

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Lauri Linros

**DESIGN OF LOG MANAGEMENT SYSTEM
INFRASTRUCTURE WITH A PROOF OF
CONCEPT IMPLEMENTATION**

Master's thesis

Supervisor: Ph.D. Enn Õunapuu

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Lauri Linros

LOGIDE HALDAMIS SÜSTEEMI DISAIN JA KATSESÜSTEEMIGA KONTROLL

Lõputöö liik: magistritöö

Juhendaja: Enn Õunapuu

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Lauri Linros

23.10.2017

Abstract

The goal of this thesis was to create an IT system that is capable of receiving and analyzing log events from several distinct sources.

The biggest challenges in doing so were that most log sources output logs in different formats and the system had to be able to work with any of them. Another problem to solve was the transport of the logs to a central location as some logs were output to files while others were sent over the network. A set of requirements was created for the system in order to be able to verify the final solution. The main requirements were that the logs need to be gathered at a central location and transformed to a unified format.

For the solution, three different software applications were considered. The applications were Splunk, Graylog and ELK stack.

The solution was designed using the Graylog software in a high-availability setup and capable of receiving large amounts of data.

The result of this work was an architectural infrastructure design for aggregating log events to a centralized location that is capable of receiving log events in any format and using files or data streamed over the network as an input. The log events are then transformed to a unified format so it could be further processed using machine learning or some other methods.

This thesis is written in English and is 61 pages long, including 26 chapters, 12 figures and 5 tables.

Annotatsioon

Magistritöö peamiseks eesmärgiks oli leida lahendus, mis oleks võimeline keskses kohas talletama IT süsteemide poolt loodud logisid. Suuremateks väljakutseteks selle saavutamiseks on see, et rakendused talletavad logisid erinevates formaatides ja asukohtades. Loodud süsteem peab olema võimeline nende tingimustega toime tulema.

Probleemi lahendamiseks loodi nõuded, mille järgi on võimalik kindlaks teha kas süsteem täidab talle seatud eesmärgid. Põhilised nõuded panid paika, et süsteem peab olema võimeline vastu võtma ja siis ühisele vormingule viima erineval kujul sisse tulevaid logisid ja erinevad võimalused, kuidas logisid transportida kesksesse kohta. Lisaks olid ka nõuded, et logidele ligipääs peab olema kontrollitud ning juba analüüsitud logisid ei tohi olla võimalik muuta.

Kui nõuded olid paigas, siis võrreldakse kolme erinevat tarkvaralist lahendust, millega on võimalik antud probleemi lahendada. Võrdluseks valiti välja Splunk, Graylog ja ELK tarkvara tooted. Antud kolme toote vahelt lõpliku lahenduse loomiseks valiti Graylog.

Graylogi kasutades loodi lahendus, mis on võimeline vastu võtma suure hulga logisid ja samas laialdaselt kättesaadav. Saadud lahendusest implementeeriti lihtsustatud variant, millega oli võimalik kontrollida vastavust nõutele.

Magistritöö tulemuseks on arhitektuuriline lahendus, mis on võimeline erinevatest allikatest, kasutades erinevaid meetodeid, kesksesse kohta kokku koondama logisid. Antud logid töödeldakse ka ühisele kujule, et oleks võimalik neid edasi töödelda kas siis masinõppega või mõnel muul meetodil.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 61 leheküljel, 26 peatükki, 12 joonist, 5 tabelit.

List of Abbreviations and Terms

Log	Record of events within organizations systems
Node	One instance of an application
JSON	JavaScript Object Notation
SIEM	Security Information and Event Management
DBMS	Database Management System
LDAP	Lightweight Directory Access Protocol
AD	Microsoft Active Directory
SAML	Security Assertion Markup Language
SSO	Single Sign-On
JVM	Java Virtual Machine
JDBC	Java Database Connectivity
PoC	Proof of Concept
KPI	Key Performance Indicator
Docker	Software for operating system level virtualization
Docker Swarm	High-availability deployment of the Docker software
DIY	Do It Yourself
Syslog	Standard for Message Logging

Table of Contents

1 Introduction	10
1.1 Background.....	10
1.2 Thesis Goals	12
1.3 Problem Description	13
1.4 Methodology.....	15
1.5 Defining Scope	16
1.5.1 Log Generators in Scope	17
1.5.2 Further Limitations	21
2 Requirements	23
2.1 Functional Requirements	23
2.2 Non-functional Requirements.....	24
3 Solution Design	25
3.1 Initial Architecture.....	26
3.2 Tool Selection.....	30
3.2.1 Splunk.....	31
3.2.2 ELK	35
3.2.3 Graylog	41
3.2.4 Comparing the Tools	46
3.3 Adapting the Tools to the Architecture	51
4 Solution Verification	57
4.1 Installation	59
4.2 Receiving Data	62
4.3 Analyzing the Data	64
4.4 Data Presentation.....	66
4.5 User Management and Alerting.....	67
5 Summary.....	68
References	70
Appendix 1 – docker-compose.yml for the PoC	71
Appendix 2 – collector_sidecar.yml for PoC	73
Appendix 3 – Parsed Nexus Access Log Entry.....	74

List of Figures

Figure 1 - Small Service Architecture Diagram [4]	18
Figure 2 - Large Service Architecture Diagram	20
Figure 3 - Basic Workflow	26
Figure 4 - Initial Component View	29
Figure 5 - Splunk High-level Architecture [13]	32
Figure 6 - ELK Stack Architecture [15]	39
Figure 7 - Graylog Architecture [16].....	42
Figure 8 - Graylog Collector Sidecar [17].....	43
Figure 9 - Component Architecture.....	54
Figure 10 - Graylog Login Screen.....	60
Figure 11 - Graylog Collectors View	61
Figure 12 - Graylog Dashboard.....	66

List of Tables

Table 1 - Expected Data Amounts.....	22
Table 2 - Feature Comparison	46
Table 3 - Server Requirements	55
Table 4 - Consolidate Server Requirements	56
Table 5 - PoC Software Versions	59

1 Introduction

1.1 Background

A log is usually a timestamped record of events that have occurred within the organization's systems and networks. Logs are generated by every application and device that is part of any IT system. All the servers, workstations, network equipment and applications are most likely constantly generating logs.

Initially, when application logs were created, they were mostly a tool for troubleshooting problems. This is also the reason for standard log levels of ERROR, INFO etc. The purpose of logs was to give an idea of how applications were behaving and if there were any problems. This enabled system administrators and engineers to better diagnose the issues that applications were having and to find solutions on how to fix the problems. In recent years this has changed and applications have also started to log data much more relevant to day to day operations.

Applications are now quite often also recording user actions and performance metrics. Having that information can be very important as it gives insight into the usability of the application and decisions can be made on how to improve the application from the user experience side rather than just the technical solution. An example would be that if an application includes forms that users have to fill and a large number of users stop at the same place some changes are needed to encourage users to move forward. Having the data from applications analyzed and centralized also helps with proactive monitoring. It is much easier to see if there are irregularities and it is possible to react to them swiftly. As an example, the log data could be tied to the monitoring system and is able to trigger alerts based on defined rules.

Centralized logging is also very helpful from a security perspective. This allows to quickly identify any wrongdoing within applications. If the application is subject to compliance standards, log aggregation and analysis might even be mandatory.

The amount of security-related logs has increased considerably, for example, every device on a computer network is capable of outputting events in some manner and if those were analyzed it could be possible to tell if someone is attacking the system. Organizations use a lot of security software that all output large amounts of logs that need to be processed. After the logs have been centralized in one location it is possible to detect the attack vectors quicker and actions can be taken against the attack faster. It also makes it easier to detect what systems were attacked to be able to verify if the attack was successful or not.

Another benefit of having logs analyzed is that it gives more transparency into change management events. Metrics are available for before and after the implementation of the change, which enables measuring the success of the change.

1.2 Thesis Goals

There is a wide variety of applications and tools available for the purpose of log management with each of them with their own strengths and weaknesses. As a result of this thesis, an infrastructure architecture design will be created using one of the tools for an organization with a vast amount of data. There are instructions on how to set up those applications on a smaller scale, but doing it for data volumes in hundreds of gigabytes requires a different approach. There will be a specific set of data sources that are used in the system, so the tools have to be chosen appropriately. In order to create the architecture design the following questions have to be answered:

- What are the requirements for handling logs
- What tools/applications can be used to meet the requirements
- What infrastructure will be needed for the IT system

After the tools have been chosen and the design created there is also need to verify that the IT system is valid so a proof of concept system will be created in order to make sure that the requirements have been fulfilled.

1.3 Problem Description

In this thesis, IT service will be designed that is able to centralize and analyze logs from several different sources with different sizes, formats, and output types, to a single service that could be accessed to view events from those log generators. The assumption is that this is all taking place in the scope of a large organization. The organization is using software that has been developed in-house and also 3rd party applications. In case of in-house applications, it is possible to change the logging setup to make centralization easier. For 3rd party applications, it is not possible to modify the logging component of the software and solution needs to take that into account. The applications that the organization is running can range from small one node applications to microservices running in hundreds of Docker containers. The log management system must also be able to handle logs from servers and workstations. Some of the biggest problems that will need to be solved are the following:

- **Log generation.** Applications are generating different types of logs to different locations on many different hosts. The number of hosts can be in the range of thousands and the hosts themselves are constantly also generating log entries. Another problem is, that the same data in different application logs might be identified differently, making comparisons quite hard. Formatting of the log lines also differs between applications. The simplest example of that would be how to present the timestamp. Some applications use the ISO standard, others use UNIX timestamp etc. Log formats can also differ as some applications output lines using a delimiter while in other cases applications output JSON lines or some other method.
- **Log storage.** It is very likely that the information that is stored in the logs should not be readily readable for everyone as quite often they can contain information that could be abused if fallen into the wrong hands. That means the solution has to have an adequate security to be able to dictate who can access what logs. There is also the question of availability. The service used to centralize the logs should be quite robust and highly available as the log data needs to be always accessible. There also might be requirements to keep some of the log data for longer periods of time meaning some sort of archiving option

has to be available. Transport of the data should also be done securely to make sure the log information is not tampered with.

- **Log analyzing.** The logs also have to be analyzed and the data inside the loglines categorized to be able to search for the data you are interested in later. In general, this means parsing the loglines into more granular parts so searches could be made over the parts that interest you. Also, some sort of rules needs to be set on naming conventions to make life easier for the people analyzing logs. An example would be that HTTP response code should always be named `responseCode`.

The problem itself is quite a complex one with many aspects that have to be taken into consideration when coming up with an appropriate solution. To achieve an adequate result is an organization-wide effort and support has to be secured from management side in order to have the resources required.

1.4 Methodology

Since this solution would be implemented in a large organization it is entirely possible that some compliance standards have to be met. To do so the following publication is in order to create the requirements for the system:

NIST Special Publication 800-92 [1]

This is a document created in 2006 by National Institute of Standards and Technology and is a guide of what steps should be done in order to set up a successful centralized log management system and also provides information on why such steps should be taken.

After the creation of the requirements comes the design phase where the architecture layout of the proposed system will be created. During the process, different software solutions will be analyzed that can be used to achieve the goals. After which the final architecture of the required service will be created.

The design process itself will be iterative where components will be added to the system as needed in order to fulfill the requirements.

The whole document can be viewed as a project where requirements and restrictions for the problem are created which then need to be solved during the course of the thesis.

1.5 Defining Scope

Creating a log management system for an organization, in general, would consist of several steps in order to come to a fully functioning solution. The following steps should be taken when starting from scratch:

- **Creation of organization-wide policies that define how to handle logs.** This would contain instructions on what logs have to be stored for how long and also would define the general approach to log centralization. Information on how the incoming data should be analyzed and what data should be present for log events.
- **Infrastructure for log management has to be created and maintained.** The designed system has to be capable of receiving large amounts of data over secure channels and store it for a predefined timeframe. Data volumes that the infrastructure has to handle can vary a lot because during incidents it is possible that the amount of incoming data can be multiple times larger than during regular operation. The data also has to be secured from accidental or intentional modification or deletion.
- **Organize support for staff with log management responsibilities.** The goal is to have unified logging procedures that can be applied throughout the organization so the staff has to be trained so they would know which logs need to be centralized and in what manner. Also, training is required for people who will use the data to know what data is available and how to search the huge amounts of data.

In the scope of this thesis, the focus is on the log management infrastructure part by creating a list of requirements that the solution should adhere to and then creating such a system and find tools that could be used to realize it.

For the creation of the requirements, several different types of log generators are used ranging from single node applications to large services that are being run in large Docker swarms. The number of different software solutions that are analyzed in this thesis will be scaled down to three in order to keep the length of the document reasonable.

1.5.1 Log Generators in Scope

For the purpose of being able to create requirements that could be applied universally, log generators that are as diverse as possible are taken as a basis. For this reason, applications that generate logs as files and applications that can send out data over the network are in scope. Application cluster size has to be taken account also to make sure that the input of the log management system can scale from single node applications to larger ones. Another large source of logs is servers and workstations used in the organization and for that reason, it has to be ensured that the solution is able to handle logs from those sources.

1.5.1.1 Single Node Application

This is an application that is running on a single host and is generating log files locally. Since it is a 3rd party application modification to logging is not possible therefore whatever logging has been set up by the provider has to be adapted. As an example of this application, Sonatype Nexus OSS application is used [2]. This application generates several types of logs:

- Access logs
- Application logs

Both log types have to be gathered as the first one gives us data about user activity with the application and the 2nd one allows us to troubleshoot in case of problems with the application and possibly also predict some issues.

1.5.1.2 Single Node Application (in-house)

This application is similar to the previous one with the main difference being that it is an in-house developed application and it is possible to change how logging works with this application. In the scope of this application, a solution will be tested that does not rely on log files at all, but events are sent directly to the logging solution. This is a simple web application using expressjs [3] andmorgan [4] middleware for logging purposes. Since it is possible to change the logging system used by this application it could be used to see what changes would need to be made to skip the creation of files on the application server and send the log events directly to the designed system.

