Implementation of Centralized Logging and Log Analysis in Cloud Transition

Antti Vainio

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 3.7.2018

Supervisor

Prof. Jukka Suomela

Advisor

MSc Cyril de Vaumas

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Antti Vainio

Title Implementation of Centralized Logging and Log Analysis in Cloud Transition

Degree programme Computer, Communication and Information Sciences

Major Computer Science **Code of major** SCI3042

Supervisor Prof. Jukka Suomela

Advisor MSc Cyril de Vaumas

Date 3.7.2018 **Number of pages** 84 **Language** English

Abstract

Centralized logging can be used to collect log data from multiple log files on multiple separate server machines and transmit the data to a single centralized location. Log analysis on top of that can automatically process large amounts of logs for various different purposes including problem detection, troubleshooting, monitoring system performance, identifying security incidents, and understanding user behavior. As the volume of log data is growing when software systems, networks, and services grow in size, the log data located on multiple separate server machines can be difficult to manage. The traditional way of manually inspecting logs has also become too labor-intensive and error-prone when large amounts of log data need to be analyzed. Setting up centralized logging and automatic log analysis systems can be used to solve this problem.

This thesis explains the concepts of log data, centralized logging, and log analysis, and also takes a look at existing software solutions to implement such a system. The solutions covered in this thesis are ones that are available for download and installation, in order to have more control over the implementation and to allow isolating the centralized logging system inside a local network. A case study is also conducted, where centralized logging and log analysis is implemented for an existing system using some of the software solutions studied in this thesis. The case study focuses on challenges that arise as the case study system is going through cloud transition and when the analyzed log data is mostly unstructured and difficult to automatically parse.

The thesis presents the benefits that the complete centralized logging and log analysis setup has brought for the original system. Additional findings about the implementation process and the finished system are also discussed. While the final setup works well and did not require changes in the original system itself, there are still some aspects that are beneficial to consider when developing a logging system for new software.

Keywords centralized logging, log parsing, log analysis, cloud transition

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Antti Vainio

Työn nimi Keskitetyn lokituksen ja lokien analysoinnin toteutus pilvisiirtymässä

Koulutusohjelma Computer, Communication and Information Sciences

Pääaine Computer Science **Pääaineen koodi** SCI3042

Työn valvoja Prof. Jukka Suomela

Työn ohjaaja MSc Cyril de Vaumas

Päivämäärä 3.7.2018 **Sivumäärä** 84 **Kieli** Englanti

Tiivistelmä

Keskitettyä lokitusta voi käyttää lokidatan keräämiseen useasta lokitiedostosta usealta palvelinkoneelta sekä datan lähettämiseen yhteen keskitettyyn sijaintiin. Tämän lisäksi lokien analysoinnilla voi automaattisesti prosessoida suuria määriä lokeja moniin eri tarkoituksiin kuten ongelmien havaitsemiseen ja ratkomiseen, järjestelmän toiminnan valvomiseen, tietoturvaongelmien tunnistamiseen ja käyttäjien käytöksen ymmärtämiseen. Lokidatan hallitseminen usealla erillisellä palvelinkoneella voi olla vaikeaa, kun lokidatan määrä kasvaa järjestelmien, verkkojen ja palveluiden kasvaessa. Perinteisestä tavasta manuaalisesti tutkia lokitiedostoja on tullut liian työläs ja virhealtis, kun on olemassa tarve analysoida suuria määriä lokeja. Keskitettyjen lokitus- ja lokien analysointijärjestelmien rakentaminen voi ratkaista tämän ongelman.

Tämä diplomityö selittää, mitä ovat lokidata, keskitetty lokitus ja lokien analysointi, sekä tutustuu olemassa oleviin sovelluksiin tällaisen järjestelmän rakentamista varten. Tässä työssä käsitellyt sovelluksen ovat sellaisia, jotka voi ladata ja asentaa omalle palvelimelle, jotta pystyisi paremmin hallitsemaan järjestelmän eri osia, ja jotta olisi mahdollista eristää järjestelmä paikalliseen verkkoon. Työssä tehdään myös tapaustutkimus, jossa toteutetaan keskitetty lokitus ja lokien analysointi olemassa olevalle järjestelmälle käyttäen joitain tutkituista sovelluksista. Tapaustutkimus keskittyy haasteisiin, kun tutkimuksessa käytetty järjestelmä tekee siirtymää pilviympäristöön, ja kun lokidata on suurelta osin ilman rakennetta sekä vaikeasti jäsennettävää.

Työ esittelee hyötyjä, joita keskitetty lokitus ja lokien analysointi on tuonut alkuperäiselle järjestelmälle. Lisäksi työssä käsitellään myös muita tutkimustuloksia liittyen toteutusprosessiin ja valmiiseen järjestelmään. Vaikka lopullinen toteutus toimiikin hyvin eikä vaatinut muutoksia alkuperäiseen järjestelmään, joitain puolia on silti hyvä ottaa huomioon, kun kehittää lokitusjärjestelmää uudelle ohjelmistolle.

Avainsanat keskitetty lokitus, lokien jäsennys, lokien analysointi, pilvisiirtymä

Preface

I would like to thank my supervisor Jukka Suomela for great guidance and useful feedback on the Thesis. I would also like to thank my advisor Cyril de Vaumas for helping with the case study system and for great suggestions regarding the case study. Lastly, I want to thank my family and especially my little brother for all the support I have received.

Otaniemi, 3.7.2018 Antti E. Vainio

Contents

Abbreviations

AIX	Advanced Interactive eXecutive – a Unix operating system
API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
gzip	(an archive file format and a software application)
HDFS	Hadoop Distributed File System
HP-UX	Hewlett Packard Unix
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
IRC	Internet Relay Chat
JSON	JavaScript Object Notation
NoSQL	non SQL
PDF	Portable Document Format
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language
ZIP	(an archive file format)

1 Introduction

The volume of log data has been growing throughout time as software systems, networks, and services have grown in size [19]. When the size of software systems grows to consist of multiple separate server machines each generating log data, there is a problem that log data resides in multiple locations, which could be difficult to manage. As a result, traditional way of manually inspecting logs has become too labor-intensive and error-prone, which then introduces a need to handle parsing and analyzing log data automatically. This is where centralized logging and automatic log analysis becomes relevant as all of the log data can be made available in a single location, and large amounts of log data can be automatically processed and analyzed.

This work concentrates on two closely connected technologies, centralized logging and log analysis systems, by having a look at existing software solutions and how those can be used. Centralized logging considers how log data can be transmitted to a centralized location from a system that is distributed on multiple server machines. Log analysis can then be used on the centralized log data to automate reading and analyzing it in order to extract useful information.

Centralized logging and log analysis are implemented in this work for a case study system that was not originally designed to be distributed on multiple server machines, but is now going through cloud transition to make this possible. This poses many challenges for the implementation, as the system also uses mixed logging standards and unstructured log messages.

The main idea in centralized logging is to have a single place, such as a server, that is dedicated to receive and store log data from multiple sources, so that all log data is available in that single location [26, 43]. It is not complex to centralize log data as it mainly consists of gathering the data, sending it through a network, and then saving it in the centralized location, which are all rather simple operations by today's standards. The log data can be read from log files or it could be sent directly to the software that is set up to receive and centralize the log data. Saving the log data into a persistent storage such as a database can still present some challenges if the amount of data is massive or the data is generated very quickly. This work mainly focuses on how to gather log data from multiple files where some of the log files can also be periodically split to keep the active log files shorter while at the same time retaining older log data in separate files.

The main use cases and motivation for automatic log analysis include problem detection, troubleshooting, monitoring system performance, identifying security incidents, and understanding user behavior, among others [26, 43]. This can be very difficult depending on the format of the log data being analyzed, as this part mainly focuses on trying to understand data that could in some cases be free-form plain text. Simpler log analysis software that expects the log data in certain format will not work properly for log data that does not follow clear standards [25]. Ideally, good analysis software would work with any log data even when the data uses unknown formats or possibly no format at all. If analysis does not seem to be otherwise possible, something could still be achieved with machine learning or anomaly detection, to find messages that occur only rarely for example, as rare messages could potentially

be important. This work focuses more on the harder case of trying to automatically extract useful information from log data that does not always use known formats and sometimes goes without any format at all.

1.1 Objectives

The main goal of this work is to look at existing centralized logging and log analysis solutions, and then implement that system for an existing system in a case study. The case study will consider possible problems that could arise from the implementation process and the final setup. An important part of this goal is to figure out how to automatically extract useful information from the log data of the case study system which is largely inconsistent as it uses many different logging formats and sometimes even no format at all. The case study has a focus on parsing multi-line log messages, detecting problems within the log data, allowing access to important textual log messages, and constructing visualizations that could help with monitoring the system. Additionally, a machine learning system is tested as part of the case study to see if such a system could provide a simple setup to analyze complex log data.

The aim of this work is to help those who are looking for a qualitative comparison of some centralized logging and log analysis solutions. Also, anyone who is planning to set up centralized logging or log analysis themselves can find the case study informative and useful as it considers practical problems with the implementation process. The already existing software solutions covered in this study can all be downloaded and installed by anyone on their own server while other types of solutions are excluded from this study. As this study focuses on the features and capabilities of the software solutions, numerical metrics, such as log data throughput and memory consumption, are not measured.

The case study is done by selecting some of the existing centralized logging and log analysis solutions from the options studied in this thesis. Those solutions are used to build a working setup on top the existing case study system while not having to make any changes to the already existing system. The complete system helps with monitoring and makes important error and warning messages easily accessible in order to facilitate problem detection and troubleshooting overall.

1.2 Structure of the thesis

This thesis consists of three distinct parts. Chapter 2 explains log data, centralized logging, and log analysis in general, and talks about their importance. Chapter 3 looks at existing software solutions for centralized logging, log parsing, and log analysis that could be used in the case study implementation. The third part about the case study consists of multiple chapters.

Chapter 4 introduces the existing system used in the case study. This chapter also sets the goals for the case study and discusses why certain centralized logging and log analysis solutions were selected for the case study implementation. In chapter 5, centralized logging and log analysis is implemented on top of the case study system using the selected software solutions. This chapter also covers problems that arise in

the implementation process, and the benefits and findings of the complete system. Chapter 6 summarizes the whole thesis and discusses further research possibilities related to this subject.

2 Theoretical framework

This chapter gives background information about logging and log data in general, and explains what centralized logging and log analysis is.

Section 2.1 talks about log data in general, what produces it, what kind of data it is, and how it is used. Examples of standard log messages are presented in subsection 2.1.1, and examples of non-standard log messages in subsection 2.1.2. The idea of centralized logging is explained in section 2.2. Log storage and log data transmission that are part of centralized logging are further explained in subsections 2.2.1 and 2.2.2 respectively. The idea of log analysis is explained in section 2.3. Some use cases for log files are covered in subsection 2.3.1 to present problems that could be solved or automated using log analysis. Subsection 2.3.2 briefly discusses log analysis tools and what they aim to achieve. Lastly, section 2.4 summarizes the structure of the whole centralized logging and log analysis system.

2.1 Log data

Log data is generally textual data generated by software systems to record runtime information of the system. Often logs can be the only data available to get detailed runtime information about the system [19, 20]. Log data could be generated by any software such as a computer program explicitly run by a user or a background service that is constantly running on an operating system. Generally, any well-established software or an operating system service writes sufficient log data, but a team of software engineers developing new software have to take this into account as logging does not come automatically and could be executed in many ways. Some things should be avoided, when creating a logging system, such mixing different log formats together and writing unstructured log messages.

Typically, log data consists of textual messages contained on a single line that are printed by a logging statement in the program code. Each log message would contain some common information, like a timestamp and the verbosity of the message, and after that a free-text message that can contain any information [19, 20]. However, software developers can implement logging in any way they want and log messages could contain practically anything without any common parts, but properly implemented software should at least follow some kind of logging standard. Generally though, smaller software and software not meant for widespread use implement only simple unstructured logging or possibly no logging at all. Although log messages are typically contained on a single line, it is also possible that log messages span multiple lines, or to have them represented in formats such as XML or JSON [25].

This kind of log data can be stored persistently in many formats but saving the logs into a file on a disk is a common straightforward way [32]. Saving logs into a file also makes a lot of sense as appending new messages at the end of the file will result in the file containing the messages in chronological order. Choosing this option also makes sense for software that is meant to be used by other people as it does not require additional configuration or database software, and the log files will be easily accessible. There are still other methods of saving the generated log data such as

storing it in a database or transmitting it directly into a log collector solution that could be a part of a centralized logging system.

Software logs are especially useful in debugging while the software is still under development and for troubleshooting when the software is being actively used [25]. System administrators are also some of the heaviest log file users as logs can be used to monitor system health and detect undesired behavior. Log data can also be used for many other purposes such as profiling and benchmarking the software, and it can provide various analysis possibilities such as user behavior analysis.

The volume of log data has been growing in general as time has passed which is understandable seeing how software systems, networks, and services have grown in size. Currently the amount of generated log data has been seen in at least one example to reach about 50 gigabytes, around 120 to 200 million lines per hour [19]. This is where centralized logging and automatic log analysis becomes relevant as the size of software systems grows to consist of multiple machines each generating log data.

2.1.1 Standard log message examples

This section presents some concrete examples of log data that follow some widely used standard format and some log data that was generated by some commonly used software.

This first example contains two log messages that follow the standard syslog [14] format used by the syslog system that is available on Linux and Unix. Generally, as an example, many Linux services that come with the operating system use syslog by default for logging purposes. The message format includes the timestamp, the host name of the system, where the message originated from, and the log message itself. A timestamp is generally an important part of any logging format to know when each event happened.

```
Dec  5 10:25:41 ubuntu kernel: [    0.987121] Freeing unused kernel memory: 1744K
Dec  5 10:25:41 ubuntu systemd[1]: Starting System Logging Service...
```

The following example shows a single message that is using the Common Log Format [13], which is used by many web servers for logging all HTTP requests. The log format works well for its purpose as it contains all the most important information about the received HTTP requests which include the timestamp, the IP address of the client, the HTTP request line, the status code, and the size of the returned object in bytes. The two hyphens can also contain certain additional information that however are rarely present for publicly accessible web sites or services.

```
10.0.2.2 - - [15/Jan/2018:15:14:04 +0200] "GET /frames.jsp HTTP/1.1" 200 4230
```

An extended version of the same log format called the Combined Log Format [13] adds two new fields at the end of the log message. These fields contain the referer and the user agent that are both provided by the client in the header of the HTTP request they sent to the server, but because this data originates from the client, it

cannot be trusted to be truthful. For example, the referer may not contain the web address from which the client came to this page and the client may not actually be using the web browser or the device that they claim to use. Still, the referer and the user agent can often contain interesting and useful data.

```
10.0.2.2 - - [15/Jan/2018:15:35:06 +0200] "GET /frames.jsp HTTP/1.1" 200 4203 "
    ↪ http://localhost/header.jsp" "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv
    ↪ :57.0) Gecko/20100101 Firefox/57.0"
```

The following example is a log message generated by Apache ActiveMQ. This time different parts of the message are clearly separated with a specific delimiter character unlike in the previous examples where some but not all of the space characters were separating different parts of the message. In addition to the timestamp and the message itself, the severity level of the message is also included which can be quite often seen in various log messages. The log message also contains information about the Java class and the thread that generated the message, which could sometimes be useful information to know with Java logs in general.

```
2017-12-01 15:29:45,761 | INFO  | Connector openwire stopped | org.apache.
    ↪ activemq.broker.TransportConnector | Thread-2
```

The following message is the first example of a log format in which each message takes two lines in the log file, and again the message uses a new timestamp format. This is the format that the standard Java logging utility `java.util.logging.Logger` [1] uses by default. The format contains the same information as the previous format except instead of the thread name there is the name of the method that created the log message. Handling multi-line log messages can be very difficult as so many existing log handling solutions expect log messages to be contained on a single line.

```
Dec 01, 2017 3:25:21 PM com.tech.cs.db.DatabaseEngine startProcessing
INFO: Starting import
```

The Java code to output this kind of message can be quite simple:

```
import java.util.logging.*;
public class example {
        private static final Logger LOGGER = Logger.getLogger(example.class.
    ↪ getName());
        public static void main(String[] args) {
                LOGGER.info("Starting import");
        }
}
```

The following example contains a program stack trace that gets automatically generated on error situations when using certain programming languages. This stack trace specifically shows what a typical Java program stack trace could look like. Generally, stack traces are very useful for debugging errors, but otherwise contain way too much information when they are not used for debugging purposes but only

to inform that a certain error occurred. If stack traces are printed into log files containing other messages, it makes parsing the logs much harder as stack traces always span multiple lines whereas other log messages generally do not. At least the format of stack traces always follows some standard which helps parsing the stack traces themselves.

```
Exception in thread "Timer-1" java.lang.RuntimeException: java.sql.SQLException:
    ↪ Data source is closed
        at com.tech.rnd.hco1.masterslave.InstanceRepository.lambda$new$21(
    ↪ InstanceRepository.java:168)
        at com.tech.rnd.hco1.masterslave.InstanceRepository$$Lambda$3/774757641.
    ↪ get(Unknown Source)
        at com.tech.rnd.hco1.masterslave.InstanceRepository.getConnection(
    ↪ InstanceRepository.java:125)
        at com.tech.rnd.hco1.masterslave.ComponentMonitoring.checkForSlavesChange
    ↪ (ComponentMonitoring.java:45)
        at com.tech.rnd.hco1.masterslave.ComponentMonitoring.run(
    ↪ ComponentMonitoring.java:6)
        at java.util.TimerThread.mainLoop(Timer.java:555)
        at java.util.TimerThread.run(Timer.java:505)
Caused by: java.sql.SQLException: Data source is closed
        at org.apache.commons.dbcp2.BasicDataSource.createDataSource(
    ↪ BasicDataSource.java:2016)
        at org.apache.commons.dbcp2.BasicDataSource.getConnection(BasicDataSource
    ↪ .java:1533)
        at com.tech.rnd.hco1.masterslave.InstanceMonitoring.lambda$new$21(
    ↪ InstanceMonitoring.java:112)
        ... 7 more
```

Caught exceptions can be printed out in Java inside the catch-block:

```
catch (Exception ex) {
        ex.printStackTrace();
}
```

2.1.2 Non-standard log message examples

The rest of these log data examples are all from the case study system that will be used in later chapters. These log messages do not follow any specific format, and because the system is quite large and consists of many separate parts, it writes log data to multiple different log files using many different formats. Quite often many different formats were used even inside a single log file.

The following example is a very simple one that does not contain even a timestamp, which is something that could be considered to be the minimum required information for each log message. However, the log message does contain the severity level of the message.

```
[WARNING] Requested parameter value was NULL: statistics_server
```

The following three examples also use very simple formats that mostly consist of the message and the timestamp with yet again a new format. The second message also contains a tag after the timestamp and in the third message the tag is replaces with the severity level. In addition, the timestamp is slightly different in the third message which can make it more difficult to automatically parse these messages when encountered together.

```
20171215 154429.986 Timer-0-ComponentManager:connect() - cached connection NOT
    ↪ CONNECTED!
```

```
20171204 105507.220 GRC: memory_alarm_limit=0.1
```

```
20171201 152636 WARNING: com.tech.rnd.hco1.ui.BaseManager log: Using character
    ↪ set ISO-8859-1
```

The following two messages again use a different timestamp and also use the semicolon character as a delimiter to clearly separate different bits of information. The second message is just an extended version of the first one that contains some additional information. Also, the second message contains some information whose meaning is not clearly apparent. For example, it would be impossible to figure out what the number `-1` right before the last delimiter means without knowledge of how the logging in this specific system is implemented. In this log format the number is always `-1` and has no meaning.

```
2017-12-01 16:53:38;Monitor started (pid 2849)
```

```
2017-12-01 15:26:36;ui.servlet.ServletBase.initMaintenance;localhost-startStop-1;
    ↪ INFO;-1;Maintenance log initialised
```

The next example shows a short snippet from a single log file. The log file indeed contains these three log messages, each with their own format, on three consequent lines. Automatically parsing this kind of log file would be very difficult as also the message format has to be detected for each line before meaningful data can be parsed.

```
Marking as a core component
 INFO | Successfully connected to tcp://HTWM1:48312?wireFormat.
    ↪ maxInactivityDuration=0&wireFormat.tcpNoDelayEnabled=true&wireFormat.
    ↪ maxInactivityDurationInitalDelay=30000
20171201 152635.591 Found 2 KFRs to re-distribute requests to
```

This last example shows a single log message that is clearly supposed to be on a single line but due to a bug or an oversight, the message has been divided to two lines. Trying to automatically parse log files that contain this kind of anomalies is definitely challenging. Ideally the log parser should be so robust that it can detect these situations, but if the log file also mixes different log formats on different messages like was seen in a previous example, it could be impossible to properly handle this kind of anomaly.

```
2017-12-01 16:55:39.893;taskengine.eb.a;UM(nwu2)_Thread(0) [umId: nwu2,
    ↪ loginAttempts: 0, dirty: false, channel: 1, ioLog: 0, sessionId:
    ↪ HCO_htech_nwu2_16, sessionState: 4 session shutting down (linger timeout)]
    ↪  (time: 1512140139893);SEVERE;-1;IO Exception, when writing output line: <
    ↪ Run logout logout
>, to macro server: Stream closed
```

It is important to note from these examples that even a single software system can write to multiple different log files and use many different ways to format log messages. Therefore, trying to automatically parse logs from even a single large system to extract meaningful information can be very complex. However for humans, reading these log files with various message formats is not very difficult as it is not so hard to figure out most of the different log formats just by looking at them. Traditionally these log files have only been read by humans, which is why this has become the current situation.

2.2 Centralized logging

The main idea in centralized logging is to have a single place, such as a server, that is dedicated to receive and store log data from multiple sources, so that all log data is available in a single place [26, 43]. In a network of multiple servers that each produce log data, it is easier to access any log when they are in a single location instead of separated on multiple machines, which is the main motivation for centralized logging. This way logs would not be lost either, if one of the log producing servers is lost. Centralized logging is also likely going to be even more important in the future as the Internet grows and services are run in cloud environments consisting of multiple virtual machines. If and when the so-called Internet of Things grows to become more prominent, centralized logging could in some cases be a good solution to monitor those devices as has already been seen in some studies [2, 11, 9].

