DM

Development of a Centralized Log Management System

MASTER DISSERTATION

Joaquim Tomás Almada Abreu
MASTER IN COMPUTER ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

August | 2020

Development of a Centralized Log Management System

MASTER DISSERTATION

Joaquim Tomás Almada Abreu

MASTER IN COMPUTER ENGINEERING

ORIENTATION

Filipe Magno de Gouveia Quintal

CO-ORIENTATION

Sérgio Miguel Rodrigues Viúla

Acknowledgements

This section is dedicated to thank everyone that, directly or indirectly helped and accompanied me in my academic course.

First of all, I would like to thank my family and friends for all the support and encouragement they have given me throughout the development of this dissertation.

A special feeling of gratitude to my parents who have always been understanding on the moments that I could not be present.

I am extremely thankful to my supervisor, Prof. Dr. Filipe Quintal for all the support, guidance, suggestions and constant encouragement throughout my dissertation work. I also thank him for his continuous availability, patience and concern.

I am extremely grateful for ams Sensors Portugal, Lda for creating excellent working conditions. I would also thank them for the opportunity given to develop this project.

A special thanks to my company's supervisor, Engineer Sérgio Viúla, for all the guidance and technical support that was given since day one. I would also like to thank Funchal's IT team and its managers for having received me and facilitated the integration process.

Resumo

Os registos de um sistema são uma peça crucial de qualquer sistema e fornecem uma visão útil daquilo que este está fazendo e do que acontenceu em caso de falha. Qualquer processo executado num sistema gera registos em algum formato. Normalmente, estes registos ficam armazenados em memória local. À medida que os sistemas evoluiram, o número de registos a analisar também aumentou, e, como consequência desta evolução, surgiu a necessidade de produzir um formato de registos uniforme, minimizando assim dependências e facilitando o processo de análise.

A ams é uma empresa que desenvolve e cria soluções no mercado dos sensores. Com vinte e dois centros de design e três locais de fabrico, a empresa fornece os seus serviços a mais de oito mil clientes em todo o mundo. Um centro de design está localizado no Funchal, no qual está incluida uma equipa de engenheiros de aplicação que planeiam e desenvolvem applicações de software para clientes internos. O processo de desenvolvimento destes engenheiros envolve várias aplicações e programas, cada um com o seu próprio sistema de registos.

Os registos gerados por cada aplicação são mantido em sistemas de armazenamento distintos. Se um desenvolvedor ou administrador quiser solucionar um problema que abrange várias aplicações, será necessário percorrer as várias localizações onde os registos estão armazenados, colecionando-os e correlacionando-os de forma a melhor entender o problema. Este processo é cansativo e, se o ambiente for dimensionado automaticamente, a solução de problemas semelhantes torna-se inconcebível.

Este projeto teve como principal objetivo resolver estes problemas, criando assim um Sistema de Gestão de Registos Centralizado capaz de lidar com registos de várias fontes, como também fornecer serviços que irão ajudar os desenvolvedores e administradores a melhor entender os diferentes ambientes afetados.

A solução final foi desenvolvida utilizando um conjunto de diferentes tecnologias de código aberto, tais como a Elastic Stack (Elasticsearch, Logstash e Kibana), Node.js, GraphQL e Cassandra.

O presente documento descreve o processo e as decisões tomadas para chegar à solução apresentada.

Palavras-Chave:Registo, Gestão de Registos, Sistema Centralizado, API, Elasticsearch, Node.js

Abstract

Logs are a crucial piece of any system and give a helpful insight into what it is doing as well as what happened in case of failure. Every process running on a system generates logs in some format. Generally, these logs are written to local storage resources. As systems evolved, the number of logs to analyze increased, and, as a consequence of this progress, there was the need of having a standardized log format, minimizing dependencies and making the analysis process easier.

ams is a company that develops and creates sensor solutions. With twenty-two design centers and three manufacturing locations, the company serves to over eight thousand clients worldwide. One design center is located in Funchal, which includes a team of application engineers that design and develop software applications to clients inside the company. The application engineer's development process is comprised of several applications and programs, each having its own logging system.

Log entries generated by different applications are kept in separate storage systems. If a developer or administrator wants to troubleshoot an issue that includes several applications, he/she would have to go to different database systems or locations to collect the logs and correlate them across the several requests. This is a tiresome process and if the environment is auto-scaled, then troubleshooting an issue is inconceivable.

This project aimed to solve these problems by creating a Centralized Log Management System that was capable of handling logs from a variety of sources, as well as to provide services that will help developers and administrators better understand the different affected environments.

The deployed solution was developed using a set of different open-source technologies, such as the Elastic Stack (Elasticsearch, Logstash and Kibana), Node.js, GraphQL and Cassandra.

The present document describes the process and decisions taken to achieve the solution.

Keywords: Log, Log Management, Centralized System, API, Elasticsearch, Node.js

Table of contents

Table of figures

Table of tables

Glossary and important terms

ams AG: Austria Mikro Systeme *Aktiengesellschaft* (anonymous society)), also known as ams;

API: An Application Programming interface is a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service;

CLS: Centralized logging system;

CQL: Cassandra Query Language;

DBMS: A Database Management System is system software for creating and managing databases, providing users and programmers with a systematic way to create, retrieve, update and manage data;**GitHub**: GitHub is a Git repository hosting service which provides a web based graphical interface, access control and collaboration features for software development projects;

GUI: Graphical User Interface;**JSON**: JavaScript Object Notation, a lightweight format for storing and transporting data, primarily used to transmit data between a server and web application;

LaaS: Logging as a Service;

Log Rotation: The process of compressing, moving or deleting no longer needed/dated logs. Log rotation is typically performed according to a schedule (e.g., hourly, daily, weekly) [1], [2].

