting logs across different sources and destinations [124].

Fluent Bit Treasure Data also developed *Fluent Bit* [47], an open-source log collection and processor, in 2015. Fluent Bit, which was written in C, was designed for a specific use case: highly distributed systems with limited capacity and little overhead [48].

Apache Kafka Kafka [9] is an open-source distributed store and stream-processing platform. The goal of the project is to provide a unified, high-throughput, low-latency platform for real-time data flows. In comparison to other systems, Kafka abstracts away the files and rather wants log or event data as a stream of messages. To collect the data, Kafka has two options: Clients are based on a Kafka library for direct message streaming or Kafka Connect, a tool that connects to other systems like databases or application servers [10].

Analysis and Storage

Depending on the solution for this layer, the focus lies more on the analysis or the storage, but normally they go together.

Prometheus The open-source software Prometheus [98] is used for event monitoring and alerting, but it was primarily designed for metrics, not logs. The records are pulled via HTTP and saved in a time series database. Since there is no option to push log data into Prometheus, it is not a good solution, since data might get lost if not pulled in time [92].

Grafana Loki This solution is often described as "Like Prometheus, but for Logs" [67]. Loki is described as a log aggregation system designed to store and query logs from all applications. It does not index the content of the log lines, but only indexes the metadata, which saves a lot of data [56].

MongoDB The system MongoDB [90] is classified as a cross-platform document-oriented NoSQL database. Functions like log aggregation, indexing and full-text search make it interesting for big-data logs. From the beginning, MongoDB is based on a scale-out design, which enables many tiny machines to collaborate to construct fast systems that can manage large volumes of data [127].

Apache Solr™ Solr [121] is a Java-based open-source search platform. It supports full-text search, real-time indexing and dynamic clustering. It provides an HTTP API to query it and get JSON, XML, CSV, or binary results [42]. Solr is currently one of the top three most popular enterprise search engines [30].

Visualization

Grafana The web-based analytics and interactive visualization program is one of the most popular metrics tools [7]. When connected to supported data sources, it produces web-based charts, graphs, and alerts [54, 58]. Prometheus, InfluxDB and other time series databases are frequently used in conjunction with it [100].

Cyclotron This open-source platform was developed for creating and hosting dashboards of metrics in time series. Cyclotron allows data visualization, text and markdown editing and table calculation, when it is connected to one of the supported databases like Prometheus, InfluxDB or Elasticsearch [25].

3.2.2 Software solutions for the stack

In contrast to the chapter before, the following solutions try to provide suitable tools for all stages, which are adjusted to each other.

Elastic Stack

The first release of *Elasticsearch* was published in 2010 [112]. Since then, it has become one of the most followed and used projects on GitHub [101] and the most popular enterprise search engine [30]. In 2012 the company *Elastic NV*, previously named Elasticsearch, was founded to provide commercial products and services around the Elasticsearch software [36].

The Elastic Stack, at the beginning called ELK-Stack, started with the three open source projects *Elasticsearch*, *Logstash*, and *Kibana*. As mentioned before, *Elasticsearch* [38] is a search and analytics engine with an HTTP web interface and schema-free JSON documents. *Logstash* [83] is a server-side data processing pipeline that simultaneously ingests data from numerous sources, processes it, and transfers it to a storage place or analytics engine. *Kibana* [70] allows users to visualize data using charts and graphs [116].

In 2015 with Elasticsearch 2, *Beats* [14], a set of lightweight, single-purpose data shippers, were added to the stack. These shippers exist for different kinds of data: files, metric data, event logs, network data and more. The Beats completed the stack seen in Figure 3.15.

Figure 3.15: Elastic Stack

In August 2021, Elastic announced the *Elastic Agent* as one unified agent for all kinds of data sources and agents spread across multiple networks [110]. Even though Elastic states "you can continue to use [Beats] alongside Elastic Agent" [110], since Elasticsearch 8 they are pushing the Agent in favor of the Beats [88], and Beats and Logstash are not listed on the product overview anymore [35].

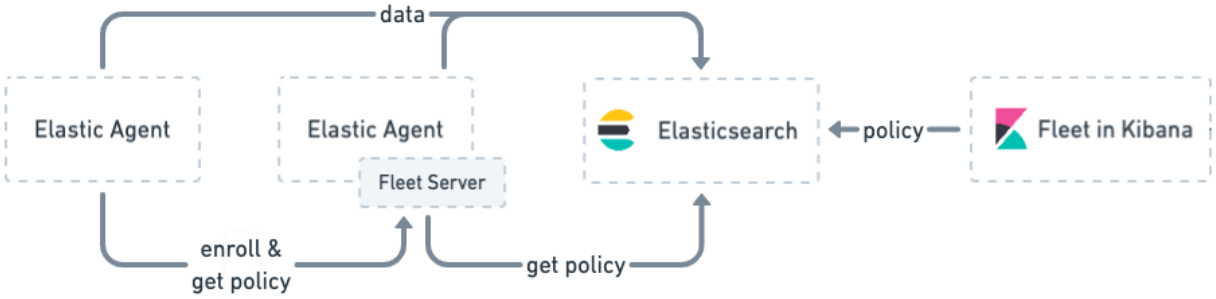


Figure 3.16: Elastic 8 Stack with Elastic Agent and Fleet Server [46]

This thesis concentrates on the Elastic 8 Stack with the Elastic Agent and the Fleet Server, as shown in Figure 3.16, since this seems to be the future of the Elastic Stack.

Amazon OpenSearch

With Elasticsearch and Kibana version 7.11, Elastic changed the Apache 2.0 licensed code to the Server Side Public License, which is not considered an Open Source license [120]. With the support of Red Hat, SAP and Capital One, Amazon introduced the forks *OpenSearch* and *OpenSearch Dashboards* under the Apache License in April 2021 [86].

A production-ready version of OpenSearch and OpenSearch Dashboards was published in July 2021 [45, 60]. Beginning with Beats 7.13, the connections to previous versions of Elasticsearch and Kibana, and therefore OpenSearch are not supported anymore. But Beats can be connected to Logstash, and Logstash can send the data via the OpenSearch output plugin to OpenSearch, as illustrated in Figure 3.17 [16, 5].

Figure 3.17: OpenSearch Stack [15]

Grafana Loki Stack

In the last section, *Grafana Loki* and *Grafana* have been introduced as tools that cover part of the stack. Even though they are used in a stack with various tools, GrafanaLabs recommends the Grafan Loki stack shown in Figure 3.18. The setup of Loki is always done with *Promtail* [99] in the official Grafana Labs documentation [65]. Promtail is the log collector specifically built for Loki. Loki then indexes the log with labels and provides the query language *LogQL*, which can be directly run in Grafana to visualize the data [56].

Figure 3.18: Grafana Loki Stack

TICK Stack

TICK was developed for time series analysis by the company InfluxData [64]. The main component *InfluxDB* is an open-source database, which is available as a cloud service, and it could be installed on-premise as well. Application metrics and IoT sensor data can be pushed via Agent *Telegraf* [114] into the database. For the processing and the anomaly detection, InfluxData provides the tool *Kapacitor* [69]. The visualization can be done directly in InfluxDB, but the company provides an open-source web application [21] for monitoring and creating alert rules as well. The whole stack is illustrated in Figure 3.19.

Figure 3.19: TICK Stack [34]

Datadog

Datadog is a cloud application software, which allows monitoring of servers, databases, tools and services via a software-as-a-service (SaaS) based data analytics platform, which runs only in the cloud. It specializes in companies, that need analysis and monitoring services. Figure 3.20 summarizes the various services by Datadog. To collect data from various platforms, Datadog uses the Agent [22]. The Datadog Agent is an open source tool, that runs on your hosts and gathers events and metrics, and delivers them to Datadog for analysis [4].

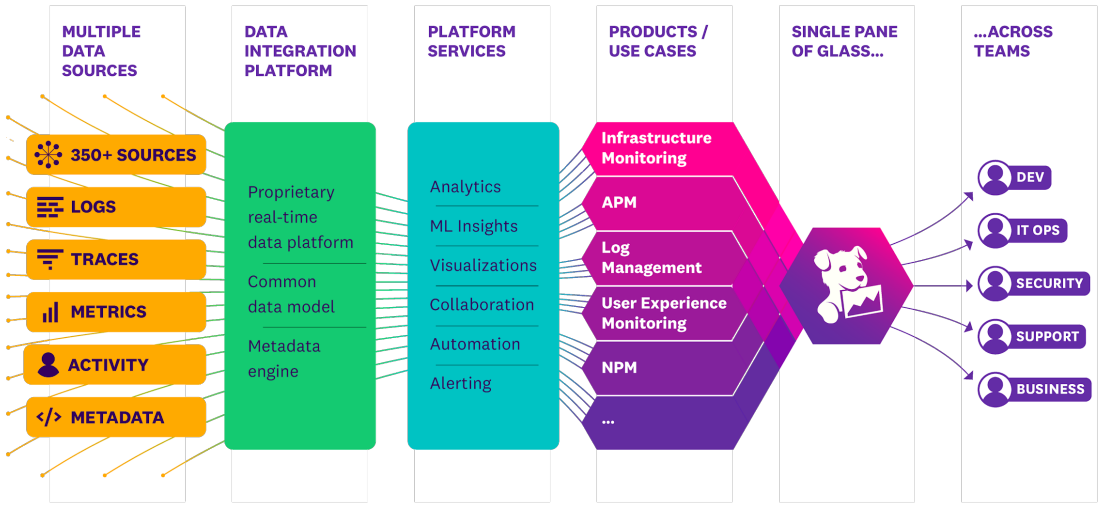


Figure 3.20: Datadog platform [26]

Mezmo previously LogDNA

During writing this thesis, *LogDNA* [75] was rebranded as *Mezmo* [72, 18]. Mezmo is a SaaS-based log management tool that centralizes logs from many applications, servers, platforms, and systems into a web viewer. Smart parsing and clever filters are among the features, and it supports a variety of ingestion mechanisms with the open-source LogDNA agent [118].

Loggly

Loggly [73] is like Mezmo a SaaS-based log management tool. It collects logs from different sources into a single location where you can track activity and evaluate patterns [78]. Even though Mezmo and Loggly are very similar, when it comes to functionality, Loggly has a bigger user community and generally has higher costs [28].

Chapter 4

Analysis and Implementation

What are the benefits of a complete solution instead of a custom assembled stack? This is the first question answered in this chapter. For further research, three systems were selected and implemented. The section 4.2 goes deeper into the selection process. The implementation and examination of Datadog, Grafana Loki and the Elastic 8 Stack follow in the last section.

4.1 Full stack vs. a custom assembled stack

The components in the last chapter provide interfaces for various inputs and outputs. At a first glance, there is the possibility to put together individual software stacks for your needs. The challenges usually become clear when these systems are used in practice. Since each component was developed on its own, there are various problems depending on the composition.

4.1.1 Drawbacks of a custom assembled stack

A first obstacle is, that an efficient specialized interface to the system of the next layer is not offered. This does not mean, the components do not have an interface at all, but rather they provide a standard interface, like a TCP port that accepts JSON messages. But these interfaces come with a lot of unwanted overhead, since keywords are different and not predefined. Sometimes additional plugins or extensions by the same company or third party developers are available to provide the required interface.

When this hurdle is cleared, there are various challenges when configuring the interfaces, such as different naming conventions for labels and keywords. To support this with an example, the documentation of Fluentd describes a few workarounds to make an export to Elasticsearch work.

Even if a custom assembled stack should work in theory, configuration and troubleshooting can be difficult, especially if there is no detailed documentation on the various combinations of the stack. Blog and forum posts can help here, which of course are mostly written by third parties and not by the developers themselves. However, it is often not clear which version a post is valid for.

Since the tools are maintained by different communities and software manufacturers, the update cycle also differs and the dependencies on different software versions become a time-consuming challenge.

With a custom stack, different underlying technologies might be used for the components. As a result, configuration might be in different languages. As an example, configuration for one component could be done in YAML, but the next component is configured in XML. Experts who master both technologies are then required for the configuration.

4.1.2 Advantages of a complete solution

The advantage of a solution that has been designed as a complete stack lies in the good interplay of the components with one another. This does not just start with the pure software products, but continues with uniform testing, integration, deployment and appropriate documentation. A stack from a single supplier is therefore easier for setup and maintenance.

Some of the features you find in a complete solution are working, because of the vertical integration of the components. An example is the monitoring and update function in the visualizer for the collection tools, as it is implemented in the Elastic Stack.

Another advantage lies in the security settings. With the stack from a single source, certificates and encryption can usually be added more easily, since it is configured in one place and there is no need to set up certificates for each component individually.

One of the important requirements is *RQ21 Scalability*. The simpler the stack in the beginning, the easier it will be to adapt and modify it, when stability and scalability will become key. Even if a distinction must be made, whether a cloud solution or an on-premise solution is used. With cloud solutions, scalability is often done by adjusting a few settings or switching the plan, while with on-premise solutions more time has to be planned for additional setup of containers or virtual machines.