A centralized logging solution mainly consists of multiple log generators and the log collector system [30]. Log generators include all the machines that are generating the interesting log data that should be accessible in a single location instead of being separated on multiple machines. The log collector could be a single server receiving the log data or it could consist of multiple servers that can all receive log data. Multiple log collectors can be used to support a larger log volume if the amount of generated log data is very large. Other reasons for having multiple log collectors include fault tolerance and high availability so that there are backup collectors for cases when some collectors fail or become unavailable [43]. Using just a single log collector would introduce a single point of failure in the centralized logging solution which may not be acceptable depending on the use case.

2.2.1 Log storage

Log storage is also an integral part of a centralized logging system which should be handled by the log collector system. Storing the log data could be done in multiple different ways which also depends on the format of the log data [30]. Generally, the

log data can be considered to be structured with a clear format, or on the other hand unstructured when there is no format. The data could also be inconsistent, with multiple different formats or sometimes having structure and sometimes not.

It would always be possible to store the log data exactly as it is received but a more suitable storage method could be selected if the log data is well-structured [25]. As an example, a well-structured log file would contain a single log entry on each line and each entry would contain a time stamp and a severity level of the message in addition to the message itself. This kind of log data could be easily stored into an SQL database with separate columns for the time stamp, severity, and message. For semi-structured data, a NoSQL database could also work better than storing the log data as is. This could facilitate searching log data from a certain timespan or logs with certain severity for example.

With log storage, there are also certain characteristics that should be taken into consideration when planning a centralized logging system. These are persistence of the log data and storage capacity, write and read access to the log collector, and security [30].

It should be decided how long the log data should be kept in the persistent storage as it is going to affect the amount of total capacity needed along with the speed at which the log data is generated. If storage cost is going to be a limiting factor, it will determine the amount of available storage capacity and also how long the old log data can be kept in the persistent storage. This time could also be extended by compressing the old log data or by only keeping some important or aggregated data instead of everything.

Write access or sending logs to the log collector is going to be limited by the bandwidth and latency of the network connection. If the amount of written log data is able to exhaust the bandwidth, the log collector could be moved closer to the log generators in the network or the log collector system could be scaled horizontally with new machines if that is allowed by the existing system. Read access mainly considers how the stored log data can be accessed, for example, whether the used storage system allows for random or sequential access of the log entries and what is the performance of the system.

Finally, the security concern of the log storage system has a focus on access control. This means that the logs should be stored in such a way that unauthorized people cannot get access to the data and that the data in the storage cannot be retroactively modified. This aspect should be takes especially seriously if the log data could contain sensitive data, such as personal information, that should not be revealed to everyone.

2.2.2 Log data transmission

The method of transmitting log data from the log generators to the log collector also has certain aspects to consider which include compression, encryption and authentication, and integrity [39].

When the log data is transmitted through a network, it may be a good idea to compress the data before sending it in order to save bandwidth. Because log

data is usually plain text, it should compress very effectively. Compression and decompression does introduce some additional CPU load and can cause some slight delay before the log data becomes available in the centralized location. Therefore, if the whole centralized logging system is inside a local network where bandwidth is not a problem, it could be a better option to go without compression.

Encryption and authentication are generally important in communication through a network and should quite likely be also implemented for the log data transmission, which would again be especially important if the logs could contain sensitive data. Encryption is required so that third parties could not read the transmitted log data from intercepted network traffic which is similarly important as the access control of the log storage system. Authentication can be used to make sure that the log collector, which receives all the log data, is the real one and not some malicious third party with a fake log collector. Similarly, the log collector could confirm that the log generator sending the log data is authorized to do so, so that the data cannot be polluted with log data sent from a malicious third party. As with compression, it could be possible to leave encryption and authentication out to save CPU resources and configuration hassle if the whole system is inside a properly secured local network, although additional layers of security are always a good idea.

Lastly, integrity of the transmitted log data could also be a concern although it would not be needed to detect tampering of the data during transmission if encryption was already used. If TCP was used for transmission, it would already handle integrity when transmitting the data through the network. Apart from that, there is likely no need for additional integrity checks although additional integrity could be achieved by using a compression format that supports integrity checks, such as ZIP.

2.3 Automatic log analysis

The main motivation for automatic log analysis is to have a system that is able to automatically do some actions, such as report error conditions, based on log data [26, 43]. Especially when more software will likely be deployed in a cloud environment in the future, they could grow to become so big that manual log analysis is no longer feasible and an automatic system becomes necessary. This kind of system could also be able to extract some valuable data from logs that would be practically impossible to acquire through manually reading the logs even if the amount of data was relatively small.

Log analysis could be thought to contain two separate parts: log parsing and log analysis [19, 20]. Log parsing is about preprocessing the log data and trying to find some structure or a better format for the log data, which could be done by finding the time stamp for each log line as an example. As was mentioned before, the log data could have a clear format, no format at all, or be inconsistently formatted. Log analysis then concentrates on discovering some actually useful information based on the log data. This is where preprocessing the log data becomes important as it is much easier to find useful information from log data that has a good format and structure.

2.3.1 Log file uses

The data stored in log files could be used for multiple different purposes which include anomaly detection, monitoring system health and performance, analyzing user behavior and usage patterns, and security [25, 30]. Some of these uses could clearly benefit from real-time analysis of the log data, such as system health monitoring and security, whereas analysis of user behavior is easier when there already exists more data for analysis.

Anomaly detection is a rather generic log analysis method which concentrates on just finding anomalies in the log data, and it could be used on most log data once the anomaly detection system has been trained. The idea is that when the expected contents of the log file is already known to some extent, anomaly detection would be able to report when the log file contains something unexpected, such as a rare error message.

System health and performance monitoring could focus on some software system or even hardware. Hardware monitoring logs could report, for example, power failures or low amount of available memory. Monitoring the system health works better if information about problems can be acquired without much delay so that the problems can be fixed quickly. Therefore, finding the cause for the problems by analyzing the relevant log files in real-time could facilitate fixing those problems faster.

User behavior and usage pattern analysis is mostly relevant for analyzing web server logs as those have many users and their journey through different web pages is clearly visible. This kind of data is primarily useful for making the existing product better once there is enough collected data for the analysis.

Lastly, monitoring security matters is a very important use case for log analysis as it may not be very easy to detect security related problems outside of log files. Security logs could be used to detect system breaches and malicious users, and also to analyze how attackers behave in a honeypot system as another use case. Web server logs could also be used here as it could be possible to track the actions of malicious users and investigate security incidents.

2.3.2 Log analysis tools

There already exist many log analysis tools but many of them only concentrate on analyzing web server or syslog [14] logs as those are clear standards and straightforward to analyze [25]. Even the analysis tools that can work on non-standard log data, mostly expect each log entry to be contained within a single line which can limit the effectiveness of those tools. As an example, multi-line messages would already be introduced if the monitored software prints stack traces among the log data. There also exist some interesting attempts at improving log analysis although these tools may only work in specific settings. One such tool analyzes source code in order to figure out the data structure of the log file [30].

More issues can emerge if the amount of log data is huge as any log analysis tool can only analyze certain amount of data in given time [30]. This can cause big problems especially if there is a need to analyze the data in real-time. Something that could be done to partially solve this problem is to sample the log data and only

analyze a part of the log entries. However, this will introduce another problem that very rare events written to the log file can be entirely missed.

2.4 Summary of a centralized logging and log analysis system

Figure 1 presents an overall setup for a centralized logging and log analysis system and the different components of this setup. The whole system could consist of multiple log producing machines that would all be running the software that produces the log data and the log transmission system that sends the log data to the centralized logging servers. The components of the centralized logging server could in fact be running on different machines as it is not required that the log collector, the log storage, and the log analysis solution are running on the same machine.

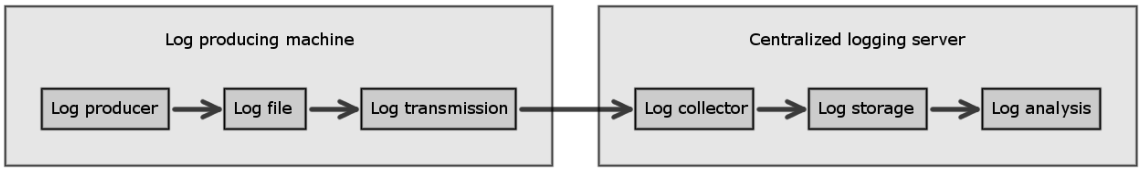


Figure 1: An example centralized logging and log analysis system and its components

This example does not contain a separate log parsing system but the parsing could be done by the log transmission or the log collector system before the log data is stored in the log storage. Some parts of this system are also not necessary in all use cases. For example, the log collector could be left out and allow the log transmission system to directly save the logs to the storage if there is no need to perform any special operations on the log data before storing the data in the database. Log producers could also send the log data directly to a log transmission system, such as syslog [14], and not use log files at all. It could also be possible that the log collector connects to all of the log producing machines to collect log files in which case the log transmission system would not be needed.

3 Existing software solutions

This chapter looks at existing software solutions that are available for centralized logging and log analysis purposes. The choice of the database solution for log data storage is also a concern for a complete centralized logging system but different database solutions are not explicitly looked at in this study. Different database solutions are still mentioned to some extent as different solutions support only certain databases, sometimes only one. The different solutions are mainly compared based on their features, while numerical metrics, such as log data throughput or memory consumption, are not measured.

Only those solutions that could be downloaded and installed on a server machine are taken into account while other kinds of solutions are not considered. The other solutions could include centralized logging and log analysis services that are available on the public Internet, that would receive the log data encrypted over the public network. When the software is available for download and installation, there is more control over the setup and the system can be isolated inside a local network.

Some notable solutions were excluded in this study, including Graylog [22] and NXLog [29]. Graylog has an open source free to use version that offers centralized logging and log analysis capabilities. NXLog also has an open source free to use version that mainly focuses on centralized logging but also has some log parsing capabilities.

Within this chapter, section 3.1 looks at solutions that enable transmission of log data over the network from one server machine to a centralized location. Section 3.2 discusses how couple of the centralized logging solutions can also be used to do log data parsing. Lastly, section 3.3 looks at different solutions for log analysis and visualization. All the software solutions covered in this chapter apart from Splunk were also installed and briefly tested. Some of the solutions were also used in the case study and are further covered in the case study chapter 5.

3.1 Centralized logging solutions

First, a simple cron job based solution is discussed in subsection 3.1.1. After that, proper centralized logging software is looked at: syslog daemons in subsection 3.1.2, Filebeat and Logstash in subsection 3.1.3, Fluent Bit and Fluentd in subsection 3.1.4, and Splunk in subsection 3.1.5. Lastly, the subsection 3.1.6 summarizes the centralized logging solutions.

3.1.1 Simple cron job based solution

Possibly one of the simplest ways to achieve centralized logging would be to set up a cron job [21] that would then use some file transmission utility, such as scp [34, 24] or rsync [40], to periodically copy log files to some server machine working as a centralized log storage location. Setting up this kind of system on the log producing machine could be done very easily by adding a line to the crontab file that could look something like this:

```
0 * * * * rsync -r /home/ubuntu/logdirectory ubuntu@centralized_server_hostname:~
```

This example would copy everything inside `logdirectory` recursively once every hour to the centralized log server. This kind of system would be quick to set up and would not even require installation of separate software, but a system as simple as this one comes with some severe limitations.

Probably the biggest limitation is that logs are only copied at certain time periods which means that the log data does not become available in the centralized location right as the log data is generated but possibly way later. Therefore it will not be possible to rely on the log data for troubleshooting purposes when there is a need to quickly handle problem situations. Additionally, if this method were to be used on ephemeral virtual machines, part of the logs would always be lost when the virtual machine is brought down. This could be especially problematic if the virtual machine is brought down due to an error condition as any log data that could contain information about the error would be lost along with the virtual machine. However, this limitation can be acceptable if permanently losing some data does not cause problems, which could be the case with user behavior analysis as an example, because something like this would be done offline when enough log data has been accumulated.

Another problem that could surface is when there are multiple log generating machines that are running the same software. If these machines just naively copied the similarly named log files to the same location in the log collecting machine, the files would overlap and get replaced. Avoiding this situation would require making separate configurations on each machine or developing some more complex solution that is able to handle this problem. Alternatively, rsync does provide a backing up system in which existing files are renamed instead of replaced when a new file with the same name is copied. In the end though, a system like this can result in a messy file system structure when there are multiple machines generating log data, unless the whole system is very carefully designed.

Finally, this kind of system can reasonably be used to support log data that is written in log files which are going to be similarly stored in log files in the centralized location. If there is a desire to introduce some database solution for the persistent storage, it is beginning to resemble reinventing the wheel as there are existing solutions for this kind of use case, such as Filebeat and Logstash introduced in section 3.1.3, and Fluent Bit and Fluentd introduced in section 3.1.4.

Despite these limitations, there are some upsides too when using scp or rsync. They both have built-in security and authentication once ssh keys have been set up and they both do integrity checks to ensure that files are copied successfully. Additionally, they both also have a support for compressing the transmitted data which is a feature that could be optionally enabled to save network bandwidth if that is required. Scp and rsync also work nicely when copying active log files that are still being written, by simply copying what is in the file at the moment it is copied. Later the file will be replaced with a newer version in the second location unless the rsync backup system is used which keeps previous versions.

In summary, a simple cron job solution can be good enough in some cases, such as in very small-scale when there are only a few machines generating logs and it is not required to have the log data available immediately, and it would not matter either if some of the data could be lost.

3.1.2 syslog daemons

Syslog [14, 36] has become the standard logging system on Unix-like systems, but is not available on Windows by default. It is a logging facility that works separately from the software that is using it for logging purposes, effectively allowing the separation of the logging system that stores the log data into files. In addition, it defines a standard message format which to a small extent forces software to write logs with a better format and also makes it easier to search and analyze the log files.

Using syslog for transmitting log data over the network has been considered in some studies [41, 33] while newer software solutions tend to get more attention for centralized logging purposes. These solutions are also covered later in this chapter: Filebeat and Logstash in section 3.1.3, Fluent Bit and Fluentd in section 3.1.4, and Splunk in section 3.1.5.

In order to write logs to the syslog, it usually has to be done explicitly in the program code. There are routines available to write to the syslog in almost all Linux programming languages, such as C and Python. Simple C and Python code examples to send a message to syslog could look like this:

```
#include <syslog.h>
syslog(LOG_INFO, "log message");
```

```
import syslog
syslog.syslog(syslog.LOG_INFO, "log message");
```

Because of this logging method, adopting a syslog based centralized logging approach for existing software that is already using some other way of logging may not be feasible as all logging related code should be refactored to use syslog. It is still possible to work around this limitation by using the `logger`-command, available in Unix-like systems, which sends any messages that it receives to syslog. Monitoring files is also possible by piping output to the `logger`-command from a `tail`-command that is set to follow the log file. The whole command could look like this:

```
tail -f logfile.log | logger
```

A syslog daemon that is running on the local machine handles receiving log messages and storing them, generally into files. It is also possible to route these message to other syslog daemons running on other hosts and this way allow transmitting log data from multiple different machines to a central location. The daemon can also be configured to take certain actions based on the priority and facility of the log message. Making these configurations will be required as syslog does not route the messages anywhere by default. Additionally, because there may also be other

programs logging to syslog it would be a good idea to filter out the uninteresting log messages.

The original syslog daemon is quite limited however, as it only accepts messages from Unix sockets, cannot route based on regular expression, and has some other limitations too. Therefore, some alternative daemons for syslog have been created that can cleanly replace the original syslog daemon. Two alternative daemons that have been around for a long time and are still being updated are syslog-ng and Rsyslog.

Syslog-ng [28] is an open source alternative syslog daemon that comes with many improvements compared to the original syslog daemon. In addition to the original syslog message standard, syslog-ng also supports JSON and journald message formats. Transmitting log messages is done reliably using TCP and by buffering the log messages on the client machine disk if the centralized logging host happens to be unavailable. Syslog-ng also supports TLS for encryption, X.509 certificates for client and log server authentication, and is also able to use the IPv6 communications protocol. For log storage, syslog-ng supports many SQL databases, the NoSQL database MongoDB, and for big data, HDFS files and Elasticsearch clusters. Additionally, syslog-ng is able to filter log messages using regular expressions and Boolean operators, and even parse and rewrite log messages. A great improvement in syslog-ng over the original syslog daemon is that it can be configured to automatically monitor log files and include any messages added to those files in the syslog system. This improvement makes it actually feasible to use syslog-ng as the centralized logging solution for existing software that uses just log files for logging purposes.

Rsyslog [15] is another open source alternative syslog daemon that could be seen as a competitor for syslog-ng as Rsyslog was released later. Rsyslog also supports basically all the features that were already listed for syslog-ng and the differences between the two daemons are not necessarily very important as they both can act as a proper centralized logging system. Still, Rsyslog is not available on some platforms, including AIX and HP-UX, whereas syslog-ng is.

Choosing one syslog daemon over the other may not even be an issue as some Unix-like systems have switched away from some older daemon as their default option, replacing it completely with either syslog-ng or Rsyslog. In such case, starting to use the available syslog daemon for centralized logging purposes would be straightforward, only requiring configuration.

3.1.3 Filebeat and Logstash

Filebeat [5] and Logstash [7] are both open source and free to use centralized logging software from Elasticsearch BV. In addition to these two solutions, the company also provides Elasticsearch database software [4] that can be used to store the log data. Filebeat and Logstash themselves do not provide any log storage but concentrate on log data transmission instead. They can transmit the log data into other log files but for more structured and distributed storage, the Elasticsearch database is well supported in Filebeat and Logstash as it is from the same company.

Especially Logstash has a lot of attention and is often mentioned [19, 43]. It is

occasionally used for IoT-solutions [2, 11] and also other solutions [42, 32] along with other software including Elasticsearch and Kibana. Kibana is covered later in section 3.3.3.

Filebeat and Logstash are very similar to each other as they can both read log data from various different sources, filter and enhance the log data, and send the data to various different destinations. The overall difference is that Filebeat is much more lightweight and Logstash supports more options and even has plugins for different inputs, filters, and outputs. The suggested configuration for a full centralized logging system using these solutions is to install Filebeat to act as a lightweight log forwarder on each machine that is generating log data. The Filebeat clients would then send the log data to one or multiple Logstash servers that would further process the log data before sending it to the Elasticsearch storage.

There are also other ways to use these solutions depending on the use case and requirements: it would be possible to go without Filebeat and use syslog [14] instead to send the log data from the clients to the Logstash server. It is also possible to do less log data processing by removing the Logstash server layer completely and sending the logs directly to the Elasticsearch storage from the Filebeat clients. Additionally, the clients could forward the log data using Logstash instead of Filebeat which would enable better processing capabilities on the client machines with the cost of additional processing overhead. Lastly, it is possible to replace the Elasticsearch log storage with almost any other database solution especially when using Logstash as it can support almost anything with its output plugin system. Figure 2 visualized all of these different setups with one option on each row.

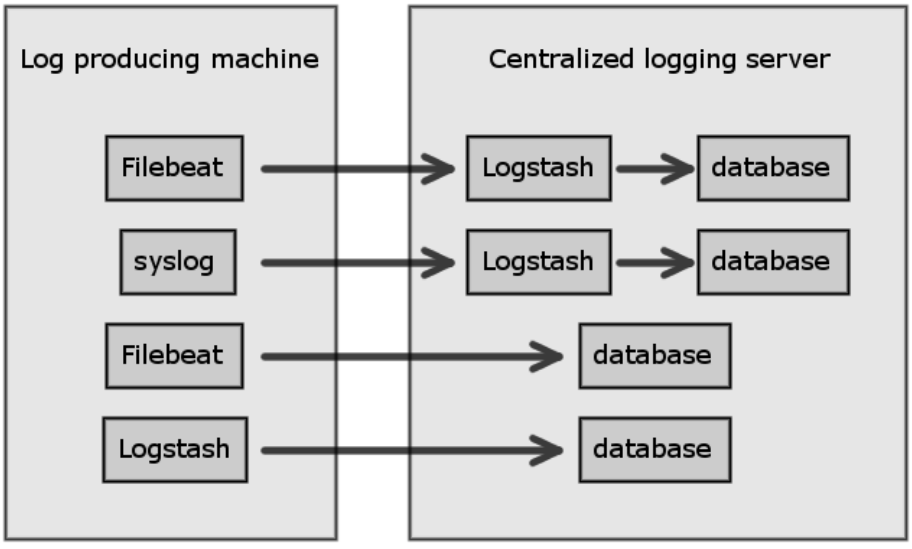


Figure 2: Different centralized logging setups with Filebeat and Logstash

While Filebeat has rather limited log data inputs and outputs, Logstash can send and receive log data from many different sources. The number of supported log collectors and databases can also be extended with plugins, some of which are supported by Elasticsearch BV. Logstash additionally supports various protocols

such as TCP, UDP, and HTTPS for input and output, which also makes it possible to write custom software to receive log data from Logstash. It is also possible to make configurations to execute commands to periodically generate new log data and receive log messages, or to use pipes allowing input and output of log messages with another program. Logstash can even read data from some Internet services including Github and Twitter, send email, and write messages to IRC. However, this list only includes features and plugins listed in the Logstash documentation and more plugins could be created by anyone.

Filebeat and Logstash can both support compression, encryption, authentication, and integrity when transmitting log data but this also depends on the used transmission method as they both support multiple methods. The communication between Filebeat, Logstash, and Elasticsearch has support for all of that as these solutions are developed together. Filebeat also guarantees that all messages are sent at least once by keeping the delivery state of each event in a registry file. Logstash can also be configured to save queued messages on disk to prevent losses in case of crashes but by default Logstash saves in-flight messages only in memory which can cause data loss.

3.1.4 Fluent Bit and Fluentd

Fluent Bit [10] and Fluentd [12] are also both open source and free to use centralized logging software originally from Treasure Data. Fluentd is now a hosted project under the Cloud Native Computing Foundation. Fluent Bit and Fluentd are very similar to Filebeat [5] and Logstash [7] respectively, essentially being log data transmission software that again require some additional database system for log storage, unless outputting the log data directly to files in the centralized location is enough.