1.5.1.3 Small Service

This service consists of several different applications, but together they make a single service so from a log aggregation point of view logs from all these applications should be looked at as a whole. The application used for examples here is Atlassian Bitbucket DataCenter edition [5]. This service consists of the following components:

- **Load balancer:** Application used in this case is HAProxy
- **Search database:** Elasticsearch is used to provide the search database
- 2 x Application nodes

Logs from the DBMS system and shared file system are not in scope for this as that is handled when setting up logging management for DMBS in general.

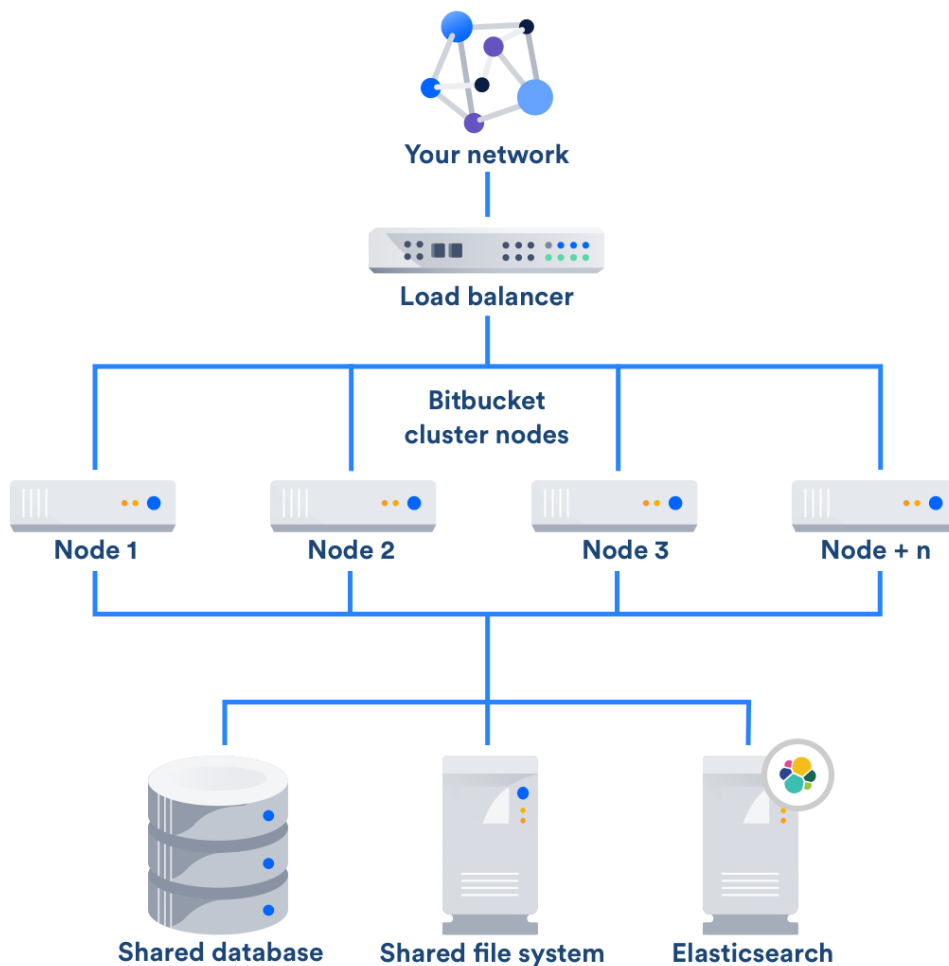


Figure 1 - Small Service Architecture Diagram [5]

There are several types of logs generated and the following types need to be consolidated:

- Cluster access logs
- Cluster audit logs
- Cluster java logs
- Elasticsearch logs
- Load balancer logs

Description of how the Bitbucket nodes do the logging can be found on their web page [6].

Documentation on how the Elasticsearch logging can be configured can be found in the documentation [7].

As a load balancer in this instance, HAProxy is used with documentation for logging available on their web page [8]. In case of HAProxy getting the logs is a bit more complicated as there are no logging components build into HAProxy itself but it relies on 3rd party tools for the logging.

1.5.1.4 Large Microservices Application

The largest system that is within the scope of this thesis is a large client facing web application that is using microservices architecturally. High availability is required by the application so at any time there are at least two instances of any component running with load balancers distributing workloads. The application is deployed using containers and an orchestration engine. Due to this architecture, this allows starting additional nodes of any microservice very fast if the load on the system requires it. This also means that containers are destroyed if they are no longer needed. This raises the problem that all the logs generated into that container would also be deleted on its destruction. Having the logs transported to a central location for storage is a good solution for this. The service consists of the following components:

- **Load Balancer.** HAProxy is once again used here to load balance between application nodes.

- **HTTP Accelerator.** Varnish is used here to cache the static data to speed up the application and also to balance between microservices.
- **Web Server.** Apache web server is used for serving static content.
- **N x Microservices.** These are in-house services that offer content over REST APIs.

Each of these components will have N number of instances running at the same time.

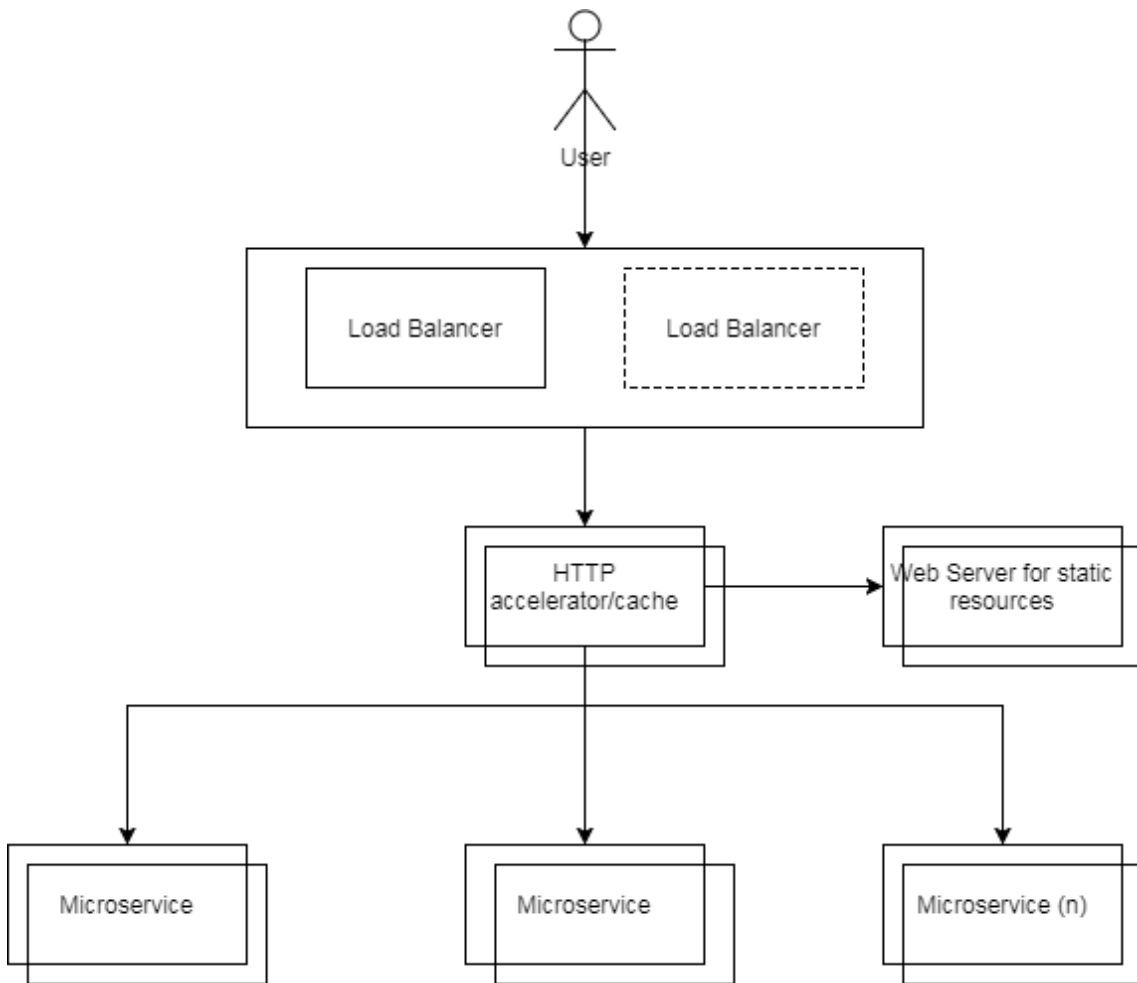


Figure 2 - Large Service Architecture Diagram

The general flow of the application is that web pages are put together using the static content supplied by Apache servers and then data shown is gathered using REST API calls to the microservices. Access logs and applications logs are generated by this application.

1.5.1.5 Servers and Workstations

One of the biggest sources of logs in any organization is the servers that run the applications and workstations used by staff in everyday work. For this reason, the designed infrastructure has to be able to grab log entries from UNIX system logs and also Windows Event Logs. In case of UNIX systems, it is assumed that rsyslog is used to do the logging on the operating system level.

1.5.2 Further Limitations

1.5.2.1 Cloud Services

There are cloud services which could be used for the purpose of storing and displaying the log entries so using those would mean that one would only have to set up the part of gathering the logs and possibly parsing them in case of custom formats, but the storage side and displaying it would be handled by the service provider. This approach is feasible in some situations but might not be worth it in others. This solution can be considered when dealing with logs that are not security related and the source of the log is not coming from the organization's data center but from applications running on client machines. The problem with using this for applications that are running inside the organization is that this could quite quickly overload the connection to the outside world due to all the traffic generated by transferring logs. This would also be more important when there are actual issues and the number of logs generated has increased meaning it has potential to cause more issues.

1.5.2.2 Data Volume

An important factor when choosing the solution will also be its ability to process the log volume. For that reason, data estimation needs to take place. In the table below the estimated data amounts for the applications that are in scope can be found. The log amounts for servers and workstations are a bit harder to estimate, but during normal working, it is expected that they generate about 1MB of data per day.

Application	Data amount per day
Single node application	1GB
Small service	5GB
Large application	50GB
Servers	2GB
Workstations	3GB
TOTAL:	60GB

Table 1 - Expected Data Amounts

The table above shows the data amounts for single applications, but the organization has several other applications also which when it comes to sizing will have to be taken into consideration. Due to this, the expected volume of data per day that is taken as a basis for this thesis amounts to 150GB of data per day.

2 Requirements

Requirements are divided into two main categories and consist of functional and non-functional requirements.

2.1 Functional Requirements

- **Logs have to be available in a central location.** There should be a single point of entry to be able to access the logs
- **Files as log source.** It should be possible to read the files that the application is generating as the source of the log data. This is the most common use-case as currently, most applications create log files on the servers that they are running on.
- **Receive log data over the network.** Sending log events directly from the application to the central solution should be possible as in some writing files with the log data can be avoided. This is especially true in Docker containers where the files that would be written are destroyed on container destruction.
- **Events have to be analyzed.** The logs have to be parsed into more manageable bits to enable searching for certain values, for example, to search HTTP access logs for all 4xx return values.
- **Standardization of input data.** As the format in which log entries are generated varies between different applications the system should take those incoming events and convert them to a unified standard.
- **User Access lists.** It must be possible to configure who can see what data. This should work on the service, application and log type level.
- **Archiving the data.** Some data has to be archived in case it is needed later for auditing or some other reason.
- **Monitoring alerts:** It should be possible to automatically create alerts based on incoming data. As the amount of data that is coming in is very large it can't be

expected that someone is constantly looking at it to be able to detect issues. For that reason, alerts should be set up with appropriate triggers. An example of it would be that in case a web application the HTTP responses with code 401 are above a threshold there might be an attack happening on the service so an alert should be raised so someone would have a look.

2.2 Non-functional Requirements

- **Read-only system.** Log data modification by users should not be allowed to preserve the integrity of the log.
- **High availability has to be achieved.** The solution needs to be able to survive interruptions in any part of its design as the access to logs at any time is critical.
- **Access has to be logged.** Who has viewed what data? The data about access has to be also stored for security reasons in order to be able to later verify what was viewed.
- **Maintenance works on the system should come with minimal downtime for the users.** Since log data has to be available at all times for users the infrastructure should have minimal or no downtime when changes to the system are required.
- **The data availability duration.** The data has to be available on the cluster within 30 seconds of the application creating the logline.
- **The service has to be easily scalable in case more applications are added to it.** As the number of logs coming into the system will only increase with time as more and more sources are added, it is required that the system could be scaled to handle the increased load.
- **Secure transport of the logs.** The transport of the logs from the application server to the central system needs to be encrypted to minimize the risk of log alteration during transport.
- **Configurable log retention.** The time that logs are kept has to be configurable depending on the service, application or log type.

3 Solution Design

The architecture design to be used will be created in three main steps:

- **Initial architecture.**
- **Tool selection.**
- **Adapt the architecture.**

In the first step, a high-level design will be created in order to be able to identify components that are part of the final design. For the creation of this, only the functional requirements are considered and what components might be needed to achieve this.

The next step would be to find suitable applications that can be used instead as those components. In this part examination of different applications that could be used in the final design takes place and the infrastructure design is expanded to match the tools that are chosen.

In the final part of the design phase, the solution has to be made compliant with all the non-functional requirements. This will have changes on the design and might even introduce new components if needed in order to fulfill all the requirements.

3.1 Initial Architecture

The current situation is that all log generation sources are writing the log events into files. The files are spread out on the hosts where the application or system is running and there are a very large number of those hosts. The files are then left on these hosts and if administrators run into issues with these applications they have to access those hosts and use command line tools in order to view the data inside the files. In some cases the files can be several gigabytes in size making the use of the command line tools rather hard as it can be quite hard to find the required data for and searching the files takes a long time. To make matters even more complicated the logging format of applications does not have any standardization making it harder for administrators to switch from working with one application to another. Unfortunately, this creates an environment where specialization on certain applications is going to happen therefore lowering the change that there are several people able to tackle a problem less likely.