4.1.3 Exclusion of a custom assembled stack

All of these points lead to the conclusion, that an individually assembled stack is not suitable for long-term use in a company for the use cases mentioned. As a result, a custom build stack is no longer considered a solution for this thesis, and the components discovered in subsection 3.2.1 *Components for one layer* are not investigated further.

4.2 Selection of full stack solutions

The stacks presented in subsection 3.2.2 are intended to be holistic, and the components of one stack are deliberately designed to work together. Nevertheless, there are differences between the individual stacks, which are listed in the next section.

4.2.1 Elastic vs. Amazon

Amazon OpenSearch is a fork of Elasticsearch that arose as a result of Elastic's license change. Since the change, both projects have diverged even further. Figure 3.17 illustrates, that with the new fork, Amazon took care of the development of the Storage & Analysis level and the Visualization level. At the time of writing this thesis, no tool for collecting data has been published by Amazon. This continues to be done with Elastic's Beats and Logstash.

With Beats 7.13 it is no longer possible to connect OpenSearch directly. Instead, Logstash must be used with a plugin to export to OpenSearch. A central option needs to be changed, when configuring OpenSearch to work with Logstash 8.0 [5].

These arguments no longer support the concept of a holistic system. For this reason, Amazon OpenSearch and Dashboards are not examined in this thesis. It remains to be seen whether the stack will be completed in the next few years and whether it will again become a system managed by one community.

4.2.2 Comparison of Cloud-Only-Solutions

Some stacks listed in subsection 3.2.2 can be set up in a company network or run in the cloud and thus meet *RQ15 On-premises and Cloud*. Other solutions run only as cloud software. This does not mean that the first layer Collection takes place exclusively in the cloud, since there must be at least one interface to the data sources, which is usually implemented with a slim software agent on the client. But Analysis & Storage and Visualization are only implemented in the cloud.

Datadog, Mezmo and Loggly are the presented solutions, which can only be run in the cloud. In general, all three providers are designed for collecting and evaluating log data. In terms of functionality, they differ considerably.

One of the most important requirements is *RF11 Extendable file parser*. When looking at the documentation of Loggly, it turns out that Loggly is not able to parse SECS or CSFW logs without an additional tool. The documentation states, that there are other tools for custom logs to structure them in JSON [77]. Datadog and Mezmo both offer ways to parse custom logs. While Datadog relies like Elastic on the Grok parser, Mezmo has developed its own resource-saving solution [93, 17].

Requirements *RF25 API for Visualization*, *RF26 Query API* and *RF38 Export selected original log data* describe how the indexed logs can be made available via interfaces and export functions, so they can be used in another program or displayed in a presentation. The solutions differ with regard to these requirements. While Datadog offers an extensive API for ingesting and export of log data and for configuration, the API for Loggly and Mezmo is mainly limited to import and export. However, the API does not help when configuring pipelines and the web interface must be used [119, 109, 76].

In addition to the technical differences, the spread of the three systems varies a lot. A look at Google Trends, Stack Overflow questions and Reddit posts show that Datadog has a distribution, which is more than ten times higher and used in significantly more stacks than Mezmo[1] and Loggly [28, 29].

These reasons have led to Datadog being included in the closer comparison, although it does not meet *RQ15 On-premises and Cloud*, which is not a must-have requirement.

4.2.3 Comparison of On-Premise-Solutions

The other stacks mentioned in subsection 3.2.2 support cloud and on-premise usage. While in the Grafana Loki Stack, Storage & Analysis are combined in one component. With the Elastic Stack and the TICK-Stack, Storage & Analysis are handled by two different components.

Looking at the documentation, all three systems meet the must-have requirements. For a more detailed examination, the systems can be tested and used in Docker containers. But not in all cases the trial was successful. The best-documented system is Grafana Loki. They provide a Docker Compose file to quickly set up the stack [33]. The Elastic Stack documentation is extensive and helps to set up the Elastic stack [104]. The TICK-Stack documentation of InfluxDB 1.x is available [31], but since 2020 InfluxDB 2.0 is published and there is no official documentation on setting up the whole stack.

The TICK-Stack has file watchers, which use regular expressions embedded in the Telegraf client, but it does not allow full-text search [115, 63]. After several attempts, a TICK-Stack could be created with a Docker Compose file seen in Listing 4.1. The connection from Telegraf to InfluxDB works. Also, the connection from Chronograf to InfluxDB works according to the settings. Nevertheless, the data from InfluxDB is not displayed in the dashboards. Since the documentation and other forums are not of any help, *RF43 Minimal IT support* cannot be fulfilled,

[1] The comparison was done with LogDNA before the rebranding as Mezmo.

```yaml
version: '3'

services:
  influxdb:
    image: influxdb:2.3.0
    volumes:
      - ./tmp/influxdbv2:/var/lib/influxdb2:rw
    ports:
      - 8086:8086

  telegraf:
    image: telegraf:1.23.0
    links:
      - influxdb
    volumes:
      - ./telegraf/telegraf.conf:/etc/telegraf/telegraf.conf
    environment:
      - DOCKER_INFLUXDB_INIT_ORG=Systema
      - DOCKER_INFLUXDB_INIT_BUCKET=telegraf
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=XXX
    depends_on:
      - influxdb

  chronograf:
    image: chronograf:1.9.4
    ports:
        - 8888:8888
    volumes:
        - ./tmp/chronograf-data:/var/lib/chronograf

volumes:
  influxdbv2:
  chronograf-data:
```

Listing 4.1: `docker-compose.yaml` of the TICK-Stack

if the setup is complex. Therefore, the TICK-Stack is not included in the further investigation, but it might be a good fit in the future.

4.3 Implementation of selected solutions

The solutions, that will be looked at in more detail, are Datadog, the Elastic 8.x Stack and the Grafana Loki Stack. These solutions seem to meet most of the important requirements.

4.3.1 Datadog

Datadog is offered only as a Cloud Service, and therefore a manual deployment is not necessary.

Licensing and Setup

To use Datadog the first step is to register for a plan. This can be done in two ways. The first one is directly using the Datadog website, the second one is using the Azure platform. Besides Azure, there is no other cloud platform that hosts Datadog. Each platform has its own pricing model. The Datadog website offers three different plans, starting with a free plan, a Pro plan for $15 and an Enterprise plan for $23 per host per month [96]. Datadog on the Azure platform has one plan called *Datadog Pro Pay-As-You-Go*. As the name suggests, you get billed by the resources you use [87]. For the investigation in this thesis, the Azure Plan was used, as it is more flexible with testing. There is currently no way to switch from the Datadog platform to the Azure platform after a platform has been chosen. On the Datadog platform you can easily switch between the plans, if you want a different plan on the Azure platform, you have to contact the Datadog Sales department for a custom offer.

On the Azure platform, you have to create a *subscription*, which has limits or quotas on the resources, which can be created. A subscription can have multiple *resource groups* with multiple *resources*. A resource group is a logical container into which the resources like Datadog are deployed, as shown in Figure 4.1. After the configuration of this is done, it takes a few minutes till the Datadog resource is up and running in the chosen data center. From the Azure platform, the Datadog system can be reached with a few clicks. Since it was created with Azure, a few Azure dashboards are pre-integrated in addition to the default Datadog dashboards. With these steps, the server side of the Datadog service is up and running.

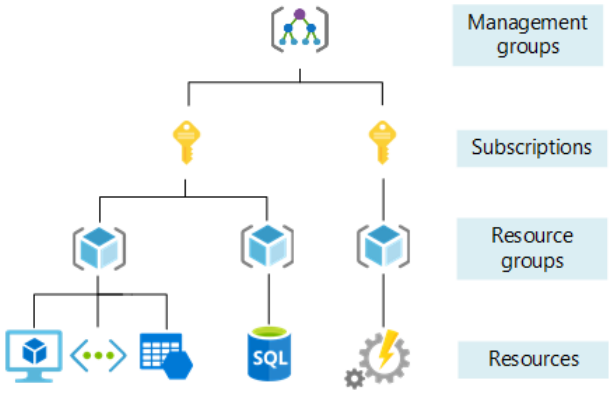


Figure 4.1: Azure platform management levels and hierarchy [91]

Collection

To get data to the server, Datadog provides three types of integration. The first way is using an installed open-source agent. The second way is authentication-based integration, where Datadog crawls metrics using the API of a service. And the last method is using a library to send data directly to the Datadog API [66]. Since this thesis focuses on logs, the Datadog Agent is used, which is the recommended way by Datadog, because it has a built-in buffer and splits up the log data into the preferred sizes for the Datadog API.

The Datadog Agent is available precompiled for many platforms. For the testing, the Windows platform is used. After the installation, the main configuration file `datadog.yaml` needs to be filled with the API key, the URL of the data intake and the enabled log collection, as seen in Listing 4.2. In the user data folder of the Agent, the same folder where `datadog.yaml` is stored, is a folder called `conf.d` with the configuration files with the various inputs.

```
1  ##########################
2  ## Basic Configuration ##
3  ##########################
4
5  ## The Datadog API key to associate your Agent's data with your
       organization.
6  api_key: XXXXXXXXXX
7
8  ## The site of the Datadog intake to send Agent data to.
9  site: us3.datadoghq.com
10
11 ###################################
12 ## Log collection Configuration ##
13 ###################################
14
15 ## Enable Datadog Agent log collection by setting logs_enabled to true.
16 logs_enabled: true
```

Listing 4.2: `datadog.yaml` for the Datadog Agent

In the `conf.d` folder, the config files for the log collection are stored. To collect logs, a config file like Listing 4.3 is needed.

```
1  logs:
2    - type: file
3      path: C:\Users\tiede\logs\current\example.log
4      service: csfw-fabric01  # name of the service owning the log
5      source: _csfw  # defines which integration is sending the log
6      log_processing_rules:
7        - type: multi_line
8          name: multi_line_rule
9          pattern: ^\d\d\d\d\.\d\d\.\d\d \d\d:\d\d:\d\d\  \- PID
```

Listing 4.3: `conf.d\conf.yaml` for the file log collection

If logs should not be written to the disk, but rather sent via network, the Datadog Agent can open a TCP port to collect the logs. A config file would look like Listing 4.4. The TCP port can then be filled by the logging application. Listing 4.5 was programmed and used for testing the TCP port with generated log lines.

```
1  logs:
2    - type: tcp
3      port: 10518
4      service: tcp_srv
5      source: _tcp
```

Listing 4.4: `conf.d\conf.yaml` for the TCP log collection

```python
import socket, time

def logtest():
    host = '127.0.0.1'
    port = 10518
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    send_loglines(s, 1000)
    s.close()

def send_loglines(s, amount):
    for x in range(amount):
        logline = f'Logline {str(x).zfill(3)}\r\n'
        s.sendall(logline.encode('utf8'))

if __name__ == "__main__":
    logtest()
```

Listing 4.5: Python script to send logs via TCP port to Datadog Agent

Analysis and Storage

Parsing and structuring of the log data take place in the cloud. Pipelines for a host, service or source are created centrally at the Datadog website. Fourteen processors and pre-created nested pipelines can be integrated into the pipeline. In addition to a Grok parser, there are also various processors for enrichment. In addition to enriching the IP address with the location, a database can be used to enrich the parsed attributes with further details. Important data can be obfuscated at this point, so it is not visible in the dashboards. Figure 4.2 shows an example of a configured pipeline for CSFW logs.

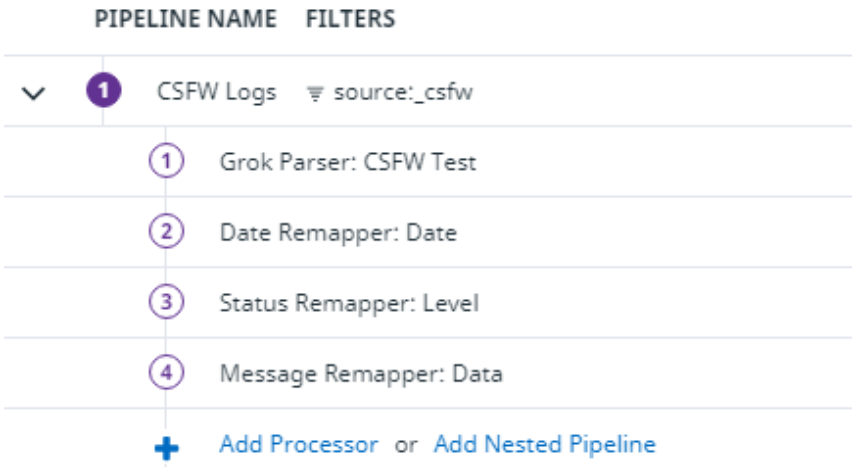


Figure 4.2: Datadog pipeline configuration

In the documentation for creating pipelines, one important detail is indicated. "Log events can be submitted up to 18h in the past and 2h in the future" [97]. Therefore, *RF15 Ingest log data later in bulk* cannot be fulfilled.