Fluent Bit and Fluentd are not often used in studies and are usually not mentioned as alternative options. Still, Fluentd was used along with Elasticsearch [4] and Kibana in at least one study to implement a system for streaming big data [18], and was also mentioned in another study [43]. Kibana is covered later in section 3.3.3.

The difference between Fluent Bit and Fluentd is again that Fluent Bit is much more lightweight but only supports certain log inputs and outputs whereas the capabilities of Fluentd can be extended with plugins to support additional log sources and database systems. Fluentd also has better log filtering and processing capabilities compared to Fluent Bit.

Comparing to Logstash, Fluentd does not have as many input and output plugins out of the box in a clean installation but Fluentd currently has about three times as many plugins listed on their website compared to the plugins listed in the official plugin repository of Logstash. However, less than one tenth of the Fluentd plugins are certified which means that the plugins were developed by Fluentd core committers or companies that made commercial commitment to Fluentd project. So, it can be easier to find niche plugins for Fluentd but the people behind Fluentd have not spent as much effort to ensure the quality of plugins. Fluentd also uses less memory compared to Logstash [31].

Comparing Fluent Bit and Filebeat, Fluent Bit offers much more sources for log

input which even includes generating system performance metrics such as CPU and memory usage. Filebeat does not offer performance metrics but there is another software, Metricbeat from Elasticsearch BV, that does. Fluent Bit combines these features into a single solution. For log output destinations, Fluent Bit also offers a few more options compared to Filebeat and they both support output to Elasticsearch, so replacing Filebeat in that case would be possible.

As with Filebeat and Logstash, there are multiple different ways to set up a centralized logging system using Fluent Bit and Fluentd. The recommended way is to use Fluent Bit to gather the logs on the machines that are generating the log data and send them to server machines running Fluentd to further process the logs and send them to some log storage system. All the alternative methods already listed for Filebeat and Logstash also work with Fluent Bit and Fluentd. This means that Fluent Bit on the log generating machines could be replaced with a syslog solution [14], the server layer with Fluentd could be removed as Fluent Bit can output directly to certain databases, and finally, Fluentd could be used in place of Fluent Bit to collect the log data. As Fluentd uses less memory compared to Logstash [31], going with the simple solution to just use Fluentd everywhere and not bother with Fluent Bit or Filebeat could be a feasible option more often. Figure 3 visualized all of these different setups with one option on each row.

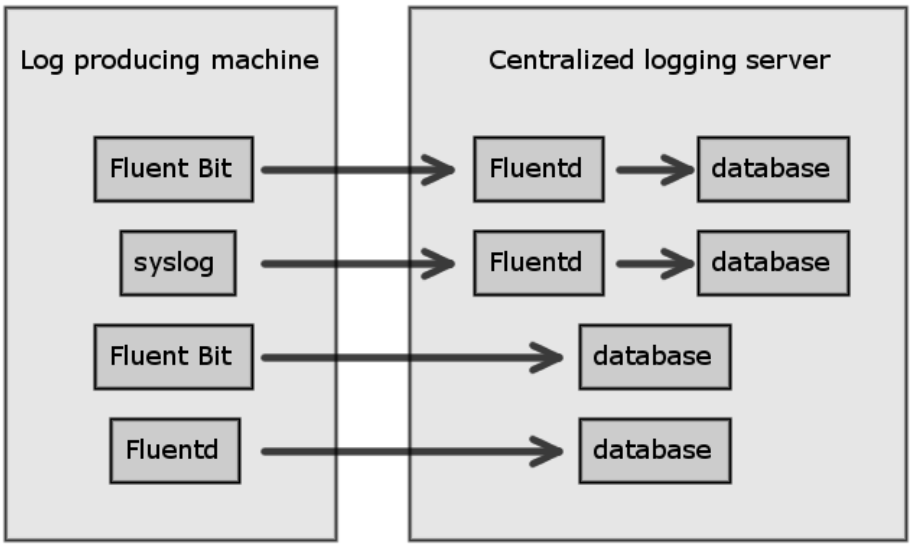


Figure 3: Different centralized logging setups with Fluent Bit and Fluentd

Fluent Bit and Fluentd can also support compression, encryption, authentication, and integrity for transmitting log data but as with Filebeat and Logstash this depends on the used transmission method as they both support various methods. These are supported in communication between Fluent Bit and Fluentd and with certain storage solutions, which includes Elasticsearch like was already the case with Logstash. Fluent Bit and Fluentd can both be configured to also buffer log data on disk so that log messages and are lost in case of crashes.

3.1.5 Splunk

Unlike the previously seen centralized logging solutions from Elasticsearch BV and Treasure Data, the products from Splunk [23, 35] are not open source and are primarily commercial solutions. A free version of Splunk is also available but compared to the other free centralized logging systems, it is much more limited in its features. For example, the maximum daily log indexing volume is only 500MB whereas Logstash [7] and Fluentd [12] can be used without limits.

Splunk is often mentioned when talking about logging solutions in general [19, 25, 30, 43], so out of the commercial options it generally gets more attention. Splunk is also occasionally used for various solutions including IoT-solutions [9], and sometimes it is mentioned as an alternative for a setup consisting of Logstash, Elasticsearch, and Kibana [2, 42]. Kibana is also covered later in section 3.3.3.

Splunk differs in quite a major way from the other centralized logging solutions in that it is not primarily just supposed to be only a system to centralize log data. Splunk always comes with its own database system to store the log data and strong indexing and analysis capabilities for the data. This way Splunk handles the whole centralized logging and log analysis stack on its own and does not need a separate system for log analysis. It is important to take this into account when considering Splunk, as it is much harder to mix and match different software with Splunk and some of the value that Splunk provides would be lost.

As with the other solution, Splunk also offers a very light software version, called the Splunk Universal Forwarder, that is recommended to be used on the log generating machines to collect the log data and to forward it to a proper Splunk instance that can further process the data. The heavier Splunk instance can also be configured to act as a lighter log forwarder because certain Splunk features can be completely disabled to make it use less resources. Because of this, it is often feasible to go without the Universal Forwarder and instead use normal Splunk instances and configure them differently on different machines.

The features of Splunk and Splunk Universal Forwarder are in general very similar to the previously seen solutions which include support for encryption and authentication of sent log data, and plugins with a large number of plugins also listed on its website. Integrity of the data transmission can also be guaranteed by Splunk so that all log data is received in the destination at least once. However, a major difference in Splunk is that it does not properly support sending data to any third-party systems or a databases, as Splunk is supposed to provide all log data storage, handling, and analyzing functionality on its own.

As Splunk cannot primarily be used for free unlike Logstash and Fluentd, it should be considered whether Splunk offers something that is desired. The strengths of Splunk may be in its log analysis capabilities, covered in section 3.3.5, and in the fact that it offers everything needed in one package without the need for a separate database solution or log analysis software. Still, Splunk has some features for centralized logging that the other solutions are not providing.

While the other centralized logging solutions seen earlier can only be configured by editing configuration files, Splunk also provides a command line interface and a

web interface that can be used to modify the configuration in a simpler way. The configuration file is still there for more complex configurations. The Splunk Universal Forwarder and the full Splunk software share the same configuration file format and it is even possible to copy the configuration file from the Universal Forwarder to a full Splunk instance and this way convert the light forwarder to a heavier forwarder that provides more capabilities. On the other hand, none of Logstash [7], Filebeat [5], Fluentd [12], and Fluent Bit [10] use the same configuration file syntax which makes configuration more difficult. Some concrete examples of these configuration files can be found in the case study chapter in sections 5.1.1 and 5.1.2. In addition, the other solutions also need separate software for log storage and log analysis which again need to be configured separately. This is not an issue with Splunk as it already contains its own solutions for log storage and analysis. All in all, when only the centralized logging aspect is considered, Splunk mainly offers an easier way to configure and set up the whole system.

3.1.6 Summary of centralized logging solutions

This chapter looked at some centralized logging software solutions that a system admin could set up on their own server. A simple cron job [21] based solution could work in some cases to transmit log files, when there is no requirement for certain features such as database storage or instant availability of the log data in the centralized location. However, if the log data is important and should be quickly available, it makes sense to use proper centralized logging software that is constantly running in the background.

In the end, centralized logging is relatively simple as it is mainly about transmitting log data from one place to another. As a result, the different solutions do not differ much from each other in their features. All of the software options can receive log data directly from the log producing software or alternatively read log files, and then send the data over the network to the centralized location while also supporting compression, encryption, and authentication in the transmission. The centralized logging solutions can output data only to certain database solutions for persistent data storage and the supported databases vary across the solutions, which will limit the options if a specific database will be used in the centralized logging setup.

Log parsing and filtering is also possible with the centralized logging solutions which could be useful before sending the log data further into the analysis or visualization system. Therefore, the choice for the centralized logging solution could be influenced also by log parsing requirements, in addition to the database choice. The final thing to note is that Splunk [23] is the only primarily commercial solution whereas the rest are free to use and open source.

The supported output channels for log data, such as databases and protocols, usually vary a lot between the different centralized logging solutions. Generally, more than ten output channels are supported in each solution with the possibility to support even more through third-party plugins. Exceptions are Splunk which uses its own database solution, and Filebeat [5] that only supports a couple databases and does not have third-party plugins. Disregarding Splunk, only the Elasticsearch

database [4] is supported in every solution and Apache Kafka in all but one. Together these solutions support over 50 databases and protocols.

3.2 Log parsing solutions

Log parsing focuses on understanding the format of the log data and extracting pieces of information from a log message. Log parsing could also focus on detecting multi-line log messages from a stream of log data where a line break does not necessarily start a new log message. It is not always quite necessary to explicitly parse log messages depending on which log analysis and visualization solutions are used. Machine learning solutions can often work with very unstructured data whereas some visualization solutions have very weak parsing capabilities.

This section looks at the log parsing capabilities of some of the centralized logging solutions. Filebeat and Logstash are covered in subsection 3.2.1, and Fluent Bit and Fluentd in subsection 3.2.2.

3.2.1 Filebeat and Logstash

Filebeat [5] and Logstash [7] both handle logs so that each single or multi-line log message is stored in a single log event that consists of fields containing various data, usually at least a timestamp, the source log file, and the log message itself. In general, Filebeat and Logstash can be used to modify these log events and existing data fields, and to delete and create new fields. This way it is possible to parse log messages by storing the data extracted from a log message to separate fields. Parsing the log data is mainly done using regular expressions but certain generic data formats, like CSV, JSON, and XML, can also be handled automatically in Logstash and JSON in Filebeat. The configurations to parse log data and to handle log events have to be done manually as Filebeat and Logstash cannot automatically detect log format and do not provide machine learning capabilities. If the log data uses many different custom formats, defining robust configurations can be a huge task.

Filebeat can parse log data to some extent but it is mainly supposed to be used only to collect log data from log files and transmit it elsewhere to be further parsed and stored. Filebeat can still be used very effectively to detect multi-line log messages, such as Java stack traces, and it should always be used to do this. This is because the correct ordering of the log messages will be lost when Filebeat transmits log data which makes it impossible to correctly detect multi-line log messages after the log messages have left Filebeat. To parse logs, Filebeat can mainly just add tags to log events and drop certain events based on the contents of the log message, the source log file or other such information. Extracting specific fields from a log message is not quite possible for an arbitrary log format but Filebeat does have a built-in support to automatically parse certain standard log formats which include the Apache2 and Nginx HTTP server logs.

If there is a need to properly parse the log data for log formats that Filebeat does not support, Logstash provides more capabilities for this purpose. Unlike Filebeat, Logstash is actually designed to parse log data and it should be able to extract data

from nearly any kind of log format. Logstash has a filter plugin called grok, that allows to simply list the used log format definitions with regular expressions making it straightforward to parse log data with mixed formats. The grok filter is then able to detect which one of the formats is used for each log message to extract the data in the log message into different fields in the log event.

3.2.2 Fluent Bit and Fluentd

Fluent Bit [10] and Fluentd [12] in general provide very similar log parsing functionalities as Filebeat [5] and Logstash [7] and also handle logs so that each single or multi-line log message is stored in a single log event that consists of fields containing various data. Again, the configurations to parse the log data has to be done manually as there are no automatic format detection capabilities although certain generic data formats like CSV and JSON have built-in parsers in Fluentd and JSON in Fluent Bit. Notably, there is no support for the XML format out of the box like there is with Logstash. As with Filebeat and Logstash, some common log formats, such as the Apache2 and Nginx HTTP server log formats, have a built-in support in both Fluent Bit and Fluentd. Otherwise, complex parsing can be done using regular expressions, and multi-line log messages can be detected in both Fluent Bit and Fluentd.

Compared to Filebeat, Fluent Bit works very much the same way but also offers more log parsing utilities when compared to Filebeat. Fluent Bit is able to fully parse log messages using regular expressions and extract different bits of data from the message to store them into separate fields in the log event, which is a feature that was not available in Filebeat at all. As Fluent Bit practically has all the same functionalities as Filebeat and can also parse logs on its own, Fluent Bit could be a better choice over Filebeat for simpler centralized logging systems that do not use the additional layer of Fluentd/Logstash between the database and Fluent Bit/Filebeat.

Fluentd again works very similarly to Logstash and has mostly the same log parsing functionalities although Logstash comes with a larger number of filtering and parsing plugins out of the box. It is also possible to install a third-party grok filtering plugin for Fluent Bit, which was available in Logstash, making it possible to simply list the definitions of used log formats with regular expressions, so that a matching format is picked from the list for each log message. The built-in regular expression parser is not as flexible which makes the grok plugin an easier option if many custom log formats are used.

3.3 Log analysis and visualization solutions

The log analysis solutions covered here have varying log parsing capabilities where some of the solutions mainly focus on visualizing already parsed data.

Lnav, covered in subsection 3.3.1, differs from the other solutions by being lightweight software run in a terminal, aiming to facilitate reading and following multiple log files. The other solutions are constantly running services that provide a web interface for configuration and visualization. Grafana and Kibana focus on manual analysis and visualization, and are covered in subsections 3.3.2 and 3.3.3

respectively. Splunk and the X-Pack plugin for Kibana also provide machine learning capabilities, and are covered in subsections 3.3.5 and 3.3.4 respectively. Lastly, the subsection 3.3.6 summarizes the log analysis and visualization solutions.

3.3.1 lnav

Lnav [37] is a good example of lightweight log analysis software that gives some comparison baseline of what a log analyzer should be able to achieve. Lnav is open source command line software that provides helpful features for reading logs on a terminal interface compared to other utilities that operating systems usually provide out of the box. Being just a command line tool and quick to set up and use, lnav is likely a good option to consider for small-scale log analysis, or during active development when setting up a full centralized logging and log analysis system would take too much effort.

Once lnav is installed, using it is as easy as just invoking a single command. The command could look like this where each log file directory and separate log file that will be followed is simply listed in the command:

```
lnav logdirectory1/ logdirectory2/ logfile1.log logfile2.log
```

Lnav will then read everything that is already in the files and then continue monitoring the files for new log entries and the directories for new log files. The lnav interface will list all log messages from all the log files mixed together and chronologically sorted in a single view. This makes it straightforward to follow multiple log files simultaneously and see which log messages appeared in different log files around the same time, which helps understand the big picture of how the monitored system is operating. Lnav also improves readability of log messages by highlighting and coloring different parts of recognized log formats. It will even highlight certain things from generic plain text data such as strings, key-value pairs, URLs, and IP addresses. Figure 4 shows how lnav displays log data inside the console window.

In general, lnav expects each log message to be contained in a single line and follow some known format that also has to include some sort of timestamp. Lnav recognizes certain standard log formats and also formats from some software solutions. Even when lnav does not immediately recognize the format, lnav can be configured to recognize these additional formats as long as they are single line and contain a timestamp. Lnav also does support multi-line log messages to some extent by grouping log lines that use unrecognized format with the most recent recognized log message. The additional log lines beyond the first line in a multi-line log message will then be treated as plain text. This means that lnav cannot detect or even be configured to detect things like the severity level from the additional lines after the first one.

Lnav also comes with some other limitations that can make it quite unusable if the log files do not follow good logging standards. While lnav is able to display any single file that contains plain text but does not use any recognizable format anywhere, the file will not be displayed at all when multiple log files are followed simultaneously.

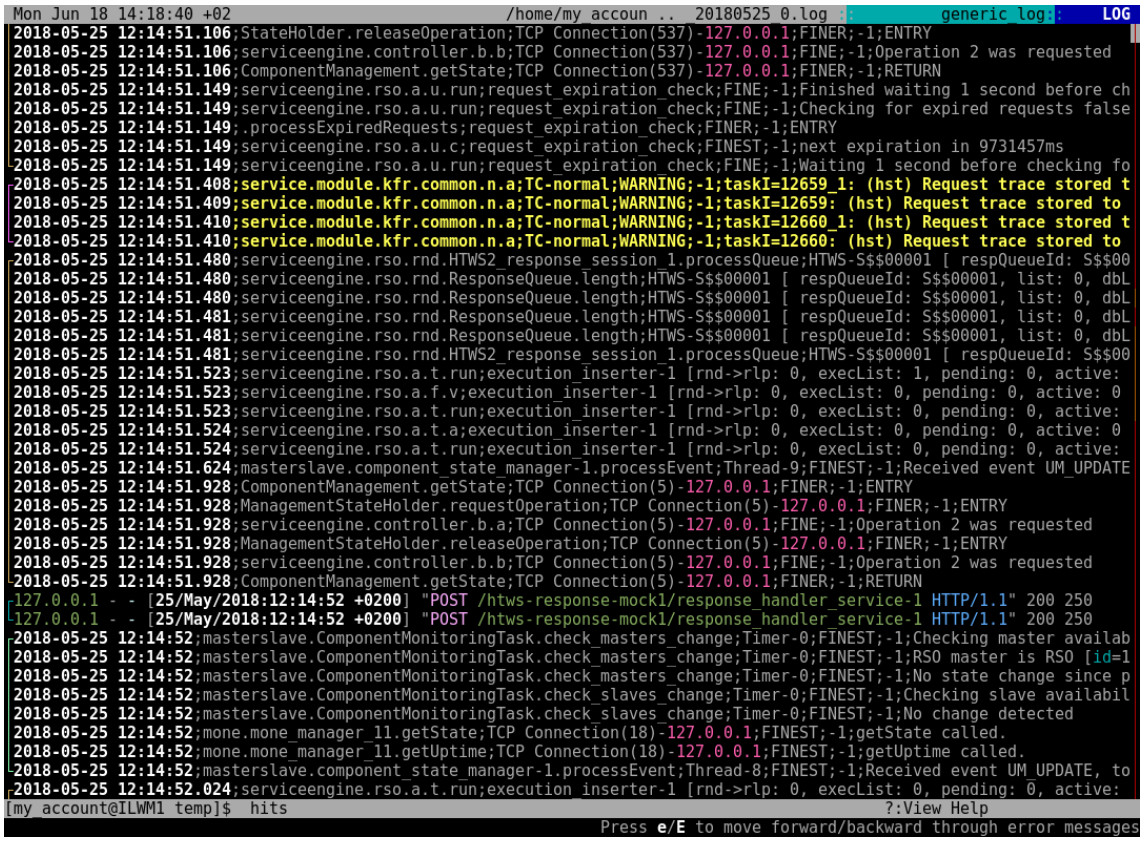


Figure 4: An example of the lnav user interface

This problem can be remedied to some extent by configuring custom log formats but files that do not contain any timestamps at all would still be left out in any case. In addition, lnav can only use one log format to parse one log file. If there are multiple different formats mixed in one log file, only one of them is recognized and the rest of the log messages are treated as plain text. Lastly, lnav trusts the timestamps provided by the log messages themselves even if a new log message appears in the log file at a different time. This means that following multiple log files simultaneously will not work very well if the timestamps are not synchronized as lnav will not be able to correctly order the messages by their actual time of appearance.

Lnav can sometimes also work incorrectly when it is used to read logs that use a format that is not properly supported out of the box. For example, the following log message is incorrectly recognized as an error by lnav because it does not use any actual logging format and happens to contain `erR` in the middle of the message:

```
2017-12-15 15:44:40;requestengine.common.MasterRSO.logKeyEvent;Timer-1;OFF;-1;
    ↪ MasterRSO.masterUnavailable(masterId=1)
```

As lnav can mostly receive its log data only from log files, it requires the logs to be saved in files and not for example in a database. At the same time lnav does not require any centralized logging system to read and index the log files if there is only one log producing server to be monitored. However, it can still be possible to use

lnav even with a centralized logging system as those can usually be configured to write all received logs also into files on the centralized machine. In the end though, lnav is probably most useful for only quick and small-scale purposes.

3.3.2 Grafana

Grafana [27] is open source and free to use analytics and monitoring software from Grafana Labs with the main goal of visualizing data by providing time series analytics and aggregated information in various different charts. Grafana runs its own web server and displays all of the visualizations in a web interface. Displaying raw log data is not very well supported in Grafana as it focuses on visualizing the data using graphs and charts. It is still possible to get the original log data visible in the Grafana interface but filtering and browsing the displayed data can be done only by manually writing database queries. As such, Grafana can mainly be considered for cases where access to the raw log data is not required, and time series and aggregated metrics can provide enough information.

Grafana is not often used in studies and is usually not mentioned as an alternative option. Still, Grafana has seen some use [3] and is also sometimes mentioned [2, 43].

Some Grafana configurations, such as the web server configurations, are done by modifying a configuration file, but otherwise the data visualization configurations are done mainly in the web interface provided by Grafana. Additionally, visualizations can be configured using an HTTP API provided by Grafana or by modifying configuration files. Grafana is able to read data from many different databases, including some time series focused databases, some SQL databases, and also Elasticsearch [4] which has been mentioned before. Creating complex data visualizations in Grafana requires knowledge on how to make proper queries to the database that contains the visualized data because the specific queries have to be typed out as part of Grafana configurations. As there are so many ways to query the different databases supported by Grafana, flexibility is achieved by allowing the user to define the exact query that is sent to the database.