First, a general workflow of how the log entities will be going through the system is created. To do this a simple diagram of the workflow was created.

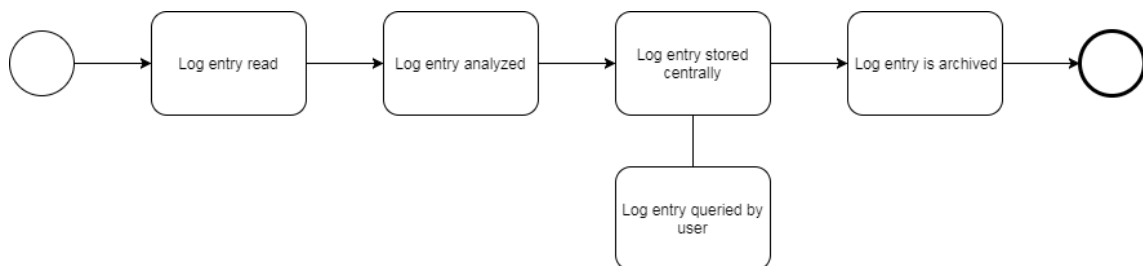


Figure 3 - Basic Workflow

From this workflow, it became apparent that since every application has its own log files which are stored with the application and the format of the logs also varies between applications, it would not be feasible to manage the reading and analyzing part of the workflow by a central team. For this reason, the reading and the analyzing of the data will need to set up by the team who is responsible for the application. This also means that analyzing of the log entries has to be done before they are forwarded to the central location for storage.

An overview of the types of log generators that are going to be used is a must and the differences between them also need to be taken into when deciding what components are required for reading in log entries.

- Applications that write logs to files
- Applications that write logs to network socket
- UNIX syslog (rsyslog)
- Windows event log

With the information gathered from the workflow, it is possible to create an initial design of how the infrastructure for the log management system could look like. When looking at the list of requirements it can be assumed that initially for each of the activities a separate component is required, therefore the following components are in scope:

- **Read log files.** The log files are read from the hosts where they currently reside on. Since all the files reside on host machines for the application this means that for each application separate agent has to be installed that will then read the logs of that application. As the number of applications is too large to manage this centrally the responsibility for the initial processing of the log entries will fall to the application teams. Reading of the files has to be reactive, meaning when a new entry is created into the file it should be read as soon as possible for further processing. In some cases, applications are not writing into files but sending the data over the network. For these cases, the log reading component should also be able to accept connections and be able to receive the data in order to send it further. Initially, two separate components are used for this.
- **Analyze log file.** Since all the applications don't follow a standard when writing the log entries it falls upon the application teams to standardize their logs. In case of in-house applications it might be possible to do this already on the application side, but in case of 3rd party applications, it is likely that the log entry generated by the source have to be transformed to a standardized format. There will be some mandatory values that have to be present for all applications and naming convention has been set up for common values between applications to make it easier for administrators to switch between applications.

- **Forward log file to central storage.** After the logs have been read in and analyzed transportation to a central storage location is required. The channel that is used to move the log entries has to be secured in order to avoid tampering with the data. There also has to be some fault tolerance set up that the system would not break in case of network outages or when the amount of data that needs to be transferred gets too large.
- **Output the data.** When the data has been stored in the central storage solution there has to be a way to view it. Since the data has been analyzed and standardized it should be possible to narrow down the data query to entries that have values of interest to the administrator at the time. This can be something simple like only to output entries from a specific application between a certain timeframe or to very complicated queries to output data from a certain host where a predetermined functionality of an application was used by a defined user.
- **Archive the data.** The amount of data that potentially has to be stored can grow to be very large. In order to not bog down the system, some log entries have to be moved to an archiving solution. For this a separate component is required that could efficiently store large amounts of data. The archiving process itself should allow the data also be restored in case a need arises. When to archive a log entry will be dependent on the application itself and the type of log event in question.
- **Monitoring alerts.** There is a requirement to be able to set up a notification in case some predefined conditions are found in the incoming data. This is a separate component that periodically checks the incoming data against some triggers. This functionality can be used as an early warning system for administrators to quickly react to potential issues.
- **User authentication.** As the data that is being gathered by the system most likely contains secure information there has to be a way to restrict access to the data. The best solution would be if it was possible to set up access lists of what types of data what person or group of people have to access. Authentication is set up as a separate component as it is not clear in what part of the solution authentication is applied at currently.

From the previous list of components, the following diagram has been created.

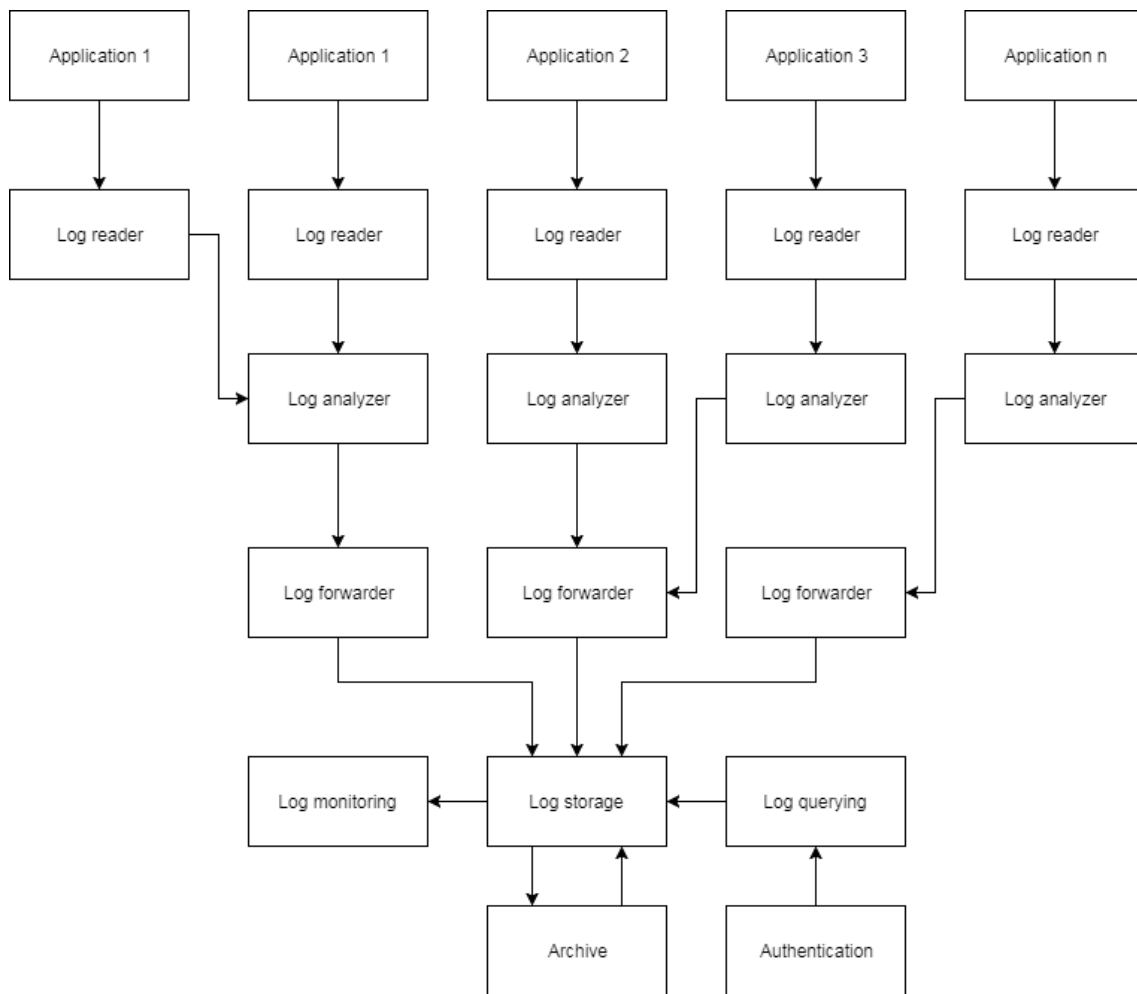


Figure 4 - Initial Component View

As can be seen, there are quite a large number of components and in some cases, there might be several instances of the same component running. This is best seen in case of log reader components as there has to be one available at every location where logs are generated. In the later parts of the workflow, it should be possible to reduce the number of component instances by reusing already existing ones. Also, some logical grouping would be beneficial for example in case of larger services the logs from different applications that make up the service could be transferred to a log analyzing instance that is capable of parsing logs from all of the applications that make up the service.

3.2 Tool Selection

Now that there is a good overview of what components are needed and what activates need to be taken in order to achieve the log management process, it is time to start looking at applications that could be used for it. Doing some searching on the internet reveals that there are several solutions available for analyzing and centralizing logs but quite a lot of them only offer cloud-based services. SaaS-based solutions have been ruled out previously so the list of possible options grows significantly smaller.

Most of the software seems to offer quite a lot of functionality in the same package and the only part that would need to be set up separately is the initial sending of the data to the application. After a lot of searching on the internet to find what the popular tools are for log management three potential candidates were chosen for the setup of the system:

- Splunk [9]
- Graylog [10]
- ELK stack [11]

These three products were chosen as from initial inspection they could be used to meet the requirements that were set. Another reason for choosing these three products was that they are quite different from each other. Splunk being a full enterprise commercial product that in theory should mean it would be simplest to set up. Graylog is an open source program that is offering a lot of the functionality that is required. ELK stack is the most DIY offering of the three with many of the features not offered out of the box but have to be added either using other applications or plug-ins.

3.2.1 Splunk

Splunk as an American company that is offering an application with the same name with the purpose of collecting analyzing and then visualizing log and machine data. Out of the chosen products Splunk is the only fully commercial product. It was chosen as one of the candidates due to being one of the first solutions around years ago, so it has had time to mature and offers a large set of features. For analyzing log data they are currently offering three different products:

- Splunk Enterprise
- Splunk Cloud
- Splunk Light

The cloud offering can be discarded immediately. For a comparison between the versions, a handy web page is hosted on their site. [12] From this page the Splunk Light version can also be discarded as it offers a max daily volume of 20GB per day which is less than the expected amount. This leaves only the Splunk Enterprise version as a possible candidate. The documentation for the application seems quite adequate and with the Enterprise version support for the product is also included. The feature list for the product is very impressive and it offers some features that were not set up as a requirement. The most useful part seems to be that by default Splunk is able to make sense from a lot of standard log formats so application teams would not have to spend time on describing their log formats for the shippers. Searching through the data also seems to be convenient in the application, as they have developed their own syntax for doing so. I've added a diagram of the high-level architecture of a system that has been set up using high-availability.

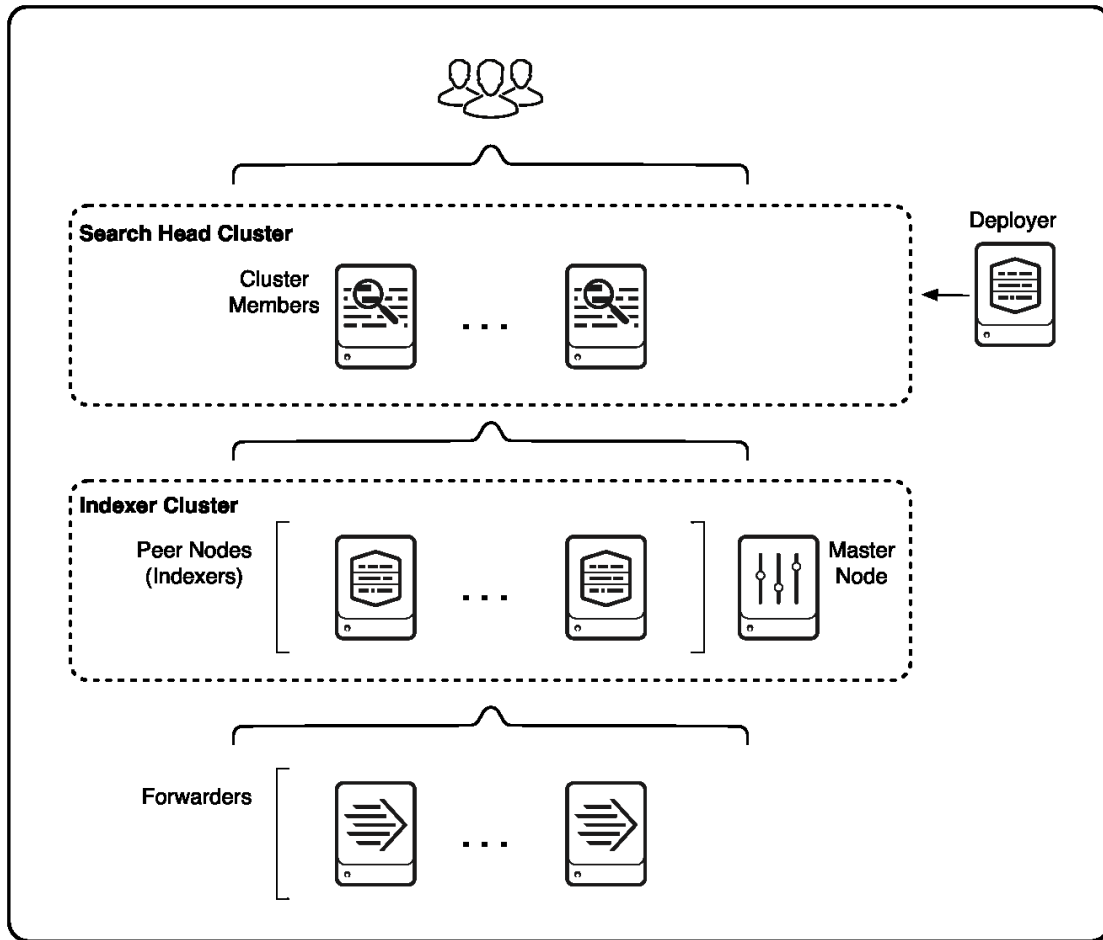


Figure 5 - Splunk High-level Architecture [13]

As can be seen, here high-availability is achieved by having more than one node of each component that makes up the application. The diagram is showing three different layers of the application

- **Data input.** External data is consumed by these forwarding nodes and is then sent to the indexers. These forwarders are also aware of all the cluster indexers available and load balance between them as needed
- **Indexing.** Here the data is stored and ready for being queried.
- **Search Management.** This is the layer of the application that handles all searches that are done on the data in the indexer layer.