By default, Datadog has an index where the indexed log lines are stored for fourteen days. With the appropriate settings, multiple indexes can be created and log aggregation can be configured. For archiving log data over a longer period, cloud storage can be added. After the expiration time in the index, the log lines are copied and stored in the cloud.

Visualization

The logs are visualized in the Datadog Log Explorer, as shown in Figure 4.3. Results can be achieved quickly even without reading the documentation. The visualization can be done in six different diagrams. Not only can filters be applied to the data, but also patterns and transactions can be recognized and combined. Statistical values can also be calculated using additional formulas.

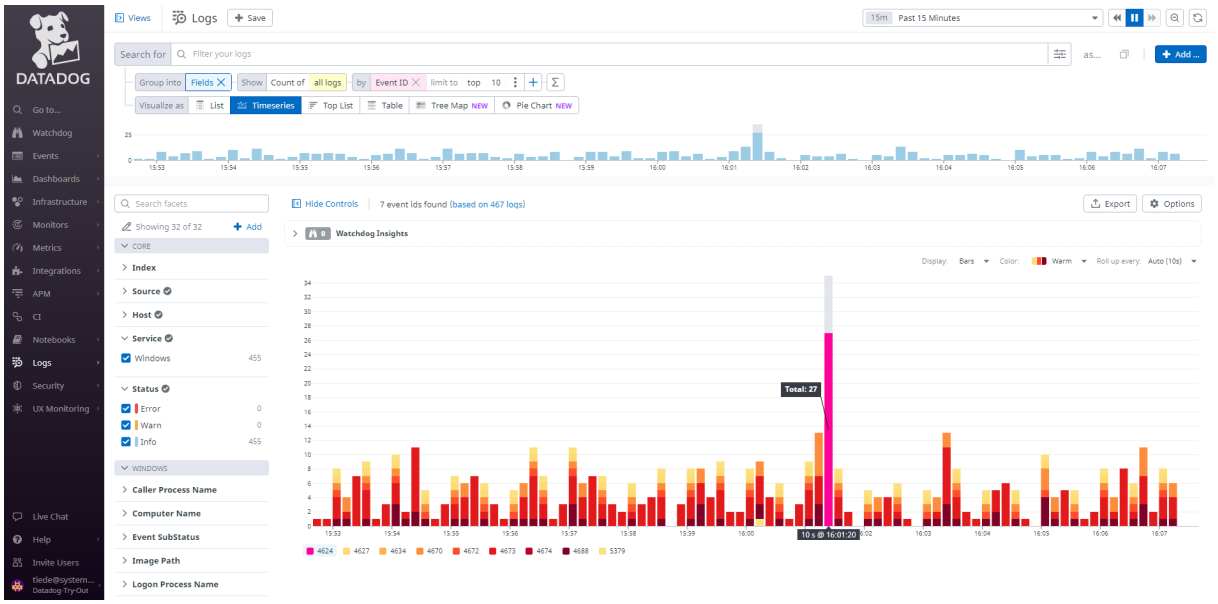


Figure 4.3: Datadog web interface

The colors in the table view are based on the log level. In the time series view, predefined color schemes are available. The other diagram types have no options to use customized colors. There is no straightforward marking or comment function. But there is a workflow system built in Datadog to handle incidents. The incident's handling in Datadog allows commenting and working on an issue together in real-time.

One feature in Datadog is the grouping of transactions, as shown in Figure 4.4. This allows a comparison between recurring sequences, and gives details about duration, the highest log level and the amount of events during that transaction. To differentiate the transactions, a unique attribute like an Event-ID or a Wafer-ID for the same transaction is necessary to identify and group the events in a sequence.

The Datadog Log Explorer allows sharing the dashboard internally if it is used with a team. In addition, it has an export function to download the data as a CSV file or the API can be used to retrieve the results of the analysis as well.

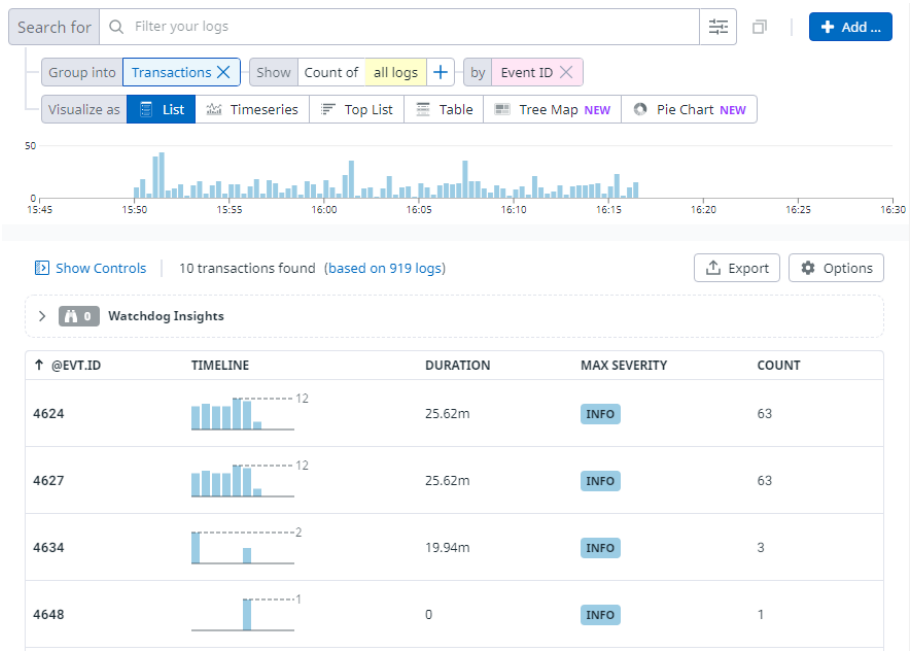


Figure 4.4: Grouping of transactions in Datadog

4.3.2 Grafana Loki Stack

Grafana Loki by Grafana Labs is an open-source solution, which can run on-premise or in the cloud. This thesis concentrates on the on-premise solution, to evaluate the setup process as well. The setup, for the cloud solution, is done by creating a Grafana Cloud account.

Licensing and Setup

Since Loki and Grafana are licensed under the AGPLv3 [52], the commercial use of the code is permitted. If changes are made to the source code and used in a productive environment, the changes have to be published under the same license. For on-premise use, the cost of the hardware and the maintenance of the stack have to be considered. Grafana Labs offers a free cloud plan for up to three users and 50 GB of logs with 14 days of retention. The Pro plan starts at $8 per month per user and includes 50 GB of logs and one month retention. For the pricing for the Advanced and Enterprise plans, Grafana Labs has to be contacted.

The setup of the test environment is done via Docker Compose 1.29.2 and the Docker Engine 20.10.14. After creating the `docker-compose.yaml` seen in Listing 4.6 the configuration file for each container has to be created. The configuration for each container is described in the following sections. After the spin-up of the Docker containers, the web interface of Grafana can be opened and logged into. Within a few seconds, the indexed log data should be ready to display.

Collection

The Grafana Loki Stack offers two tools for log collection. The *Grafana Agent* is used for sending metrics and log data of the client. And *Promtail* is used specifically for shipping local logs to a Grafana Loki instance. This thesis uses Promtail as a log collector, since it was developed with the focus on log shipping.

```yaml
version: '3'

networks:
  loki:

services:
  loki:
    image: grafana/loki:2.5.0
    ports:
      - 3100:3100
    volumes:
      - ./loki/:/mnt/config
    command: -config.file=/mnt/config/loki-config.yaml
    networks:
      - loki

  promtail:
    image: grafana/promtail:2.5.0
    volumes:
      - ./promtail/:/mnt/config
      - ./csfw-log-folder/:/mnt/logs
    command: -config.file=/mnt/config/promtail-config.yaml
    networks:
      - loki

  grafana:
    image: grafana/grafana:8.5.6
    volumes:
      - ./grafana/datasources:/etc/grafana/provisioning/datasources
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=XXXX
    ports:
      - 3000:3000
    networks:
      - loki
```

Listing 4.6: `docker-compose.yaml` of the Grafana-Loki-Stack

The collection of the log data is done via Promtail. Promtail needs to run on the client and is used to read, parse, transform, filter and push the log lines to Loki. The configuration file `promtail-config.yaml` is used to set the input and the output of the collector. Listing 4.7 shows an example for collecting a CSFW log file. The pipeline configuration of the log collector is also done in this file and not centrally in Loki. This means, there is no central configuration file. In the case of a distributed system, it must be ensured, that all configuration files are up-to-date. The Grafana Loki Stack does not provide a solution to achieve this, and therefore a third-party tool needs to be used.

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:
- job_name: csfw
  static_configs:
  - targets:
      - localhost
    labels:
      job: csfw
      __path__: /mnt/logs/
  pipeline_stages:
  - match:
      selector: '{job="csfw"}'
      stages:
      - regex:
          expression: '(?P<timestamp>\d\d\d\d.\d\d.\d\d
              \d\d:\d\d:\d\d.\d\d\d).*PID: (?P<pid>\d+).*LEVEL:
              (?P<levelint>\d) (?P<level>[a-zA-Z]+) (?P<content>.*)'
      - labels:
          timestamp:
          levelint:
          level:
          pid:
          content:
```

Listing 4.7: Extract of `promtail-config.yaml` of the Grafana-Loki-Stack

The pipeline in the `promtail-config.yaml` consists of a set of stages. Promtail differentiates four types of stages. Typically, the first stage is a *parsing stage* to extract the data out of a log line and make it available for the next stages. The next two types of stages are the *transform stages* and the *action stages*. The action stages can modify the label, change the timestamp or content, and can calculate metrics. The final stage is the *filtering stages*. The presented config file uses the regex parsing stage and the labels action stage to structure the log data.

Promtail does not have a cache. This means that if Loki is down or cannot receive the log data, then Promtail will retry as often as configured and then drop the batch of log entries and proceed with the next ones.

Analysis and Storage

Loki stores indexes of the structured log lines. In doing so, it is only indexing the metadata rather than the content of the log lines. This means a full-text search is not possible. A workaround can be achieved, if a label is put on the whole log message, so the message is indexed and can be searched using placeholders.

Besides the structured input, Loki's settings are set by the `loki-config.yaml` seen in Listing 4.8. It provides the setting for the HTTP port Loki is listening to and the setting for the database schema. The database schemas are important for updates of Loki and thus the database structure. The last line is the configuration of the built-in supported Prometheus Alertmanager [6], which can be used to send notifications and warnings defined by rules.

```
server:
  http_listen_port: 3100
  grpc_listen_port: 9096

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

limits_config:
  ingestion_rate_mb: 512
  ingestion_burst_size_mb: 512

ruler:
  alertmanager_url: http://localhost:9093
```

Listing 4.8: Extract of `loki-config.yaml` of the Grafana-Loki-Stack

Visualization

The visualization in this stack is implemented in Grafana. For Grafana to be able to display the data from Loki, the configuration for a data source must be created. This happens either via the web interface or via a YAML configuration file. The testing was done via the YAML file seen in Listing 4.9. The most important settings are the URL of Loki and that the type of the data is set to `loki`.

The Grafana web interface is divided into the menu items Dashboards, Explore and Alerting. Dashboards offer the possibility to put together a customized interface of diagrams and log data. Explore can be used to search for entries in the log data or to calculate and display metrics, as shown in Figure 4.5. The tables created can be added to dashboards if they are used more often. Alerting allows the creation of rules for monitoring and, if necessary, notification when limit values are exceeded.