In Grafana, visualizations are contained inside so-called dashboards that can contain multiple charts, and to make dashboard configuration faster, it is possible to export and import these dashboards in JSON format. Grafana Labs also lists and hosts some dashboards on their website for various common log analysis purposes. However, attempting to use dashboards made by others may not be easy or even possible as dashboards require usage of a specific database because the database queries are part of the dashboard configuration. Additionally, the data stored in the database should follow a specific format as otherwise the queries would not be able to fetch the data properly. Because of this, there would not exist database configurations to visualize arbitrary data. The available configurations are mainly for visualizing some common data formats stored in one of the databases supported by Grafana, for example Apache HTTP server access logs stored inside Elasticsearch. Therefore, if there is a specific need to visualize logs from some popular server software, it may be possible to do it very quickly by just importing a dashboard and using the specific database required by the dashboard. Figure 5 shows an example of a dashboard in

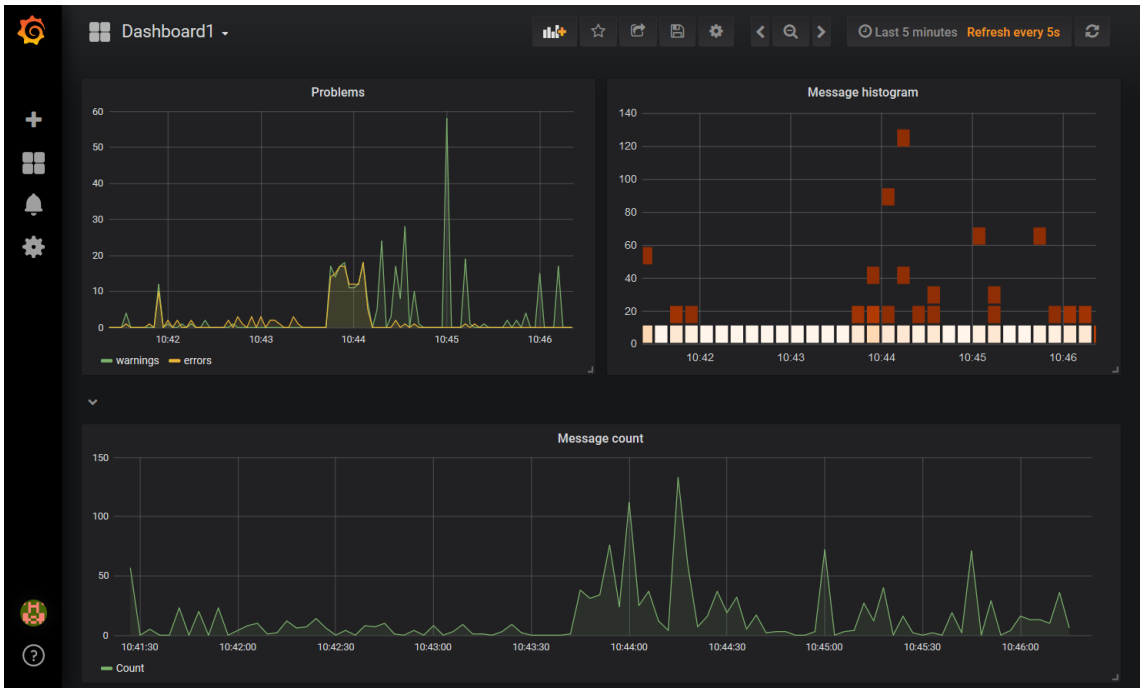


Figure 5: An example of a Grafana dashboard

Grafana containing couple different visualizations.

Grafana also provides various other features including user permissions, alerting, and plugin support. The user permissions allow there to exist separate users who can create and edit dashboards, and other users who can only view. These permissions can also be defined for each dashboard separately, allowing certain users to view only specific dashboards for instance. Grafana also makes it possible to automatically send alerts through various channels, such as email or group chats like Microsoft Teams and Slack, when certain events occur. The alerting rules can be configured for any chart when the metric followed by the chart acts in a specific manner. Lastly, Grafana has a support for third-party plugins that can introduce new visualization panels, and allow Grafana to support additional databases and data sources. Grafana Labs again has a list of some available plugins on their website, and installation could be done in a simple way by giving the plugin name for a command line tool that comes with Grafana.

All in all, Grafana specifically focuses on visualizing time series and aggregated data in graphical charts. Configuring the dashboard and the charts can take a lot of time when the visualized data does not follow some common format as the configurations have to be done manually. However, setting up a visualization for a common use case can be very fast with already existing dashboards.

3.3.3 Kibana

Kibana [6] is open source and free to use software from Elasticsearch BV and in many ways similar to Grafana [27], aiming to provide data visualizations for data

that is stored in an Elasticsearch [4] instance. Indeed, Kibana does not directly support other database solutions apart from Elasticsearch although it can be possible to make complex setups that also use other database solutions [17]. As Filebeat [5], Logstash [7], Elasticsearch, and Kibana are all from Elasticsearch BV, they integrate well together. Kibana also runs its own web server and displays the visualizations in the web interface in which new visualizations can also be configured. Unlike Grafana, Kibana does not have as much focus in visualizing data in graphs and charts, and it provides very good tools to search and browse the textual data stored in the database. While Kibana still requires knowledge on how to write queries to browse the database, Kibana has a proper interface to display the raw JSON format messages stored in the Elasticsearch database, which helps writing proper queries as it is easier to understand the structure of the data. The browsing interface also makes it possible to quickly find messages from certain time spans without having to write specific queries for that purpose. This way compared to Grafana, Kibana is better suited for the use cases in which the actual content of the log data is also very important and should be accessible and easily browsable.

Kibana gets a lot of attention and is sometimes just discussed [43] but often used for various purposes [42, 17, 18, 32] and also IoT-solutions [2, 11]. Quite often Kibana is used with Logstash although other centralized logging software could also be used.

Some Kibana configurations, such as the web server configurations, are done by modifying a configuration file, but otherwise the data visualization configurations have to be done in the web interface provided by Kibana. Apart from that, Kibana does not provide additional APIs or configuration files for configuration like Grafana, but it is possible to export and import visualizations and dashboards like it is also in Grafana. While the visualizations in Kibana are configured in a similar manner, it can be easier than in Grafana, as queries can be written using the simpler lucene query syntax instead of having to write proper database queries. This is possible because Kibana focuses solely on using Elasticsearch as its database solution instead of supporting multiple different databases.

As was already briefly mentioned, Kibana also uses dashboards to present multiple charts and to list log messages in a single page. While it is possible to export and import charts and dashboards also in Kibana, Elasticsearch BV does not list ready-made dashboards for common use cases anywhere. As such, this functionality is mainly useful when reinstalling Kibana or copying configurations to another Kibana instance using the same database. Compared to Grafana, it would still be easier to share dashboards with others as the database would always be Elasticsearch and only the format of the data stored in the database would need to match. Some examples of the Kibana interface can be seen in the case study chapter in section 5.3.2.

Kibana does not provide many additional features such as user permissions or alerting out of the box but instead provides these features through a plugin, although not free of charge. This plugin, called X-Pack, is covered in the following section 3.3.4. That said, in addition to the user permissions and alerting, Kibana plugins can provide new visualizations and other tools that could work with the data stored in Elasticsearch for example. As was the case with Grafana, Elasticsearch BV lists

some plugins on their website, and installation could be done in a simple way by giving the plugin name for a command line tool that comes with Kibana. This list includes many free to use third-party plugins.

All in all, Kibana is in many ways similar to Grafana, focusing on time series visualization, aggregated data, and also searching and browsing the data stored in the database. However, the database choice is restricted to Elasticsearch, and it would not be possible to use Kibana if the data is already stored somewhere else. As was the case with Grafana, configuring the dashboard and the charts can take a lot of time because the configurations have to be done manually and there are no dashboards available for quick configuration.

3.3.4 X-Pack

X-Pack [8] is a plugin for Elasticsearch [4] and Kibana [6], developed by Elasticsearch BV. The plugin has a free to use basic version that contains only some of the features, and a paid version to use all of the available features. The idea of the plugin is to introduce some more advanced optional features to Kibana for those people who need these features. The plugin brings some of those missing features to Kibana that were already there in Grafana [27], which include authentication, roles and user permissions for the Kibana user interface, and alerting. New features related to log analysis itself include graph analytics and visualization, and machine learning functionalities. Other new features include reporting functionalities and monitoring the different components of the centralized logging system that are from Elasticsearch BV.

X-Pack is not mentioned in many studies which is understandable as the software is still relatively new. It is still getting some mentions in few recent studies [16, 38] that mainly focus on Elasticsearch or Kibana.

The user permission system in X-Pack is very similar to that in Grafana, making it possible to allow different users to view and configure specific things. The alerting system uses Elasticsearch queries and can be configured to trigger on certain conditions. This allows alerts to be triggered based on any data stored in the Elasticsearch database which could be log data or for example monitoring data from the centralized logging components. As was the case with Grafana, these alerts can be automatically sent through various channels, for example email or some group chats like Slack. However, neither of these features are in the free to use basic version of X-Pack.

X-Pack provides a way to visualize data by using graphs consisting of vertices and connections between them. The tool is entirely visual, operated by manually issuing queries, and mostly useful for exploring data. For example, when exploring data extracted from an HTTP server access log, the graph tool could be used to find out which web pages are accessed most by certain users by examining the connections between pages and users. The machine learning capabilities introduced in X-Pack can make the detection of anomalies from the log data quite automatic. There are mainly two distinct things that the machine learning system is able to achieve. First, it can build a model based on time series values and detect anomalies when new values in the time series do not follow the learned model. Second, the system is

able to parse completely unstructured textual data and categorize similar messages. It is then possible to detect if certain message categories occur very rarely, more often, or less often than usual, and then report these cases as anomalies. Again, neither of these tools are in the free to use basic version of X-Pack. Some examples of the X-Pack interface for machine learning can be seen in the case study chapter in section 5.3.1.

The monitoring functionality makes it straightforward to view the status of Logstash [7], Elasticsearch, Kibana, and their server machines directly in the Kibana user interface, while all of the monitoring data will also be made available in the Elasticsearch database. The monitoring information of the server machines contain time series data about metrics like available disk space, memory consumption, and CPU utilization. Additionally, a lot of time series data is provided about the components themselves, such as information about requests sent to the systems, response times, and the amount of data stored in Elasticsearch. Then, the reporting tool makes it possible to generate PDF format reports of Kibana dashboards and visualizations, and CSV format reports from search data. Both of these tools are free to use features available in the X-Pack basic version apart from exporting PDF files which is a paid feature.

In summary, X-Pack provides monitoring and some reporting functionalities for free, and in the paid version, a number of additional features including machine learning capabilities for log analysis. The machine learning capabilities are likely one of the more useful features if there is a need to deal with unstructured log data, as there currently does not really exist free machine learning options tailored to analyze log data.

3.3.5 Splunk

Splunk [23, 35] has already been covered to some extent in section 3.1.5 regarding its centralized logging capabilities. As was already discussed, Splunk is primarily a commercial solution while there also exists a free to use version with limitations. On the other hand, all the other solutions apart from X-Pack [8] are open source and free to use.

The log analysis and visualization part of Splunk is overall very similar to the capabilities provided in Grafana [27] and Kibana [6]. Like the other solutions, Splunk also runs its own web server to provides a web interface for displaying the visualizations and browsing the log data. Setting up and configuring visualizations is also done in the same interface because Splunk aims to make it possible to do all important configurations, also for centralized logging, without having to touch configuration files. As an all-encompassing centralized logging and log analysis solution, Splunk also provides its own database solution and does not directly support any other databases for data storage. This means that the Splunk analysis solution would always get its data from its own centralized logging solution which discourages using other centralized logging solutions along Splunk.

In general, Splunk resembles Kibana the most as it also focuses on facilitating searching and browsing the textual log data, in addition to providing visualization

tools. Splunk provides its own query language that aims to be simple for most purposes, such as browsing of log messages, while also being capable of complex tasks, like aggregation, data lookups, and creating statistics. Selecting the timespan for the displayed data can be done without having to write specific queries with the query language, similar to Kibana.

Splunk also uses the concept of dashboards, which are similar to the dashboards in Grafana and Kibana. These dashboards are mainly edited using the interactive tools in the web interface and could contain multiple visualizations and reports, and also list raw log messages. However, the concept of reports is somewhat different in Splunk when compared to Kibana. Reports are created from the search interface in Splunk and then saved so that the same reports can be quickly generated from new data again in the future. The reports are viewable in the web interface but could also optionally be exported into PDF files. Whole dashboards could similarly be exported into PDF files.

The dashboards are saved in XML format which can be used to import, export, and also edit the dashboards. As an additional feature, Splunk allows converting the XML-format dashboards into HTML to allow even more control over the layout, formatting, and everything else that is possible in HTML. The HTML-dashboard will reside outside Splunk itself and use a REST API to get data from Splunk. HTML dashboards can no longer be interactively edited, exported to PDF, or exported to other Splunk instances, and they may stop working if the Splunk instance is updated. There also does not exists any list of ready to use dashboards on Splunk's website, so the export functionality is mainly useful when reinstalling Splunk or copying configurations to another Splunk instance.

Splunk also provides many additional features out of the box, that also existed in Grafana and X-Pack, such as user permissions, alerting, and plugin support. User permissions and alerting is very similar to Grafana and X-Pack, being able to set specific permissions on reports, dashboards, and other things, and to set up alerts that trigger on certain events. The plugins for Splunk can provide their own user interfaces and can introduce new functionalities, such as new visualizations, new data sources, and possibility to send data elsewhere. A list of existing plugins is maintained on Splunk website, and the plugins can be installed from the web interface of Splunk once the plugin file is downloaded. Lastly, Splunk also provides machine learning capabilities which are similar to the capabilities in X-Pack. Splunk is able to group together unstructured log messages and detect anomalies based on these groups and also within time series data. In addition, the machine learning tools in Splunk aim to reduce false positives and cluttering from detected events.

In summary, Splunk is very similar to Kibana, enabling browsing of log data using its own query language and allowing visualization of log data. Splunk also provides all the additional features in X-Pack that are otherwise missing in Kibana. However, Splunk does not support other databases apart from its own, so getting data from some other database solution to Splunk could be difficult.

3.3.6 Summary of log analysis and visualization solutions

This chapter looked at some log analysis and visualization solutions that a system admin could set up on their own server. Some of the solutions are able to do parsing and recognize log formats, while other solutions have weak parsing capabilities and focus more on visualizing already parsed data.

Lnav [37] is the only solution of these that is run on a terminal and is not run as a background service whereas the other solutions are constantly running and provide a web interface. As such, lnav does not function well in large log analysis setups as it can only read log files, has some limitations in its log parsing capabilities, and does not offer visualization. Lnav could still be used in addition to other log analysis solutions on the side for direct log file reading as it offers more features compared to other utilities that operating systems usually provide out of the box.

Rest of the solutions are quite similar to each other, being constantly running services that provide a web interface. One of the biggest limitations of these solutions is the database support. While Grafana [27] does support 10 different data sources, Splunk [23] only uses its own database and Kibana [6] supports only Elasticsearch [4]. If the log data is already in some database, it can severely limit the options unless another system is set up that moves the data to Splunk or Elasticsearch.

Grafana mainly focuses on visualizing time series data and has very limited interface for displaying textual log data. If there is no need to access the textual log data, Grafana could be a more attractive option as it provides additional features like alerting and user permissions for free, whereas Kibana and Splunk do not. However, Kibana and Splunk also focus on displaying textual log data in addition to building visualizations, and if access to the textual log data is important, Grafana is likely not going to be good enough.

Kibana is a free solution for viewing textual log data and building visualizations. Additional paid features, including alerting, user permissions, and machine learning capabilities, can be brought into Kibana with the X-Pack plugin [8]. With this plugin, Kibana is very similar to Splunk in features and functionality as Splunk is not missing any features that would be available in the other solutions. If Kibana alone does not provide enough features, the choice between X-Pack and Splunk could come down to preference or small details between the two options.

4 Research material and methods

This chapter discusses how the case study is conducted and what are the different software solutions used in this study. Section 4.1 introduces the system, for which the centralized logging and log analysis setup is implemented, and describes all the relevant characteristics of this case study system. Section 4.2 then defines the overall goal for this case study, which is further detailed in section 4.3 regarding centralized logging and in section 4.4 regarding log analysis. In addition, the sections 4.3 and 4.4 already present some problems that should be solved during the case study. Lastly in section 4.5, certain third-party solutions are selected to be used in the case study, and the reason for the selection is explained.

4.1 Introduction to the case study system

The case study system is a relatively large Java application that has been simultaneously in production use and under active development for more than a decade, and it still is. Currently, the only platforms that are supported for the case study system are the CentOS and Red Hat Linux operating systems, which means that the different parts of the centralized logging system will be set up on Linux environments. The system is sold to customers who set up their own production environment and run the software on their own servers, meaning that there is no production environment being run in the company that develops the software. In essence, the software is some very old technology that has been gradually updated and modernized to meet today's standards and customer requirements.

The main function of the system is to process requests that are sent to it, and for most of its existence, the software has been run as a single instance on a single server machine where scaling the system throughput had to be done vertically. Nowadays, larger software systems are aiming for horizontal scaling in which multiple instances of the software are run on multiple server machines communicating with each other, as horizontal scaling is able to go further than vertical scaling [43]. Following this trend to some extent, the case study system has been recently updated to support running multiple instances on separate server machines that are able to communicate with each other. This however, does not bring true horizontal scaling as some parts of the request processing cannot be distributed to the other instances of the software but has to be always executed on one of the instances. While this configuration still retains a bottleneck in the system, it improves the availability of the system as the remaining instances can take over the request processing duties if some of the instances fail and become unavailable.

The case study software produces a significant amount of log data which is all saved into files within the file system of the server machine that is running the software. While the software was still a single instance running on a single server, this did not cause too much of a problem as the log files could easily be located on that single server machine. As the software was updated to run as multiple instances, this also introduced the challenges that manifest with such a system, one of them being the fragmentation of log files on multiple separate server machines. As the

logging system remained the same it had been so far, all of the log data would become located in multiple separate locations which makes it way too difficult to debug specific problems or to get the big picture of the whole system. Therefore, proper centralized logging and log analysis is required for this system going forward. While centralized logging can be used to tackle the problem of having logs in multiple locations, log analysis can be used to further improve what the logging used to be even before this update.

4.2 Overall goal of the case study

The overall goal of this case study is to implement centralized logging and log analysis for the case study system as it still uses the traditional way to simply write all log data to files which is not going to be enough in the future. Difficulties arise from the matter that the case study system was never really designed to be run as multiple instances on separate server machines, and improving the old logging system implemented in the software itself is not easily possible.

The main problem caused by this transition is that it will be more difficult to find the correct log file from multiple instances of the case study system running on separate server machines. Monitoring the system and troubleshooting problem situations is also going to be more difficult as a result. The most important goal of centralized logging is to have all log data from every server available in a single centralized location where it should be easy to browse the log data.

Log analysis is then also implemented which will make the whole logging system even better than what it has been so far. The logs have generally been browsed by simply opening log files and reading their contents manually whereas with log analysis, many things could be made automatic and there would not be a need to manually read log files anymore when looking for certain pieces of information. In general, the log analysis system should make it possible to easily monitor everything that is considered important, and highlight problems and errors to make it easy to troubleshoot any occurring problems, while still allowing to browse the raw log data when needed.

The case study is done by installing and configuring some of the centralized logging and log analysis software discussed earlier in this work, depending on which ones would best seem to answer our needs. An additional focus in the case study is to consider and solve any problems that could occur when implementing such a logging system for a system that is in many ways traditional single server software with an existing relatively complex logging setup.

4.3 Centralized logging

The case study system consists of multiple modules that are run on separate Java processes that can communicate with each other. In addition to processing requests sent to the system, the system also provides a web interface for configuration and monitoring that is run on an Apache Tomcat server. Each of these modules write log data to one or more log files of their own, usually separating error messages to a

separate file and sometimes using additional log files to further separate different kinds of log data. At least some of the log files can also be configured not to be written at all, such as the Apache Tomcat access log. In addition, the system supports a couple plugins that could be optionally installed that also introduce additional log files. Some of these plugins also use their own file location for storing the log data while some use the default log folder with the rest of the system. In total the number of different log files is close to 100 files but the amount of generated log data is not so massive that the throughput of the centralized logging system would become an important factor.

The log files are generally always located in specific directories although the actual location, where the log files are stored, is saved in the database used by the system. However, there is currently no interface to change the logging location, and in order to change the location, the database value has to be manually modified. It is still good to consider this as the log data location will affect the centralized logging configuration. Most of the different log files are also automatically split to multiple separate files to avoid single log files growing without limit. Usually old log files are never renamed or packed and a new log file is started each day and each time the system is restarted, so that the name of the new log file, as it is created, is already its final name. The resulting log files could be named for example in the following way:

```
UI_20171201_0.log
UI_20171201_1.log
UI_20171204_0.log
UI_20171204_1.log
UI_20171204_2.log
UI_20171204_3.log
UI_20171205_0.log
UI_20171212_0.log
```

In addition to this, there are a couple log files that are emptied and started over every time the system is restarted, and also a single log file that is rotated in a different way than the other files. The log file is divided to multiple files each with a running number, and every time the system is restarted, each of these files is renamed by increasing the running number. The log file `shl.log.0` is always the one being written and files beyond `shl.log.9` will be automatically deleted, so that the list of files eventually looks like this:

```
shl.log.0
shl.log.1
shl.log.2
shl.log.3
shl.log.4
shl.log.5
shl.log.6
shl.log.7
shl.log.8
shl.log.9
```

Lastly, the system also provides some maintenance scripts to be run manually or by using cron [21], where some of the scripts can also work with the log files. There are a couple things that can be achieved with the maintenance scripts regarding log files, namely, old enough log files can be moved to an arbitrary location and optionally compressed, old log files can be completely deleted, or log files currently still in use can be split to multiple files.

The main goal when implementing centralized logging for the case study system is to mainly have it working so that all useful log data from each server machine is available in the centralized location. Implementing centralized logging is going to be a relatively simple task for this system when comparing to the complexity of the log analysis. As all of the log data is written to log files, it is going to be mostly enough to read the files and transmit the data to a centralized database, and for this purpose there have been many software options already discussed earlier. The different plugin setups, the log file splitting, and the various maintenance scripts could still pose problems, such as causing some files to not be read or some log messages showing up multiple times in the database, which is something that should be sorted out.