Inside the cluster, there is also an additional master node. The purpose of this node is to regulate the functioning of all the other nodes.

The most important thing would be to make sure that Splunk is able to handle all the data sources that are required. By default, it seems that the Splunk instance can monitor files and directories that are available on the host where it is installed to. This is not something that is very useful as installing Splunk instances on all the hosts in the organization is not feasible as it uses quite a lot of resources. It is also possible to upload files directly from the interface. This functionality could be useful in some cases, but for analyzing application logs it would not work as everything would be done as a reaction to some event and having someone to upload logs from all the hosts is a task that is not feasible. Splunk does offer the option of setting up TCP/UDP ports to listen to incoming data, this would satisfy the requirement to be able to receive information over the network. The incoming data is also be secured with SSL which was a requirement that was given. For reading files on remote hosts and forwarding them to the Splunk instance a separate application is offered named Splunk Universal Forwarder. This application would have to be deployed on the source host and it would then monitor files and directories on that host. After reading the files the data would be sent to the central indexers for processing over a secure channel. With these options available it would be possible to send in the log data from all of the sources that are in scope. It is also possible to manage the configurations of all the Universal Forwarders from a central deployment server once initial installation has been done making maintaining the system much simpler.

For viewing the stored data Splunk has developed its own Search Processing Language. The language allows you to search for specific values from the log lines that one might be interested in. There is a decent amount of conditionals included allowing for the creation of specific queries. There are also visualization tools available in order to make dashboards with a wide variety of graphs and such and from which regular reports about the data could be made. It is also possible to export these into PDF format for use in other documentation. From the security perspective access to this data can be controlled with granular access control. There are several options available for user directories with LDAP/AD availability but also support for SAML single sign-on solutions.

Splunk also allows for the creation of monitoring alerts. It is possible to set it up in a way that e-mail alerts will be sent when certain conditions are met. The more interesting part is that it is also possible to make the system to trigger actions instead of sending out

just e-mails. This means that it would be possible to automatically remedy a situation that is affecting the system.

The application also has archiving part covered with two main options on how to deal with it. One option is to keep the data in Splunk itself and it seems to compress it into a different format. In which case searching it would be much slower, but it would still be visible. The second option offered is to use an external Hadoop cluster for long-term storage or even to use Amazon S3 instances.

Overall it can be said that it is an excellent tool and seems to meet all the requirements that were set forward. The major drawback of all these features is that it all comes with a hefty price tag. The cheapest option available on their website has a price of 600\$ per GB for a month. [14] With the estimation of 150GB of data per day this would come at a cost of 90000\$ per year. Also one would have to take into account that if the amount of data grows, so will the licensing fees.

3.2.2 ELK

ELK is shorthand used for Elasticsearch-Logstash-Kibana. It is a set of tools offered by Elasticsearch company that if used together provide the ability to collect logs, store them and the display them. Out of the three options considered in this thesis, this is the most customizable solution on offer. The only part of the ELK stack that has to be used is the Elasticsearch database server. The other parts can be replaced with alternative applications that might suit the needs better than the default offering. When searching for solutions for log centralization ELK is one of the most popular options that one would be presented with. The main reason for this is, that the by default this solution does not have any license fees attached to it so it's a very likely first choice for organizations to create PoC solutions in order to see how log management could benefit them. There are some features that are offered for the ELK stack by the company that does require you to have an active subscription, but adding those features is a matter of installing a plug-in to an already existing installation so it can be added as the need arises. As mentioned before ELK consists of three separate applications.

- Elasticsearch
- Logstash
- Kibana

From these 3 applications, Elasticsearch is the storage solution where the data is stored. Logstash is the transport and analyzing agent used to move the data into the database. Kibana is used in order the view and visualize the data present in the Elasticsearch database. There are popular alternatives available for both Logstash and Kibana.

3.2.2.1 Elasticsearch

This is the heart of the ELK cluster, where all the data is stored and queried from. The main reason why Elasticsearch is used for storing the data instead of traditional relational DBMS solutions like PostgreSQL or Oracle is the speed at which Elasticsearch is able to return results for queries. This is most apparent when doing full-text queries over very large data amounts. Elasticsearch is usually installed in a cluster that consists of many nodes and that cluster is very easily scalable. It is possible to add or remove nodes without any downtime and when set up correctly without any data loss.

To achieve this Elasticsearch is using the Apache Lucene indexing and search library. Data in the Elasticsearch cluster is stored in indexes. The index can be looked at as a grouping of similarly defined documents. It can also be looked at as if indexes are database tables.

These indexes can then be divided into shards. Each shard in itself is a fully-functional index. The reason for this is to allow horizontal scaling of the data. It can be possible that a single index can contain more data than available on a single node inside the cluster, for that reason indexes can be split into separate shards that can be then distributed between the available nodes in the cluster. As a result of this, it decreases the load on a single node of the cluster also increasing the throughput.

Another core feature of Elasticsearch is replication. At any time it is possible to have more than one instance of a shard available. Having more shards available has two main benefits. Firstly it ensures that in case there are node failures inside the cluster there are replicas available so no data loss can occur. And as data searches are executed on all replicas it can speed up search performance.

Elasticsearch does have a featured named snapshots and restore that can be used for the purpose of archiving. It is possible to create snapshots of images on local hard drives, Hadoop clusters or Amazon S3. After a snapshot has been created it can be removed from Elasticsearch and quickly restored when needed. In order to make the management of this simpler log data for each day should be stored in separate indexes.

A separate tool named Curator has been released by Elastic to curate and manage Elasticsearch indexes and snapshots. This is a command line tool that would then need to be set up to run periodically to snapshot the indexes you want to archive and also remove the indexes that you no longer need.

3.2.2.2 Logstash

Logstash is a data collection engine with real-time pipelining capabilities. From log management view Logstash is an application that is deployed on software hosts in order to read in the data from logs and to transform the log events into an improved searchable format. Logstash comes with a wide array of input plug-ins out of the box which can be used for reading the data from hosts. Reading from files comes with the default installation. There is also the possibility to open TCP/UDP sockets and receive

events over the network. As a bonus it comes with a syslog plug-in which can read syslog events over the network, this allows to set up a Logstash instance in order to read events from a large number of infrastructure devices that don't have file-based logging capabilities.

After the data has been read into Logstash it can then be transformed to a better format for indexing. What one would want to do in this part depends a lot on the quality of the incoming data. In case of reading from the log file, the usual format is one big string which contains the whole logline. In this case, the log lines probably follow some logic which can be applied to it in order to extract data from it. As an example, this would mean extracting usernames, response times, response codes etc into separate fields. This allows for much simpler searching for results after the log event has been stored in the database. It is also common that log events are written to file as JSON string with the meaning of values already designated. This makes life a lot easier as log events already have a good structure when reading into Logstash. With Logstash it is also possible to enrich the data before sending it for storage. For example, you might want to populate a certain field with a value that depends on response time in order to quickly see how many slow requests are being processed by the system. After the processing of the data has been completed it is then sent to an output. There is a decent list of output options available by default. In this case, the Elasticsearch output is used in order to send the data for storage.

There seems to be a general opinion that Logstash uses a bit too many resources in order to do what it does, this is due to the fact that it runs on the JVM as it is written in JRuby. Another common issue seems to be that it is very hard to debug what is happening with the log event inside the application so in the case where log events are not analyzed correctly fixing it is very complicated. Due to this another log gatherer has emerged that is gaining popularity that could be used as a substitute for Logstash, that tool is named Fluentd. The main difference between them comes from the design focus each of them has. Logstash is focusing on flexibility and interoperability whereas Fluentd prioritizes simplicity and robustness. As a result, there are two different products that can be used to do pretty much the same thing. The biggest factor that has to be taken into account when choosing between the two is that they use different approaches when processing events. Logstash uses algorithmic if-then statements to decide on what to do with events whereas Fluentd uses tags. In case of complex message processing the tags system can

end up being more understandable, but in most cases, it is not needed. The big difference comes on how transport of messages is handled. This is where Fluentd has an advantage as it has the built-in reliability that ensures that messages get to where they need to be. Logstash by default keeps the data that it is processing only in memory and in case there is a problem with the Logstash instance the data can be lost. Another issue coming from this is that since the memory queue uses a fixed size and cannot be configured. In situations where there is a lot of incoming messages, this can cause some messages to be missed because the memory buffer was full. In the recent versions, a new feature has been added named persistent queues which protect against data loss on application crashes, but it does not help against data peaks. In order to solve the issue with data peaks, an external buffering solution should be used. This could be a Redis server or Apache Kafka for example. Both of these applications can be set up to act as FIFO buffering applications in order to allow Logstash to pull new events at the pace that it is able to process them.

In order to cut down the resource usage on application hosts for log management, a separate log reader was created for Logstash named Filebeat. Filebeat is a lightweight application that reads data files and then forwards the events to Logstash for further processing. It is also able to do simple buffering for Logstash in case the volume gets too high for Logstash to handle at any given time. In some common formats like Apache web server logs, for example, it is able to parse the log events into analyzed data and can send the data directly to storage.

3.2.2.3 Kibana

Kibana is the application that is used for searching data and also for visualizing the data. Theoretically, you could query the Elasticsearch server directly using its REST API and get the results in JSON format. The query language to get the data from the REST API is not that simple and as all the results are just documents in JSON understanding what is happening can be quite tricky. For that reason, Elastic offers an application named Kibana as the frontend from where it is possible to do searches and visualize the data you have stored. It can also be used to create dashboards to get a quick overview of the status of the application.

For visualizing data there are also alternatives that can be considered. One of the possible options is Grafana which can be used to create visualizations to present the data

you have. The main drawback of Grafana is that it does not support searching and exploring the data inside the cluster and is just for visualizing the data. This means that it can be used for the purpose of visualizing metrics and KPIs that are aggregated from log data, but in case of troubleshooting issues, it is not much help. A feature that Grafana does offer that is missing by default from Kibana is user management. Grafana allows setting up a role-based access internally to control who has access to what dashboards. It would also be possible to run both applications in parallel and for example to use Kibana for operational issues while using Grafana to display metrics and KPIs of the application.

3.2.2.4 Conclusion

While the ELK stack does offer the most freedom when coming up with designs for log management it is also the setup that would require most work to maintain. There is chance, that there will be someone dedicated to just keeping the stack operational and making changes as needed. The whole infrastructure that would be required to set up can be found in the documentation for Elasticsearch and it is quite large.

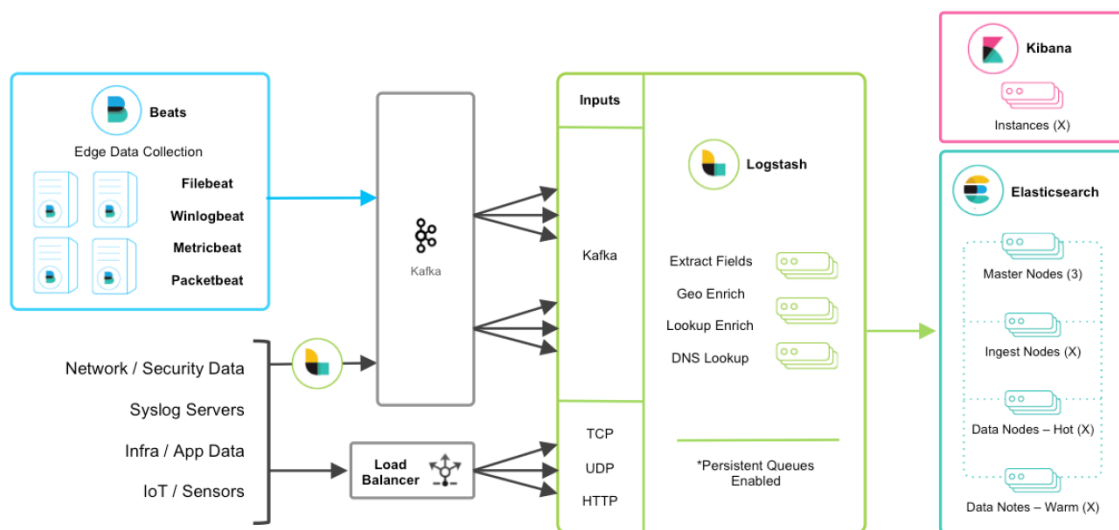


Figure 6 - ELK Stack Architecture [15]

The number of parts in this system would be very large and someone would have to keep all this running at all times. The big issue is also a lack of any security in the default configuration. To have officially supported plug-in to enable security for the ELK stack a subscription is required which in case of a 10 node Elasticsearch cluster can cost up to 35000\$ yearly. There are also free alternatives available for

authentication and authorization like search-guard, the downside of the non-official solutions is that they are updated slower than the official ones which are released at the same time as new Elasticsearch stack versions. This can lead to situations where updates are required due to bugs in the stack but a new version of the 3rd party plug-in is not available.

Another feature that is missing from the default installation but can be achieved with the subscription is monitoring and alerting. Theoretically one could use an existing monitoring solution to directly query the Elasticsearch database for parameters and trigger events that way, but this can lead to creating a lot of load on the Elasticsearch cluster which in turn can lead to instability which you would want to avoid.

3.2.3 Graylog

Graylog is an open-source log management solution. In many ways, it seems to be similar to Splunk with the difference in the cost. Most of the features on offer by the system are free, but there are some features that are considered enterprise and licensing fees are required for it. With the enterprise, version support is also included in addition to the added features.