```
apiVersion: 1

datasources:
  - name: Loki
    type: loki
    access: proxy
    orgId: 1
    url: http://loki:3100
    basicAuth: false
    isDefault: true
    editable: true
```

Listing 4.9: `datasource.yaml` of the Grafana-Loki-Stack

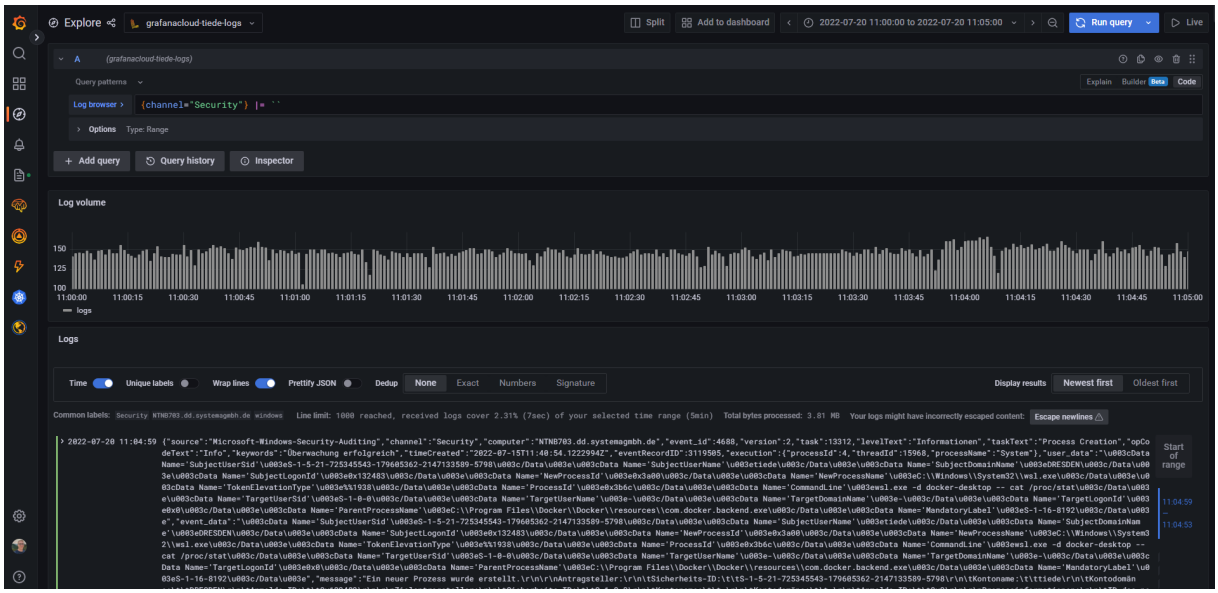


Figure 4.5: Grafana web interface

The language *LogQL*, inspired by the Prometheus Query Language, is available for querying the log data from Loki. LogQL consists of log queries for querying log lines and metric queries for calculating metrics based on the log data. Extensive queries are not possible without looking at the documentation [82].

The colors of log data are determined based on the Syslog level. There is currently no way to customize this. There is also no option to mark log lines or add a comment. In the diagrams, Grafana offers the possibility to add annotations to interact with other users.

The export at Grafana allows downloading the structured data as JSON and CSV. If the whole log line is indexed, it can be exported to a text file again, which results in the original log file. The data can be limited by time or by additional filters.

4.3.3 Elastic 8 Stack

Of the stacks examined in this thesis, Elasticsearch has been on the market for the longest time. As explained in subsubsection 3.2.2 the Elastic 8 Stack is used for this research.

Licensing and Setup

Since Elasticsearch 7.11, Elastic changed the binary distribution and the source code licensing to the Elastic License v2. This affected all components of the Elastic 8 stack, i.e. Elastic Agent, Elasticsearch and Kibana. Therefore, if an application that embeds and redistributes Elasticsearch is built, a direct agreement with Elastic for redistribution is required [41]. The Elastic Stack can be used as a cloud service and self-managed on-premise. The Cloud plans of Elastic reach from $95 to $175 per month for 120 GB storage. For the Enterprise solution on-premise, the sales department needs to be contacted.

Elastic provides a Docker Compose file for setting up part of the Elastic 8 stack, but the Elastic Agent is not part of it. Listing 4.10 shows an excerpt, which creates a cluster of three Elasticsearch containers with a Kibana container on top. The communication between the individual components in the stack takes place in the test environment with self-created certificates. To use the Elastic Agent, additional environment settings must be set in the Docker Compose file [104].

Once the Docker environment is running, the Elastic Agent needs to be installed at a client. At least one Elastic Agent has to be configured as a Fleet Server, which runs as a subprocess inside an Agent. One Fleet Server process supports numerous Elastic Agent connections, and acts as a control layer for updating agent policies, gathering status data, and organizing activities among Elastic Agents. The Agent settings are managed in one central place in Kibana, and in a distributed system, configurations are updated automatically by the Fleet Server. Agent updates can also be rolled out globally via the Kibana web interface or the REST API for Kibana.

The Elastic Agent can either be run directly or installed as a service. Before the first start, an agent police has to be created in Kibana. The agent policy consists of general settings for the agents and the selected integrations. The next step is the check and distribution of the CA certificate, a digital certificate issued by a certificate authority. In the case of a self-signed certificate, this has to be copied and installed at the client [1, 62].

Collection

For the log collection, the agent police needs to include the *Custom Logs* integration [61]. In addition to the log file path, the custom configuration has to be filled with the pipeline and the multiline setting as shown in Listing 4.11 [85].

```
1 pipeline: csfw-pipeline
2 multiline:
3   type: pattern
4   pattern: '^\d\d\d\d\.\d\d\.\d\d \d\d:\d\d:\d\d\.\d\d\d'
5   negate: true
6   max_lines: 20000
```

Listing 4.11: Custom configurations of the Custom Logs integration in Kibana

```yaml
version: "2.2"

services:
  setup:
    image: elasticsearch:8.2.3
    volumes:
      - certs:/usr/share/elasticsearch/config/certs

  es01:
    depends_on:
      setup:
        condition: service_healthy
    image: elasticsearch:8.2.3
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - esdata01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200

  kibana:
    depends_on:
      es01:
        condition: service_healthy
      es02:
        condition: service_healthy
      es03:
        condition: service_healthy
    image: docker.elastic.co/kibana/kibana:8.2.3
    volumes:
      - certs:/usr/share/kibana/config/certs
      - kibanadata:/usr/share/kibana/data
    ports:
      - 5601:5601

  elastic-agent:
    image: docker.elastic.co/beats/elastic-agent:8.2.3
    container_name: elastic-agent
    restart: always
    user: root
```

Listing 4.10: `docker-compose.yaml` of the Elastic-Stack

The Elastic Agent with the Fleet Server has to be enrolled with the settings shown in the following command, and can then be started or installed, as seen in Listing 4.12.