Another option when implementing centralized logging would be to have the log producing software directly send the log data to the log transmission system, such as syslog [14], and not write the log data to files at all. This would however require changing the logging routines in the software itself, and with the case study software, this kind of approach would not be feasible as many of the different modules have implemented their own separate logging systems and it would be a large overhaul to fix everything.

4.4 Log analysis

As the case study system consists of multiple modules each writing to their own log files, it means that each of the modules could be using their own logging routines. While some of the modules may be sharing common logging utilities, many of them indeed have implemented their own logging functions. As a result, there are many different log message formats in use across all of the modules and sometimes even multiple formats used inside a single log file. Most of the time the log message formats are also custom ones that do not follow any common logging standards already used in other software. Java stack traces are also often written to the log files among the rest of the log data which causes the log files to contain a mix of single and multi-line log messages.

It is also possible to configure the verbosity level of logging for most of the modules separately so that less important messages could be either included or filtered out. Generally, the configurable verbosity levels used in this system are severe, warning, info, and all, where the warning level is used by default in order to have only severe and warning messages included in the log files. However, it is not possible to configure the verbosity of every log file, meaning that those files will always contain the same log data. In practice, the verbosity levels are the same ones that are introduced in `java.util.logging.Logger` [1], which also introduces additional verbosity levels of config, fine, finer, and finest, coming after the info level. These additional verbosity

levels can also appear in the log files although some other levels, such as error, can also appear as `java.util.logging.Logger` is not used to generate all of the log data. It would be better if the log analysis system could handle the log data at the most verbose level as it should then be able to handle the log data also on the less verbose levels.

Some examples of the log formats used in this case study system were already presented in the section 2.1.2 along with some explanation of the structure of the different formats.

Some of the log files produced by this system are used for different purposes although most of them simply log what the corresponding module is doing or what errors and problems it encountered. Often these error logs also contain the Java stack traces when those are available. The system produces a few standard log files, namely the Apache ActiveMQ log and some Apache Tomcat access logs that use either the Common [13] or the Combined [13] log format. Otherwise, rest of the produced log data is often non-standard, although occasionally the log format of the standard Java logging utility `java.util.logging.Logger` is used. Some of the log files also contain some very specific data that could have special uses and should likely be given special attention to properly leverage the information written to those files.

To quickly go over some of the specific log files, the MA log file tracks successful and unsuccessful login attempts, logouts, and other user account related events such as creating and deleting users. The SOAP web service access log can be used to monitor the number of requests received by the SOAP API of the system, and the log files inside the `um_logs` folder use some very special format that stores information of processed requests that are sent outside the system. The task engine performance log is periodically appended with some performance statistics in CSV format. Rest of the useful log files mainly write arbitrary informational, warning, and error messages in various different log formats.

There are a couple different goals when the log analysis system is implemented. Most importantly, troubleshooting problems that occur in the system is currently very difficult with multiple servers and log files. To fix this situation, the log analysis system should be able to detect which messages contain information about problems and errors, and display all those in a single location to make finding information about problems easier. Additionally, it would be beneficial if the log analysis system could alert when some severe problems can be detected from the log messages before the effects of the problems manifest in the case study system itself. As there are multiple logging formats used with many that do not even include the severity level, trying to detect which messages are error messages could be challenging. In addition to error messages, the log files also contain a lot of general information in the access and performance logs, and also in the more generic log files alongside any error messages. This data could be leveraged to monitor the general state of the system and also build visualizations in some of the data visualizing software that have been discussed earlier. And important part of this goal is to also figure out which of the log data should be considered important as there is a lot of uninteresting log data mixed within all of the data. Additionally, if it is possible, the final analysis system

should be somewhat robust so that the system would still work in the future as the case study software is further developed and expected to follow the relatively chaotic logging standards used so far.

4.5 Choosing software solutions for the case study system

The centralized logging and log analysis will be mainly set up for the case study system using software from Elasticsearch BV, Filebeat [5] to read the log files generated by the case study system, Logstash [7] to do some log parsing, Elasticsearch [4] as the log storage database, and Kibana [6] for visualizing the log data. In addition, the X-Pack plugin [8] is tested to see what can be achieved with machine learning. While the focus will be mainly on these choices, some remarks about the other options will also be mentioned as those have also been tested to some extent with the case study system.

One important aspect of the choice is that it is free to use and also open source. As we are mainly selling the case study system to be used by our customers, it would be preferable if also the customers adopted centralized logging as it would help us when we need to troubleshoot their problems. The centralized logging system being free to use would lower the threshold to adopt the system which is preferred. This would also make it possible to have a personal centralized logging system for each developer of the case study system as licensing would not be an issue. A non-free solution could be more easily considered if we were running just one production environment ourselves and not selling the case study system for others to use, as licensing would then be more straightforward and we would not have to take into account our customers running the case study system. But in this situation, a primarily commercial solution like Splunk [23] would not be such a good choice. X-Pack can still be tested even though the interesting machine learning features of X-Pack are a paid feature, because X-Pack is an optional plugin for Kibana, it is straightforward to install, and has a trial version that can be used to test the product. In that sense, it is worth to take a look at its machine learning capabilities to see what could be done with it, because when the trial expires, the rest of the centralized logging and log analysis system still remains free to use without any interruptions.

As Kibana is being used for visualizing the log data, it requires that Elasticsearch is used as the database to store the log data, and that requires that the log collecting system can store data in Elasticsearch. In this case a pure syslog [14] based solution would not work although it would be possible to send data to Logstash or Fluentd [12] using syslog. However, Filebeat and Fluent Bit [10] would be better choices as they are more capable of doing some log data parsing and detecting multi-line log messages which is especially important in the case study system.

Fluent Bit and Fluentd would also be entirely valid choices over Filebeat and Logstash for this case study, as they work very similar to each other. They could both send data to Elasticsearch, and Kibana is basically able to visualize any data stored in the Elasticsearch database. However, there already exists some knowledge of Filebeat and Logstash in this company and these software solutions are already used in some cases whereas Fluent Bit and Fluentd have not been in use. Choosing

something already familiar would make maintenance work and applying software updates easier in the long run. The case study system is not going to generate massive amounts of log data either, so benchmarking different options to get more throughput is not really required.

Both, Filebeat and Logstash, will be used even though it would be possible to drop either one of them from the centralized logging chain. Because multiple instances of the case study system can be run on multiple separate servers, memory and CPU resources can be saved by using Filebeat to collect log data instead of Logstash, as Filebeat is more lightweight. A single Logstash instance is then used to do some log parsing and processing which is an important step for this system as the log data can be quite difficult to handle. Filebeat alone does not provide as good data processing capabilities and it makes sense to do log processing in a single location as that also simplifies configuration changes.

5 Case study

This chapter goes through the process of implementing a centralized logging and log analysis solution for the case study system with log parsing additionally covered separately in its own section. The installation and configuration process of the different software components is concretely covered while also taking any potential problems into account. The chapter also discusses the benefits of the complete system along with other findings about the implementation process and the finished setup.

First, the section 5.1 covers the implementation of centralized logging and the configuration of Filebeat, Logstash and Elasticsearch in subsections 5.1.1, 5.1.2 and 5.1.3 respectively. The section 5.2 then covers how log parsing is configured for Filebeat in subsection 5.2.1 and for Logstash in subsection 5.2.2. Log analysis is then covered in section 5.3 by taking a look at machine learning with X-Pack in subsection 5.3.1 and manually setting up Kibana for log analysis in subsection 5.3.2. Lastly, the benefits of this complete system are discussed in section 5.4 and other findings in section 5.5.

5.1 Centralized logging

The centralized logging implementation for the case study system consists of installing Filebeat [5], Logstash [7], and Elasticsearch [4], and configuring them properly. Filebeat will be installed on each server that is running an instance of the case study software. Additionally, a single instance of Logstash and Elasticsearch are installed on a separate server that is not running the case study software. Logstash and Elasticsearch could also be running on two separate servers as they communicate with each other over the network. In this system the Filebeat instances collect all the log data that is sent to Logstash which then sends it to the Elasticsearch database for storage. The Elasticsearch database acts as the centralized location for all the log data.

While Filebeat, Logstash and Elasticsearch can all be installed also on Windows and Mac, the Linux version will be used as the case study system itself only runs on Linux, specifically CentOS and Red Hat Linux. Installation is very straightforward as Elasticsearch BV provides deb- and rpm-packages that can be installed with the package managers of different Linux distributions. These software solutions are installed as services that can be controlled on command line with the `service`- and `systemctl`-commands. As enabled services are automatically started with the rest of the system, bringing the centralized logging system up can be very quick if the server machines ever have to be restarted. Fluent Bit [10] and Fluentd [12] were also briefly tested. Their installation was similarly straightforward and also they were installed as services.

In this case study, Filebeat was installed on CentOS 7 to collect data from log files generated by the case study system running on the same server. Logstash and Elasticsearch were installed on a Ubuntu 16.04 server to receive and store the data. In this setup, getting log data to appear on the Elasticsearch database was quick and went without problems with only a few configuration changes. However, problems

can occur if the log directories or files cannot be read by everyone as the log collector may not be able to read the log data. When the case study system is installed, it sets the Linux permissions so that only the user who executed the installation scripts has the read- and execute-permissions for the log directories, meaning that only that user can see the contents of those directories. With Filebeat this was not a problem because by default the Filebeat service was run on the root user that is not restricted by file permissions.

In addition to testing Filebeat as the log collector, Logstash, Fluent Bit, and Fluentd were also tested as log collectors. While Fluent Bit worked without problems as its service was also run as root, Logstash and Fluentd were not able to see the contents of the log directories and failed to read any log data because they were not run on the root user. To fix this, the Logstash or Fluentd service should be run on a user that is allowed to see the directory contents, or the directory permissions should be modified to allow Logstash or Fluentd to see the contents. This matter further reinforces the point that Logstash and Fluentd are not really designed to do the actual log data collecting from log files even though it would still be possible.

5.1.1 Configuring Filebeat

Filebeat is the software that is installed alongside the case study software on the same server machine. Considering centralized logging, the job of Filebeat is to read log files for text data as it appears and send these log messages over the network to a Logstash instance running on another server. To configure Filebeat, it would be enough in this case to do all of the configuration changes in the `filebeat.yml` configuration file. There is also the `fields.yml` configuration file that can be used to set which fields Filebeat includes in the JSON format messages that it sends to Logstash or somewhere else. The default fields included by Filebeat already contain all of the important data, including the timestamp, the hostname of the server, and the name of the log file. Otherwise, the log parsing and field modifications shall be done in Logstash.

Configuring Filebeat to send logs to a server that is running Logstash can be done using the following example. SSL settings can also be optionally specified to ensure that Filebeat sends the data encrypted and only to trusted Logstash servers. The certificate and key files would have to be configured separately and set up so that the certificate authority signs the Filebeat and Logstash server certificates. However, setting up encryption and authentication would not be entirely required if the Filebeat and Logstash servers are both inside a closed network. Compression of the sent data is also enabled by default using a relatively light gzip compression level.

```
output.logstash:
  hosts: ["logstash_server_hostname:5044"]
  ssl.certificate_authorities: ["/etc/ca.crt"]
  ssl.certificate: "/etc/client.crt"
  ssl.key: "/etc/client.key"
```

To read data from files, Filebeat can be configured to look for log files in certain locations and even look into folders recursively. The configuration could look like this:

```
filebeat.prospectors:
- type: log
  paths:

    # read all files with .log extension in the path1 folder
    - /home/user/path1/*.log

    # read all files with .log extension in each folder inside the path2 folder
    - /home/user/path2/*/*.log

    # search recursively for all files with .log extension inside the path3
    ↪ folder
    - /home/user/path3/**/*.log
```

Filebeat wants the full file paths which may pose some problems if the log files could reside in different locations. The case study system is typically installed in the home folder of the user who installed it which causes the full path to the log files to be different quite often. Additionally, it is possible to have the case study system write logs to different locations by modifying the database values that define these locations. Because of this, a single Filebeat configuration file would not work everywhere, and it makes automatic installation and sharing configuration files more difficult. Having our customers use configuration files provided by us would then always require additional manual configuration or the use of scripts that can generate working configuration files.

Considering the log files produced by the optional plugins for the case study system that may or may not be installed, Filebeat can be configured without issues to monitor file paths that do not exist, so that the optional log files are included when they actually do exist. At the same time, the manner in which the different log files are split into multiple files, do not cause problems either when taking into account the following examples from the case study system.

```
UI_20171201_0.log
UI_20171201_1.log
UI_20171204_0.log
UI_20171204_1.log
UI_20171204_2.log
UI_20171204_3.log
UI_20171205_0.log
UI_20171212_0.log
```

In this example a new log file is started each day and every time the system is restarted. Filebeat simply picks up each new log file and starts following them because it is configured to follow every file with the `.log`-extension. As the old log files are no longer updated, there will not be any problems caused by that either,

although performance can be eventually impacted as the amount of different log files keep growing.

In the next example, every time the system is restarted, each of these log files are renamed with an increasing running number and a new log file with the name `shl.log.0` is started.

```
shl.log.0
shl.log.1
shl.log.2
shl.log.3
shl.log.4
shl.log.5
shl.log.6
shl.log.7
shl.log.8
shl.log.9
```

In this case Filebeat can be configured to only follow the `shl.log.0` log file as the other files only contain older data and are no longer updated. Filebeat is able to track files using their file handles and will detect when `shl.log.0` is renamed and a new file with the same name is started. As a result, each new `shl.log.0` file will be picked up and monitored properly. There could be an additional concern that something is written in the log file right before it is renamed and Filebeat would miss this message, because by default Filebeat only scan files every ten seconds if the file has not been written to for some time. But because Filebeat tracks files using their file handles and not the filename, it would still be able to scan the new messages in the renamed file. However, there is still an additional issue that Filebeat closes file handles after five minutes of inactivity and starts checking files using the filename again, which could indeed cause some messages to be missed.

Lastly, there were a couple files that are emptied and started over each time the case study system is restarted. In general, if the size of a monitored file becomes smaller than what is was before, Filebeat will read the whole file from the beginning for new log messages, so generally it would not be a problem that a log file is emptied before writing new log messages to it. The previous log data would not be lost either because log messages that are already stored in the Elasticsearch database will not be deleted. However, this can cause a problem if the emptied log file is quickly rewritten to contain at least the same amount of data that it did before emptying. As Filebeat only scans files at certain time intervals, if it does not see the file being emptied before the file grows back to its previous length, the new contents of the file will not be read by Filebeat and the log messages never reach Elasticsearch. This also happens in the case study system with one log file, where the file is very quickly emptied and then generally written with the exact same content apart from timestamps which means that the file usually never appears in Elasticsearch. This could be a severe problem in some cases but a good solution to fix the problem does not really exist as Filebeat only monitors the file size ever so often. The best that can be done is to increase the scanning frequency of Filebeat but that can also cause some performance impact. An additional problem exists, that something could be

written to the file right before it is emptied, kind of similar to the previous case with the `shl.log` files. In this case, if Filebeat does not happen to scan the new messages before the file is emptied, these messages will simply be lost.

The maintenance scripts that come with the case study system can delete, move, and split log files, but using these scripts should not cause problems with Filebeat most of the time. Deleting log files and even starting new files with the same name is not a problem as was already found out. The script that moves log files, only targets files that are at least a day old as to not move away files that could still be written to, and from Filebeat point of view this is same as deleting the files. Moving active log files around should never be done anyway, because if the log producing software still holds a file handle to the log file, the software could still continue writing data to the log file in its new location, which would probably not be a deliberate result. Problems could arise if log files were moved, possibly accidentally, to locations monitored by Filebeat, as the files would then be picked up and the whole contents immediately dumbed to Elasticsearch. The purpose of the script is to move and compress old log files away to an archive location, so generally it should not cause problems. Finally, the splitting script copies contents of all, even active, log files into new files inside a new directory and truncates the existing log files to zero length. As the new files are located inside a new directory, Filebeat would not pick them up unless it is configured to recursively search inside directories, which should not be done with the case study system. Truncating the existing files and then continuing to log to the emptied files also works nicely with Filebeat as was already found out. Using these scripts occasionally to delete or compress old log files in order to free disk space would still be required going forward, as Filebeat does not offer any functionality like this, but at the same time, some additional care should be taken to not interfere with Filebeat.

In summary, there can be some small problems when using Filebeat but generally it handles all situations in some proper way by default. Filebeat also tries to make sure that it does not miss or lose any log messages with the default settings although in some very specific situations some data could be lost which should be taken into account if data loss cannot be tolerated.

As was mentioned earlier, the configuration file formats are different in basically all of these software solutions making it more difficult to try and to configure the different solutions. The following example shows what a Fluent Bit configuration file could look like for reading a single log file sending the data to Fluentd:

```
[INPUT]
    Name  tail
    Path  /home/user/path1/*.log

[OUTPUT]
    Name   forward
    Match  *
    Host   fluentd_server_hostname
    Port   24224
```

5.1.2 Configuring Logstash

In a centralized logging system, a single Logstash instance running on a single server machine can be enough if the log processing throughput does not need to be too high. Multiple Logstash instances could also be run to scale them horizontally so that Filebeat would then transmit log data evenly to each instance. However in the case study system, the amount of log data is not so large that more than one Logstash instance would be needed. Logstash should also be installed on its own server not running the case study software to minimize performance impact. Considering centralized logging, the purpose of Logstash is to receive log events from one or more Filebeat instances and forward all these events into a single Elasticsearch database.

Logstash also uses the same configuration file format as Filebeat for some of its configuration settings, such as the amount of worker threads and internal event queue settings. These values do not necessarily need configuration as the defaults are quite reasonable. However, the other configuration file to set inputs, outputs and log message processing, uses a different format. For the Filebeat connection, setting `ssl_verify_mode` to `force_peer`, which is not the default, forces all connecting Filebeat instances to authenticate using their own certificates to prevent accepting log data from any untrusted party. Again, encryption and authentication are not entirely required inside a closed network but it can be set up in a similar manner as for Filebeat. The following example shows this configuration:

```
input {
  beats {
    port => 5044
    ssl => true
    ssl_certificate_authorities => ["/etc/ca.crt"]
    ssl_certificate => "/etc/server.crt"
    ssl_key => "/etc/server.key"
    ssl_verify_mode => "force_peer"
  }
}
```

Unlike the connection between Filebeat and Logstash, the connection to Elasticsearch uses HTTP or HTTPS as the communication with Elasticsearch has to be done using the REST API it provides. HTTPS can be used for secure connection and server authentication but because client authentication is rarely implemented with HTTPS, Elasticsearch and Logstash do not support it either, and Logstash instances sending data to Elasticsearch cannot be authenticated. This problem could be avoided if the amount of produced log data is not too large, as Logstash and Elasticsearch could be run on the same server machine using only the loopback address for communication. Also, inside a closed network, authentication would not be as necessary either. Compression of the sent data is also supported through the HTTP compression method which uses gzip. The following example shows the configuration that uses HTTPS and compression.

```
output {
  elasticsearch {
    hosts => ["https://elasticsearch_server_hostname:9200"]
    ssl => true
    cacert => "/etc/ca.pem"
    http_compression => true
  }
}
```

The following example shows what a Fluentd configuration file could look like to receive data from Fluent Bit and sending it to an Elasticsearch database. Again, the format of the configuration file is different compared to Filebeat, Logstash, and Fluent Bit, as was mentioned earlier.

```
<source>
  @type forward
  @id input_forward
  port 24224
</source>

<match **>
  @type elasticsearch
  host elasticsearch_server_hostname
  port 9200
  index_name fluentd
  type_name fluentd
</match>
```

5.1.3 Configuring Elasticsearch

Elasticsearch is the database that will persistently store all the log data that it receives from Logstash, and then serve the stored data for the log analysis system to analyze. Configuration of Elasticsearch is relatively simple as it is only supposed to provide an API that other software use to store and retrieve data. However, Elasticsearch can only provide an HTTP API and does not support HTTPS alone. HTTPS support can still be enabled for Elasticsearch by putting it behind a reverse proxy server that supports HTTPS, or by using the security features that the X-Pack plugin introduces for Elasticsearch. Enabling SSL communication between Elasticsearch and Logstash using X-Pack is the official way of securing the connection as X-Pack is developed by Elasticsearch BV just like Elasticsearch and Logstash are. However, the security features of X-Pack are a paid feature and therefore not suitable for this case study as it aims to be implemented using free software. A reverse proxy server in front of Elasticsearch would still work well as Logstash is able to connect to an HTTPS end point without any additional plugins.

Elasticsearch does not necessarily need any configuration as it binds to loopback addresses and listens on port 9200 by default, and if Logstash is running on the same server machine, it will already be able to connect to Elasticsearch. However, if this is not the case, the following lines could be modified in the configuration file:

```
network.host: 192.168.0.1
http.port: 9200
```

This configuration allows connecting to Elasticsearch inside a private network but only using HTTP. To use HTTPS, it would require some additional configuration of either X-Pack or some reverse proxy server.

5.2 Log parsing

As Kibana [6] is going to be used for visualizing the log data, some log parsing should be done before storing the data in the Elasticsearch database [4], as Kibana is not able to parse log messages as well as Filebeat [5] or Logstash [7]. While Filebeat does provide a lot of log parsing capabilities, Logstash is still better suited for parsing the log data as it has many features that are useful for this case study. Another reason to include Logstash in this setup is that it makes sense to parse the log data in just one central location instead of parsing log data in Filebeat on multiple servers. This setup simplifies the configuration and also makes maintenance and further development easier in the future. However, some log parsing must already be done in Filebeat which is covered in the following section 5.2.1.

5.2.1 Configuring Filebeat

For log parsing, Filebeat has a rather major role as it should detect multi-line log messages and combine the lines into a single log event before sending it to Logstash. When Filebeat reads log data from log files, it does not do it in real time and instead scans the files at a certain frequency. Therefore, Filebeat is often going to see multiple new lines in the log files at the same time and give those lines the same timestamp. After this the log data is sent to Logstash out of order, so it would not be possible to find out the correct ordering of log messages in Logstash anymore. Inside Filebeat, the ordering is still preserved and multi-line messages can be properly constructed.