NPM: Node Package Manager;**OAS**: OpenAPI Specification;

OKD: Origin Community Distribution;

PaaS: Platform as a Service;

PL/SQL: Procedural Language for SQL is the Oracle Corporation's procedural extension for SQL and the Oracle relational Database;

URL: Uniform Resource Locator;

1 Introduction

In this chapter, we present the motivation and major challenges of this thesis. This work was mainly motivated by the need of having a central logging framework sufficiently capable of handling logs generated by various applications in ams, the company that proposed the work. Keeping all the log data in a central location makes it easy to extract and query later when this need arises.

Logs are a crucial piece of any system and give a helpful insight into what a system is doing as well as what happened in case of failure. Every process running on a system generates logs in some format. Generally, these logs are written to storage resources on local disks.

Before the existence of computers and operating systems, the act of keeping a record of an event was performed by taking a picture with a camera, writing in a sheet of paper or making use of punched cards to document what happened [3]. The first mainframe computers had output lights on the control panel, which served as a hardware based system for logging and monitoring. As operating systems started appearing, so did the recording of operations of such systems. The output of these systems consisted of little more than simple logs and core dumps, triggered in the event of a crash. This was mostly due that, at the time, most systems were batch-oriented[1] and had little real-time input or output. These systems were also under use by trained technicians who knew how to interpret and handle the output[5]. Unix[2], the operating system that had its origin with Bell Labs in 1969, generated more detailed logs than its predecessor systems, but it did not have user friendly Interfaces nor did it come with tools for aggregating log files from multiple sources or automatically converting them to a single format. Instead, administrators had to use basic text manipulation and search tools to make sense of log files, due to the primitive nature of software at the time [8].

As systems evolved, the number of logs to analyze increased, and, as a consequence of this progress, there was the need of having a standardized log format, minimizing dependencies and making the analysis process easier[9]. Operating systems kept logs in separate locations and applications also often kept their own logs, not necessarily in a centralized location [8].

[1] A batch oriented system is one that takes a batch of commands or jobs, executes them and returns the results, rather than having a user executing one a time [4].
[2] Unix is a stable, multi-user, multi-tasking system for servers, desktops and laptops [6]. It was developed as a self-contained software system, comprising the operating system, development environment, utilities, documentation, and modifiable source code [7].

When a system grows to multiple hosts, the access and management of such logs can get complicated. Searching for a specific error across multiple log files on several servers is difficult without the right tools. A common approach to this problem is to mount a centralized logging solution so that multiple logs can be put together in the same location. In the early 2000s, proprietary log management and analysis tools were beginning to appear in the market[8]. These tools collect log data from multiple locations, as well as multiple log types, allowing administrators and developers to view and study the log data in a single location.

The following sections of the chapter aim to further describe the thesis motivation in detail. Section 1.1 aims to present the concept of storing the occurrence of events and how it evolved through time. Section 1.2 introduces ams, the company that proposed such work. Finally, section 1.3 introduces the main challenges that must be solved.

1.1 Logging and tracing

In computing, a log is a record of the events that occurred within an organization's systems and/or networks. Logs are composed of log entries, and each entry contains information related to a specific event that occurred within the system or network. Logging is the act of obtaining, transferring, storing and parsing information about a certain resource, state, activity, object or action [10], [11]. Log data is usually consumed by developers and system administrators, for analysis purposes and or during maintenance, monitoring or to solve a particular problem.

Logs are not exclusive to computer science, in fact the term log appeared from the field of behaviorism, which is the observation and study of actions, thoughts and feelings [3]. The idea of logging comes from a more open view of behaviorism, where it is emphasized the observed behaviors[3] while also considering the aspects that accompany such behaviors. Initially, the observation of behaviors was assisted with some kind of recording device, such as a camera or a pen and paper. However, the concept of observation has been extended to include other recording devices, notably logging software. Originally, logs were used primarily for troubleshooting problems, but now serve many functions, such as optimizing system and network performance, recording the action of users and supply data which is useful for the examination of malicious activity.

Logs have evolved to contain information related to many different types of events occurring within networks and systems [2]. One of the most used standards of

[3] Behavior: In the context of behaviorism, a behaviour is an observable activity of a person, animal, team, organization, or system. It is something that can be detected and, therefore, recorded.

sending and receiving logs from several network devices is the Syslog [12]. This standard requires the presence of a timestamp, device-id, facility code, severity level, message number and message text fields in the message, as can be seen in Figure 1.

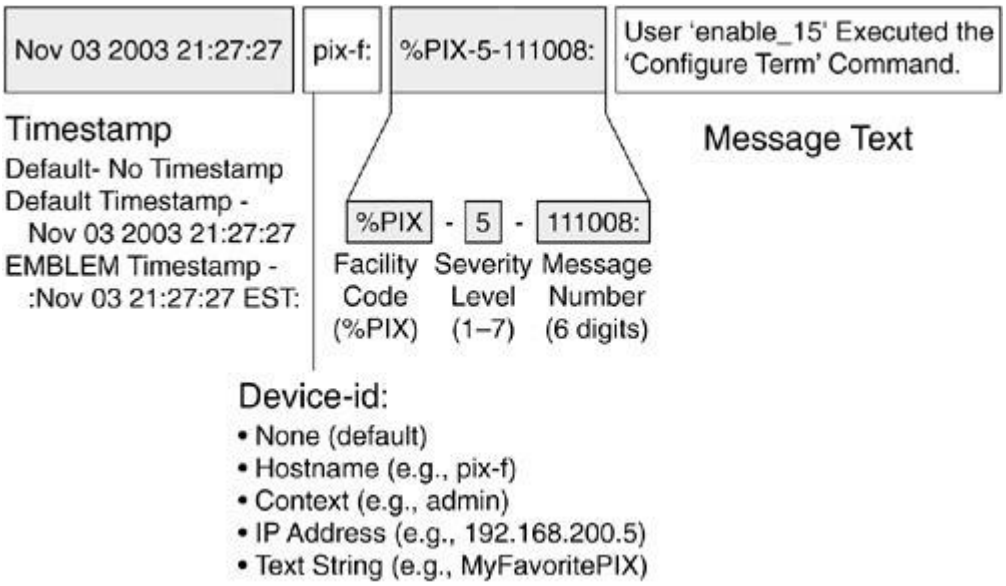


Figure 1. A standardized Syslog message [12].