When compared to the offering from Splunk, Graylog more customizable, meaning you have a lot more control over the final setup. Where with Splunk you had a single install file that encapsulated everything, Graylog consists of several applications that have to set-up in tandem. At its core, Graylog seems to be further development of the ELK cluster that was looked at previously.

The main database used is an Elasticsearch cluster where all the log data is stored. In order to make the usage of the database as fast as possible, the Graylog application connects to the Elasticsearch cluster as a node and does not use the REST API for pulling and storing the data. This is much faster as in this case the data does not have to be transformed into JSON and then back in order to be used. This behavior has been changed starting from Graylog version 2.3 where they started to use a lightweight HTTP client to connect to the Elasticsearch cluster. This has resulted in much less coupling with the Elasticsearch versions allowing for a wider variety to be used.

The application configuration itself is stored in a MongoDB database set. This is where all the users, visualizations and dashboards are stored. The application is set up in a way that if this database is down, the data input part of the application would still be working but viewing the data is not possible. It is recommended to set up both Elasticsearch and MongoDB in a high-availability setup.

In order to achieve full high availability, it is possible to run several Graylog nodes in parallel and load balance between them. In case one of the nodes would fail the operation of the cluster would not be affected.

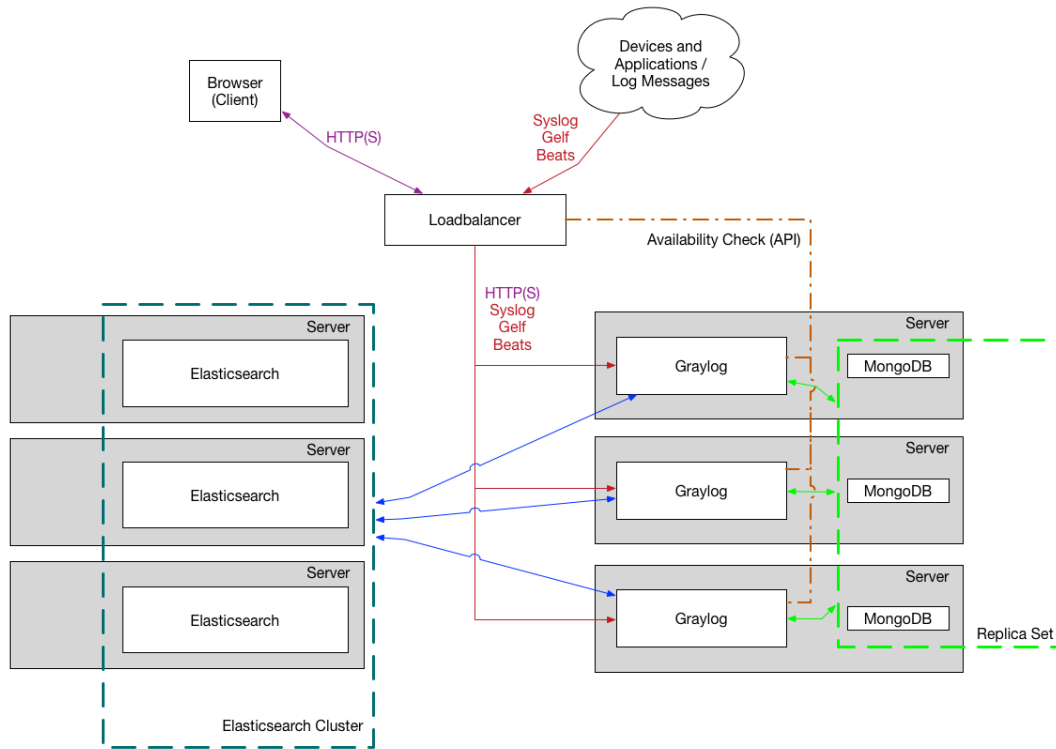


Figure 7 - Graylog Architecture [16]

The typical setup for a larger installation of the Graylog application would look something like this. This setup is fully high-available and failures in any part of the system should not affect the application as a whole. The load balancer also has to have a failover node available for this.

For inputting the data there are several options available. It is possible to use the Logstash and Fluentd applications that were discussed in when looking at the ELK cluster. Both of these applications have output plug-ins for GELF format. GELF is Graylog Extended Log Format which defines some structure to the messages that are being sent. This means that everything that was possible to be sent to ELK is also possible to do with Graylog. However, there is also the option to collect the data using Graylog Collector Sidecar. It can be deployed on Windows or Linux hosts and have the following structure.

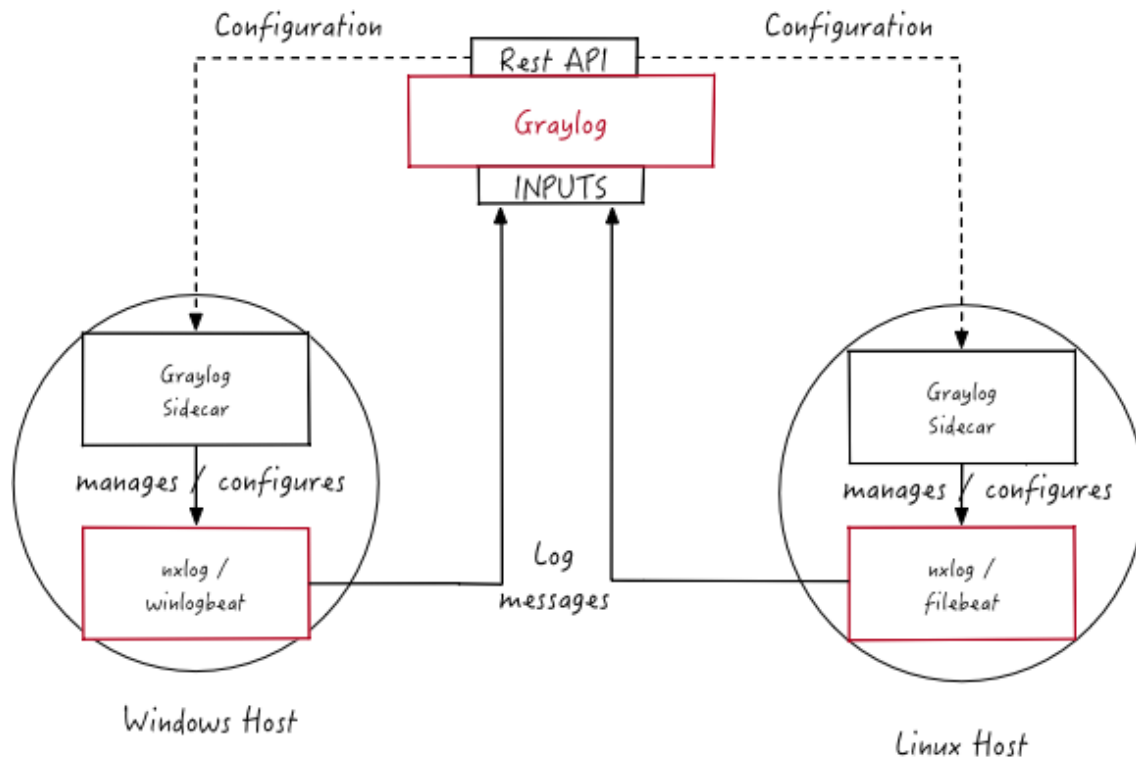


Figure 8 - Graylog Collector Sidecar [17]

When sidecars are deployed you have to assign them tags and then you can centrally configure what data is being sent from the hosts. This is a very good feature for larger application deployments as configurations on a large number of hosts can be maintained in a central location making changes much simpler. The sidecar itself has been built so it has minimal resource usage on the host.

From a security side, all the data being transported can be encrypted in order to avoid tampering. It is also possible to encrypt the connections between the Graylog application and the Elasticsearch cluster.

The analyzing part of the log entries can happen in several locations. If logs are being transferred using Logstash they can be parsed there before being sent to the Graylog server. If the Sidecar is used to collect the logs from the servers there isn't much that can be done on the host itself. Basic filtering can be done in order to exclude some data that you are not interested in but the general analyzing of the log events will take place on the Graylog server. For this Graylog has a featured name Extractors that can be configured using regular expression or grok patterns in order to parse log events in custom formats.

User management has been built into Graylog by default a built-in user directory is used, It is possible to use LDAP or Active Directory as user directories in order to avoid having to set up users locally in the applications. Access is managed via roles that can be defined. By default, every user will have some role attached to them. It is also possible to use access tokens for authentication in case you want to set up some automated process that extracts data from Graylog for some purpose. It is also possible to extend the possible options for authentication providers via plug-ins to include for example SSO solutions.

To have archiving set up by the Graylog application it is required to buy the enterprise license for it. The cost of the license is 9000\$ per node of the application used, so with a 2 node cluster that would be a cost of 18000\$ yearly. Since Graylog stores everything in an Elasticsearch server it would be possible to set up a custom solution as described in the ELK section of the thesis. In order to do this, indexes need to be set up to rotate daily, which allows to set up a job to snapshot the indexes from previous days that are no longer getting any new data.

Graylog does come with a built-in monitoring and alerting system. One has to set up conditions that would be creating these alerts in the monitoring system. The types of alerts can look for values in some fields or use aggregated values in order to make more complex alerts. It would be possible to check that if the number of incoming requests is outside of some standard deviation number an alert is triggered. What happens after the alert is triggered is also configurable. It is possible to send out e-mails or to make some HTTP requests which in turn can trigger actions to remedy the situation. This could easily be used to monitor the load of some application and if it gets above a threshold trigger creation of more application nodes in order to not overload the existing nodes. The alerts once triggered remain visible in the application until the condition that triggered the alert is no longer active, then the alert is moved to a resolved state.

The application also has an interface for searching for certain messages from the dataset. The search functions quite similarly to how it is set up in Kibana and both of them use Lucene syntax in order to search for messages. In fact, it is quite simple to set up a Kibana instance to be used with the data gathered by Graylog. It is also possible to create dashboards which include pre-defined views of the data. This can easily be used

to gather metrics about applications and to get a quick overview of what is happening with them.

As a conclusion, it can be said that Graylog works quite similarly to the ELK cluster, but some work has been put into in order to make the operation of a log management system simpler as some of the work is done for you by the application. It could be said that Graylog is a development upon the ELK stack. The main benefits being that the management of indexes is done by the application and a central configuration platform for the log gather agents. It is also beneficial that it comes with a user management system and alerting options which are missing from the free version of ELK.

3.2.4 Comparing the Tools

All of the log management solutions that were examined boasted a lot of features that could all be used when setting the system up. To have a better overview of the situation I've created a table for comparison.

Feature	Splunk Enterprise	Graylog	ELK
Read logs from files	X	X	X
Receive logs from network	X	X	X
Parse logs	X	X	X
Search logs	X	X	X
Create visualizations	X	X	X
Control user access	X	X	X/License required
Monitoring and alerting	X	X	X/License required
Configure log retention times	X	X	X
Archive old logs	X	X/License required	X
Secure transportation	X	X	X
High availability	X	X	X

Table 2 - Feature Comparison

It should be noted that Splunk Enterprise is an application that requires a license and there is no free version available for it.

For the purpose of reading the log data from files, all three applications have similar approaches. They require you to run an agent on the host machine where the files are located which then reads in the data and forwards it for processing. Splunk uses a custom application to achieve this while both ELK and Graylog use similar solutions. The main difference between Graylog and ELK comes from the fact that it is possible to install an agent on the host machine that takes care of configuring the application that gathers the log data. This allows for Graylog to manage the data collectors from a central location giving it an edge over the standard ELK stack deployment. Central management of log forwarders is also a feature that is present in the Splunk Enterprise application.

For the purpose of being able to receive logs over network sockets, Splunk and Graylog have an advantage over the ELK stack. In case of an ELK stack, a separate Logstash or Fluentd instance would have to be set up in order to receive the data over the network and then parse it. With Graylog and Splunk, the support for receiving the data is directly in the application so no extra steps are needed. Both of the applications support receiving the data over TCP/UDP but also support for getting data as HTTP messages with JSON payloads is supported. It is also possible with all of the applications to receive data over TCP/UDP in syslog format meaning it's possible to send the logs from Linux servers.

For the purpose of sending logs from Windows event logs, both ELK and Graylog use Winlogbeat. This application is very similar to Filebeat that is used by those applications to read files but instead of files, it is able to read Windows Event Logs when running on Windows hosts. The reason that a separate application is required is that files where Windows stores its log data are not directly accessible and can only be accessed using Window API. In case of Splunk, it is also possible to send data from Windows logs but it is using the same Universal Forwarder that is used for sending data from files on both Windows and Linux hosts.

Another data source that might need to be used is databases. For Splunk, there is a tool named Splunk DB Connect that can be used in order to connect to a traditional database and pull structured data from there. For ELK and Graylog Logstash can be used with its JDBC input plug-in, there are also plug-ins available for this for Fluentd.

Parsing the logs to extract the data you are most interested in from the log events is a critical part of any log management system. After the data has been parsed it is much simpler to navigate the huge amounts of logs and find the relevant events that you are looking for. Splunk is able to extract fields from XML and JSON documents without any problems. In case of custom log formats, it is possible to use regular expression in order to extract the data from log events. In case of ELK stack, the parsing takes place in the Logstash pipeline before being sent for storage in Elasticsearch. For this purpose, Logstash comes with multiple plug-ins in order to achieve this. There are possibilities to parse data automatically from events in JSON, XML, CSV or events that are using delimiters. In most cases, the grok filter is used in order to analyze the log event. Grok is a language where you define the patterns for log events and then the patterns are used in order to extract the data from log events according to the pattern. In case of Graylog by default, the messages are sent to the application without being parsed by sidecars. After the message has been received it is possible to extract the data. It is possible to use grok and regular expressions in order to extract data. There are also extractors available for JSON and for cases where the data is in key/value pairs. In general, it is possible to parse the entries with all of the applications but in case of ELK cluster, the parsing is done in a distributed manner before being sent to the central location as opposed to Splunk and Graylog where the parsing happens after being transferred to the application.