When there are multiple different log files where some of them contain some very specific data and some use certain logging standards, it makes sense to separate these files in the Filebeat configuration and provide tailored configuration for the specific files. These specific files can also be given tags that are included with the log event sent to Logstash and eventually stored in Elasticsearch so that the files can be more easily found and separated in the log analysis system. Some special log files also exist in the case study system which include web server access logs, user account audit log, performance data logs, and logs for received and sent requests. As these log files use a clear structure that can be parsed, they should be tagged and handled separately, whereas rest of the generic log files mostly contain very unstructured and varying data. The Filebeat configuration can be done with multiple prospectors, with their own configuration, that follow only certain specific log files:

```
filebeat.prospectors:
- type: log
  paths:
```

```
    - /home/my_account/HT/hco/ui_logs/access_log.*
  fields:
    type: access_log

- type: log
  paths:
    - /home/my_account/HT/hco/logdir/TaskEnginePerformance*
  fields:
    type: task_engine_performance
  exclude_lines: [',id,host load,']
```

The above example also shows a case in which log lines can be excluded if the line contains a match for one of the given regular expressions. In this example however, there is only one regular expression and it only contains literal characters.

As the special log files are often stored inside the same folders that contain the rest of the unstructured generic log files, the already handled special log files can be similarly excluded from the generic prospector. This way those log files are not processed in multiple different prospectors and duplicated as a result. The following example shows how the generic log file prospector is configured for the case study system so that it ignored the special log files located inside the common log folders:

```
- type: log
  paths:
    - /home/my_account/HT/hco/logdir/*.log
    - /home/my_account/HT/hco/logdir/*.err

    - /home/my_account/HT/hco/ui_logs/catalina.out

    - /home/my_account/HT/hco/ws_logs/catalina.out
    - /home/my_account/HT/hco/ws_logs/htws.*.log

    - /home/my_account/HT/hco/shl_logs/catalina.out
    - /home/my_account/HT/hco/shl_logs/shl.log.0

  exclude_files: ['TaskEnginePerformance', 'access_log', 'MA_.*\.log']
```

What is left at this point is to configure the detection of multi-line log messages inside each prospector when needed. In the case study system there is one special log file that uses multi-line log messages that uses its own format, whereas rest of the special log files only contain single-line messages. In addition, the unstructured generic log files may contain multi-line log messages in various different standard and non-standard formats.

The UM IO logs produced by the case study system always contain data blocks that follow the same format. While each data block may start with a line of either `=` or `-` characters, the data blocks do not differ from each other and should be handled similarly. The following example shows two different data blocks:

```
========================================
New task started 20180516144556
-> Task 1063594_1
```

```
<- (CONTROL) Send parameters
-> task-type=create
-> product-version=ANY
-> product-name=DUMMY
...snip...
-> Stop
<- (CONTROL) Task 1063594_1
<- (CONTROL) Ok - Ready

----------------------------------------
New task started 20180516144557
-> Task 1063595_1
<- (CONTROL) Send parameters
-> task-type=create
-> product-version=ANY
-> product-name=DUMMY
...snip...
-> Stop
<- (CONTROL) Task 1063595_1
<- (CONTROL) Ok - Ready
```

The Filebeat prospector reading the UM IO logs can now be configured to construct multi-line messages from these data blocks. In this case a simple solution would be to detect the empty line that comes after each data block and construct a single log message from all the non-empty lines before the empty line. Normally, every line matched by the regular expression would be added to the multi-line message, but now we instead want to include the non-empty lines that are not matched by the regular expression, which is possible by negating the regular expression. Additionally, the log message would normally be constructed by adding the non-empty lines after each empty line. This behavior can also be changed to add the non-empty lines before the empty line instead. The following configuration example shows this setup:

```
- type: log
  paths:
    - /home/my_account/HT/hco/um_logs/*.io
  fields:
    type: um_logs_io
  multiline.pattern: '^$'
  multiline.negate: true
  multiline.match: before
```

However, this case was straightforward and the unstructured generic log files contain a mix of single and multi-line log messages in various different formats. Some multi-line log message examples were already seen in section 2.1.1, such as Java stack traces and the messages produced by `java.util.logging.Logger` [1]. However, the system also uses other multi-line formats and can even print Java stack traces using other formats. The following examples show some Java stack traces in different formats:

```
"JMX server connection timeout 96" #96 daemon prio=5 os_prio=0 tid=0
    ↪ x00007f4a6800f000 nid=0x7cf2 in Object.wait() [0x00007f4a342d8000]
```

```
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at com.sun.jmx.remote.internal.ServerCommunicatorAdmin$Timeout.run(
    ↪ ServerCommunicatorAdmin.java:168)
        - locked <0x00000000fae61d00> (a [I)
        at java.lang.Thread.run(Thread.java:745)
```

```
20180420 143908 WARNING: org.apache.catalina.loader.WebappClassLoaderBase
    ↪ clearReferencesThreads: The web application [hco1] appears to have started
    ↪  a thread named [SelectorThread] but has failed to stop it. This is very
    ↪ likely to create a memory leak. Stack trace of thread:
 sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
 sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:269)
 sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:79)
 sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:86)
 sun.nio.ch.SelectorImpl.select(SelectorImpl.java:97)
 com.sun.corba.se.impl.transport.SelectorImpl.run(SelectorImpl.java:273)
```

A similar example but with a differently set up timestamp:

```
Apr 20, 2018 2:39:26 PM org.apache.catalina.loader.WebappClassLoaderBase
    ↪ clearReferencesThreads
WARNING: The web application [htws] appears to have started a thread named [
    ↪ ActiveMQ Transport: tcp://HTWM1/127.0.0.1:48333@63006] but has failed to
    ↪ stop it. This is very likely to create a memory leak. Stack trace of
    ↪ thread:
 java.net.SocketInputStream.socketRead0(Native Method)
 java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
 java.net.SocketInputStream.read(SocketInputStream.java:170)
 java.net.SocketInputStream.read(SocketInputStream.java:141)
 org.apache.activemq.transport.tcp.TcpBufferedInputStream.fill(
    ↪ TcpBufferedInputStream.java:50)
...snip...
```

The system can also occasionally produce other multi-line messages that are not stack traces, for example:

```
error while checking connection component: java.io.IOException: Failed to
    ↪ retrieve KFRServer stub: javax.naming.ServiceUnavailableException [Root
    ↪ exception is java.rmi.ConnectException: Connection refused to host: HTWM1;
    ↪  nested exception is:
        java.net.ConnectException: Connection refused]
```

```
20180420 154018 FINE: com.tech.rnd.hco1.environinfo.a.j$1 a: Processing
    ↪ ht_ServerModel.pid<--3957-->F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN
    ↪   TTY          TIME CMD
0 S  1000  3957  3948  0  80   0 - 773723 futex_ pts/1   00:00:31 java
```

In some cases, the log files may even contain almost free-form text with varying and complex structure, such as the following examples:

```
"VM Thread" os_prio=0 tid=0x00007f03ac076000 nid=0xf57 runnable

"GC task thread#0 (ParallelGC)" os_prio=0 tid=0x00007f03ac021800 nid=0xf54
    ↪ runnable

"GC task thread#1 (ParallelGC)" os_prio=0 tid=0x00007f03ac023800 nid=0xf56
    ↪ runnable

"VM Periodic Task Thread" os_prio=0 tid=0x00007f03ac1a2800 nid=0xf64 waiting on
    ↪ condition

JNI global references: 965

Heap
 PSYoungGen      total 33280K, used 20197K [0x00000000d5580000, 0
    ↪ x00000000d7880000, 0x0000000100000000)
  eden space 32256K, 59% used [0x00000000d5580000,0x00000000d6853af0,0
    ↪ x00000000d7500000)
  from space 1024K, 89% used [0x00000000d7700000,0x00000000d77e5c70,0
    ↪ x00000000d7800000)
  to   space 512K, 0% used [0x00000000d7800000,0x00000000d7800000,0
    ↪ x00000000d7880000)
 ParOldGen       total 209920K, used 91769K [0x0000000080000000, 0
    ↪ x000000008cd00000, 0x00000000d5580000)
  object space 209920K, 43% used [0x0000000080000000,0x000000008599e470,0
    ↪ x000000008cd00000)
 Metaspace       used 62930K, capacity 64328K, committed 64768K, reserved 1105920
    ↪ K
  class space    used 7245K, capacity 7545K, committed 7680K, reserved 1048576K
```

```
Database::connection start
db: com.tech.rnd.hco1.ui.SourceDatabase@58c58d8e
get log server host "HTWM1"
Start tag: (START) 87294_2
End tag: (STOP) 87294_2
file_infos_size:1
search exp for log files:[nwu1.*ml]
filtered_files_size:1
file_info{nwu1.ml}:1525865997
Loop search found 3 lines when searching the log
```

It would be difficult and labor-intensive to try and detect multi-line messages, using only regular expressions, from this kind of data as it appears among a mix of stack traces, structured data, and unstructured data. Taking into account that the software is still being developed and new multi-line structures could be introduced in the future, it does not seem like a good idea to configure Filebeat to detect all of these formats with regular expressions.

Another approach to construct multi-line log messages from mixed log data would be to detect timestamps at the beginning of each line and group lines without a timestamp with the most recent line that had a timestamp. This is also how

lnav [37] groups lines together to form multi-line log messages, and in general it worked reasonably well for the case study system. Considering all the different log formats seen so far, apart from the syslog [14], Common [13], and Combined [13] log formats, they all have the timestamp at the beginning of the log message. As syslog, Common, and Combined log format messages are generally generated by some specific software and used in their own files, these formats do not usually get mixed inside the generic unstructured log files. Therefore, using this solution to detect timestamps would work well. Another benefit of this solution is that also the `java.util.logging.Logger` messages get correctly handled as those messages have the timestamp at the beginning of the first line of the multi-line message. This solution is not perfect but it can be quickly implemented, and it is still preferred to sometimes have log messages incorrectly grouped together than to have a long multi-line message, such as a stack trace, broken into multiple separate pieces and out of order.

At this point the task is to find out all the different timestamp formats that are used in the logs produced by the case study system. Only a relatively few different formats are used, and the following list contains examples of the different timestamps that can be encountered:

```
Apr 09, 2018 9:30:48 AM
Apr 25, 2018 12:10:16 PM
20180420 143541
20180420 143716.389
2018-04-09 14:39:13
2018-04-09 14:35:25,258
2018-04-09 10:31:53.630
```

When we have to use regular expressions to detect these timestamps, there is no need to write separate regular expressions to detect `20180420 143716.389`, `2018-04-20 14:35:25,258`, and `2018-04-23 10:31:53.630`, because those are already detected together with `20180420 143541` and `2018-04-20 14:39:13`. The following example shows the final Filebeat configuration for the generic log files, that detects these timestamps from the beginning of each line and includes the subsequent lines without a timestamp as part of the multi-line message:

```
- type: log
  paths:
    - /home/my_account/HT/hco/logdir/*.log
    - /home/my_account/HT/hco/logdir/*.err

    - /home/my_account/HT/hco/ui_logs/catalina.out

    - /home/my_account/HT/hco/ws_logs/catalina.out
    - /home/my_account/HT/hco/ws_logs/htws.*.log

    - /home/my_account/HT/hco/shl_logs/catalina.out
    - /home/my_account/HT/hco/shl_logs/shl.log.0

  exclude_files: ['TaskEnginePerformance', 'access_log', 'MA_.*\.log']
```

```
multiline.pattern: '^\w{3} \d\d, \d{4} \d\d?:\d\d:\d\d [AP]M|^\d{8} \d{6}|^\d
  ↪ {4}-\d\d-\d\d \d\d?:\d\d:\d\d'
multiline.negate: true
multiline.match: after
```

This regular expression is a bit more permissive than is required as it always allows any digits when there is a digit in the timestamp, and it allows any three-letter combination to work as the month abbreviation. As a result, the regular expression is much simpler but it also permits impossible dates. However, as it is supposed to parse at least somewhat reasonable log data, it could be assumed that there are no log messages that are deliberately trying to break the regular expression. Currently this regular expression does not produce false positive detections for the case study system.

5.2.2 Configuring Logstash

Regarding log parsing, Logstash should be used to process the log events so that the messages can be easily searched and visualized in Kibana. As Kibana does not provide as good log parsing tools as Logstash, it would be preferable to do as much parsing already in Logstash as possible.

First of all, the log events that Logstash receives from Filebeat have many data fields that do not contain any useful data, such as the version number of Filebeat, hostname repeated in multiple fields, and the tags that are not used in any analysis. These additional data fields also clutters the Kibana interface when browsing log events as can be seen in figure 6. Dropping these unneeded fields would also save some storage space in Elasticsearch.

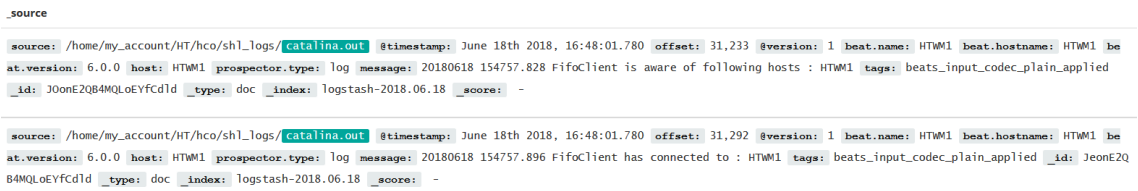


Figure 6: Log events with many data fields in Kibana

These unneeded data fields can be dropped from the event with a Logstash filter. In addition, it makes sense to remove the file path from the source field if it is not needed, and only keep the filename. However, log files in different folders could have the same name and this should be taken into account when designing the system. The following Logstash configuration shows how certain fields can be removed and how to drop the file path using the grok filter:

```
filter {
  mutate {
    remove_field => [ "beat", "prospector", "tags", "@version" ]
  }
  grok {
```

```
    match => { "source" => "%{GREEDYDATA}/%{GREEDYDATA:source}" }
    overwrite => [ "source" ]
  }
}
```

As a result, the log event browsing interface becomes cleaner in Kibana as can be seen in figure 7.

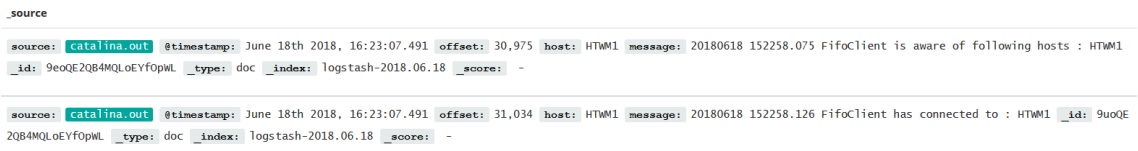


Figure 7: Log events with less data fields in Kibana

After this, the log messages itself should somehow be handled before storing them in Elasticsearch. First, the task engine performance log contains data in CSV format with the column names occasionally printed also in CSV format among the actual data. Logstash provides a filter to parse CSV data and even automatically set the column names based on the first message received by the filter once Logstash is started. As the case study system always prints out the column names before any actual data after a restart, Logstash should be started before the case study system to ensure that the first message received by the CSV filter contains the column names. If the filter detects wrong column names, it would always be possible to restart Logstash to also reset the CSV filter.

In addition, the CSV filter can be configured to also skip any lines that contain the column names, so it would not matter that the column names are occasionally repeated among the performance data. Lastly, the CSV filter can be applied conditionally to only certain log data by using `if`-statements in the configuration. A simple way to do this would be to check the type field that was set in Filebeat to mark the task engine performance data. The following configuration shows how to set all of this up for Logstash:

```
filter {
  if [fields][type] == "task_engine_performance" {
    csv {
      autodetect_column_names => true
      skip_header => true
    }
  }
}
```

It would be possible to set the column names manually for the CSV filter if the column names are not printed in the log file. It would also cause problems if multiple columns have the same name as the filter would only keep one of the columns with the same name and discard the rest. Manually naming the columns would again help in this case.

Next, the grok filter can be used to parse log messages that follow some format and to store any data extracted from the message into a separate field. First of all, the grok filter has a built-in support for access logs that use the Common [13] and Combined [13] log formats, and it is possible to handle all access logs with either format inside a single filter. With this configuration, the different parts of the log formats would be automatically put in separate fields within the log event that is then stored in Elasticsearch. The following Logstash configuration shows how this would be done:

```
filter  {
  if [fields][type] =~ "(ht|shl|ws)_access" {
    grok {
      match => {
        "message" => [ "%{HTTPD_COMBINEDLOG}", "%{HTTPD_COMMONLOG}" ]
      }
    }
  }
}
```

Some of the other special log files also contain data that would be useful to store in a separate field as the data could then be easily accessed and searched for in Kibana. It is possible to write custom regular expression to be used with the grok filter instead of just using the built-in parsers. With the UM logs, it would be useful to store the task id in its own field, to make searching for the UM logs of specific tasks straightforward in Kibana. The following configuration example shows how the task id could be parsed from the UM IO multi-line log messages and stored in a field named `id`. Examples of these logs were already seen in the previous section 5.2.1 about Filebeat.

```
filter {
  if [fields][type] == "um_logs_io" {
    grok {
      match => { "message" => "-> Task (?<id>[0-9_]+)" }
    }
  }
}
```

Next, we take a look at what could be done with the generic unstructured log data. The log data generally contains a lot of information that is often not very important, such as the Java class/method/thread related to the message, or the timestamp of the message because that is provided by Filebeat and already stored in its own field. Looking at the various log formats mixed among the unstructured data, the only very important bit of data would be the severity level of the message which should be detected when possible and stored in its own field. In addition to that, it would be beneficial to detect which log messages contain a Java stack trace and mark those with a Boolean value.

Detecting the severity level could be done in a simple manner by just looking for certain words within the log message instead of trying to properly parse the

various different log formats. In the case study system, the severity level is always printed out in capital letters which makes the severity level detection contain less false positives as the regular expression matching is case sensitive. There exist many different lists of severity levels that could be used in a software system in general. The case study system uses severity levels defined by the Apache Log4j logging utility and the standard Java logging utility `java.util.logging.Logger` [1]. For example, the Apache ActiveMQ that is used in the case study system uses the Log4j logging utility to write its log files. The following list combines the severity levels of both of these utilities quite arbitrarily from most severe to least severe.

```
FATAL
SEVERE ERROR
WARNING WARN
INFO
CONFIG
FINE DEBUG
FINER
FINEST TRACE
```

It would be possible to group the severity levels in different ways depending on the use case. In addition to storing the textual severity level in a separate field, it also makes sense to include a numerical severity level in another field that could then be used with arithmetic operations, for example to search for messages with a severity level of less than three. Due to the nature of the unstructured data and the way in which multi-line messages are constructed, a single message could contain multiple different severity levels. Therefore, the levels should be checked from most severe to least severe, so that each message gets the highest severity level possible. Additionally, if the severity level is not present in the message, a default level of `WARNING` is given for messages that contain a Java stack trace, and `INFO` for the rest of the messages.

To detect whether a message contains a stack trace, it would be enough to detect just one part of the stack trace. The following regular expression would be able to detect at least one part of any of the examples presented in the previous section 5.2.1 about Filebeat, while also trying to avoid false positives:

```
\s[a-zA-Z0-9$_.]+\([a-zA-Z0-9_. -]+\.java:\d+\)
```

The following example configuration combines all of this by first checking for a stack trace and setting a default severity level, and then checking the message content for an explicitly set severity level. The Logstash configuration format is not very flexible which causes this configuration to grow longer. The `if`-statements have to be directly inside the `filter`-block and could not be placed inside `mutate`-, `add_field`- or `update`-blocks to optimize the length. The grok filter could also be used to more easily detect Java stack traces as it has a built-in support also for that. However, it is not possible to check if the grok filter failed which is the reason it cannot be used in this case.

```
filter {
  if ![fields][type] {
    # Try to find parts of a Java stack trace and set some default severity level
    if [message] =~ "\s[a-zA-Z0-9$_.]+\([a-zA-Z0-9_. -]+\.java:\d+\)" {
      mutate {
        add_field => {
          "fields.trace" => true
          "fields.level" => "WARNING"
          "fields.level0" => 2
        }
      }
    } else {
      mutate {
        add_field => {
          "fields.trace" => false
          "fields.level" => "INFO"
          "fields.level0" => 3
        }
      }
    }

    # Set the severity level based on the message content
    if [message] =~ "FATAL" {
      mutate {
        update => {
          "fields.level" => "FATAL"
          "fields.level0" => 0
        }
      }
    } else if [message] =~ "ERROR" or [message] =~ "SEVERE" {
      mutate {
        update => {
          "fields.level" => "ERROR"
          "fields.level0" => 1
        }
      }
    } else if [message] =~ "WARN" {
      mutate {
        update => {
          "fields.level" => "WARNING"
          "fields.level0" => 2
        }
      }
    } else if [message] =~ "INFO" {
      mutate {
        update => {
          "fields.level" => "INFO"
          "fields.level0" => 3
        }
      }
    } else if [message] =~ "CONFIG" {
      mutate {
        update => {
```

```
          "fields.level" => "CONFIG"
          "fields.level0" => 4
        }
      }
    } else if [message] =~ "FINE[^RS]" or [message] =~ "DEBUG" {
      mutate {
        update => {
          "fields.level" => "FINE"
          "fields.level0" => 5
        }
      }
    } else if [message] =~ "FINER" {
      mutate {
        update => {
          "fields.level" => "FINER"
          "fields.level0" => 6
        }
      }
    } else if [message] =~ "FINEST" or [message] =~ "TRACE" {
      mutate {
        update => {
          "fields.level" => "FINEST"
          "fields.level0" => 7
        }
      }
    }
  }
}
```

This configuration seemed to work very well at least for the case study system and did not produce any false positives for the severity level or the stack traces. The case study system also does not seem to produce any stack traces that would not be detected with this configuration.

5.3 Log analysis

Setting up log analysis for the case study system on top the already implemented centralized logging and log parsing only requires setting up Kibana [6] and optionally also the X-Pack plugin [8]. X-Pack is the plugin for Kibana that brings some machine learning capabilities for log analysis, although these features can be considered to be only some additional utility on top of what Kibana already provides, and not a necessity as becomes apparent in the following section 5.3.1. The machine learning in X-Pack is also a paid feature which increases the threshold to start using it. On the other hand, installing X-Pack is straightforward as Kibana comes with a plugin installation utility which removes the need to manually fetch installation packages from their website. After installation, X-Pack also automatically starts a 30-day trial allowing to test the paid features for free, and to see whether machine learning could provide useful information in addition to what Kibana already provides on its own.