In normal situations, only logs that can be acted upon should be stored, in other words, logs that report errors and that represent state changes should be kept [10]. This means that the system administrators will see logs with high level information and very little noise[4].

Logs can be related with each other to provide extra information about an issue, allowing managers and developers to analyze a program's flow in a more detailed manner. This particular use of logging is designated as tracing.

Tracing is a specified use of logging to record information about a program's execution. A trace is the complete processing of a request. It represents the path taken through an entire stack of an application or system[10], storing information that is more detailed than the one generally seen in common logging.

Each trace is comprised of a number of spans. A span, occasionally mentioned as segment, is an activity or operation that takes place within individual services or components of the application or system. Each span is a step in the total processing of a request [13].

[4] Having "noisy" logs means that the system contains many duplicate events or information that is not helpful to the intended audience

```
{ SyntaxError: Unexpected token : in JSON at position 0
    at JSON.parse (<anonymous>)
    at createStrictSyntaxError
(C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\body-
parser\lib\types\json.js:158:10)
    at parse
(C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\body-
parser\lib\types\json.js:83:15)
    at C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\body-
parser\lib\read.js:121:18
    at invokeCallback
(C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\raw-
body\index.js:224:16)
    at done
(C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\raw-
body\index.js:213:7)
    at IncomingMessage.onEnd
(C:\Users\jald\Dropbox\Tese\code\nodejs\log_rest_api\node_modules\raw-
body\index.js:273:7)
    at IncomingMessage.emit (events.js:189:13)
    at endReadableNT (_stream_readable.js:1103:12)
    at process._tickCallback (internal/process/next_tick.js:63:19)
  expose: true,
  statusCode: 400,
  status: 400,
  body: ':"{logsSearch(searchInput: {source_system: \\"ORACLE\\",
application_code: \\"MZE\\", log_guid: \\"69043178\\",created_at_gte:\\"2019-03-
30\\"}) {log_guid}}"}',
  type: 'entity.parse.failed' }
```

Figure 2. Trace of a node.js exception

Tracing is usually used for debugging purposes and helps to follow a program's flow and data[14]. It also helps to find out where the performance bottlenecks are, and, if an error occurs, assessing each span within a trace gives the chance to detect the source of a problem. A Node.js error trace log is a good example of the path taken by a program up to the point of an exception, as can be seen in Figure 2.

The process of logging is of great significance to understand what happens in a system, and when such system spans to several machines and applications, it is important to gather these logs in a centralized location for posterior analysis. This is a challenge that is addressed in chapter 1.3.1.

1.2 ams introduction

ams is a company that develops and creates sensor solutions used in applications that require a compact format, low energy, high intensity and sensor integration models. The company offers sensors (including optic sensors) and also software interfaces for consumer, industrial, automotive, medical and communications markets. It has over

thirty five years of experience in the design and industrial fields, where it employs over eleven thousand professionals worldwide.

With twenty-two design centers and three manufacturing locations, ams serves to over eight thousand clients worldwide. One design center is located in Funchal, which includes a team of application Engineers that design and develop software applications to clients inside the company [15], [16]. The application Engineers development process is comprised of several applications and programs, each having its own logging system. There are inherent problems with the manner the log system is structured, and these will be presented in the section below.

Figure 3. ams Logotype

1.3 Problem statement

As presented in section 1, handling logs in separate locations can be an arduous job due to the lack of a general data structure and the difficulty of knowing where and how to search for relevant logs. These tasks become more urgent if services that generate considerate amounts of money are down, such incidents should be taken care of as quickly as possible. The problems described in each subsection of this section are the main challenges encountered when faced with handling a logging infrastructure. Sections 1.3.6 and 1.3.7 describe use case specific business requirements and restrictions, respectively.

1.3.1 Distributed logs

Log entries generated by different applications are kept in separate storage systems. If a developer or administrator wants to troubleshoot an issue that includes several applications, he/she would have to go to different database systems or locations to collect the logs and correlate them across the several requests. This is a tiresome process and if the environment is auto-scaled, then troubleshooting an issue is inconceivable [17].

1.3.2 Storage

Log data needs a destination. The storage system needs to be able to handle the growth in data over time. Each day will add a certain amount of data that is related to the number of systems and processes that are generating log data.

The way to store this kind of data depends on how long it should be stored. Logs can be stored for long-term, require no immediate analysis, or needed for a time

span of months, days or even hours for real time querying. Depending on how long data is stored, a suitable storage system needs to be employed to accommodate the needs of its users and allow for it to scale out horizontally if the data volume becomes too large.

Some storage systems are not appropriate for real-time analysis. Data archiving and long-term backup systems, such as Amazon S3 Glacier5, can take hours to load a file. These are not adequate for production troubleshooting [19]. Similarly, SQL based storage solutions, such as MySQL, can become slow with large tables, degrading the performance and reliability of the system [20].

1.3.3 Querying and Analysis

Logs generated and stored on a centralized storage system will not help much in resolving issues, if there is no fast and efficient way to analyze and make use of them. There could be the need to supply this data to other developers or external sources. Particularly, in ams, the company expects some kind of API to be able to fetch data as it is needed. For this, the solution to implement should allow data filtering and be capable of providing such data in a simple and effective manner.