Searching the stored logs is possible with all three applications and in case of ELK and Graylog, the options to do so are very similar. Both use Lucene syntax in order to query the underlying Elasticsearch cluster in order to retrieve the data. In case of Graylog, it is possible to use a Kibana application in parallel as the data is stored in a similar manner for both. For the purpose of searching the data Splunk has developed its own language in order to simplify searching the data. It must be said that the Lucene syntax is not that easy and when trying to do searches it is highly probable that a reference sheet is required in order to get the results that you expect. The general rule is that complexity of searching depends a lot on the data being looked at. If the data has been properly analyzed and relevant data has been extracted from log events, it makes searching for relevant data a lot simpler. The Splunk query language is trying to improve the situation and it is a bit simpler in some cases but in general, it is not that much better. Due to this, it is possible to save searches in all of the applications. This allows you to work out the

syntax once for the type of search you want to do and then you can just modify some parameters according to what you are searching for.

Another common use-case is to visualize the data in order to get a better view of what is happening with the system. This can be used in order to identify peak times quickly or to get an overview of the system usage at times. All of the applications allow for the creation of dashboards which usually contain one or more visualizations on them. In most cases, this means that log data is turned into metrics and then displayed as graphs or charts. In case of access logs, the usual suspects are showing the rate of different HTTP response code, average response times to different request and to display possible sources for problems. This allows for users to get a good understand of how the system is currently behaving from once source as the dashboards are usually set up to contain relevant data. Another use of dashboards is for KPI tracking where it is possible to measure the metrics and give an overview of the current values for the KPIs in one screen and possibility to export the view for reporting.

For user access control ideally one would want to use an SSO solution in order to cut down the needed maintenance works. As a minimum, a local user directory would also do, but is not ideal. Unfortunately, none of the applications come with support for SSO solution by default. For ELK reverse proxy solution should be used in front of the application in order to communicate with SSO server and provide the username. In case of Graylog and SSO integration can be achieved with plug-ins and with Splunk it is included in the application. All of the applications can use AD or LDAP as the user directory but in case of the ELK stack that requires the installation of the X-Pack plug-in which requires a license to operate. There is a free alternative available also for Elasticsearch to be used instead of X-Pack. All the applications use role-based approach to access management allowing AD groups to be mapped to roles. Or in case of local user directories assign them to the users directly.

Alerting is a feature available for all applications, but in case of the ELK stack is part of the Watcher plug-in which is included in the licensed X-Pack feature set. The alerting and monitoring systems offer a very similar feature sets which enable users to set up conditions in which the alerts are triggered and what should happen if an alert is created. It is possible to set up simple e-mail notifications or call out webhooks with relevant

data. The implementation of the ELK stack also has the possibility to use machine learning in order to detect anomalies which might give it an edge over the competition.

Archiving old logs is possible with all three applications. All of them allow to store the old logs in a Hadoop cluster and restore them from there as needed. As an extra feature, Splunk also allows compressing of older logs inside the system so no separate application is required in order to preserve the data for longer. With Graylog it is possible to use a feature that comes with the enterprise server to manage the data moving to Hadoop or to set it up directly in Elasticsearch. Only the latter option is available when using the ELK stack. With Elasticsearch it is also possible to store the snapshots on the file system or in a cloud-based storage solution like the Amazon S3.

From a security perspective, no application stands out from the pack. Transportation of the data can be done using encrypted channels with all the solutions and since the hosts where the data should be coming from rules can be set up to only accept data from those sources. User activity can also be monitored with all of the systems through access logs which can be configured. With the ELK stack X-Pack plug-in, it is also possible to get detailed information on what data was queried by what user.

As a conclusion, it must be said that all three chose applications are up to the task. The main difference between them is the amount of work required in order to set up the system and later the amount of maintenance work. From the operations side, Graylog and Splunk have a more centralized approach where most of the functionality can be configured from the application itself and is then propagated down to the collector agents whereas, in case of the ELK stack, agent configuration takes place on the hosts themselves. From the three applications, Splunk seems to be the one offering most features but some of them are not needed for the present requirements. All of those features do come with a price tag that is a bit too hefty so that rules out Splunk for the final solution. Choosing between Graylog and the ELK stack is more difficult but in the end, the Graylog solution was chosen since a lot of the work that would have to be done manually has already been done in case of Graylog. Also with Graylog, it is possible to do everything that the ELK stack offers due to the architecture of the system as the data is stored in the Elasticsearch database for both of them and where needed, Logstash can still be used in order to send the data.

3.3 Adapting the Tools to the Architecture

The system will be set up using Graylog as the primary application, but due to the fact of how Graylog works, it is also possible to use every ELK stack feature with it. This will be important as maintaining the agents that send the log data to the system is responsibility of the operations teams and not all of them want to use the Graylog Sidecar Collector, but are looking to deploy their agents with the application, in order to keep the configuration out of Graylog and in their own deployment tools.

To be able to analyze all of the incoming data effectively, some mandatory fields are required, in order to sort the incoming data and to apply the right rules to the right data. All the incoming data needs to contain the following fields:

- service
- application
- logtype

These 3 fields are used in order to assign the entries into correct index sets in Graylog. This allows to set up viewing permissions so people only see indexes that they are supposed to and also to configure removal or archiving of old data according to preset rules. These fields should be set during the initial data gathering by the agents and if not present the data is discarded. Service and application are separate fields as in some cases the whole system can consist of several applications. From the example applications, the Bitbucket service has several other applications that generate logs in order to function.

The whole system consists of the following components:

- **Elasticsearch.** All the data is stored in an Elasticsearch cluster which consists of several nodes. The cluster itself can be scaled horizontally or vertically as needed by either increasing the resources available or adding additional nodes. Initially, the setup contains seven nodes out of which four are strictly dedicated to storing the data and three are handling the coordination of the cluster, also known as master nodes. The three master nodes that are coordinating the cluster are also the ones used for data input and output. HTTP interface on the data nodes is disabled to make sure they can dedicate all their resources to data indexing and saving the data. The reason for having three coordinating nodes is to ensure that node failures do not cause an outage for the cluster. If only two master nodes were present there could be present you would require both of them to be working in order to have a cluster or there is risk splitting the cluster into two separate instances which would lead to data loss.
- **HAProxy.** Load balancer software that is used in order to split the traffic between available nodes for applications. This also helps with the high availability setup, in case of node failures traffic will be diverted to working nodes by HAProxy. There are two instances of HAProxy configured, a master instance which is used in most cases and a backup instance which takes over if a problem with the master instance is detected. To achieve this, VIP (Virtual Internet Protocol) address is set up that is used as the entrance point for the cluster and the HAProxy servers are listening on that IP for connections. Routing software named keepalived is used by the servers to determine if the master is available or backup server should be used.
- **Redis.** An in-memory data structure store that is used in order to buffer incoming traffic. There are two separate instances of this Redis running in the system and HAProxy is used to balance the load between them. High availability is guaranteed by HAProxy as it only routes the connections to working instances of the software.
- **Graylog.** The main application of the system where most of the work is done. Graylog is responsible for extracting the data from log events, storing them in

the database and searching through the data in order to display results that. Initially, the application is running on three nodes as they are set up on the same servers and the MongoDB databases. The three nodes form a cluster and the load balancer spreads out the connections between the servers.

- **MongoDB.** A database engine that is used for storing the application configuration of Graylog. Not much load is expected for this application, but it does have to be a high availability system as Graylog cannot work without it. To achieve this MongoDB replica set needs to be configured and a minimum of three nodes is required to do so.
- **Graylog Sidecar.** The default agent for sending the data from servers to the Graylog application. These are deployed to all the servers and they connect back to the main application in order to get their configurations based on the tags that have been assigned to them. A deployment of these agents has to be integrated into the creation of new servers for both Windows and Linux systems as they are sending the server logs to the central system. Additional tags will have to be configured based on what applications are running on the server and configurations for those tags have to be created. If the application is running inside a Docker container a separate collector needs to be added to the container or the application inside the container has to log to standard output.
- **Logstash/Fluentd/Filebeat.** Alternative agents that can be used in order to collect the data from the servers and send it to Graylog. These are available for operations teams who don't or can't use the default Graylog Sidecar Collector that is installed on the servers for some reason. For these agents, the configuration needs to be handled by the operations teams and extracting the data can happen already on the servers so the data reaching Graylog is already analyzed.
- **Kibana.** The default application of the ELK stack that is used to view the data inside the Elasticsearch server. This will not be set up in a high availability setup and mostly be used for the administration of the Elasticsearch database. Users are not granted access to this system as they should be viewing the data from the Graylog application.

- Apache Hadoop.** A distributed storage system that is used for archiving purposes. Older logs that need to be preserved for an extended timeframe, but don't need to be available for search are moved from Elasticsearch to the Hadoop cluster that is set up for this purpose.

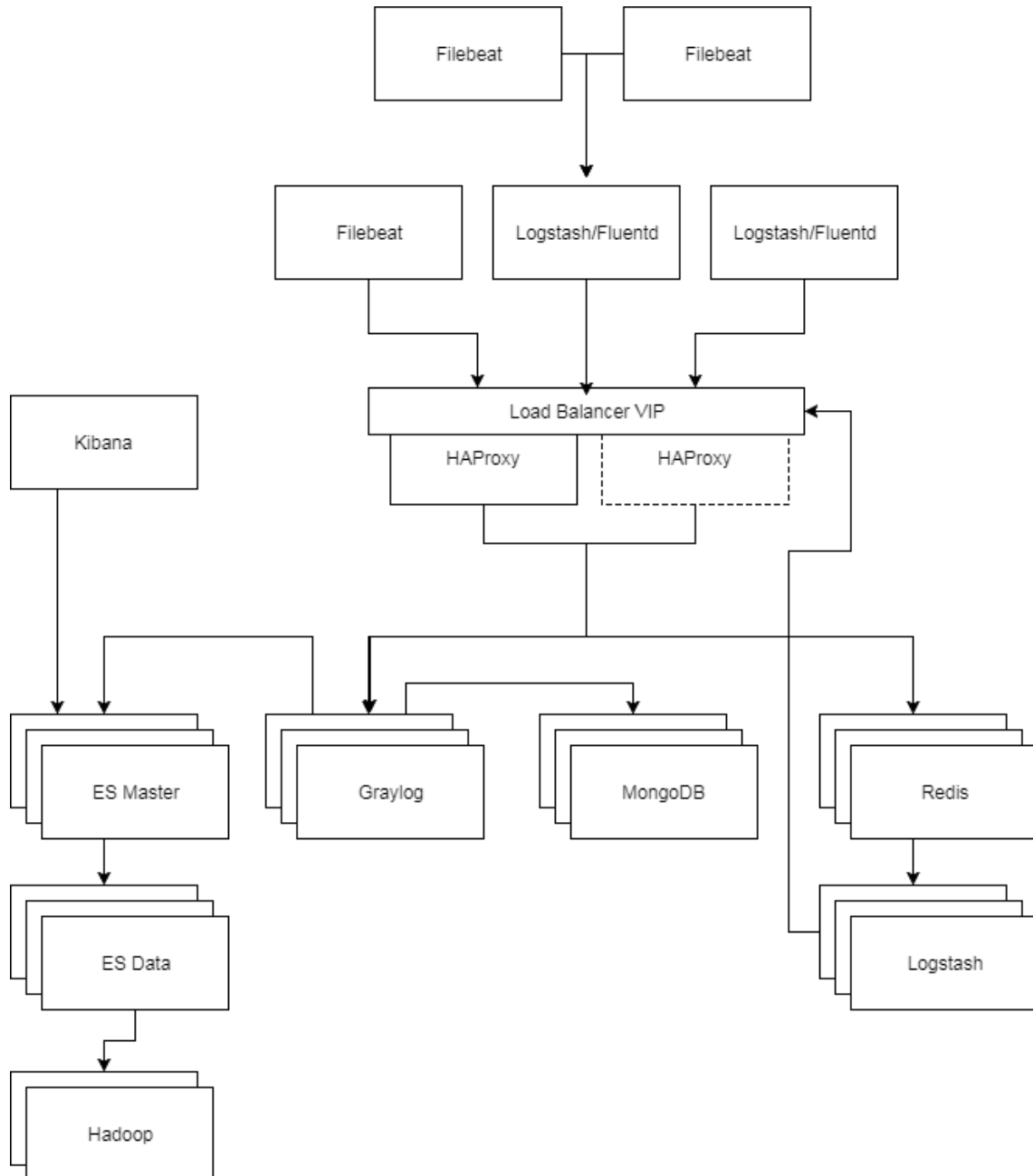


Figure 9 - Component Architecture

The previous figure shows the interactions between the components of the system. The parts before the load balancer show possible setups for the teams that wish to use their own agents instead of the Graylog Sidecar Collector. The Redis and Logstash are also there in order to do buffering for the Logstash instances to avoid data loss. There are

two main ways to collect the data. The first one would be with the Graylog Sidecar Collector, which collects the data and sends it through the load balancer to a Graylog instance for processing. The other option is to use some alternative agent, but in this case, the data is sent through the load balancer to a Redis instance in order to do buffering. After that, there is a separate Logstash instance that pulls from the Redis server and forwards the data to the Graylog instance through the load balancer.

System	Servers
ES Master	3
ES Data	4
Graylog	3
MongoDB	3
Redis	3
Logstash	3
HAProxy	2
Kibana	1

Table 3 - Server Requirements

If all this software was installed on separate servers the number of servers would go quite high so it would be better to consolidate some applications on the same server. Since some systems will be creating very little load it is possible to run several of them on the same server. Since 3 servers have to be used for MongoDB and Graylog to achieve high availability, it would make sense to also add Redis and Logstash instances to the same servers as that would reduce the number of servers from twelve to three for those applications. The Elasticsearch cluster would still require seven servers.