```
.\elastic-agent.exe enroll  ‘
  --fleet-server-es=https://localhost:9200 ‘
  --fleet-server-service-token=XXXX ‘
  --fleet-server-policy=fleet-server-policy ‘
  --fleet-server-es-insecure ‘
  --url=https://localhost:8220 ‘
  --enrollment-token=XXXX
```

Listing 4.12: Enrollment of the Elastic Agent with Fleet Server

After the start, it takes a few seconds until the Fleet Server is online, and the log data is sent to Elasticsearch via data streams.

Analysis and Storage

Pipelines for parsing and indexing the log data are configured in Kibana and executed in the Elasticsearch ingest node. For the pipelines, 39 processors are shipped for parsing, transforming, trimming, splitting and enrichment. Direct enrichment with data from an external database is not possible. Data can only be enriched via an Elastic enrich index, which is created by execution of an enrich policy. An enrich policy combines data from the policy's source indices to create the enrich index [39]. In addition, it is possible to integrate your own processors. For this trial with a CSFW log file, the dissect processor [32] was used for parsing the log lines and the date processor for the parsing of the time, as seen in Figure 4.6. Like the Grok parser, dissect also pulls structured fields from a single text line within a document. But Dissect does not make use of Regular Expressions like the Grok parser does, which makes it quicker than the Grok parser in most cases.

Elasticsearch can contain multiple indices which consist of types, and the types consist of documents with properties. These indices allow the full-text search of the log data.

Visualization

The log data query in Kibana can be found under Analysis, which lists the log lines in a table, as the screenshot in Figure 4.7 shows. Two query languages for filtering the data are available in Kibana, the Lucene query language and the Kibana Query Language (KQL). Only the latter language supports auto-completion. This is helpful for quickly applying filters without reading the documentation beforehand.

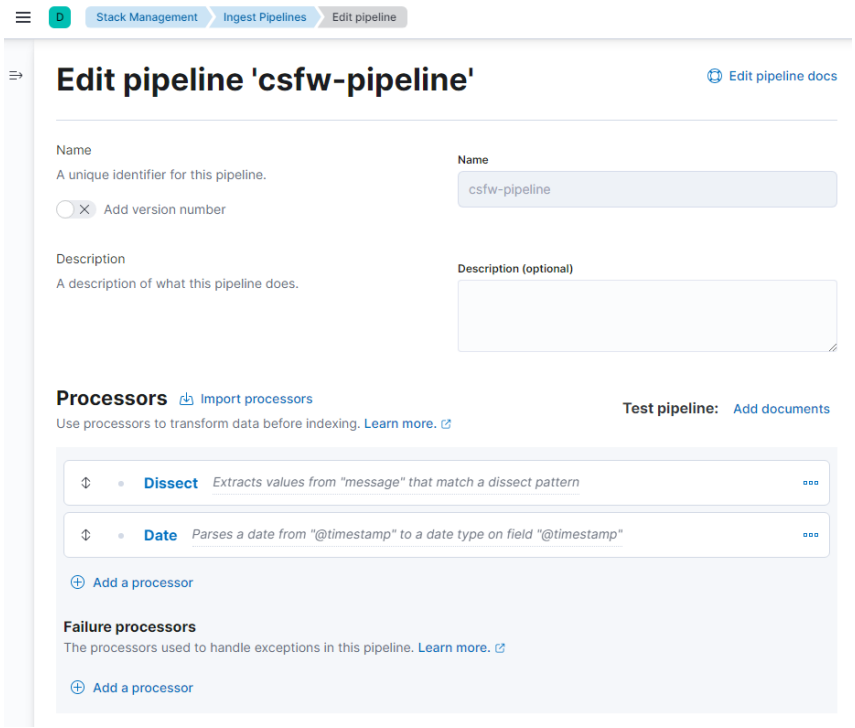


Figure 4.6: Elastic pipelines

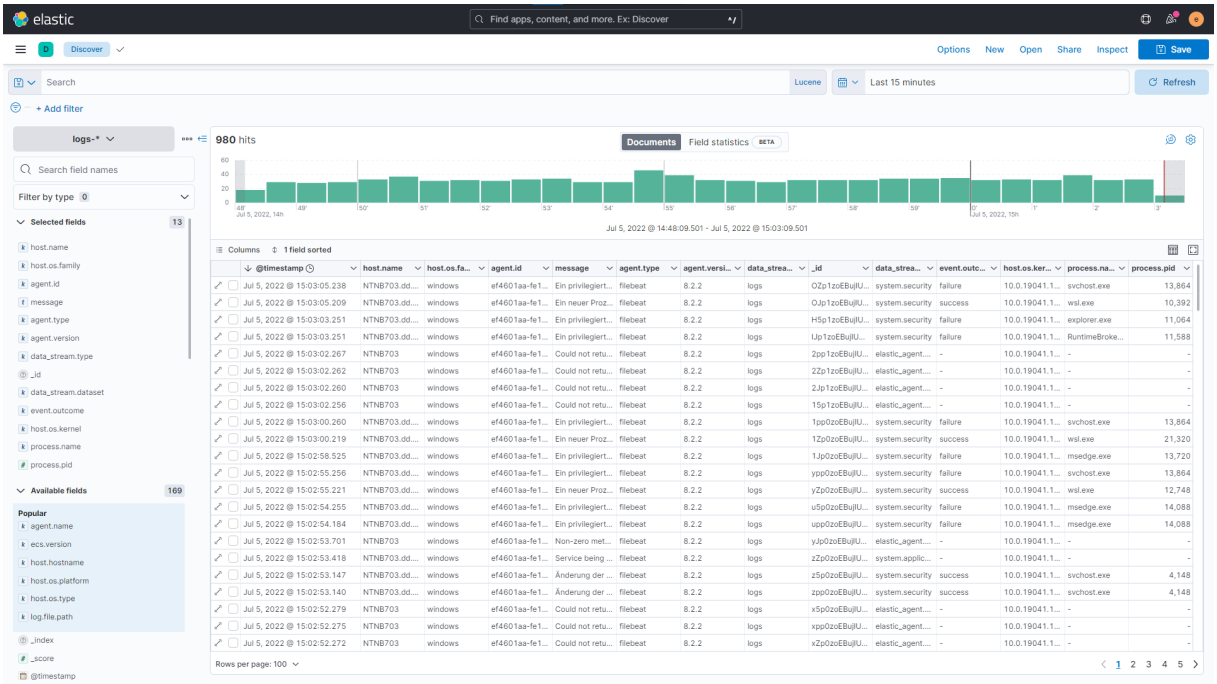


Figure 4.7: Elastic Kibana web interface

Dashboards in Kibana are composed of elements, referred to as *visualizations*. The individual visualizations are stored in the *Visualize Library* and can be combined into a dashboard. The strength of Kibana lies in the creation of the visualizations. Worth mentioning is the large selection of chart types and the break-down feature, which allows the data to be divided into different segments with different colors, as shown in Figure 4.8.

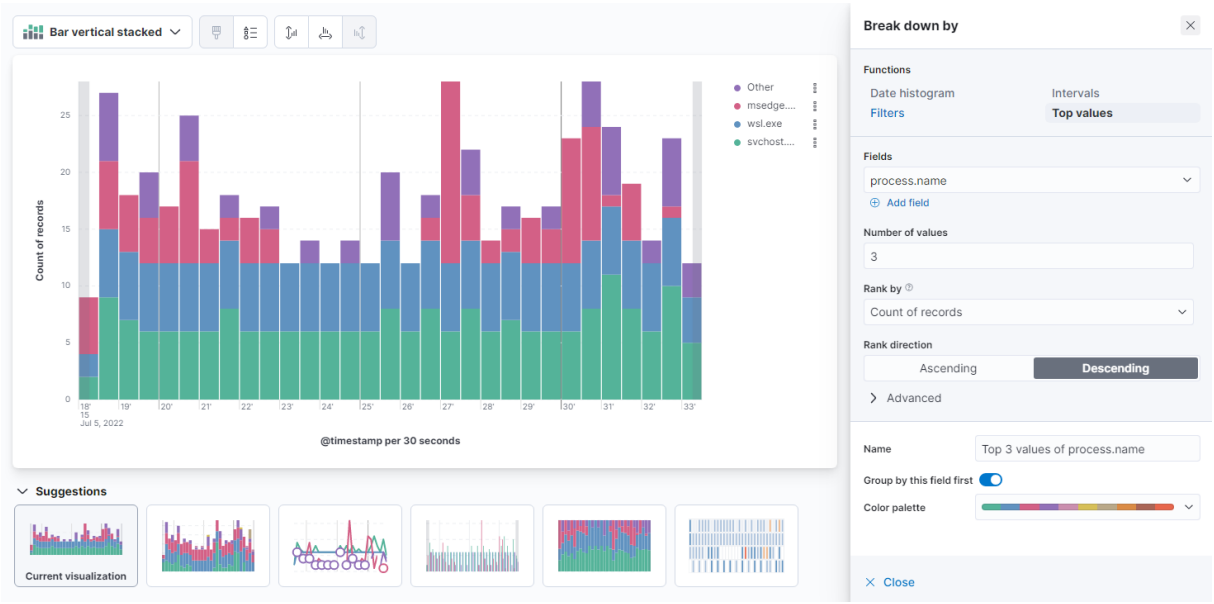


Figure 4.8: Visualization setup in Elastic Kibana

If multiple entries should be exported, the web interface of Kibana is limited to CSV files. The JSON for a single entry can be copied in the web interface, but for the export of multiple log entries, the API has to be used. Diagrams can be shared via a link, which either represents a snapshot or the current data. With Kibana there is no way to mark log lines or add comments.

Chapter 5

Comparison

In this chapter, Datadog, Grafana Loki and the Elastic 8 Stack are compared based on the three components of the architecture and the requirements collected in section 2.3.

5.1 Comparison of components

The comparison of the three examined solutions is made using the architecture breakdown described in subsection 3.1.6.

5.1.1 Collection

With all three solutions, an agent is installed on the client to collect the log data. Only Elastic still needs the intermediate step with the Fleet Server. Overall, Datadog is the easiest to install and set up. With Grafana Agent, the effort is greater due to additional YAML settings. The setup of the Elastic Agent takes the most time, since in addition to various settings, the certificates must also be set up properly, especially for the Fleet Server.

The resource consumption differs considerably. A distinction is made between the idle time and times when actual log data is processed. Datadog's process uses 90 MB and almost no calculator time if no data is coming in. The Grafana Agent wants 110 MB and uses a little more calculation time. The Elastic Agent starts multiple subprocesses depending on the data collected. In the testing, the Elastic Agent used 35 MB and the subprocesses Filebeat 140 MB, Metricbeat 100 MB and the Fleet Server 30 MB, in total 305 MB. The Fleet Server can be outsourced to save resources at the client.

During the ingest, the Grafana Agent used the most calculation power, because the pipeline for parsing and structuring the data is executed on the client. With Datadog and Elastic, this process takes place in the cloud or the Elasticsearch cluster. This leaves only the load of multiline detection on the client. During the ingest, Datadog does not consume much more resources compared to the idle state. Even though the Elastic Agent does not run the pipelines on the client, it almost uses as many resources for the ingest of the same file as the Grafana Agent.

All three tools can be controlled via the command line. The Datadog Agent also offers a web interface that shows the status, and it is also possible to adjust the settings and restart the agent. The Elastic Fleet server has a web API which shows the status as JSON.

When it comes to agent settings, Elastic allows most options to be managed centrally from Kibana. Datadog allows pipeline management via the cloud interface, and with Grafana everything has to be done locally via YAML. Updates must be installed manually for Datadog and Grafana Loki. With Elastic, the update can be started via Kibana.

5.1.2 Analysis

The analysis of the compared systems starts with the ingest pipelines. Even though the concept of pipelines is the same for all systems, they are implemented in different ways. For the Grafana Stack, the pipeline is running in Promtail. Datadog's analysis is done in the cloud, and Elastic runs the pipeline in Elasticsearch.

The Grok parser is implemented as a processor for all pipelines. Elasticsearch goes one step further, including a more resource-friendly parser with the dissect processor. With Datadog and Elastic, setting up the parser is much more convenient via the web interface, since it can be executed there immediately with examples. Setting up Grafana Loki via YAML is more

demanding, and the log import has to be run after each change for testing. This means only Datadog's and Elastic's analysis settings can be changed via the REST API.

Nothing is known about the structure of the index at Datadog. Loki's two index mechanism, called Single Store Loki, uses one object store, with two different types of data: chunks and indexes. As log entries from a stream arrive, they are compressed as "chunks" and saved in the chunks store. The index stores a label set of the log events and links them to the individual chunks. A single Elasticsearch cluster can have several indices, which can hold different documents, each of which has properties.

Performance and scalability

The cloud versions of all three solutions automatically scale within the boundaries of the selected usage plan. To evaluate the performance and scalability of the on-premises version of Grafana Loki and Elasticsearch, the collecting and indexing of 12.4 GB and more than 10 million log events have been tested. The test data was the events from a single instance of a semiconductor plant over a period of two months, and they were structured as a CSFW log. The test was performed on a workstation with a 2.4 GHz processor with 4 cores, 32 GB RAM and an NVMe SSD. Both versions had to parse two custom fields or labels out of a log event. Grafana was configured with a Grok parser and Elastic used the dissect parser.

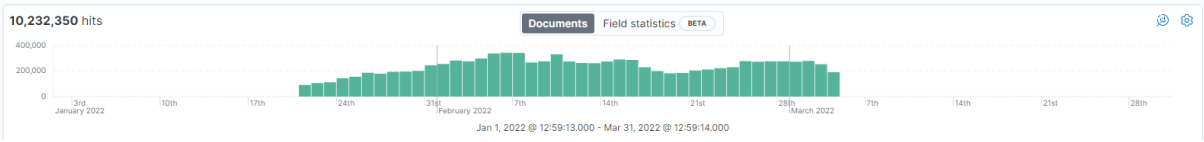


Figure 5.1: Result of the Elastic test run

The Elastic Stack 8 stack indexed the data in 2 hours and 56 minutes. The resulting diagram is seen in Figure 5.1. In the end, the index had a total size of 8.8 GB, but the index size did grow to over 11 GB while the index was generated. During and after the indexing, accessing the log data via Kibana was possible without a noticeable delay.

The first runs with Grafana Loki returned many errors. The agent aborted the transmission of many log events with "server returned HTTP status 429 Too Many Requests" and less than 10,000 log events were indexed by Loki. Since the Grafana Agent or Promtail have no built-in cache, the agent discarded the log data after a specified number of retries. After increasing the `limits_config` in the of `loki-config.yaml`, as seen in Listing 4.8 Loki was able to handle the data load. The complete indexing took Loki 23 minutes, and it ended up with an index of 1.3 GB. Compared to the Elastic Stack, there were noticeable delays when accessing the statistics and charts. Sometimes the query took several seconds until the log evaluations were displayed in the dashboard.

Grafana Loki's ingest was faster and ended with an index, which was almost ten times smaller than the original log data, as shown in Figure 5.2. Elasticsearch took nearly eight times longer for the indexing, and finished with a two-thirds smaller index. Larger amounts of data increase the retrieval time in Grafana significantly, which is confirmed by [23].

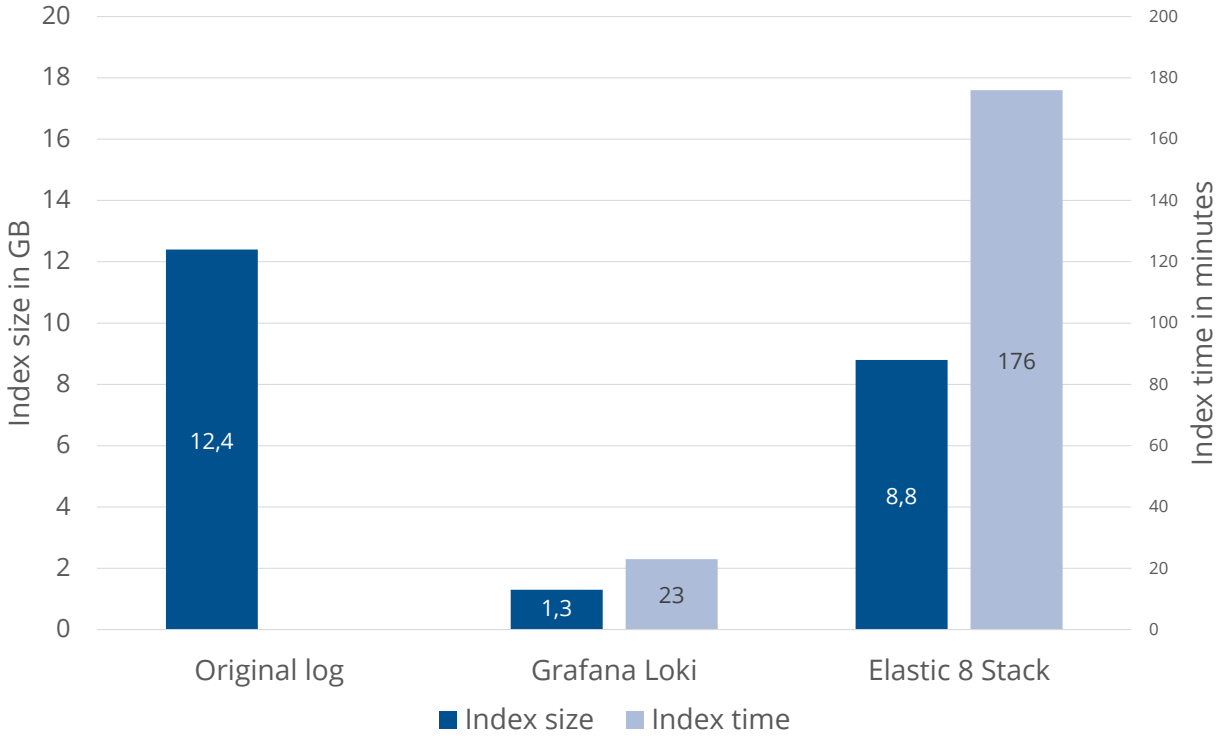


Figure 5.2: Performance of the on-premises stacks

5.1.3 Visualization

The visualization in the stack is the most important component for the developers, as it is used to evaluate anomalies in the log data and to find incorrect behavior.

All three systems offer an interface with automatic updates for real-time monitoring of the log data, which are built similarly. In the upper right corner is the selection of the time range, and the largest part is taken up by a table view with the selected attributes. Datadog and Grafana color the log lines according to the log level. Elastic has no option to color the log lines in table view.

When searching for specific log data, someone can quickly come up to speed with the filters and the structure of the interface with Datadog and Kibana, even without prior knowledge. Grafana Loki filtering can be used quickly for simple requirements using selection lists, but the query syntax must be known for more complex filters using the search bar. The integrated auto-completion does not help as well as with Datadog or Kibana.

The visualization options are well-developed on all three platforms. Nevertheless, Kibana stands out with features that are not available on its competitors. The different coloring in the diagrams, depending on any attributes, leaves behind the coloring based on the log levels.

Datadog is the only tool that offers sequence detection. This allows log lines to be grouped, and changes in the sequence order can be detected.

All three tools do not have the desired functions to record the knowledge gained or to share it with employees. It is not possible to mark or comment on individual log lines. Datadog offers

a kind of shared document with integrated links to the dashboards for collaboration. With Grafana, comments can be left in the diagrams and Elastic is limited to creating reports.

The export functions are necessary in the use cases for transferring the data to other special applications. JSON and CSV export are supported by all stacks, and only Grafana allows export as a text file. This is an interesting feature because if the log line is fully indexed, the original log file can be restored without additional scripts.

Based on usability and structure, Datadog is the preferred choice, followed by Elastic. Grafana can also be used well with a little more training.

5.2 Comparison of requirements

Based on the requirements in section 2.3, the following part goes into more details, if and how the requirements are met by Datadog, Grafana Loki or the Elastic Stack. It follows the same order as in chapter 2.

At the end of each section, the requirements are summarized in a table. "–" means, the requirement is not fulfilled. "+" means the requirement is fulfilled, and "++" means, it was extraordinarily well implemented.