Having a front-end platform (GUI) also facilitates the analysis process, allowing querying and inspection of the data with the help of tables and dashboards. The use of such platforms allows for a more near real-time and interactive access to the data, but is not the best suited for mass batch processing. In these situations the solution to implement should only present relevant information back to the developers or administrators [19]. Choosing and configuring a tool that is capable of performing log analysis and can centrally store logs is a complex decision, even more so if the tool must fulfill specific user and infrastructure needs [21].

1.3.4 Alerting

Having someone constantly watching the log data for specific events is a tiresome and repetitive process. This routine task steals important time from the developers or administrators, who could be spending it on the development of new features. The last thing a developer wants is to find out about a critical problem from a client or a superior. An important part of any log management system is alerting. Rather than repeatedly searching for events, it would be much more effective if the system could be able to monitor them and notify or email interested parties when such events occur [19], [22], [23].

[5] Amazon S3 Glacier: Long-term object storage for data archiving[18]

1.3.5 Price

Before deploying a centralized log management solution, those responsible for the system must decide if they will use a proprietary tool or an in-house solution. Either choice will have an associated cost, and the cost of proprietary log management solutions is not cheap (for example, sumo logic, a company that provides log management solutions, has an enterprise plan that charges an annual price of 19,800$ to users that ingest 10GB of logs per day[24]). When using a proprietary tool, most maintenance and configuration work will already be implemented, but this choice makes the system dependent of the proprietary tool provider.

If it is decided to implement a centralized logging solution from the ground up, the cost of this solution will depend on the labor needed to implement and maintain it, such as the provisioning of the infrastructure, ensuring high availability, installing, configuring, tuning the tools, maintaining, upgrading and building features specific for the log management system. When all this work is added up, the time and price needed becomes significant [25].

1.3.6 Case study specific business requirements

ams also has business requirements to follow, and several of these requirements are motivated by the problems referred in the previous sections.

Initially, the requirements were obtained from an interpretation of the project proposal documents. At a later phase of the project and after discussions with ams' supervisors, additional requirements were included. This addition allowed us to better understand the initial general requirements and to add extra functionalities to the final product. The initial requirements had a focus on database performance and the need of a data interaction interface through REST, but as the project evolved, the need to add visualization interfaces, alerting and log rotation (see definition in Glossary and important terms) functionalities emerged.

Bellow we present the business requirements, and, while these are still general and have some room left for a more flexible implementation, they are certainly more concrete than the initial business requirements. In this elicitation phase, besides clarifying and adding new requirements, the original ones were rewritten according to the good practices proposed by Mavin et al.[26]:

- BR1: Database with performance for writings;
    - BR1.1: the system shall be able to handle 100 log inputs per second without losing any log;
    - BR1.2: the system shall implement a database with better write performance than the currently used by ams (Oracle);

- BR2: Integration with a performant database for reads;
    - BR2.1: the system must provide a better read performance than the currently used by ams' Oracle system;
- BR3: Validate the log inputs, checking if values are acceptable according to the defined schema;
    - BR3.1: the system must specify a schema to be used by all log inputs;
    - BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema;
- BR4: Implement an interface (REST API) that allows client applications delivering log files to also selectively retrieve log information for usage within an application;
    - BR4.1:the system must provide means to modify, visualize and access log data;
    - BR4.2 the system must provide an API to insert and consume data;
    - BR4.3: the system must provide a visualization platform for the log data;
- BR5: Provide documentation on how to consume data from the log interface;
    - BR5.1: the system must provide developers with documentation for the available API interfaces;
- BR6: Implement a log rotation job to address the requirements of each application regarding no longer needed/dated logs;
    - BR6.1: the system must comply with ams' defined log persistence schedule;
- BR7: Implement a basic and extendable alerting mechanism;
    - BR7.1: the system must alert each application stakeholder by email;
    - BR7.2 the system must update the alerting method according to ams' defined heuristics;
- BR8: Define a security concept for secure communication with the logging server, should the service be accessed from outside the intranet;

1.3.7 Restrictions

As the project evolved, some development restrictions emerged, due to ams' management and development procedures, having an impact in the final solution development. In software development a restriction differs from a requirement, and can be imposed by external stakeholders, systems, or even by other sections of the

development process, such as the maintenance process. According to Wiegers and Beatty[27], a restriction can be derived from:

- Technologies, tools, programming languages or databases that need to be used (or avoided);
- A system's operating environment, such as browser versions or operating systems;
- Conventions or patterns that have to be followed;
- Backward compatibility or the need to have future compatibility;
- Regulations or business rules.

More specifically, in terms of application deployment, ams requires the deployment of the application in OpenShift, a container application platform. As for the storage system restrictions, it is important to note that, due to storage issues with the company's Openshift platform, the databases to deploy will be running in a reserved Windows Server. Besides this restriction, the Oracle database will be used as a backup storage system and will contain company related information required to perform log rotation and alerting tasks. The Log Management System's restrictions are described below:

- R1: the system must be deployed in Openshift's container application platform.
- R2: the system must maintain an Oracle persistent database.
- R3: the system's read and write databases must be deployed in a Windows Server;
- R4: the system must only be accessed and developed inside the company's intranet;
- R5: the system's alerting and rotation jobs heuristics must stay stored in Oracle Database;

1.4 Summary

The above sections introduced the main challenges that must be solved in this work. The approach taken must address the distributed nature of logging systems (see section 1.3.1), keeping the logs in adequately structured storage systems (see section 1.3.2) and provide enough features for later analysis and monitoring (see section 1.3.3), all while complying with the business requirements and restrictions, described in sections 1.3.6 and 1.3.7, respectively. The correct approach must be chosen to best fit the use case at hand.

2 Approach

In this chapter we present the solution we will follow, chosen from two suitable common approaches. Considering all the challenges presented above, we have reached a set of features and architectural decisions that we believe provide the best fit for the proposed use case.