System	Servers
ES Cluster	7
HAProxy	2
Graylog/MongoDB/Redis/Logstash	3
Kibana	1

Table 4 - Consolidate Server Requirements

The system will still be installed on a quite a large number of servers but it is required in order to have a high availability solution. Most of the servers are used by the Elasticsearch cluster, but the amount of expected incoming data is rather large so a cluster with sufficient resources is needed.

4 Solution Verification

In order to make sure that the system works as is expected, a small-scale proof of concept system will be set up logs events will be sent to it. This will allow to verify that it is possible to send logs and analyze them. In the proof of concept the following: functionality will be tested

- Sending logs
- Analyzing logs
- User management
- Viewing log entries
- Sending events directly to the API

The setup of the system is done using Docker [18] as that is the simplest way to have a working cluster with all the required components. Testing alternative agents for log sending is not part of the proof of concept, this makes the setup of the test system much simpler as the only things required are a Graylog server, Elasticsearch server, and a MongoDB server.

Data for the test comes from the Nexus application. This application was chosen as it is simple enough that the setup does not take too long, but there is enough data to test the solution. There are three files that are needed to be forwarded and parsed.

- Nexus access log
- Nexus Java log
- Apache access log

There is an Apache reverse proxy in front of the application so it would be possible to connect to it directly over ports 80 and 433 as by default the application runs on port 8081. The server is running a Red Hat operating system.

Since the system consists of several containers that need to be able to communicate with each other the Docker Compose tool is used to coordinate the deployment and interactions between the containers. This allows for quick deployment of the system and if needed it would allow scaling the system so high availability could be tested.

Before work can start on the application it has to be deployed on a host running Docker, and the Collector has to be installed on the host that is providing the testing data. The Docker Compose file used can be found in Appendix 1 with the configuration of the deployed Collector in Appendix 2.

4.1 Installation

The Log Management system PoC is installed on a freshly installed Red Hat 7 virtual machine. In order to be able to deploy the system, some software has to be installed. Mainly what is needed is Docker itself and the Docker Compose command. Both of these are available in the package repositories provided by Red Hat so the default package management tool *yum* can be used in order to install the required software.

Software	Version
Docker	1.12.6
Docker Compose	1.9.0

Table 5 - PoC Software Versions

Another thing that has to be verified is that the kernel setting *net.ipv4.ip_forward* has been set to 1. This is required in order to forward incoming connections to the virtual machine to the Docker containers running on that machine.

The next step would be to run the `docker-compose up` command in the directory where the `.yaml` file is located. The file describes what containers have to be started and variables that can be configured for those containers. It also defines what containers can connect to each other and what ports on the host machine are mapped to ports inside the container.

After running the command Docker downloads the defined images from Docker Hub [19], which is the official container repository used by Docker. After the containers have been downloaded they are started using the variables that have been provided in the configuration file.

The compose file that was used worked without any problems and after the setup had finished the login screen on port 9000 of the server could be seen.

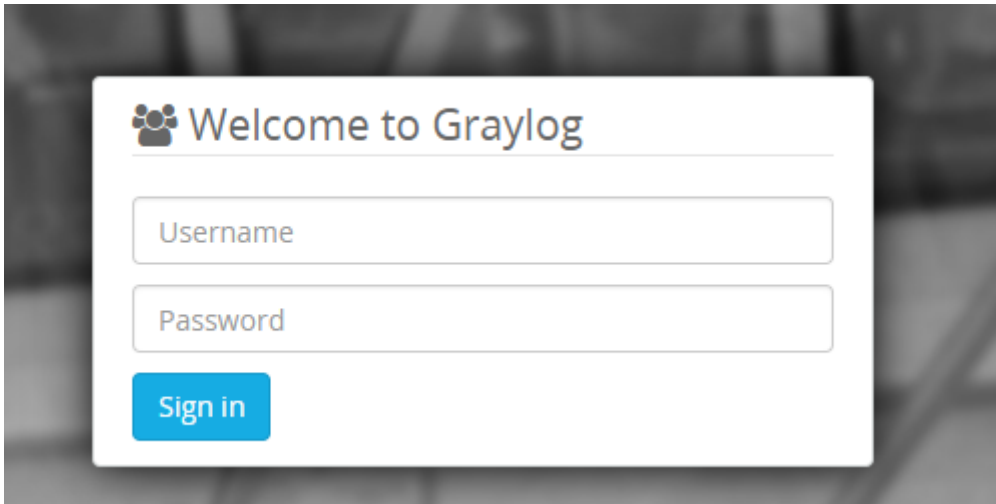



Figure 10 - Graylog Login Screen

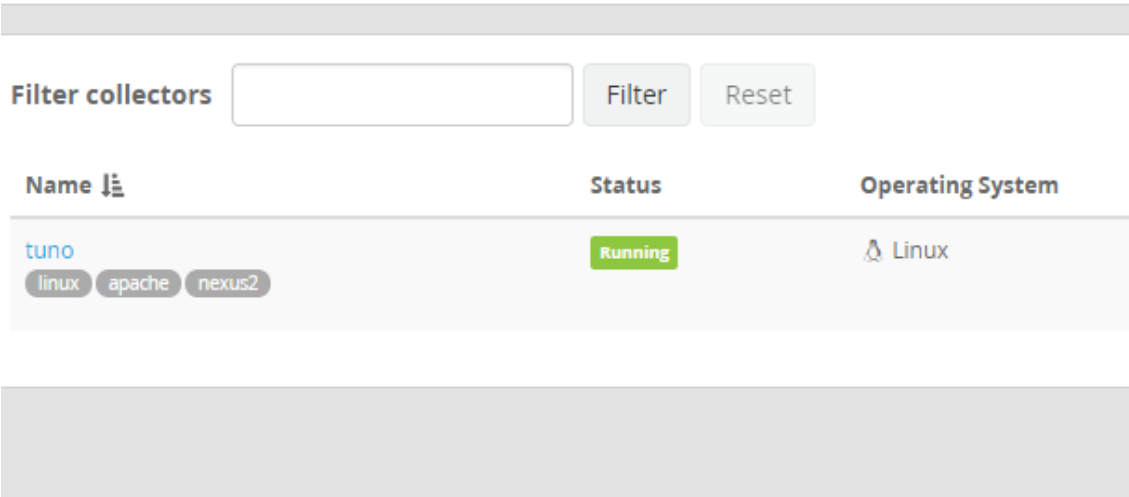
Logging in with the default user and password worked without any problems so it can be concluded that the server setup was successful.

The next step would be to set up a Collector Sidecar on the server that is used as the data source for testing. For the installation of the Collector Sidecar an installer needs to be downloaded from their Github page. Currently, there are three types of installers available. RPM packages for Red Hat based systems, Debian packages for systems based on that Linux distribution and an installer for Windows-based systems. Since the server used is running Red Hat operating system the RPM package has to be downloaded to it. After installing the package a separate command has to be run in order to set it up as a system service in order to start/stop it during boot. Before the Collector Sidecar can be started some modifications have to be made in the configuration also. For the PoC there were two values that were changed from the default, the name of the node and the server URL. Also, the tag nexus2 was added to it, as Graylog uses these tags in order to determine what configuration should be deployed by this Collector Sidecar. After the changes to the configuration file were made and the Collector Sidecar was started it showed up nicely in the Graylog application.

Collectors in Cluster

The Graylog collectors can reliably forward contents of log files or Windows EventLog from your servers.

 Read more about collectors and how to set them up in the [Graylog documentation](#).



Name	Status	Operating System
tuno linux apache nexus2	Running	Linux

Figure 11 - Graylog Collectors View

If more of the Collector Sidecars would be installed on other machines they would show up in this list. Forwarding syslog data to the Graylog host over UDP in rsyslog configuration has been enabled also. Installation of required software for the PoC can be considered complete after this and work with the application itself can begin.

4.2 Receiving Data

All the data that is sent to the Graylog application is stored in index sets. When creating the index set rotation rules are set. This is used in order to remove old log entries from the system. By default it is configured to store a certain amount of messages and if the number gets larger old ones are rotated out. In most cases this is a good rule to use as the data usage by the system is very predictable. There are however some log files that contain audit data that have to keep for a certain time period. In order to fulfill this goal time-based rotation strategy can be used. When using this strategy it is defined, for how many days you want to keep the data searchable. In some cases, the time period can be very long and data should be moved to an Apache Hadoop cluster before it is removed from the Graylog system. Since these rules are set up on an index set basis this means that each type of file that is read has to be stored in a separate index set. For example, for the nexus application, a total of 3 separate index sets are required as rules for each are different. A separate index set was configured for syslog data also.

To decide to which index set data should be stored Graylog uses streams. It checks incoming messages for rules and assigns them to a stream. Each stream is mapped to an index set. Since every incoming event includes the service application and log type fields those are used to assign the data to the correct stream in Graylog.

After these steps are done successfully the configuration on where to store the data can be considered done, next on the agenda should be receiving the data. The first step is to define inputs for the Graylog application. This part defines what ports Graylog is expecting to receive what types of messages. For the purpose of the PoC two inputs were set up. One input was set up to receive messages from the applications and another one to receive the syslog data. Since rsyslog had already been configured on the server to send the data to Graylog, as soon as the input was set up, the system started to receive data from it and store it in the syslog index set. As analyzing of the messages has not yet been set up they are saved as strings in the system, so it's quite hard to search for anything. This is solved a bit later as for now the logs from the Nexus application stored also need to be stored on the server.

The next step in order to get the data from Nexus application would be to configure the Collector Sidecar to start sending the data. The tag nexus2 was applied to the collector

on installation so now a configuration should be created that gets applied to this tag. The Sidecar configurations consist of two parts. The first part is where to send the data, and the second part is what data to read. When configuring files to read it is also possible to add fields to the read entries, this can be used to write the required fields to all the log entries. It is also possible to write separate advanced configurations for the Filebeat application that are used to read the logs and use the Collector Sidecar to distribute it to hosts. Tags can be added to every collector configuration that you create if a tag applied to a collector configuration matches tags present for a Collector Sidecar that host pulls the configuration and applies it. After creating a collector configuration to read the nexus access and java logs with the output defined as the Graylog server, new log events from the server started to appear in the configured index sets.

4.3 Analyzing the Data

Having the raw data from log files can be useful, but to truly benefit from collecting the log events they should also be analyzed in order to simplify searching for relevant data. In the scope of the PoC, the log entries from nexus access logs should be analyzed so visualizations using the data could be created in order to verify that it works as expected. Data coming to Graylog can be analyzed using the extractors. For every input, it is possible to define extractors which will then be run on incoming messages. Unfortunately, it is not possible to define sufficient conditions on when to run the extractor so if logs from many applications were sent to one input and extractors for each log type was defined all of them would be run on every incoming message leading to very high load on the Graylog host. This can be used in for syslog messages as those are coming in on a separate input from applications and all of the machines sending the data are using the same standard so only one extractor is required in order to parse the incoming messages. For the purpose of applications logs, in order to not overload the server, each application should use a separate input in order to only run extractors relevant to that application. That approach would cause a lot configuration in order to set up new applications as changes in the load balancer and Docker port forwarding would be required each time an application was added. Fortunately, it is also possible to use pipelines in order to process the data. The pipeline system is much more flexible then what was possible to do with extractors. A pipeline is a set of rules that will be applied to all messages going through it. Each pipeline is attached to one or more streams. This means that for each log type it is possible to define a unique pipeline that will analyze the messages being stored. It is also possible to reuse pipeline if, for example, you have HAProxy set up for several services, then the HAProxy pipeline can be attached to all the streams containing messages from different services. Another advantage of using pipelines is that it's possible to do more with it then what extractors offer. It is possible to add or remove fields that the message contains or to modify already existing fields or create new fields that have data dependent on values of other existing fields.

Depending on the log format different methods can be used in order to analyze data. In some cases, it is possible that delimiters are used in order to split values in log entries, but usually, the structure of the log entries is more complex and regular expressions are used in order to define the format. Most commonly grok patterns are used which is a

collection of named regular expressions. Graylog application comes with a large number of patterns already implemented by default and it is possible to define custom patterns. In order to create the pattern for the nexus access logs, an online Grok Constructor [20] is used. The following log line was taken as the basis for building the pattern:

```
10.61.45.245 - lauri.linros [19/Oct/2017:10:07:28 +0000]
"GET
/nexus/content/groups/public/org/codehaus/plexus/plexus-
archiver/2.9/plexus-archiver-2.9.jar HTTP/1.1" 200 145376 6
```

Using the tool a grok pattern was created and some type definitions were added which is a Graylog feature.

```
%{IPORHOST:clientip} - %{USERNAME:user}
\[%{HTTPDATE:timestamp}\] "%{WORD:verb} %{NOTSPACE:request}
HTTP/%{NUMBER:httpversion}" %{INT:response;int}
%{INT:bytesSent;int} %{INT:ms;int}
```

This pattern was then added to the Graylog application with the name NEXUS_ACCESS_LOG that it can be used with the pipeline attached to the stream containing the nexus access logs. The following rule was created for the pipeline:

```
rule "Extract nexus access log fields"
when
  true
then
  let m = grok("%{NEXUS_ACCESS_LOG}",
to_string($message.message), true);
  set_fields(m);
end
```

As the result of this, all log messages in the stream now have been parsed into more fields that can be used in order to do searches on the data to find out trends or to create a visualization in order to have an overview of how the system is behaving and what it is begin used for. The value of the when directive for the pipeline rule is set to true since this rule has to be run on all the entries, It is also possible to set up rules that are only run of some fields that contain certain values.

4.4 Data Presentation

Searching the data is quite simple in Graylog, one has to open the stream they are interested in and then run a query over a selected timeframe and Graylog outputs the records that match the query. While the syntax of the search engine might not be the easiest to use, it is quite powerful and complex queries can be written with it. Graylog documentation on the matter is quite extensive and should be consulted when creating more complex queries.

The system allows for the creation of visualizations and then to group them together in dashboards. The process of creating these visualizations is quite simple. When looking for data it is possible to quickly create visualizations from the search interface and add them to existing or new dashboards. After all the desired visualizations have been added to the dashboard it is possible to position them and do some minor modifications to them.