5.2.1 Functional requirements

With the functional requirements, there are no differences between the cloud solution and on-premise of Grafana Loki and the Elastic Stack except for *RF43 Minimal IT support*. Table 5.1 summarizes the result of the functional requirements.

Requirements for collection

RF11 Extendable file parser M

All three stacks come with a file parser in the form of a Grok parser. While in Grafana the configuration is done in a YAML file, Datadog and Elastic offer a web interface where the parser can be tested using examples. Elastic also offers different kinds of parsers.

RF12 SECS parser and CSFW parser C

None of the stacks offers a SECS parser or a CSFW parser. They have to be built with the available parsers.

RF13 Enrichment S

Grafana Loki has no option to enrich log lines with external data. Elastic allows enrichment if the data is stored in another Elastic index and an enrich policy is used. Only in Datadog, other databases can be used to enrich logs with additional information.

RF14 Combine data from multiple sources S

All solutions have agents, which push the log data to a central point, and have the option to collect logs from multiple files. Since the logs are stored in the same index, they are also shown together in the visualizer.

RF15 Ingest log data later in bulk S

For Grafana Loki and Elasticsearch, the ingest can be done at any time and of any size. Datadog has restrictions when it comes to the time, logs can only be ingested up to 18 hours in the past and two hours in the future.

RF16 Pseudonymization S

Datadog has in its settings multiple options to replace data with pseudonyms. In Elastic, there is a pipeline processor called fingerprint, which accomplishes this same feature. Grafana does not fulfill this requirement.

RF17 Load of log collection client S

The subsection 5.1.1 gives a more detailed description of this requirement. Datadog uses the least resources. The Grafana Agent and the Elastic Agent have much higher needs, when it comes to resources for the logging client, and therefore do not meet this requirement.

Requirements for analysis

RF21 Filter by attributes M

All three stacks allow the filtering of the various attributes.

RF22 Full-text search M

Datadog and Elastic allow the full-text search. Grafana Loki's form of indexing does not allow it.

RF23 Sorting by time M

This basic feature is implemented in all three stacks.

RF24 Auto-completion S

Basic auto-completion is supported by all platforms, but Elastic and Datadog show possibilities based on the indexed data in addition to the suggestions of the query syntax.

RF25 API for Visualization M

The three stacks have an API for their visualizer and third-party tools as well.

RF26 Query API C

All tools have a REST API for querying the indexed data. Only Elastic offers a querying with SQL statements.

RF27 Anomaly detection C

Anomaly detection can be activated in Datadog and in Elastic. Grafana does not support this feature.

Requirements for visualization

RF31 Table view M

The table view is available in all three stacks. On each platform, there are various tools to change the columns, sort by columns or select the size of the log data excerpt.

RF32 Histogram S

All solutions can show the log events in a histogram for different time ranges, where the bars are colored according to the log level. Kibana of Elastic is the only solution, which allows custom coloring and breakdown of the data according to different attributes.

RF33 Scatter plots S

Datadog and Grafana have built-in options to generate a scatter plot. Kibana has no built-in option to produce scatter plots, but there are Kibana plugins available to add this feature.

RF34 Grouping sequences C

Datadog is the only service that allows, with the transaction feature, the specification of sequences, which then can be grouped.

RF35 Real-time log tail C

All systems provide a real-time log view, which updates automatically to show the latest log lines.

RF36 Mark log lines C

None of the systems have an option to mark log lines. There also seems to be no plugin available.

RF37 Comment log lines C

And none of the systems have the capability to comment on log lines, but there are other options to collaborate with your coworkers. Datadog has a feature called *Notebook*, which are essentially shared documents with the option to link diagrams of the log data. Grafana has an option to annotate the diagrams.

RF38 Export selected original log data S

All programs have an API to export data in JSON and a web interface to download data in CSV or other formats. Grafana stands out in comparison, because it allows a wide range of formats and has various filter options integrated. The original log line is saved in Elastic by default. In Datadog and Grafana Loki the original log line needs to be added as attribute to allow a later export.

RF39 Sequence diagram W

Only Datadog fulfills *RF34 Grouping sequences*, but it cannot show the data in a sequence diagram.

Feature	Datadog	Grafana	Elastic
Requirements for collection			
M RF11 Extendable file parser	++	+	++
C RF12 SECS parser and CSFW parser	-	-	-
S RF13 Enrichment	++	-	+
S RF14 Combine data from multiple sources	+	+	+
S RF15 Ingest log data later in bulk	-	+	+
S RF16 Pseudonymization	+	-	+
S RF17 Load of log collection client	+	-	-
Requirements for analysis			
M RF21 Filter by attributes	+	+	+
M RF22 Full-text search	+	-	+
M RF23 Sorting by time	+	+	+
S RF24 Auto-completion	++	+	++
M RF25 API for Visualization	+	+	+
C RF26 Query API	+	+	++
C RF27 Anomaly detection	+	-	+
Requirements for visualization			
M RF31 Table view	+	+	++
S RF32 Histogram	+	+	++
S RF33 Scatter plots	+	+	-
C RF34 Grouping sequences	+	-	-
C RF35 Real-time log tail	+	+	+
C RF36 Mark log lines	-	-	-
C RF37 Comment log lines	-	-	-
S RF38 Export selected original log data	+	++	+
W RF39 Sequence diagram	-	-	-
Commercial requirements			
M RF41 Usage of existing solution	+	+	+
M RF42 User Management	+	+	+
S RF43 Minimal IT support	+	-	-
M RF44 No publishing of source code	+	+	+
Count of fulfilled requirements (+)	25	17	24

Table 5.1: Comparison of Functional requirements

Commercial requirements

RF41 Usage of existing solution M

The solutions already exist, and no further development is necessary.

RF42 User Management M

Datadog and Grafana Loki have three user roles predefined. Elastic has 30 roles at the beginning. All systems allow adding more custom defined user roles.

RF43 Minimal IT support S

Datadog is easy to maintain, since only the agent and the backup need to be taken care of. This is similar if the cloud solutions of Grafana or Elastic are used. When installing on-premise, there is a significantly greater effort required as well as much more IT support.

RF44 No publishing of source code M

The Datadog Agent and Grafana Loki are open-source and can be modified without publishing the source code. In the case of source code modification, for the Elastic stack, the code has to be published under the Elastic License again. In the case where no source code change have been made, Elastic fulfills this requirement as well.

5.2.2 System requirements

The system requirements depend heavily on whether a cloud or an on-premise system is used. They can be drawn up more precisely if concrete project needs are known, but this is beyond the scope of this work. Table 5.2 summarizes the result of the two system requirements.

RS01 Hard disk space M

The used hard disc size of Datadog Agent Windows client is 640 MB, and is the only part that needs to be installed. The Grafana Agent for Windows uses 100 MB and the Grafana Promtail container, which was used in this implementation, uses 200 MB. The Windows Elastic Agent takes the most space with 760 MB.

The Grafana Loki Stack adds up to 550 MB with Promtail as the collector, as seen in Figure 5.3. The Elastic 8 Stack needs 2.7 GB to run properly. The size of the index is covered in *RQ22 Volume of indexed information*.

RS02 Secure transmission M

With Datadog, the connection from the agent to the cloud is encrypted. With the Grafana test system, the data transport between the components is unencrypted. The certificates for secure data exchange have to be manually configured. At Elastic, the test system is created with self-signed certificates and the data is transported encrypted between the components. A subsequent exchange of the certificates is possible.

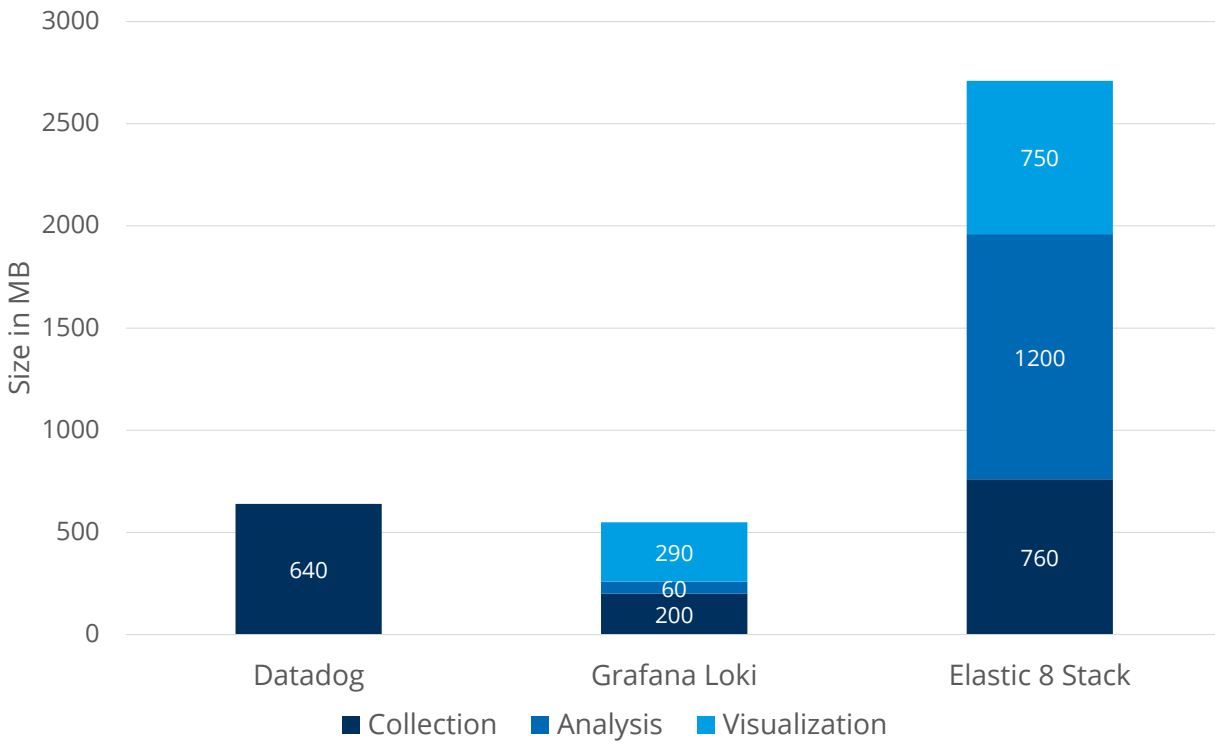


Figure 5.3: *RS01 Hard disk space* comparison

Feature	Datadog	Grafana	Elastic
M RS01 Hard disk space	++	++	+
M RS02 Secure transmission	++	+	++
Count of fulfilled requirements (+)	4	3	3

Table 5.2: Comparison of System requirements

5.2.3 Quality requirements

There are differences between the cloud and the on-premise version for some quality requirements. The results are summarized in Table 5.3.

Availability

RQ11 High availability S

This requirement compares the cloud solution, since an on-premise solution depends on the company infrastructure. Datadog terms state 99.8 % availability [27]. The Grafana Cloud ensures 99.5 % availability without reduction in costs [55]. Only the Elastic SLA mentions a 99.9 % availability without a cost reduction [37].

RQ12 Storing of full data S

All platforms save the indexed data for at least fourteen days.

RQ13 Storing of filtered data S

All three solutions allow configuring a retention policy, which allows the removing or archiving of selected log data.

RQ14 Archived data S

Datadog allows setting up external storage for archiving the data. Grafana Loki has no archive feature and the retention policies need to be used if the data needs to be stored for a longer time. Elastic has an archive feature which is called *Snapshot* and is used for creating backups.

RQ15 On-premises and Cloud S

Datadog is a cloud-only solution. Grafana Loki and the Elastic Stack offer on-premises and cloud versions.

Performance

RQ21 Scalability M

This requirement differentiates between the cloud service and the on-premise version. Starting with the cloud services, Datadog can be used with the Pay-as-you-go service at a larger scale without changing a plan or the settings. The Elastic Cloud resizing needs to be done manually, but also has no downtime. The Grafana Cloud advertises high scalability, but does not mention any settings in the documentation.

The on-premise version of Grafana Loki and the Elastic Stack allow horizontal scaling. During the test presented in subsubsection 5.1.2, both systems are able to handle data streams as they arise in current production facilities.

RQ22 Volume of indexed information S

For Datadog, the size of the index is not available. The size of Grafana Loki and Elastic index varies depending on settings like compression rate. Therefore, this comparison is based on the default settings and compression algorithm of Grafana and Elastic.

Grafana Loki's index size is normally less than the size of the raw data. The Loki index in the conducted testing was about ten times smaller than the original data. The Elastics index before version 8, took more than double the size of the raw data [23]. The test performed with version 8 had significantly better results, and the index was two-thirds smaller than the raw data, which can be seen in Figure 5.2.

RQ23 Real-time update interval C

In all tests of the three stacks, the ingested data was available seconds after it was generated, or the agent was started.

Feature	Datadog	Grafana	Elastic
Availability			
S RQ11 High availability	-	-	+
S RQ12 Storing of full data	+	+	+
S RQ13 Storing of filtered data	+	+	+
S RQ14 Archived data	+	-	+
S RQ15 On-premises and Cloud	-	+	+
Performance			
M RQ21 Scalability	++	+	+
S RQ22 Volume of indexed information	+	++	+
C RQ23 Real-time update interval	+	+	+
Count of fulfilled requirements (+)	7	7	8

Table 5.3: Comparison of Quality requirements

5.3 Results

In terms of the functional requirements that were met, Datadog and the Elastic Stack are differentiated by only one point, as seen in Table 5.4. The Grafana Loki Stack cannot compete with the two other systems, and falls back eight points in comparison to Datadog. If *RF15 Ingest log data later in bulk* or *RQ15 On-premises and Cloud* become a "must-have" criteria, Datadog cannot be used, and the Elastic Stack should be used for the project.