In this case study, Kibana is installed on the same Linux server machine as Logstash [7] and Elasticsearch [4]. Similar to the other software used for centralized

logging, Kibana also has Windows and Mac version, and on Linux is installed as a service by default. Alternatively, Kibana could also be installed on a completely separate server machine as it uses the Elasticsearch HTTP API for fetching data over the network. However, if the amount of log data is not too large, having Logstash, Elasticsearch and Kibana all on the same machine simplifies the system as there are less server machines and Elasticsearch can be restricted to disallow communication outside that single server machine.

Getting Kibana to successfully fetch data from Elasticsearch should not require any configuration changes after installation if Kibana is installed on the same server as Elasticsearch and the default Elasticsearch ports are used. This is possible due to the matter that Elasticsearch is the only database solution that is supported by Kibana. Kibana worked mostly without any problems but it did not use a log file of its own by default to report problems and other information. This made troubleshooting Kibana initially harder but it was possible to configure Kibana to write a log file and get access to any error messages generated by Kibana itself.

Grafana [27] was also very briefly tested as a log visualizer, and its installation and usage was similarly straightforward without any notable problems. But unlike Kibana, Grafana does not provide proper tools to view the textual log data as it focuses on visualizing time series data. Therefore, Grafana was not chosen for this case study.

5.3.1 Machine learning with X-Pack

The X-Pack machine learning capabilities shall be explored before taking a look at a more manual method of configuring log analysis in Kibana to see if machine learning could solve log analysis problems on its own without the need for manual analysis.

X-Pack mainly provides two kinds on machine learning functionalities, one that works with time series data and the other that works with textual data. Time series data in general is a series of numerical values that occur at certain time intervals. In the case study system, a good source for time series data could be the amount of received and sent requests through time. The time series analysis in X-Pack works with time buckets of some length, and it could be for example configured to monitor how many requests were seen in total every two minutes. X-Pack could then be able to determine if there is some sort of cyclic behavior in the data and display the estimated expected number of requests inside each time bucket, and point out when the time series data does not behave as expected.

Figure 8 shows how X-Pack displays this analysis in its user interface. The highlighted area contains the estimated expected number of requests, and the line shows the actual number of seen requests.

As is visible, X-Pack has been able to detect that the throughput of requests goes up and down at a certain cycle and has figured out some upper and lower thresholds for the throughput. X-Pack also marked some anomalies where the throughput has not fallen inside the expected range with larger deviations marked with higher severity level. Alerting could additionally be set up, so that X-Pack for example sends an email when it detects anomalies.

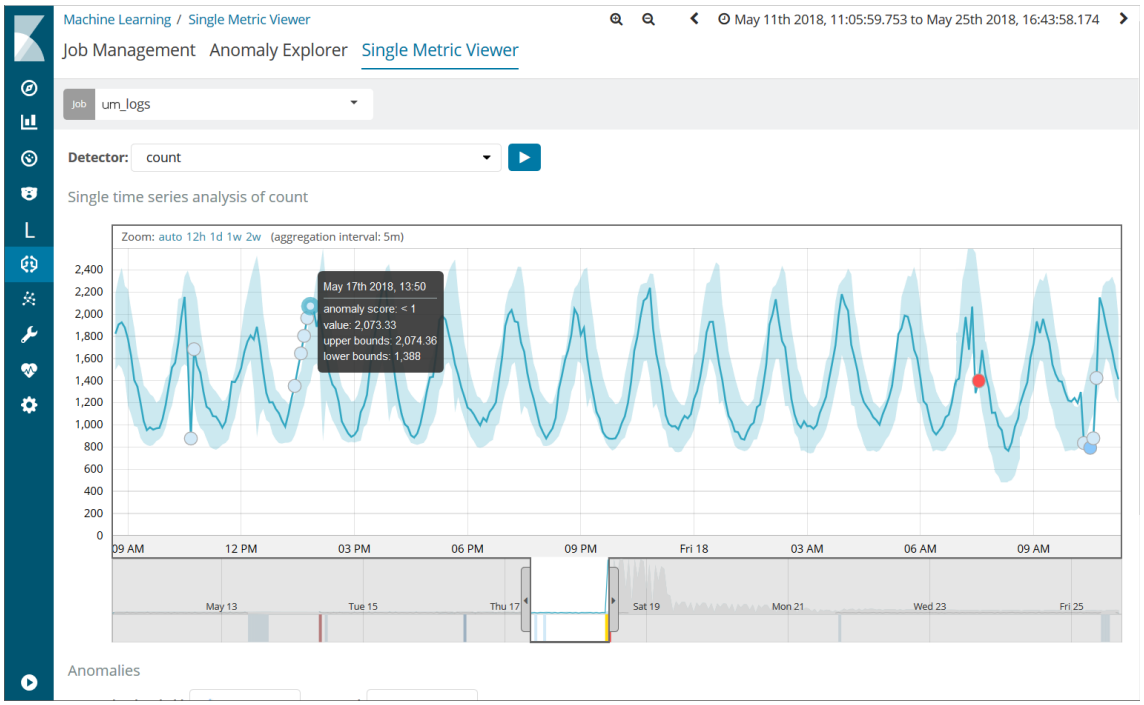


Figure 8: X-Pack time series analysis

The time series analysis is apparently able to detect cycles of any length although it did require multiple days' worth of data before it displayed the detected the cycle in this case. In this experiment, X-Pack was able to detect the cycle that was about 100 minutes long, whereas a cycle with a length of an hour or a day would probably be more common in general. When comparing to a manually set up solution with static throughput thresholds, this machine learning solution should be more sensitive to slight problems and would adjust the thresholds automatically.

However, this solution only works for time series data which cannot be used to solve all log analysis problems, and the system also has some problems. It seems to take a relatively long time to converge back to reasonable estimations after major disturbances in the time series data, so that in the worst case, the machine learning system could be virtually unusable and unable to detect anomalies for multiple days after a large incident. The system also has a delay of about 6 minutes for a relatively short time interval of 2 minutes before it is able to detect and report anomalies. As such, the system is not necessarily that good for detecting problems within a minimal delay, as the reported problem would have been occurring for some time already before being reported. X-Pack also does not provide proper context for the detected anomalies in the time series data, such as the log messages that contributed to the anomalous time intervals or log messages that occurred in any log files during the anomaly. With time series analysis, X-Pack only reports when there are problems after which it would be up to the system administrator to find out from other sources what is actually causing the problems.

The other machine learning system that X-Pack provides is able to work with unstructured textual data, which operated by recognizing similar log messages and

grouping them together. This system seems to work very well with the log data produced by the case study system, where the messages inside a group are clearly similar but the message format of a single group is not overly strict either. Figure 9 shows some examples of log message groups that X-Pack has been able to detect from the case study system logs.

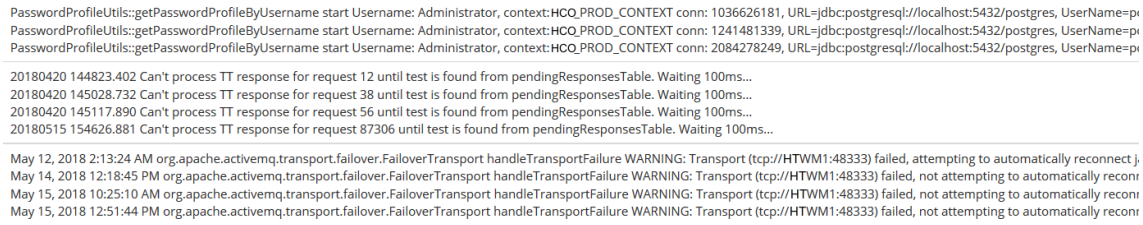


Figure 9: X-Pack log message grouping

Using these detected log message groups, X-Pack is able to perform various different analytics and point out anomalies. The most useful operations that X-Pack can do with unstructured log data is to count log messages in each group at each time interval, detect rarely occurring groups, and measure the informational content of the messages occurring inside each group. The machine learning system learns from the previously seen data, how often messages from each group usually occur.

When the system is configured to count the messages at each time interval, it can be set to report when there are unusually high or low number of messages occurring in any of the groups. At least with the case study system when major failures are occurring, X-Pack tends to produce a lot of reports about common informational messages suddenly missing from log files, which is not helpful when trying to troubleshoot what is causing the problems. However, it is possible to configure X-Pack to ignore completely missing log message groups, so that other anomalies, like a large amount of error messages, could be more easily noticed.

X-Pack can also be configured to focus on very rarely occurring log message groups and only report when one of the rare messages appear in a log file. This setting would ideally be able to focus on rarely occurring error messages or unique problem situations. On the other hand, this setting would also report log messages caused by some rarely performed maintenance tasks or other harmless events and could potentially hide actual problems caused by these same events. While this system would help with detecting rare problems, it is good to remember that some rare problems could drown among other less important detections.

Lastly, the info content function is supposed to measure the overall informational content in the messages of each group seen during a time interval. In practice, this is measured by measuring how much the messages inside a same group differ from each other. As the groups mostly determine the format of the log messages, the textual or numerical content of the messages could be quite different inside a single group as long as they have the same format otherwise. An example is given in the X-Pack documentation [8], that the info content function could be quite useful for analyzing query strings sent to a web server, and could be used to detect when a malicious party is trying to scan for vulnerabilities in the website. However, using

the info content function to analyze the unstructured log data produced by the case study system did not seem to be any useful at all as the variations in the log message contents were usually not very large. The info content function mostly reacted to the timestamp changing and practically only brought up commonly occurring messages that contained a timestamp. It would be quite preferable that the function would not react to the timestamp, as timestamps are expected to be constantly changing in any case, and it does not provide any value that the info content function points this out. As such, the info content function should probably only be used for messages that do not contain any kind of a timestamp.

Figure 10 shows how X-Pack displays anomalies it has detected from textual log data. As can be seen, the filenames for the anomalous log data is visible which should make it easier to find the relevant log data for troubleshooting purposes. The colors of the boxes indicate the number of anomalies detected at each point of time.

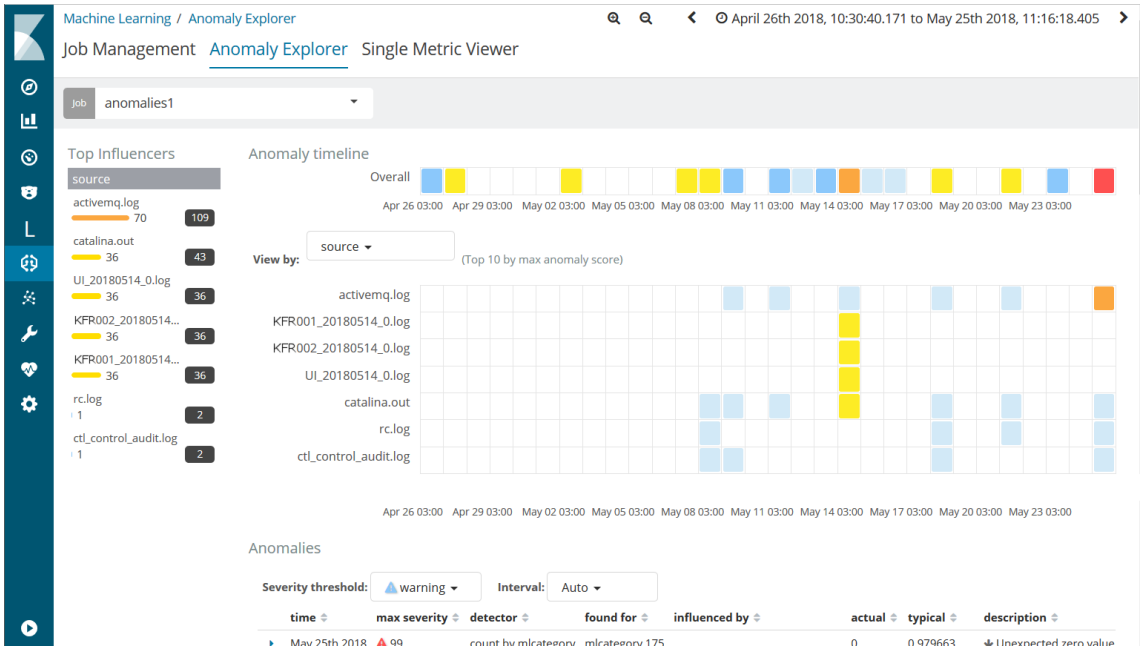


Figure 10: X-Pack displaying anomaly detections

While the machine learning could be very useful during the normal operation of the case study system, there would still be times when this automatic analysis becomes practically unusable. This could happen when there are large configuration changes in the system, or when the system is restarted, as these actions produce more than usual amount of log data that is not otherwise seen. This usually causes the machine learning system to report large amounts of anomalies about the completely normal log messages that occur during configuration changes and restarts. Because of this, the automatic anomaly detection system would likely not provide any help if there are problems that require troubleshooting during configuration or restarts. Even then, when the system is working quite normally, any detected anomalies usually come together with a large proportion of informational messages, detected as anomalies, that do not often help with troubleshooting.

In addition, it is possible to configure many parts of the case study system to output more fine-grained log messages that are usually uninteresting but could rarely be useful for troubleshooting. If any fine-grained logging is enabled in the case study system, the anomaly detections will usually just consist of messages with severity levels of `FINE`, `FINER`, and `FINEST`, as the system would be constantly outputting log messages even when it is not doing anything. The machine learning system of X-Pack does still calculate severity levels for the anomaly detections and allows filtering out less severe anomalies, which could help with hiding those anomalies consisting of informational messages.

The last thing to mention is that the machine learning system does not always provide good context for these anomaly detections, which was also the case with the time series analysis. It does show some log message examples from the detected group and which log file contains the anomaly, so that the correct log file can be found to further investigate the problem. However, the system does not automatically show what other log messages appeared in the same file or what message the system produced to any log files at the time of the problem. So, unless the detected error message alone manages to give enough information about the problem, troubleshooting the issue is likely going to require manually inspecting other log data related to the detected anomaly.

5.3.2 Setting up Kibana

Kibana makes it possible to visualize time series and other data using various different charts and also list log messages that match certain filters. These visualizations have to be configured manually but this is often quick to do with the Kibana user interface although it may require knowledge of the different query languages used in Kibana. There are three query languages that can be used in different parts of the Kibana user interface although it is not necessary to use all of the languages to achieve most of the things that are possible in Kibana. One query language is the JSON format language used by Elasticsearch, used in some parts of Kibana to send exact queries to Elasticsearch. Other language uses the lucene query syntax which is very simple and could be also used inside the JSON queries sent to Elasticsearch. The third and relatively complex language can be used to build time series visualizations in the Timelion visualizer which is one part of Kibana.

The important task is to figure out how the log data, produced by the case study system, should be visualized and made available so that the visualizations provide most support for monitoring and troubleshooting the system. The monitoring part could provide metrics that show indications when the system is experiencing problems such as partial crashing, software bugs, or slowdown in request processing. The troubleshooting part should then focus on displaying error messages and other things that would provide information about the problems and help with troubleshooting.

In the case study system, logs that provide good monitoring information are task engine performance logs, the web service access log listing received requests, and the UM logs providing data about outgoing requests. In addition, the unstructured logs can be monitored for instances of warning and error messages, and also Java stack

traces. An increased amount of such messages would be a rather clear indication of problems in the system.

Creating visualizations, including time series visualizations, can be done in Kibana mostly by using the graphical user interface without having to write too many complex queries. It is even possible to avoid writing Timelion queries as Kibana provides graphical tools to create rather advanced time series visualizations, although Timelion makes even more complex visualizations possible. Generally, only the simple lucene queries are required to control which data is used in the visualizations. The following list contains some examples of lucene syntax queries which would be useful for building the visualizations to monitor the case study system. The queries take advantage of the type field set in Filebeat, and the severity flags and the stack trace Boolean value that were set in Logstash. The second and third examples in the list in fact achieve the same thing, but are there to demonstrate some features of the lucene query syntax.

```
fields.type:ws_access
fields.type:um_logs_*
fields.type:um_logs_err OR fields.type:um_logs_io
fields.level:WARNING
fields.level0:<2
fields.trace:true
```

Figure 11 shows a visualization in Kibana consisting of the amounts of the web service access log messages and UM log messages through time. The visualization is similar to the X-Pack time series analysis but this time there is no automatic anomaly detection and reporting.

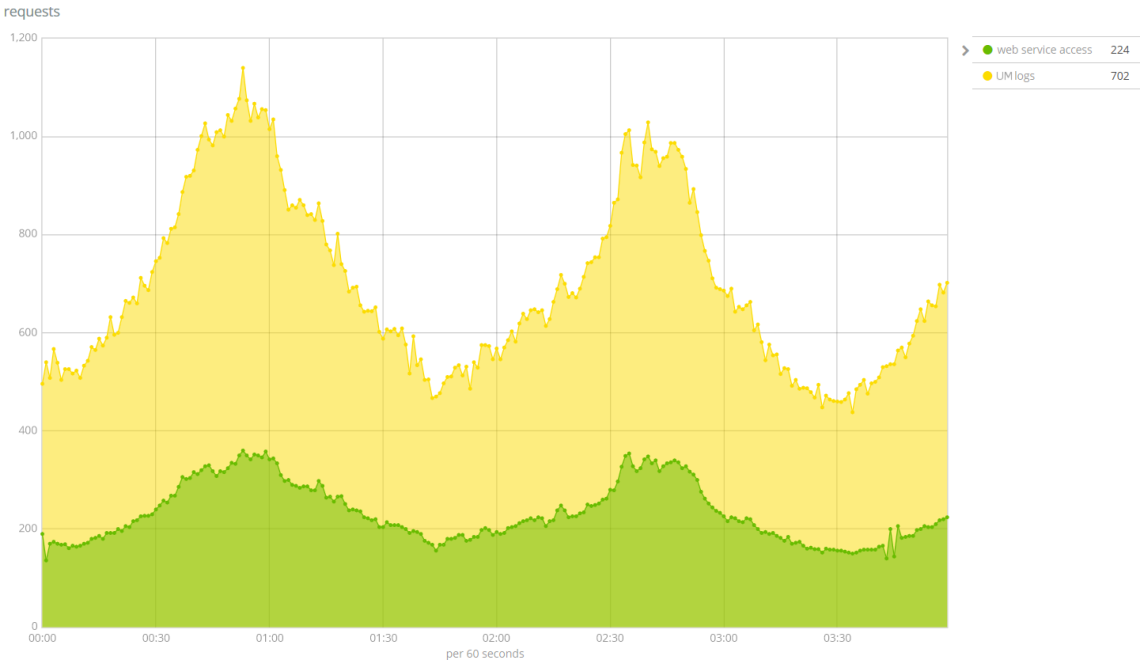


Figure 11: Time series visualization of request throughput

Similar to the previous visualization, it would also be possible to show the amount of log messages with different severity levels and the amount of stack traces within a single chart as can be seen in figure 12. This would facilitate detecting problems that cause error messages, stack traces, or an increased amount of warnings among the log data generated by the whole system.

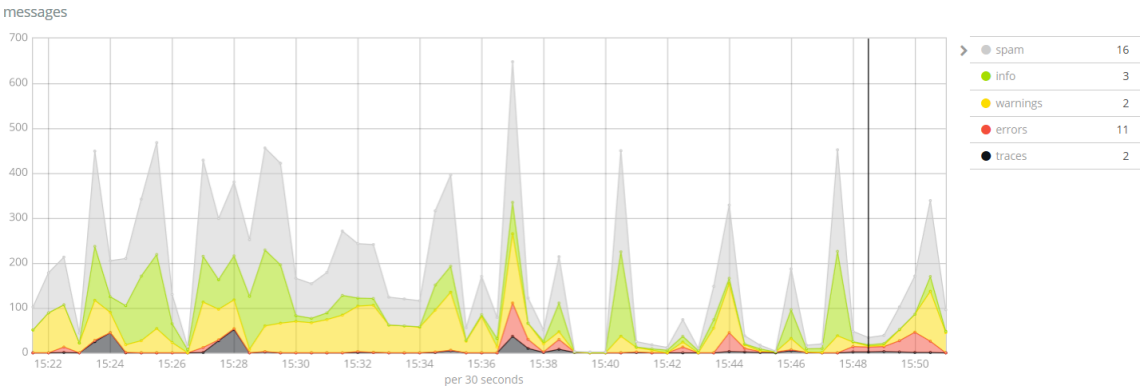


Figure 12: Time series visualization with log severity levels

It could also make sense to just track the amount of errors and stack traces as seen in figure 13.