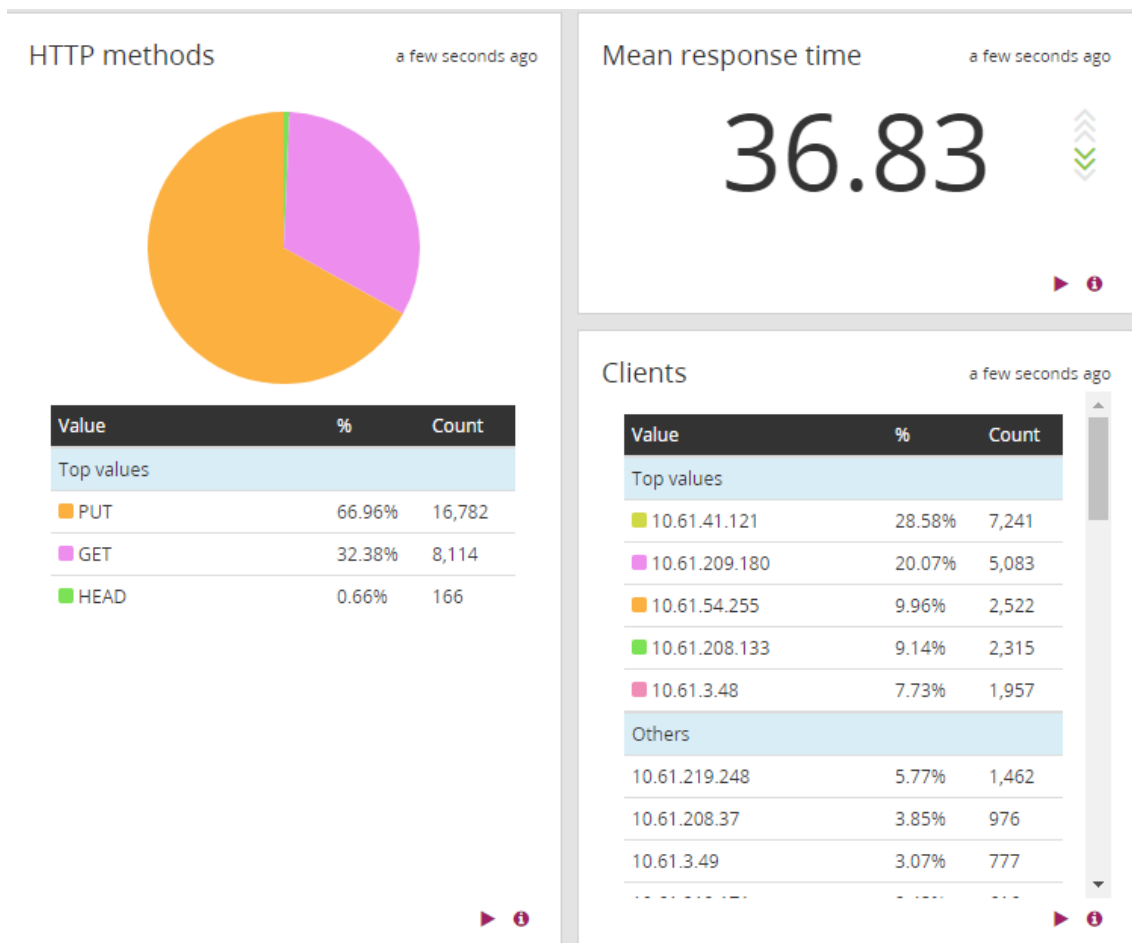


Figure 12 - Graylog Dashboard

4.5 User Management and Alerting

To manage users the goal is to do everything from Active Directory. Ideally, roles are mapped to AD groups and whoever is part of that AD group will be able to log into the Graylog system and have the appropriate permissions related to that group. The idea behind it is that the AD groups would be owned by the teams themselves so they can manage their user permissions without having to request it from someone else making it more convenient for everyone. It was possible to set up the system in this way in Graylog configuration for LDAP/AD. How it works is that there is a main access group in the AD. All the groups that are members of that group get imported into Graylog with their users and each of those groups is mapped to a role in Graylog. The roles permissions in Graylog are quite simple it is possible to set read/edit permissions on streams and dashboards granting appropriate access to it. The permissions part also affects alerting. The way alerts work is, that some conditions are defined on a stream basis and if the rule is broken an alert is raised. The way this works with permissions is that users who have permissions to read a stream also can see alerts from that stream. As many conditions can be set up as you want. Alerts remain visible until the triggering condition is no longer valid. It is also possible to set up notifications for each stream. By default, it is possible to send an e-mail or to call out an URL with the HTTP POST method. The notifications are configured on a stream level, so each of the conditions configured for the stream triggers all notifications configured for that stream. This is not ideal when trying to set up an automatic response for triggers as an extra layer is needed to be created that receives the HTTP request and decides what needs to be executed in order to fix it. It also means that it's not possible to filter what alerts to send to what e-mails on a condition basis for a stream.

5 Summary

The problem that was tackled was to set up a log management system for a large organization. There are many benefits for doing this and in some cases, it might even be required by some compliance standards that the company must adhere to. In the larger scope, this activity is part of security information and event management.

In order to get a good overview of the work ahead of us, documentation provided by NIST Special Publication 800-92 [1] written by Institute of Standards and Technology was consulted. This document gives a very good overview of the benefits of setting up log management but the more important part is the description of how such a system should work. This document was relied on in order to create the requirements that the final solution would have to adhere to.

The main requirements were that the system should be able to handle several different data sources as the data could come from files, posted over HTTP or sent by syslog. There were also requirements dealing with access to the data and the rules for retention of it. Audit and security data can be more important than an applications operational data and it is possible that it needs to be stored for a long period of time. This means some sort of conditional archiving of the data has to be set up. Among the list of requirements was also the ability to alert and notify in case of some conditions are broken in the data. This has two main purposes. First one being able to detect security issues and alert the proper people to act on it and secondly, it can be used to monitor application behavior and if it is not without predefined conditions alert the operations team in order to pre-empt a possible issue or to quickly respond to an incident. There were also several non-functional requirements that dealt with data security and the availability of the system.

With the requirements created it was possible to create an initial design so that the necessary components of the system could be identified. With that knowledge, the search for tools that could be used was started and three possible candidates were chosen which could be used to set up the log management system.

The three tools chosen were Splunk, Graylog and ELK Stack. All of them were analyzed in order to be able to tell how well they would perform against the

requirements. In the end, the conclusion was that it would be possible to set up the system using any of them but the difficulty of doing so and maintaining the system would be the deciding factor on which to use. Out of the three options in consideration, Splunk was the one with the best feature set and would have been the one to use if it didn't come with a very hefty price tag attached to it. The other two options are both software that can be used without any cost. From the remaining two options Graylog is similar to Splunk where most of the configuration is done in a web interface and kept in a central location. For the ELK stack web interface only exists for viewing the data in form of Kibana, but the whole configuration part for receiving the logs is spread on all the hosts that are sending and storing the data. In the end the decision was made to use Graylog as that system automated a lot of the work that would have to be set up manually for the ELK stack. Maintenance of the Geaylog system is also a lot simpler as system administrators won't have to search servers for the correct configuration files but can do so straight from the interface.

When the tools had been chosen it was possible to create a final design for the log management system that would successfully fulfill all of the requirements that were set forth beforehand. Due to some of the requirements, mostly the volume of the data and high availability, the final design used quite a lot of servers but in order to get the number down some of the applications that were not creating a lot of system load, were consolidated to using the same servers. Depending on the deployment method used the actual server number might even be lower, but it will not be able to be less than 3 as that is the minimum that is required for high availability.

Having completed the final design it was time to create a proof of concept(PoC) setup in order to verify that the system works as expected. To simplify the setup for the PoC some of the functional requirements were ignored so that everything could be set up on just one server. During the PoC all the requirements were verified and quite a few details were clarified on how things should be set up inside the application in order to have the system working as expected.

With the successful completion of the PoC, it could be concluded that the design is indeed valid and could be implemented for production use.

References

- [1] N. I. o. S. a. Technology, “Special Publication 800-92,” [Online]. Available: <http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf>.
- [2] Sonatype, “Sonatype Nexus OSS,” [Online]. Available: <https://www.sonatype.com/nexus-repository-oss>.
- [3] Express, “ExpressJS,” [Online]. Available: <https://expressjs.com/>.
- [4] Morgan, “Morgan,” [Online]. Available: <https://github.com/expressjs/morgan>.
- [5] Atlassian, “Bitbucket Data Center,” [Online]. Available: <https://confluence.atlassian.com/enterprise/bitbucket-data-center-668468332.html>.
- [6] Atlassian, “Bitbucket Server Log Formats,” [Online]. Available: <https://confluence.atlassian.com/bitbucketserverkb/how-to-read-the-bitbucket-server-log-formats-779171668.html>.
- [7] Elasticsearch, “Elasticsearch Logging Configuration,” [Online]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html>.
- [8] HAProxy, “HAProxy Logging Configuration,” [Online]. Available: <https://cbonte.github.io/haproxy-dconv/1.8/configuration.html#8>.
- [9] Splunk, “Splunk,” [Online]. Available: <https://www.splunk.com/>.
- [10] Graylog, “Graylog,” [Online]. Available: <https://www.graylog.org>.
- [11] ElasticSearch, “ElasticSearch,” [Online]. Available: <https://www.elastic.co/>.
- [12] Splunk, “Solunk version comparison,” [Online]. Available: https://www.splunk.com/en_us/products/features-comparison-chart.html.
- [13] Splunk, “High availability deployment,” [Online]. Available: <http://docs.splunk.com/Documentation/Splunk/6.6.3/Deploy/Indexercluster>.
- [14] Splunk, “Pricing,” [Online]. Available: https://www.splunk.com/en_us/products/pricing.html#tabs/tab1.
- [15] Elasticsearch, “Deploying and Scaling,” [Online]. Available: <https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html#integrating-with-messaging-queues>.
- [16] Graylog, “Graylog architecture,” [Online]. Available: <http://docs.graylog.org/en/2.3/pages/architecture.html>.
- [17] Graylog, “Graylog Collector Sidecar,” [Online]. Available: http://docs.graylog.org/en/2.3/pages/collector_sidecar.html#.
- [18] Docker, “Docker,” [Online]. Available: <https://www.docker.com/>.
- [19] Docker, “Docker Hub,” [Online]. Available: <https://hub.docker.com/>.
- [20] H.-P. Störr, “Grok Constructor,” [Online]. Available: <http://grokconstructor.appspot.com>.

Appendix 1 – docker-compose.yml for the PoC

```
version: '2'
services:

  # MongoDB: https://hub.docker.com/_/mongo/
  mongodb:
    image: mongo:3
  #
  # Elasticsearch:
  # https://www.elastic.co/guide/en/elasticsearch/reference/5.6
  # /docker.html
  elasticsearch:
    image:
    docker.elastic.co/elasticsearch/elasticsearch:5.6.3
    environment:
      - http.host=0.0.0.0
      - transport.host=localhost
      - network.host=0.0.0.0
      # Disable X-Pack security:
      # https://www.elastic.co/guide/en/elasticsearch/reference/5.5
      # /security-settings.html#general-security-settings
      - xpack.security.enabled=false
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
      mem_limit: 1g
  # Graylog: https://hub.docker.com/r/graylog/graylog/
  graylog:
    image: graylog/graylog:latest
    environment:
      # CHANGE ME!
      - GRAYLOG_PASSWORD_SECRET=d9oRehQ8mIBHChZN
      # Password: admin
      -
      GRAYLOG_ROOT_PASSWORD_SHA2=8c6976e5b5410415bde908bd4dee15df
      b167a9c873fc4bb8a81f6f2ab448a918
      -
      GRAYLOG_WEB_ENDPOINT_URI=http://10.61.123.34:9000/api
    links:
      - mongodb:mongo
      - elasticsearch
    depends_on:
      - mongodb
      - elasticsearch
    ports:
      # Graylog web interface and REST API
      - 9000:9000
      # Syslog TCP
```

```
- 514:514
# Syslog UDP
- 514:514/udp
# GELF TCP
- 12201:12201
# GELF UDP
- 12201:12201/udp
# Beats input
- 5044:5044
# Gelf HTTP
- 8081:8081
```


Appendix 2 – collector_sidecar.yml for PoC

```
server_url: http://10.61.123.34:9000/api/
update_interval: 10
tls_skip_verify: false
send_status: true
list_log_files:
node_id: tuno
collector_id: file:/etc/graylog/collector-
sidecar/collector-id
cache_path: /var/cache/graylog/collector-sidecar
log_path: /var/log/graylog/collector-sidecar
log_rotation_time: 86400
log_max_age: 604800
tags:
  - linux
  - apache
  - nexus2
backends:
  - name: nxlog
    enabled: false
    binary_path: /usr/bin/nxlog
    configuration_path: /etc/graylog/collector-
sidecar/generated/nxlog.conf
  - name: filebeat
    enabled: true
    binary_path: /usr/bin/filebeat
    configuration_path: /etc/graylog/collector-
sidecar/generated/filebeat.yml
```

Appendix 3 – Parsed Nexus Access Log Entry

application	nexus2
bytesSent	230883
clientip	10.61.209.202
collector_node_id	tuno
facility	filebeat
file	/data/maven/nexus/sonatype-work/nexus/logs/request.log
httpversion	1.1
input_type	log
logtype	nexus_access
message	10.61.209.202 - - [19/Oct/2017:13:27:33 +0000] "GET /nexus/content/groups/public/org/codehaus/plexus/p230883 5
ms	5
name	tuno.int.kn
offset	238323074
request	/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/3.0.10/plexus-utils-3.0.10.jar
response	200
service	nexus2
source	tuno.int.kn
tags	["linux", "apache", "nexus2"]
timestamp	2017-10-19T13:27:38.905Z
type	log
user	-
verb	GET