The proposed solution consists of a centralized log management system that will be capable of handling logs from a variety of sources, as well as to provide services that will help developers and administrators better understand the different affected environments. To address the problems accounted in section 1.3, existent tools will be deployed in conjunction with custom made services.

2.1.1 Logging as a service

The log management issues revealed in the sections above, have been subject to research both on academic projects[28] and commercial products[29]. These efforts converged into a service known as logging as a service (LaaS). Logging as a service relies on an architectural model for centrally collecting any kind of log files coming from different sources or locations. The log data is normalized or filtered for reformatting and forwarded to other system to be processed as data. This data can then be managed, displayed and finally taken away, according to a retention schedule based on defined criteria [30].

LaaS allows teams to standardize the collection, persistence and analysis of events, and, at the same time, accelerate delivery without sacrificing quality [31]. These logging services enable the finding of insights about performance, security, bugs in code, access patterns and other operational data, which could be difficult to obtain without the proper tools.

With adequate practices in place, logs can be one of the main sources of both operational and business information. Making this information available to others inside an organization can be quite valuable, something that is made easily possible through a LaaS framework [32]. For the problem stated above, the final solution should be a LaaS product that complies with ams requirements.

2.1.2 Overview of already existent solutions

There are two ways to provide a Centralized logging system (CLS): using an already built LaaS system from a paid provider or constructing our own Logging framework from the ground up. Either choice implies the need of five essential components: Log collection, transport, storing, analysis and alerting. LaaS commercial solutions, such as

Loggly[29], Splunk[33], Datadog[34] or Logz.io[35], have their specific features and differences but their products are based around these common five essential components. Some of these companies take advantage of open source tools, as is the case with Logz.io[35], which utilizes the Elastic stack[6] as a platform for log analytics.

Each LaaS provider needs to adapt its solution to the specific needs of the client, taking into account what type of logs they are ingesting, knowing how the data should be presented and what data should be preserved. There is a big variety of types of logs that go into these sort of systems, as there are also several kinds of procedures for doing so. To handle this diversity, providers allow the ingestion[7] of logs from common operating systems and applications, and let the client choose the most appropriate method of uploading. As an example, Loggly lets its clients upload logs using either syslog or HTTP/s and handles logs from a variety of log sources, including JSON[8], JavaScript, Docker, AWS[9], and more[39]. Logs then need to be quickly searched, filtered and analyzed. To achieve this, clients are provided with monitoring and analytics tools, and interactive dashboards capable of adapting to their system's specific needs. Datadog[34] lets clients build real time interactive dashboards, allowing them to visualize and search log data, while also offering high-resolution metrics and events for manipulating and graphing, as can be seen in Figure 4.

[6] The Elastic stack (Elasticsearch, Logstash, Kibana) is a collection of open source projects for aggregation, analysis and visualization of logs[36].
[7] Log ingestion refers to the process of receiving, validating and forwarding of log entries to appropriate storage systems.
[8] JSON (JavaScript Object Notation) is a lightweight data-interchange format[37].
[9] AWS (Amazon Web Services) provides on-demand cloud computing platforms[38].

Figure 4. Datadog summary dashboard customization[40]

Most of these companies focus their log management services around enterprise customers who need concise tools for searching, diagnosing and reporting any events surrounding data logs. Some even provide cloud-based solutions for fast deployment and easy management, as is the case with Splunk Cloud[41], a product from Splunk[33].

It is safe to say that the already existent solutions will be able to solve the problems present in this use case, but there may be other obstacles that the company will have to overcome in order to fully employ them. Such obstacles include the high monetary cost needed to buy and maintain the solution, as well as the lack of flexibility needed for complying with the specific use cases of some companies (such as the ones in 1.3.6).

Besides these already existent full stacked solutions, it is also possible to build a centralized logging system with existing open source tools. As seen in Figure 5, there are many tools available to solve particular aspects of the whole log management problem, which means that, to build a complete and robust application, several tools must be used in conjunction with each other.