The system requirements differ by a maximum of one point, but as already mentioned, the system requirements depend heavily on a specific project. If the cloud variant of the platforms is used, the system requirements do not play a major role, since only the resources of the log collector have to be taken into account. One on-premise system can already handle a large load of log data, as tested in subsubsection 5.1.2.

When it comes to quality requirements, the advantages and disadvantages of Datadog and Grafana balance each other out. Only the Elastic Stack fulfills all the set quality requirements.

Solution	Datadog	Grafana	Elastic
Functional requirements	25	17	24
System requirements	4	3	3
Quality requirements	7	7	8
Sum of fulfilled requirements	36	27	35

Table 5.4: Comparison of all requirements

When summarizing all the requirements, Datadog and the Elastic Stack are the systems that meet the most requirements. Grafana Loki fulfills significantly fewer requirements than the other two systems, and should not be used in the use cases mentioned.

Chapter 6

Conclusion and Future Work

This chapter summarizes the main findings and identifies further steps that could be taken in the future research.

6.1 Conclusion

A big-data logging solution does not only help to store the log data, but the various dashboards help to monitor a system, assess the state of the system and find errors using tools like filters or combined logs of multiple instances. The use of a uniform system, which is comprehensive to other areas of application, allows know-how to be shared across the various use cases.

Another important finding of this work is, that solutions consisting of different components of different communities are limited in interoperability, performance, scalability and security, in comparison to a solution, which covers the whole stack.

Smaller providers of logging systems do not have important features, are not designed for a large amount of data, and do not have the stability needed for such a system.

Based on all the use cases presented in section 2.4 the Elastic 8.x stack is the preferred solution. If the Elastic Stack cannot be used for certain reasons, Datadog should be used as an alternative. Grafana Loki should not be used for big-data logging because it does not meet many of the requirements and the queries of the database take a longer time as the data size increases.

When looking at the use cases individually, the recommended system to be used differs. If *UC01 Finding specific communication sequence* has a high priority, then Datadog should be used since it is the only solution with a transaction feature. *UC02 Watching system changes* and *UC03 Comparison with expected production path* can be best realized with Elastic. If the data in *UC04 Enrichment with metadata* comes from an external system, Datadog should be used. There is the possibility to program a processing plugin for Elastic to realize this feature. If *UC05 Decoupled log analysis* should be necessary, then Elastic must be used, since Datadog has a limited period of time for indexing log events.

6.2 Future Work

There are some additional opportunities to dig deeper into this work. The performance and scalability of Grafana and the Elastic Stack can be examined more closely, especially when using an on-premises system. In addition to the various platforms for scaling, which can be used, the question of load distribution among the individual nodes arises.

With precise data and facts, a better cost estimate can also be made, especially with regard to the cloud and on-premises variant. In the same way, the required bandwidth for the log data transmission could be examined, since this is a problem mentioned by the developers.

The research used for this document at this time used the latest versions and presents the current situation. In the future, technologies will have advanced, and a renewed comparison would be necessary.

List of Figures

List of Tables

List of Listings

Bibliography

[1] *[Fleet] Add support for custom Certificate Authorities, Certificate and Private keys. · Issue #73483 · elastic/kibana*. URL: https://github.com/elastic/kibana/issues/73483 (visited on 07/27/2022).

[2] *3 great engineering roles to apply for this week | VentureBeat*. URL: https://venturebeat.com/2021/08/06/3-great-engineering-roles-to-apply-for-this-week/ (visited on 07/27/2022).

[3] Philip Achimugu et al. "A systematic literature review of software requirements prioritization research". In: *Information and Software Technology* 56 (6 June 2014), pp. 568–585. ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2014.02.001.

[4] *Agent*. URL: https://docs.datadoghq.com/agent/ (visited on 07/27/2022).

[5] *Agents and ingestion tools - OpenSearch documentation*. URL: https://opensearch.org/docs/latest/clients/agents-and-ingestion-tools/index/ (visited on 07/27/2022).

[6] *Alertmanager | Grafana documentation*. URL: https://grafana.com/docs/grafana/latest/datasources/alertmanager/ (visited on 07/27/2022).

[7] George Anadiotis. *DevOps and observability in the 2020s | ZDNet*. Jan. 2022. URL: https://www.zdnet.com/article/devops-and-observability-in-the-2020s/ (visited on 07/27/2022).

[8] *Apache Hadoop*. URL: https://hadoop.apache.org/ (visited on 07/27/2022).

[9] *Apache Kafka*. URL: https://kafka.apache.org/ (visited on 07/27/2022).

[10] *Apache Kafka Documentation*. URL: https://kafka.apache.org/documentation/ (visited on 07/27/2022).

[11] *asf - Revision 1901055: /kafka/site/logos*. URL: http://svn.apache.org/repos/asf/kafka/site/logos/ (visited on 07/27/2022).

[12] Hany F. Atlam et al. "Integration of Cloud Computing with Internet of Things: Challenges and Open Issues". In: IEEE, June 2017, pp. 670–675. ISBN: 978-1-5386-3066-2. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.105. URL: http://ieeexplore.ieee.org/document/8276823/.

[13] A. Terry Bahill and Azad M. Madni. *Discovering System Requirements*. 2017. DOI: 10.1007/978-3-319-43712-5_4. URL: http://link.springer.com/10.1007/978-3-319-43712-5_4.

[14] *Beats: Data Shippers for Elasticsearch | Elastic*. URL: https://www.elastic.co/beats/ (visited on 07/27/2022).

[15] *Brand Guidelines · OpenSearch*. URL: https://opensearch.org/brand.html (visited on 07/27/2022).

[16] *Breaking changes in 7.13 | Beats Platform Reference [7.17] | Elastic*. URL: https://www.elastic.co/guide/en/beats/libbeat/7.17/breaking-changes-7.13.html (visited on 07/27/2022).

[17] *Bring Structure to Your Logs with Custom Parsing on LogDNA | Mezmo*. URL: https://www.mezmo.com/blog/bring-structure-to-your-logs-with-custom-parsing-on-logdna (visited on 07/27/2022).

[18] Tucker Callaway. *LogDNA is now Mezmo | Mezmo*. May 2022. URL: https://www.mezmo.com/blog/logdna-is-now-mezmo (visited on 07/27/2022).

[19] Carlos, Gonçalves Ramiro Santos Leonel, and Rabadão. "Flow Monitoring System for IoT Networks". In: ed. by Hojjat et al. Springer International Publishing, 2019, pp. 420–430. ISBN: 978-3-030-16184-2.

[20] *Chapter 10: MoSCoW Prioritisation | The DSDM Handbook | Agile Business Consortium*. URL: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation (visited on 07/27/2022).

[21] *Chronograf 1.9 Documentation*. URL: https://docs.influxdata.com/chronograf/v1.9/ (visited on 07/27/2022).

[22] *Cloud Monitoring as a Service | Datadog*. URL: https://www.datadoghq.com/ (visited on 07/27/2022).

[23] *Comparing Logging Solutions. Loki vs ELK vs SPLUNK. A Case Study. | by CrashLaker | Medium*. URL: https://crashlaker.medium.com/which-logging-solution-4b96ad3e8d21 (visited on 07/27/2022).

[24] A Costa et al. "Big Data Architectures for Logging and Monitoring Large Scale Telescope Arrays". In: https://doi.org/10.18429/JACoW-ICALEPCS2019-MOPHA032. JACoW Publishing, Geneva, Switzerland, May 2020, pp. 268–271. ISBN: 978-3-95450-209-7. DOI: 10.18429/JACoW-ICALEPCS2019-MOPHA032. URL: https://jacow.org/icalepcs2019/papers/mopha032.pdf.

[25] *Cyclotron*. URL: https://www.cyclotron.io/index.html (visited on 07/27/2022).

[26] *Datadog Application Monitoring*. URL: https://aws.amazon.com/financial-services/partner-solutions/datadog-on-aws/ (visited on 07/27/2022).

[27] *Datadog Service Terms and Agreement | Datadog*. URL: https://www.datadoghq.com/legal/terms/2014-12-31/ (visited on 07/27/2022).

[28] *Datadog vs LogDNA vs Loggly | What are the differences?* URL: https://stackshare.io/stackups/datadog-vs-logdna-vs-loggly (visited on 07/27/2022).

[29] *Datadog, logdna, loggly - Erkunden - Google Trends*. URL: https://trends.google.com/trends/explore?q=Datadog,logdna,loggly (visited on 07/27/2022).

[30] *DB-Engines Ranking - popularity ranking of search engines*. URL: https://db-engines.com/en/ranking/search+engine (visited on 07/27/2022).

[31] *Deploy the InfluxData Platform (TICK stack) in Docker containers | InfluxData Platform Documentation*. URL: https://docs.influxdata.com/platform/install-and-deploy/deploying/sandbox-install/ (visited on 07/27/2022).

[32] *Dissect processor | Elasticsearch Guide [8.2] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/8.2/dissect-processor.html (visited on 07/27/2022).

[33] *Docker | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/installation/docker/ (visited on 07/27/2022).

[34] *Downloads / InfluxData Branding Docs*. URL: https://influxdata.github.io/branding/logo/downloads/ (visited on 07/27/2022).

[35] *Elastic Products: Search, Analytics, Logging, and Security | Elastic*. URL: https://www.elastic.co/products/ (visited on 07/27/2022).

[36] *Elasticsearch Changes Name to Elastic to Reflect Wide Adoption Beyond Search | Elastic*. URL: https://www.elastic.co/about/press/elasticsearch-changes-name-to-elastic-to-reflect-wide-adoption-beyond-search (visited on 07/27/2022).

[37] *Elasticsearch Service Level Agreement*. URL: https://www.alibabacloud.com/help/en/legal/latest/elasticsearch-service-level-agreement (visited on 07/27/2022).

[38] *Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic*. URL: https://www.elastic.co/elasticsearch/ (visited on 07/27/2022).

[39] *Enrich your data | Elasticsearch Guide [8.3] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/ingest-enriching-data.html (visited on 08/09/2022).

[40] *Eric Allman | Internet Hall of Fame*. URL: https://www.internethalloffame.org/inductees/eric-allman (visited on 07/27/2022).

[41] *FAQ on 2021 License Change | Elastic*. URL: https://www.elastic.co/pricing/faq/licensing (visited on 07/27/2022).

[42] *Features - Apache Solr*. URL: https://solr.apache.org/features.html (visited on 07/27/2022).

[43] *File:Grafana logo.svg - Wikipedia*. URL: https://en.wikipedia.org/wiki/File:Grafana_logo.svg (visited on 07/27/2022).

[44] *File:Prometheus software logo.svg - Wikimedia Commons*. URL: https://commons.wikimedia.org/wiki/File:Prometheus_software_logo.svg (visited on 07/27/2022).

[45] Eli Fisher et al. *OpenSearch 1.0 launches | AWS Open Source Blog*. July 2021. URL: https://aws.amazon.com/blogs/opensource/opensearch-1-0-launches/ (visited on 07/27/2022).

[46] *Fleet Server | Fleet and Elastic Agent Guide [8.2] | Elastic*. URL: https://www.elastic.co/guide/en/fleet/current/fleet-server.html (visited on 07/27/2022).

[47] *fluentbit*. URL: https://fluentbit.io/ (visited on 07/27/2022).

[48] *Fluentd vs. Fluent Bit: Side by Side Comparison | Logz.io*. URL: https://logz.io/blog/fluentd-vs-fluent-bit/ (visited on 07/27/2022).

[49] Qiang Fu et al. "Where Do Developers Log? An Empirical Study on Logging Practices in Industry". In: (2014). DOI: 10.1145/2591062.2591175. URL: http://dx.doi.org/10.1145/2591062.2591175.

[50] *Geoffrey Moore on Twitter: "Thoughts from the week: Without big data analytics, companies are blind and deaf, wandering out onto the Web like deer on a freeway." / Twitter*. URL: https://twitter.com/geoffreyamoore/status/234839087566163968?lang=en (visited on 07/27/2022).

[51] Rainer Gerhards. *The Syslog Protocol*. Mar. 2009. DOI: 10.17487/RFC5424. URL: https://www.rfc-editor.org/info/rfc5424.