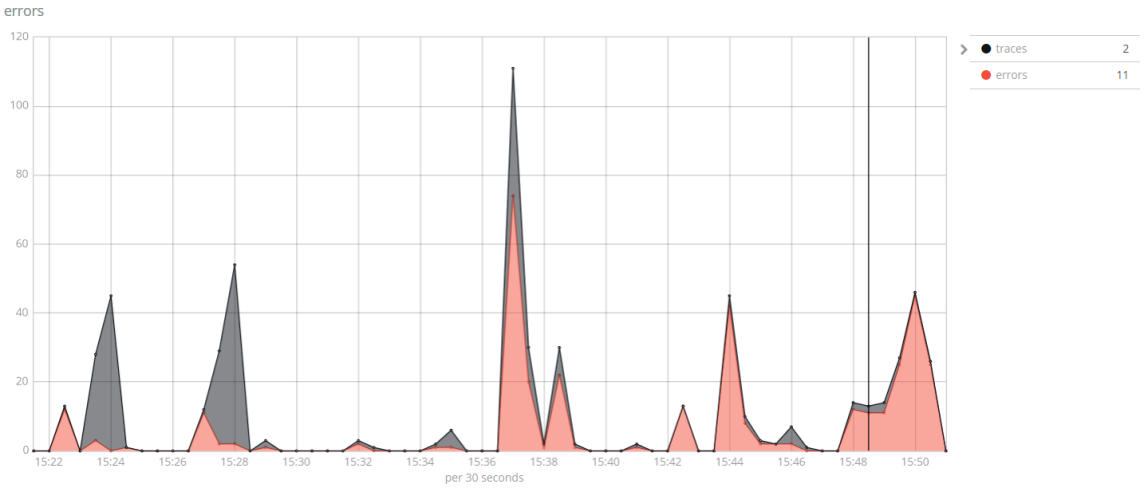
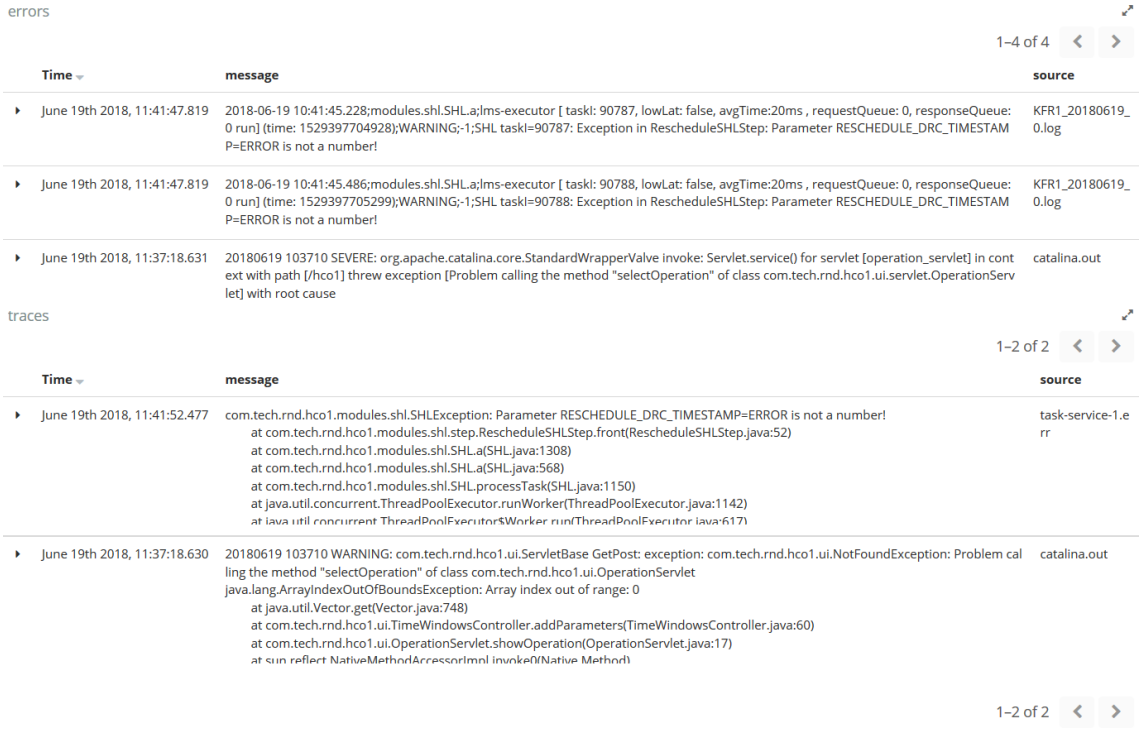


Figure 13: Time series visualization of error messages

In addition to graphical charts, it is also possible to produce lists of log messages using lucene queries. In this case it makes sense to create lists that contain the latest error messages and stack traces, so that when a problem situation is detected, the most recent error messages could be easily found to troubleshoot the problem. Figure 14 shows how two separate lists of error messages and stack traces could be shown in Kibana.

Additionally, any time Kibana produces lists of log messages, it also provides links to view an arbitrary number of other messages, that appeared right before or after the log message in question, in any log file produced by the case study system.


errors

1–4 of 4

Time	message	source
June 19th 2018, 11:41:47.819	2018-06-19 10:41:45.228;modules.shl.SHL.a;lms-executor [taskI: 90787, lowLat: false, avgTime:20ms , requestQueue: 0, responseQueue: 0 run] (time: 1529397704928);WARNING;-1;SHL taskI=90787: Exception in RescheduleSHLStep: Parameter RESCHEDULE_DRC_TIMESTAMP=ERROR is not a number!	KFR1_20180619_0.log
June 19th 2018, 11:41:47.819	2018-06-19 10:41:45.486;modules.shl.SHL.a;lms-executor [taskI: 90788, lowLat: false, avgTime:20ms , requestQueue: 0, responseQueue: 0 run] (time: 1529397705299);WARNING;-1;SHL taskI=90788: Exception in RescheduleSHLStep: Parameter RESCHEDULE_DRC_TIMESTAMP=ERROR is not a number!	KFR1_20180619_0.log
June 19th 2018, 11:37:18.631	20180619 103710 SEVERE: org.apache.catalina.core.StandardWrapperValve invoke: Servlet.service() for servlet [operation_servlet] in context with path [/hco1] threw exception [Problem calling the method "selectOperation" of class com.tech.rnd.hco1.ui.servlet.OperationServlet] with root cause	catalina.out

traces

1–2 of 2

Time	message	source
June 19th 2018, 11:41:52.477	com.tech.rnd.hco1.modules.shl.SHLException: Parameter RESCHEDULE_DRC_TIMESTAMP=ERROR is not a number! at com.tech.rnd.hco1.modules.shl.step.RescheduleSHLStep.front(RescheduleSHLStep.java:52) at com.tech.rnd.hco1.modules.shl.SHL.a(SHL.java:1308) at com.tech.rnd.hco1.modules.shl.SHL.a(SHL.java:568) at com.tech.rnd.hco1.modules.shl.SHL.processTask(SHL.java:1150) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)	task-service-1.err
June 19th 2018, 11:37:18.630	20180619 103710 WARNING: com.tech.rnd.hco1.ui.ServletBase GetPost: exception: com.tech.rnd.hco1.ui.NotFoundException: Problem calling the method "selectOperation" of class com.tech.rnd.hco1.ui.OperationServlet java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 0 at java.util.Vector.get(Vector.java:748) at com.tech.rnd.hco1.ui.TimeWindowsController.addParameters(TimeWindowsController.java:60) at com.tech.rnd.hco1.ui.OperationServlet.showOperation(OperationServlet.java:17) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)	catalina.out

1–2 of 2

Figure 14: Lists of recent error messages and Java stack traces

In this way it is quite straightforward to find all other log messages that could be possibly relevant to the problem.

In addition to just building visualizations, Kibana enables doing searches at any time using lucene queries without having to build visualizations for each specific query. This was also one of the original reasons to do centralized logging, to be able to search any log data from any host and any file in a single centralized location. This makes it possible for example to read any specific log file without having to look for them in the file system or to search UM logs of any request that was sent into the system. Especially in a multi-host environment with multiple instances of the case study software, the UM logs of each request are only stored in one of the hosts depending on which host processed the request. The only way to find the correct logs would be to go through the logs on each host to see which one of them processed the request, which is very time consuming. Kibana on the other hand, can instantly find the log data as the request id is included with the UM logs when Logstash stores them into Elasticsearch. The following lucene query could be used to find all UM logs for one specific request:

```
fields.type:um_logs_* AND id:86659_*
```

As can be seen, Kibana provides many different tools to monitor metrics and also the textual log data, which could be used to get indication about the system performance and any problems. The monitoring still needs to be manual as Kibana does not provide any automatic alerting tools which would only be available in the

paid version of the X-Pack plugin. Much of the usefulness of Kibana also depends on how the data is stored in Elasticsearch which depends on the Filebeat and Logstash configurations.

5.4 Benefits of this system

The implemented centralized logging and log analysis system is able to achieve goals set for it, and it manages to provide many benefits in this case study. Most importantly all of the log data is now available in one single location, so that there is no need to look inside multiple different log files located on multiple separate server machines anymore, which was the only major goal of the centralized logging part. The centralized logging system also did not bring any significant additional problems into the whole setup, although it does always require some manual configuration changes as the log files could reside in different locations on different installations of the case study system. Apart from some manual configuration, the centralized logging system is straightforward to install and it did not require making any changes in the case study system. This means, that centralized logging could be set up on top of any existing system that writes all of its log data into files without having to make changes in the original system.

The log analysis system also achieves goals set for it and provides many useful benefits for monitoring and troubleshooting the system. The time series visualizations in Kibana are able to provide information about the overall system health and show whether error messages are occurring on a quick glance. This system is also able to detect the severity level of log messages and conveniently show the most recent messages containing errors, stack traces, or other important data, in one single place. This makes it trivial to find error messages when they are needed to troubleshoot the system.

Automatically detecting errors and stack traces could also be used for additional testing of the system during its development. The testing could be manual or there could be an automatic testing setup, but currently log files are not closely inspected as the goal of testing is usually to check that only certain specific things are working correctly. Log data could be checked for additional errors during all kinds of testing as there could be side effects that are not explicitly checked, or there could exist less visible problem that are not clearly apparent apart from an error message in a log file. This also benefits the case study system as the log analysis setup was already able to detect some error messages and stack traces printed in the log files while the case study system was simply started and was operating seemingly normally. Of course, these detected problems would not be quite major at the moment, but could cause worse problems in the future as the case study system is further developed.

Machine learning could also be used to provide some additional utility for log analysis, as it could automatically monitor time series data and detect additional important log messages other than simple error messages or stack traces. But at the same time, it was apparent that a machine learning system alone is not quite robust enough to spot all important events. The system also tends to produce a lot of notifications about messages that are not very important at all which could make

notifications about actual problems much harder to notice among the noise. In the end though, machine learning would work best when the system is operating normally for long periods of time so that any detected disturbances are likely to be actual problems. During things like system configuration and restart, any log messages produced due to these actions would likely be reported as anomalies because these actions are so rarely executed.

Some additional benefits could be revealed in the future because this setup has been in use and under testing for a relatively short period of time, while the system shall continue to be used for many more years.

5.5 Findings

In addition to the already listed benefits, there were also some additional findings about the complete system.

There is some delay before log messages become visible the Kibana interface after they are written in the log file and this delay seemed to vary between 5 and 23 seconds in the setup used for the case study. First of all, it took between 1 and 8 seconds before Filebeat read a new message that appeared in a log file which would manifest as a difference between the timestamp contained within the message and the separate timestamp that Filebeat assigns for each log event. In theory the maximum time should be 10 seconds as that is the maximum time Filebeat waits between scans, but such delay was not recorded in this experiment. After that, is takes between 4 and 10 seconds to transmit the log message to Logstash, parse the message, store it in Elasticsearch, and have Kibana fetch it from Elasticsearch for display. This delay mainly depends on the complexity of the log parsing configuration and the amount of log messages going through the system at that moment. Lastly, Kibana can be set to automatically refresh any visualization or other display every 5 seconds at quickest, which causes there to be an additional delay between 0 and 5 seconds depending on how quickly a new log messages would become available after a refresh. It would also be possible to reduce this delay to some extent by configuring Filebeat to scan more frequently, and by using better hardware or horizontal scaling. With the machine learning system, this delay was much longer of around 6 minutes, and there did not seem to be any way to reduce this delay further.

In general, log data will not be lost even when there are problems in the network as Filebeat and Logstash queue up unsent messages and hold on to them until they can finally be transmitted to the next destination. However, there are few cases in which Filebeat could completely miss a log message that is written to a log file. When a log message is written to a file that is then renamed shortly after and Filebeat is not configured to follow the file with its new name, the log message could be missed if Filebeat does not happen to scan the file before it gets renamed. Log data could also be missed if a file is emptied after a new log message is written into it as Filebeat may not scan the file in time before it is emptied. There is no robust solution for this problem that would always work, and the only way to alleviate the problem would be to configure a faster scanning frequency for Filebeat.

One more concern could be the throughput of the whole system which considers

the amount of log messages that can be processed within some time span. The maximum throughput of this setup was not measured in this case study as the amount of produced log data was relatively small and limited throughput did not become a problem at any point. Limited throughput could become a problem if new log data is produced too quickly, but it would be possible to increase this limit by scaling the system horizontally, adding more Logstash and Elasticsearch instances. Still, the system was able to handle at least 10 thousand log events per minute in this case study.

In the end, this log analysis system worked reasonably well even though the log data was very unstructured and difficult to parse. In this case study, the most important thing that requires attention, is the detection of the severity level for each log message, so that important warning and error messages can be spotted. To achieve this in a robust manner, each log message should clearly state the severity level, as error messages that do not clearly state the severity could be missed by an automatic analysis system that relies on regular expressions. Therefore, when software developers are developing new software, it would be important for them to make sure that the severity level is included in each log message.

6 Conclusion

This thesis took a look at different software solutions for centralized logging and log analysis, and compared the features and capabilities of these solutions. The discussed solutions are available for download, so that a system admin could install those on their own servers, while other kinds of solutions were excluded from this study. This way the complete system could be isolated inside a local network with more control over the system. Multiple software solutions, mostly free to use and open source, were introduced for different purposes and with different capabilities in chapter 3. This kind of qualitative study can help developers and system administrators with making a choice for good centralized logging and log analysis solutions that fit for their case.

In addition, a case study was conducted in which a centralized logging and log analysis system was set up for an existing software system in chapter 5. The complete system was set up using some of the software that was discussed in chapter 3 while focusing on problems that can occur with the implementation of centralized logging, and the parsing and visualization of log data.

Implementing centralized logging for the case study system was straightforward and did not have any severe problems. Only some minor problems were present while the selected centralized logging solutions did not have problems reading the log files even when some files were occasionally renamed, deleted, or emptied. Log parsing had more things to consider, and a lot depends on the specific case and the format of the log data. In this case study, log parsing mainly concentrated on parsing multi-line log messages and detecting warnings, errors, and Java stack traces. Log analysis then looked at how to visualize log data and how to make the important textual log messages easily available, while also considering some machine learning possibilities. With machine learning it was possible to categorize unstructured log messages, and then focus on detecting anomalies in the unstructured log data, or alternatively in time series data.

The complete centralized logging and log analysis system brought in some clear benefits. The system facilitates monitoring and finding important messages that contain errors and stack traces, and it was already able to detect some errors in the log data even though the case study system was simply started and was operating seemingly normally. This automatic problem detection could also be included as part of an existing automatic testing system to detect additional errors. It was also found out that a solution based purely on machine learning is likely not going to work well enough, and manually doing some log parsing and analysis configurations is still required for a properly working and useful log analysis solution.

Some small problems or concerns were also present in the final system. The system had a small delay below half a minute before new log data was visible in the analysis system, and a longer delay of multiple minutes in the machine learning system. Some log data could also go completely missing in certain rare cases. Lastly especially for this case study, one of the biggest concerns would be to make sure that the severity level of log messages is always properly detected. Therefore, good advice for software developers would be to always include the severity level clearly in each

log message so that it could be easily detected with regular expressions.

There is still more research that could be done related to this subject. This thesis only considered solutions that are available for download and installation, and could still have included couple additional options in the study, notably Graylog [22] and NXLog [29]. Other types of solutions also exist for this purpose, such as log analysis services that are hosted on the Internet, that would receive the log data encrypted over the public network. Also in the case study, a closer look was only given to few of the studied solutions. Numerical metrics, such as the log data throughput or memory consumption of the different solutions, were not measured either as this study focused on the features and capabilities of the solutions. Additional research could also be done regarding the analysis and visualization of other specific log formats that were not considered in this case study. Such formats could include the Common [13] and Combined [13] log formats used in some web server software.

References

[1] Oracle and/or its affiliates. Logger (Java platform SE 8). https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html, 2018. Accessed: 2018-07-03.

[2] M. Bajer. Building an IoT data hub with Elasticsearch, Logstash and Kibana. In *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 63–68. IEEE, August 2017.

[3] M. Brattstrom and P. Morreale. Scalable agentless cloud network monitoring. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 171–176. IEEE, June 2017.

[4] Elasticsearch BV. Elasticsearch reference. https://www.elastic.co/guide/en/elasticsearch/reference/6.2/index.html, 2018. Accessed: 2018-07-03.

[5] Elasticsearch BV. Filebeat reference. https://www.elastic.co/guide/en/beats/filebeat/6.2/index.html, 2018. Accessed: 2018-07-03.

[6] Elasticsearch BV. Kibana user guide. https://www.elastic.co/guide/en/kibana/6.2/index.html, 2018. Accessed: 2018-07-03.

[7] Elasticsearch BV. Logstash reference. https://www.elastic.co/guide/en/logstash/6.2/index.html, 2018. Accessed: 2018-07-03.

[8] Elasticsearch BV. X-pack for the Elastic stack. https://www.elastic.co/guide/en/x-pack/6.2/index.html, 2018. Accessed: 2018-07-03.

[9] Y. J. Chen and H. Y. Chien. IoT-based green house system with Splunk data analysis. In *2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)*, pages 260–263. IEEE, November 2017.

[10] Treasure Data. Fluent Bit v0.12 documentation. http://fluentbit.io/documentation/0.12/, 2018. Accessed: 2018-07-03.

[11] S. Dharur and K. Swaminathan. Efficient surveillance and monitoring using the ELK stack for IoT powered smart buildings. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 700–705. IEEE, January 2018.

[12] Cloud Native Computing Foundation. Fluentd documentation. https://docs.fluentd.org/v1.0/articles/quickstart, 2018. Accessed: 2018-07-03.

[13] The Apache Software Foundation. Log files. https://httpd.apache.org/docs/trunk/logs.html, 2018. Accessed: 2018-07-03.

[14] R. Gerhards. The syslog protocol. RFC 5424, RFC Editor, March 2009.

[15] R. Gerhards et al. rsyslog 8.36.0 documentation. http://www.rsyslog.com/doc/v8-stable/index.html, 2017. Accessed: 2018-07-03.

[16] James Hamilton, Manuel Gonzalez Berges, Brad Schofield, and Jean-Charles Tournier. SCADA statistics monitoring using the Elastic stack (Elasticsearch, Logstash, Kibana). In *International Conference on Accelerator and Large Experimental Physics Control Systems*, page TUPHA034. 5 p. JACoW Publishing, 2018.

[17] R. Hanamanthrao and S. Thejaswini. Real-time clickstream data analytics and visualization. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 2139–2144. IEEE, May 2017.

[18] Zirije Hasani. Implementation of infrastructure for streaming outlier detection in big data. In Álvaro Rocha, Ana Maria Correia, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *Recent Advances in Information Systems and Technologies*, pages 503–511. Springer International Publishing, 2017.

[19] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. An evaluation study on log parsing and its use in log mining. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, pages 654–661. IEEE, 2016.

[20] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *Proceedings of the 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, pages 33–40. IEEE, 2017.

[21] IEEE and The Open Group. The open group base specifications issue 7, 2018 edition – IEEE std 1003.1-2017 (revision of IEEE std 1003.1-2008) — crontab. http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html, 2018. Accessed: 2018-07-03.

[22] Graylog Inc. Graylog. https://www.graylog.org/, 2018. Accessed: 2018-07-03.

[23] Splunk Inc. Splunk documentation. http://docs.splunk.com/Documentation, 2018. Accessed: 2018-07-03.

[24] SSH Communications Security Inc. SSH (secure shell). https://www.ssh.com/ssh/, 2018. Accessed: 2018-07-03.

[25] D. Jayathilake. Towards structured log analysis. In *JCSSE 2012 – 9th International Joint Conference on Computer Science and Software Engineering*, pages 259–264. IEEE, 2012.

[26] Michael J. Kavis. *Architecting the Cloud : Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. John Wiley & Sons, Incorporated, Somerset, 2014.

[27] Grafana Labs. Grafana documentation. http://docs.grafana.org/, 2018. Accessed: 2018-07-03.

[28] One Identity LLC. syslog-ng open source edition 3.16 – administration guide. https://www.syslog-ng.com/technical-documents/doc/syslog-ng-open-source-edition/3.16/administration-guide, 2018. Accessed: 2018-07-03.

[29] NXLog Ltd. NXLog. https://nxlog.co/, 2018. Accessed: 2018-07-03.

[30] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, February 2012.

[31] N. Peri. Fluentd vs. Logstash: A comparison of log collectors. https://logz.io/blog/fluentd-logstash/, 2015. Accessed: 2018-07-03.

[32] T. Prakash, M. Kakkar, and K. Patel. Geo-identification of web users through logs using ELK stack. In *Proceedings of the 2016 6th International Conference – Cloud System and Big Data Engineering (Confluence)*, pages 606–610. IEEE, 2016.

[33] M. Rajiullah, R. Lundin, A. Brunstrom, and S. Lindskog. Syslog performance: Data modeling and transport. In *2011 Third International Workshop on Security and Communication Networks (IWSCN)*, pages 31–37. IEEE, May 2011.

[34] T. Rinne and T. Ylönen. scp(1) – Linux man page. https://linux.die.net/man/1/scp, 2013. Accessed: 2018-07-03.

[35] Betsy Page Sigman. *Splunk Essentials*. Packt Publishing Ltd, Olton Birmingham, 2015.

[36] Bob Smith, John Hardin, and Graham Phillips. *Linux Appliance Design : A Hands-On Guide to Building Linux Applications*. No Starch Press, San Francisco, 2006.

[37] T. Stack. lnav 0.8.3 documentation. https://lnav.readthedocs.io/en/stable/, 2018. Accessed: 2018-07-03.

[38] Wataru Takase, Tomoaki Nakamura, Yoshiyuki Watase, and Takashi Sasaki. A solution for secure use of Kibana and Elasticsearch in multi-user environment. *CoRR*, abs/1706.10040, 2017.

[39] P. M. Trenwith and H. S. Venter. Digital forensic readiness in the cloud. In *2013 Information Security for South Africa – Proceedings of the ISSA 2013 Conference*. IEEE, 2013.

[40] A. Tridgell, P. Mackerras, and W. Davison. rsync(1). https://download.samba.org/pub/rsync/rsync.html, 2018. Accessed: 2018-07-03.

[41] H. Tsunoda and G. M. Keeni. Managing syslog. In *The 16th Asia-Pacific Network Operations and Management Symposium*, pages 1–4. IEEE, September 2014.

[42] Richard Underwood. Building bridges: The system administration tools and techniques used to deploy bridges. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC17, pages 5:1–5:5, New York, NY, USA, 2017. ACM.

[43] C. Vega, P. Roquero, R. Leira, I. Gonzalez, and J. Aracil. Loginson: a transform and load system for very large-scale log analysis in large IT infrastructures. *Journal of Supercomputing*, 73(9):3879–3900, 2017.