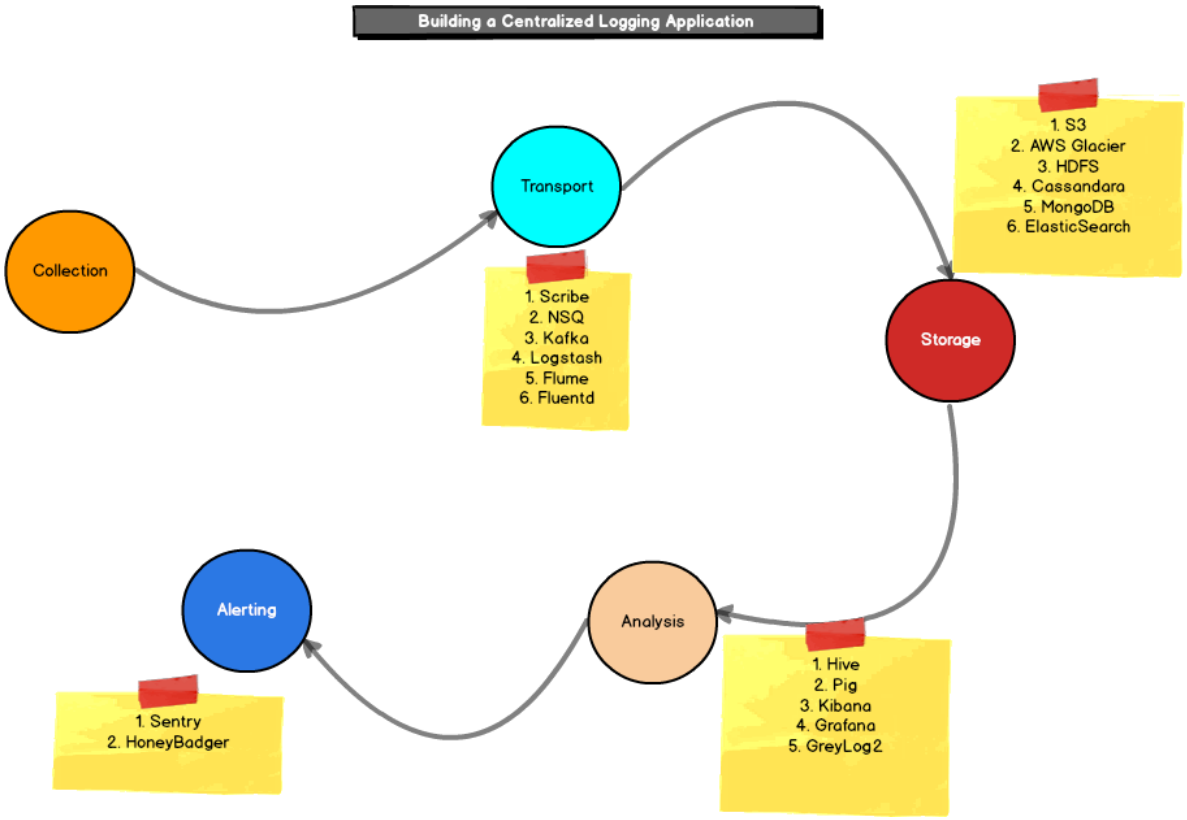


Figure 5. Centralized logging application flow[42]

An example of a stack of open source tools used as a solution to log management is the Elastic Stack, which comprises three popular open-source projects: Elasticsearch, Logstash and Kibana. This stack allows for the transport (Logstash), storage (Elasticsearch) and analysis (Kibana) of log data from different systems and locations. Deploying this stack with some custom made service to handle the log management system works as an alternative to the existent commercial products on the market.

Choosing to build a log management system from the ground up has its associated complexity costs, but comes with the advantage of having the freedom to choose the right patterns and tools for the specific needs and type of infrastructure.

2.1.3 Proposed solution

Considering the specific use case problems and suitable approaches to it, the proposed centralized log management solution should consist of several custom-made services integrated with open source products, that when deployed in conjunction solve the specific problems presented in section 1.3.

The entry point of the log management system should be a REST API, which will be expecting HTTP requests with JSON formatted data in its body. After the custom-built REST API correctly validates its inputs, according to a predefined logs schema, it should forward the logs to a data transport component and, eventually, store these in

adequate database systems. These components must ensure data durability and failure handling. Furthermore to fully comply with the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log** and the **BR4.3: the system must provide a visualization platform for the log data** business requirements, appropriate storage and interfacing systems must be introduced. This means that there should be a database for permanent storage and one for query and analysis. Alerting and Log rotation jobs should be constructed using an adequate programming language (see sections 4.2.8 and 4.2.9) and be deployed in a system with access to the storage systems, thus complying with the **BR6: Implement a log rotation job to address the requirements of each application regarding no longer needed/dated logs** and the **BR7: Implement a basic and extendable alerting mechanism;** business requirements. The CLS should be developed and deployed in a platform that allows for easy management of each component. Documentation of the existent API interfaces should be defined using a common language understandable by all users of the CLS, in order to comply with the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement.

The decision to develop the CLS from the ground up instead of deploying an already existent service came from the high monetary cost of deploying such services, as discussed in section 1.3.5, as well as due to the specific needs and restrictions that came from ams' use case.

In the following chapter we present state of the art technologies and procedures which have been used to address the decision presented in this chapter.

3 State of the art

In this chapter we provide an overview of technologies and solutions that address the problem stated above. We will focus on the technologies that were eventually selected to implement the proposed solution. However, other relevant technologies/solutions are defined in order to better contextualize our work decisions. This chapter is organized as follows, in the next subsection (section 3.1) we define and present database technologies capable of solving the problem of log management. In section 3.2, we present the technologies/stacks/programs that will work as the environments for interacting with the selected databases and to will help to build and manage the log management system.

3.1 Database technologies

To achieve the proposed solution, the log entries will have to be saved using some kind of DBMS and this data needs to be stored in a reliable and performant database. There are several kinds of database management systems available, the two main database types being the Relational databases and non-relational databases. They differentiate in how they are built, the type of information they store and how they store it [43]. Each type has its own use cases and sometimes either one can be used to solve the same problem. Although in certain situations the choice of a database type has high impacts in the performance and reliability of the system.

In the next chapters we will present solutions of Relational and non-relational databases that address the requirements and problems of this thesis.

3.1.1 SQL/Relational Databases

A relational database, or an SQL (Structured Query Language) database, is a set of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. This structure, called a schema, allows for the identification and access of data in relation to another piece of data in the database [44]. The Structured Query Language (SQL) is the declarative[10] programming language used to design and interact with relational databases. One good reason to choose an SQL database is that it ensures ACID (Atomicity, Consistency, Isolation, Durability) compliancy, reducing anomalies and protecting the integrity of the database. Also, relational databases provide transaction support, assuring the atomicity of the data in

[10] Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow [45].

the database. These databases are well understood and widely supported, which can be a major advantage when running into problems [43].

In the next subsections we will feature the Oracle and MySQL database management systems to understand how data is structured and kept in a relational scope.

3.1.1.1 Oracle Database

The Oracle Database is a relational database management system (RDBMS) from the Oracle Corporation. The system is built around a relational database framework in which data objects may be directly accessed by users through SQL. It has a fully scalable architecture and is often used by global enterprises, which manage and process data across wide and local area networks [46]. Oracle has been a leader in the worldwide relational database management systems due to being a portable, scalable, reliable, performant and general purpose solution for clients with the need of coping with large data sets and enterprise grade information [47]. Coca-Cola Femsa, one of the largest multinational bottler of Coca-Cola trademark beverages, has an Oracle Database system in use, allowing for improved database flexibility and centralization of data from all countries it works on, thus improving performance and assuring availability of its infrastructure [48]. Oracle has its own proprietary procedural extension to SQL called PL/SQL. It enables the mixing of SQL statements with programmatic constructs such as procedures, functions and packages. It also allows to control the flow of a SQL program, use variables, and write error-handling procedures [49]. One of PL/SQL's main benefits is the ability to store application logic in the database itself. The Oracle Database also has transaction support, as it is designed to work as a multiuser database, allowing multiple users to work concurrently without worrying about data corruption.