[52] *GNU Affero General Public License - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/licenses/agpl-3.0.en.html (visited on 07/27/2022).

[53] Jeffrey O Grady. *System requirements analysis*. Elsevier, 2010.

[54] *Grafana | Grafana Labs*. URL: https://grafana.com/oss/grafana/ (visited on 07/27/2022).

[55] *Grafana Cloud SLA | Grafana Labs*. URL: https://grafana.com/legal/grafana-cloud-sla/ (visited on 07/27/2022).

[56] *Grafana Loki | Grafana Labs*. URL: https://grafana.com/oss/loki/ (visited on 07/27/2022).

[57] *Grafana Loki | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/ (visited on 07/27/2022).

[58] *Grafana® Features | Grafana Labs*. URL: https://grafana.com/grafana/ (visited on 07/27/2022).

[59] Eric Hansen and Robert J. Bush. "Understanding Customer Quality Requirements: Model and Application". In: *Industrial Marketing Management* 28 (2 Mar. 1999), pp. 119–130. ISSN: 0019-8501. DOI: 10.1016/S0019-8501(98)00007-8.

[60] Andrew Hopp et al. *1.0 is released! · OpenSearch*. July 2021. URL: https://opensearch.org/blog/updates/2021/07/opensearch-general-availability-announcement/ (visited on 07/27/2022).

[61] *How to integrate custom logs with Elastic Agent | by Benoit Luttringer | Zenika*. URL: https://medium.zenika.com/how-to-integrate-custom-logs-with-elastic-agent-7f80aef5aec7 (visited on 07/27/2022).

[62] *How to manage Trusted Root Certificates in Windows 11/10*. URL: https://www.thewindowsclub.com/manage-trusted-root-certificates-windows (visited on 07/27/2022).

[63] *InfluxDB vs. Elasticsearch for Time Series Data & Metrics Benchmark | InfluxData*. URL: https://www.influxdata.com/blog/influxdb-markedly-elasticsearch-in-time-series-data-metrics-benchmark/ (visited on 07/27/2022).

[64] *InfluxDB: Open Source Time Series Database | InfluxData*. URL: https://www.influxdata.com/ (visited on 07/27/2022).

[65] *Installation | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/installation/ (visited on 07/27/2022).

[66] *Introduction to Integrations*. URL: https://docs.datadoghq.com/getting_started/integrations/ (visited on 07/27/2022).

[67] *Introduction to Loki: Like Prometheus, but for Logs | Grafana Labs*. URL: https://grafana.com/go/webinar/intro-to-loki-like-prometheus-but-for-logs/ (visited on 07/27/2022).

[68] *Introduction to SEMI SECS/GEM for beginner*. URL: https://www.insphere.com.sg/post/introduction-to-secs-gem (visited on 07/27/2022).

[69] *Kapacitor 1.6 Documentation*. URL: https://docs.influxdata.com/kapacitor/v1.6/ (visited on 07/27/2022).

[70] *Kibana: Explore, Visualize, Discover Data | Elastic*. URL: https://www.elastic.co/kibana/ (visited on 07/27/2022).

[71] Heng Li, Weiyi Shang, and Ahmed E Hassan. "Which log level should developers choose for a new logging statement?" In: 22 (2017), pp. 1684–1716. DOI: 10.1007/s10664-016-9456-2. URL: http://commons.apache.org/proper/commons-logging.

[72] *Log Analysis & Log Management Software for Observability Data | Mezmo*. URL: https://www.mezmo.com/ (visited on 07/27/2022).

[73] *Log Analysis | Log Management by Loggly*. URL: https://www.loggly.com/ (visited on 07/27/2022).

[74] *Log Formats – a (Mostly) Complete Guide | Graylog*. URL: https://www.graylog.org/post/log-formats-a-complete-guide (visited on 07/27/2022).

[75] *Log Management Software - Centralized Logging & Analysis - LogDNA*. URL: https://www.logdna.com/ (visited on 05/17/2022).

[76] *Loggly | API Overview*. URL: https://documentation.solarwinds.com/en/success_center/loggly/content/admin/api-overview.htm (visited on 07/27/2022).

[77] *Loggly | Automated Parsing Log Types*. URL: https://documentation.solarwinds.com/en/success_center/loggly/content/admin/automated-parsing.htm#custom (visited on 07/27/2022).

[78] *Loggly | Loggly Overview*. URL: https://documentation.solarwinds.com/en/success_center/loggly/content/admin/about-loggly.htm# (visited on 07/27/2022).

[79] *Logo - Fluentd*. URL: https://docs.fluentd.org/quickstart/logo (visited on 07/27/2022).

[80] *logo-square.png (800×583)*. URL: https://dashboard.snapcraft.io/site_media/appmedia/2020/02/logo-square.png (visited on 07/27/2022).

[81] *Logos and Assets - Apache Solr*. URL: https://solr.apache.org/logos-and-assets.html (visited on 07/27/2022).

[82] *LogQL | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/logql/ (visited on 07/27/2022).

[83] *Logstash: Collect, Parse, Transform Logs | Elastic*. URL: https://www.elastic.co/logstash/ (visited on 07/27/2022).

[84] Chris M Lonvick. *The BSD Syslog Protocol*. Aug. 2001. DOI: 10.17487/RFC3164. URL: https://www.rfc-editor.org/info/rfc3164.

[85] *Manage multiline messages | Filebeat Reference [8.2] | Elastic*. URL: https://www.elastic.co/guide/en/beats/filebeat/current/multiline-examples.html (visited on 07/27/2022).

[86] Carl Meadows et al. *Introducing OpenSearch | AWS Open Source Blog*. Sept. 2021. URL: https://aws.amazon.com/blogs/opensource/introducing-opensearch/ (visited on 07/27/2022).

[87] *Microsoft Azure Marketplace*. URL: https://azuremarketplace.microsoft.com/en-us/marketplace/apps/datadog1591740804488.dd_liftr_v2?tab=PlansAndPrice (visited on 07/27/2022).

[88] *Migrate from Beats to Elastic Agent | Fleet and Elastic Agent Guide [8.0] | Elastic*. URL: https://www.elastic.co/guide/en/fleet/8.0/migrate-beats-to-agent.html#why-migrate-to-elastic-agent (visited on 07/27/2022).

[89] *MongoDB Brand Resources | MongoDB*. URL: https://www.mongodb.com/brand-resources (visited on 07/27/2022).

[90] *MongoDB: The Application Data Platform | MongoDB*. URL: https://www.mongodb.com/ (visited on 07/27/2022).

[91] *Organize your Azure resources effectively - Cloud Adoption Framework | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-setup-guide/organize-resources (visited on 07/27/2022).

[92] *Overview | Prometheus*. URL: https://prometheus.io/docs/introduction/overview/ (visited on 07/27/2022).

[93] *Parsing*. URL: https://docs.datadoghq.com/logs/log_configuration/parsing/?tab=matchers (visited on 07/27/2022).

[94] Liane Pfeifer. *Syslog; Geschichte; Aussichten; Facility Levels; Schweregrade; Format eines Syslog-Paket*. Dec. 2016. URL: https://web.archive.org/web/20181023120209/http://lebendom.com/article/syslog (visited on 07/27/2022).

[95] Ameer Pichan, Mihai Lazarescu, and Sie Teng Soh. "A Logging Model for Enabling Digital Forensics in IoT, in an Inter-connected IoT, Cloud Eco-systems". In: IEEE, July 2020, pp. 478–483. ISBN: 978-1-7281-6823-4. DOI: 10.1109/WorldS450073.2020.9210366. URL: https://ieeexplore.ieee.org/document/9210366/.

[96] *Pricing | Datadog*. URL: https://www.datadoghq.com/pricing/ (visited on 07/27/2022).

[97] *Processors*. URL: https://docs.datadoghq.com/logs/log_configuration/processors/?tab=ui#geoip-parser (visited on 07/27/2022).

[98] *Prometheus - Monitoring system & time series database*. URL: https://prometheus.io/ (visited on 07/27/2022).

[99] *Promtail | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/clients/promtail/ (visited on 07/27/2022).

[100] Ernie Regalado. *Open Source Monitoring Stack: Prometheus and Grafana - Bizety*. Jan. 2019. URL: https://www.bizety.com/2019/01/25/open-source-monitoring-stack-prometheus-and-grafana/ (visited on 07/27/2022).

[101] *Repositories Ranking - Gitstar Ranking*. URL: https://gitstar-ranking.com/repositories (visited on 07/27/2022).

[102] *Revit Software | Get Prices & Buy Official Revit 2023*. URL: https://www.autodesk.com/products/revit/overview (visited on 07/27/2022).

[103] Fanny Rivera-Ortiz and Liliana Pasquale. "Automated modelling of security incidents to represent logging requirements in software systems". In: *ACM International Conference Proceeding Series* (Aug. 2020). DOI: 10.1145/3407023.3407081. URL: https://doi.org/10.1145/3407023.3407081.

[104] *Running the Elastic Stack ("ELK") on Docker | Getting Started [8.2] | Elastic*. URL: https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-stack-docker.html#get-started-docker-tls (visited on 07/27/2022).

[105] Igor Saenko, Igor Kotenko, and Alexey Kushnerevich. "Parallel Processing of Big Heterogeneous Data for Security Monitoring of IoT Networks". In: *Proceedings - 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2017* (Apr. 2017), pp. 329–336. DOI: 10.1109/PDP.2017.45.

[106] *SECS/GEM in Maschinen integrieren - Mehrwert für Ihr Produkt und Ihre Kunden steigern - Smart Industry*. URL: https://smart-industry.at/secs-gem-integration/ (visited on 07/27/2022).

[107] *SECS/GEM Introduction – by Agileo Automation*. URL: https://secsgem.eu/ (visited on 07/27/2022).

[108] *SEMI E37 High-Speed SECS Message Services*. URL: https://www.cimetrix.com/hsms (visited on 07/27/2022).

[109] *Send Log Lines*. URL: https://docs.mezmo.com/reference/ingest (visited on 07/27/2022).

[110] Jason Skowronski. *Elastic Agent and Fleet make it easier to integrate your systems with Elastic | Elastic Blog*. Aug. 2021. URL: https://www.elastic.co/blog/elastic-agent-and-fleet-make-it-easier-to-integrate-your-systems-with-elastic (visited on 07/27/2022).

[111] *Syslog - Definition and Details*. URL: https://www.paessler.com/de/it-explained/syslog (visited on 07/27/2022).

[112] *Tags · elastic/elasticsearch*. URL: https://github.com/elastic/elasticsearch/tags?after=v0.6.0 (visited on 07/27/2022).

[113] True Tamplin. *What Is Material Control? | Definition, Objectives, Principle and Advantages*. Sept. 2021. URL: https://learn.financestrategists.com/explanation/cost-accounting/material-costing/material-control/#:~:text=Material%20control%20is%20a%20system%20that%20ensures%20the,the%20required%20time%20with%20the%20minimum%20capital%20investment. (visited on 07/27/2022).

[114] *Telegraf 1.22 Documentation*. URL: https://docs.influxdata.com/telegraf/v1.22/ (visited on 07/27/2022).

[115] *telegraf/plugins/inputs/tail at master · influxdata/telegraf*. URL: https://github.com/influxdata/telegraf/tree/master/plugins/inputs/tail (visited on 07/27/2022).

[116] *The ELK Stack: From the Creators of Elasticsearch | Elastic*. URL: https://www.elastic.co/what-is/elk-stack (visited on 07/27/2022).

[117] David Tiede. "Analyse von Anforderungen zur Gestaltung einer Cloud-basierten Lösung im Kontext von Industrie 4.0". Technische Universität Dresden, July 2019, pp. 1–40.

[118] *Trusted centralized log management software | Mezmo*. URL: https://www.mezmo.com/product (visited on 07/27/2022).

[119] *Using the API*. URL: https://docs.datadoghq.com/api/latest/using-the-api/ (visited on 07/27/2022).

[120] Steven Vaughan-Nichols. *Elastic changes open-source license to monetize cloud-service use | ZDNet*. Jan. 2021. URL: https://www.zdnet.com/article/elastic-changes-open-source-license-to-monetize-cloud-service-use/ (visited on 07/27/2022).

[121] *Welcome to Apache Solr - Apache Solr*. URL: https://solr.apache.org/ (visited on 07/27/2022).

[122] *What Are Prioritization Techniques? MoSCow Method*. URL: https://blog.ganttpro.com/en/prioritization-techniques-and-methods-for-projects-with-advantages-of-moscow-model/ (visited on 07/27/2022).

[123] *What is an Embedded System? Definition and FAQs | HEAVY.AI*. URL: https://www.heavy.ai/technical-glossary/embedded-systems (visited on 07/27/2022).

[124] *What is Fluentd? | Fluentd*. URL: https://www.fluentd.org/architecture (visited on 07/27/2022).

[125] *What Is Structured Logging and Why Developers Need It*. URL: https://stackify.com/what-is-structured-logging-and-why-developers-need-it/ (visited on 07/27/2022).

[126] *What is Structured Logging? | Sumo Logic*. URL: https://www.sumologic.com/glossary/structured-logging/ (visited on 07/27/2022).

[127] *Why Use MongoDB And When To Use It? | MongoDB*. URL: https://www.mongodb.com/why-use-mongodb (visited on 07/27/2022).

[128] I-Chen Wu and Chi-Chang Liu. “A Visual and Persuasive Energy Conservation System Based on BIM and IoT Technology”. In: (). DOI: 10.3390/s20010139. URL: www.mdpi.com/journal/sensors.

[129] Ravi Teja Yarlagadda. “Future of Robots, AI and Automation in the United States”. In: *IEJRD-International Multidisciplinary Journal* 1 (5 2015), p. 6.

[130] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. *Characterizing Logging Practices in Open-Source Software*. ISBN: 9781467310673.