This database management system has a complex architecture, centralized around a database server[11]. The database server is designed to reliably manage large amounts of data in multiuser environments. It also prevents unauthorized access and provides efficient solutions for failure recovery [50].

The Oracle Database will have an important role in the log management system, firstly because it is the database of choice for ams, thus complying with the **R2: the system must maintain an Oracle persistent database**. Since it is already in use for other ams services, the database system has proven itself reliable, performant and consistent to ams' enterprise requirements. The Oracle Database can be applied in the log management system as a persistent storage solution, assuring data integrity and

[11] An Oracle database server consists of a database and at least one database instance. The instance refers to a set of memory structures that manage database files [50].

durability, satisfying the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log** business requirement. This database will also retain data that will be essential to accomplish the **BR6: Implement a log rotation job to address the requirements of each application regarding no longer needed/dated logs** and **BR7: Implement a basic and extendable alerting mechanism** business requirements, containing configuration values set by ams' developers/administrators.

3.1.1.2 MySQL

MySQL[51] is an Open Source relational database management system based on SQL, which is developed, distributed, and supported by the Oracle Corporation. The MySQL Database Software works as a client/server system that consists of a multithreaded SQL server that supports different back ends, several different client programs and libraries, administrative tools, and a wide range of APIs [52]. MySQL is capable of replicating data and partitioning tables for better performance and durability [53]. It has become widely popular, because of its speed, simplicity and portability, as well as being a low-cost database management system. Also, by being a multithreaded database, it allows for multiple clients to connect to it at the same time, providing, as a security measure, access control and encrypted connections using the Secure Sockets Layer (SSL) protocol [54]. MySQL is commonly used in small web-based applications along with PHP scripts to create server-side applications. It is commonly used along with open source projects that require a full-featured database management system. While it may prove to be a complete solution for small/simple applications, it should not be applied in solutions that will eventually require a scaling up of its data storage system. MySQL should also not be applied when the project requires complex queries and a high concurrency rate [55].

3.1.2 NoSQL/Non-relational databases

Non-relational databases, also known as NoSQL databases, meaning "Not Only SQL", refers to a group of non-relational data management systems, where databases are not built primarily on tables, and generally do not use SQL for data manipulation. These database management systems are useful when working with large quantities of data and when its nature does not require a relational model [56]. In addition to this, there are four main ways for a NoSQL database to store data in: it can be column-oriented, document-oriented, graph-based or organized as a key-value store.

NoSQL databases are the preferred choice for large data sets, due to being horizontally scalable[12] (it is easier to handle more traffic by *sharding*, or adding more

[12] Horizontal scalability means that the scaling is done by adding more machines into the pool of resources.

servers in the NoSQL database), unlike SQL databases, which are vertically scalable[13] and mainly limited by the capacity of a single machine [58].

In the next subsections Elasticsearch, Cassandra, InfluxDB and MongoDB will be presented as a way to better understand how NoSQL databases cope with data in a log a management scope.

3.1.2.1 Elasticsearch

Elasticsearch is an open source distributed and near real-time capable[14] full-text search engine written in Java that explicitly addresses issues of scalability, big data search, and performance, which can be problematic with relational databases [59]. ElasticSearch stores data in one or more indices. An index is a collection of documents that have similar characteristics [60]. Each document in ElasticSearch is a JSON object. The open source Java Apache Lucene library handles document indexing. This full-text search library is used to perform searches against the indexed documents [61], [62]. It is perhaps better to consider *Elasticsearch* as an interface to Lucene designed for big data from the ground up. The features that Lucene provides for searching data are directly available through *Elasticsearch*. The features that *Elasticsearch* provides around Lucene are designed to make it a powerful tool for full-text search on big-data [63].

A running instance of ElasticSearch is called a node, and two or more nodes can form an ElasticSearch cluster. ElasticSearch stores/retrieves objects through a REST API, with convenient methods to create, retrieve and delete indexes. The queries are submitted using the Query DSL[15] as formatted JSON in the body of an HTTP request. It is also possible to configure the sharding and replication on a per index basis, making ElasticSearch multi tenant[16] and quite flexible.

In terms of scalability, ElasticSearch automatically distributes shards of an index across the nodes of a cluster and controls that they are loaded equally. This technology is a good choice if more data is expected in the future, as the database scales horizontally. ElasticSearch can also handle heterogeneous data due to each shard being indexed/refreshed independently and also because its indices are constantly refreshed with a fixed time interval, making it unlikely that a shard has accumulated a high number of unrefreshed data. Besides this, ElasticSearch is also agile because it does not impose a schema on the documents in indices, automatically updating the

[13] Vertical scalability is the ability to increase the capacity of existing hardware or software by adding resources, such as increasing the CPU, RAM or SSD [57].
[14] In ElasticSearch, near real-time search means that there is a delay between the time the document is indexed and the time it is made visible to search.
[15] The Query DSL is Elasticsearch own query language based on JSON [59].
[16] Multi-tenancy is an architecture in which a single instance of a software application serves multiple costumers [64].

mapping if a document with a new field is being indexed. In addition, it can automatically alter the data type of a field (for example, changing from integer to long).

Regarding performance, Elasticsearch can search for documents satisfying a defined filter criteria concurrently, and afterwards the results from all are combined and returned, unlike most RDBMS, who do the filtering after gathering the data. While scalability and schema-free documents are common for NoSQL systems, the combination of scalability, agility and performance in one system is what makes Elasticsearch stand out from the other systems [59].

ElasticSearch can be used in several use cases, such as full text search, analytics store, auto completer, spell checker, alerting engine, and as a general purpose document store. It is often used to implement "search" capabilities. For example the BBC Archives[17] are working with ElasticSearch for its ability to process data at speed and scale, allowing large volumes of data to be managed and searched at scale with accuracy and speed [65].

ElasticSearch is an important tool for the solution, as it is one of the most powerful search engines for high performance searches, which is one of the requirements imposed by AMS. Besides this feature, it also takes into account some of the main issues addressed in the problem statement, by being a distributed, real-time and easy to scale system.

3.1.2.2 Cassandra

Apache Cassandra is an open-source distributed database for managing large amounts of structured data across many servers, providing a highly available service and no single point of failure. It offers continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones [66].

Cassandra has a master less "ring" design that uses a consistent hashing algorithm to distribute data [67]. This means that there is no concept of a master node, where all nodes play an identical role, communicating with each other equally and automatically distributing data across all nodes that participate in the "ring". Its built-for-scale architecture means that, even across multiple data centers, it is capable of handling large amounts of data and thousands of concurrent users or operations per second. Cassandra's architecture also means that there is no single point of failure and therefore is capable of offering continuous availability and uptime [66]. Cassandra is horizontally scalable due to being deployed as a cluster of instances that are all aware

[17] The BBC Archives are a project from BBC that contains material dating back to the 1880s and aims to preserve the BBC's content as a cultural record and for the benefit of future generations [65].

of each other. Data can be read or written to any node of the clusters, which will forward the request to the instance where the data actually belongs. It is highly available, having no single point of failure for reading or writing data. Each piece of data is replicated to multiple nodes, but none of these nodes holds the authoritative master copy.

Cassandra is highly optimized for write throughput, never modifying data on disk, only appending to existing files or creating new ones. Its records are structured much in the same way as they are in a relational database, by using tables, rows, and columns. Cassandra supports secondary indexes, a structure that allows for the efficient lookup of records by an attribute different than the primary key. These secondary indexes cannot be used for result ordering, but tables can be structured such that rows are always kept sorted by a given column or columns, called clustering columns. In terms of consistency, Cassandra provides eventual consistency, which means when data is updated, the system will reflect that update at some point in the future. Developers are willing to give up immediate consistency because it is a direct tradeoff with high availability[68]. The Cassandra database features have proven useful when handling large data sets, as is the case with Netflix, which made its transaction to Cassandra, because it could not build new datacenters fast enough as capacity growth was accelerating and unpredictable product launch spikes were causing availability issues [69].

Cassandra has an important role in the solution of the log management system, as it is a durable, highly available, write optimized and horizontally scalable database, addressing the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log business requirement**.

3.1.2.3 InfluxDB

InfluxDB is an open source time series database[18] developed by InfluxData designed to handle high write and query loads. It is meant to be used as a backing store for any use case involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics [71]. InfluxDB supports multiple data ingestion protocols, leaving flexible options for how the data should be collected and from where it is being pulled from [72]. What separates time series data from regular data is that we are commonly requesting a summary of data over a time. Properties that make time series data different than other workloads are data lifecycle management, summarization, and large range scans of many records. This data is best shown with the help of graphs, as can be seen in Figure 6 [70].

[18] A Time series database is a database optimized for time-stamped or time series data. Time series are measurements or events that are tracked, monitored, down sampled and aggregated over time [70].

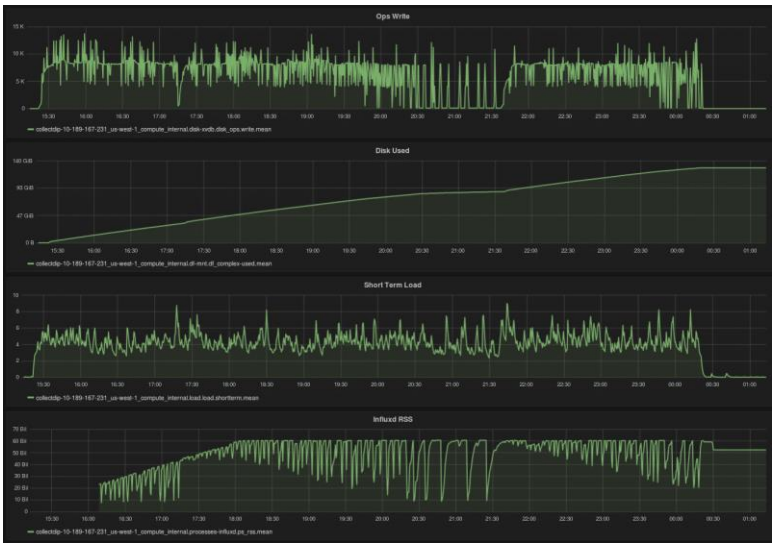


Figure 6. InfluxDB time series graph [70].

Optimizing the time series use case entails some tradeoffs to increase performance at the cost of functionality. For instance, InfluxDB cannot store duplicate data, as it may be overwritten in rare circumstances. It also only keeps data around for a short period of time, aggregating and down sampling it into longer term trend data. As the data is mostly searched by a time range, InfluxDB does not have a full text search functionality. Other characteristics include write-append-mostly, rare updates, sequential reads and occasional bulk deletes [70], [73].

If the plan is logging textual data (log messages, exceptions, requests, for example) in the database and later query them by content, then InfluxDB is not the best solution since it does not specialize in textual data searching. Using InfluxDB for this kind of logging and later querying may require the use of an additional search engine, which can cause further problems in terms of synchronizing the data between the two systems.

3.1.2.4 MongoDB

MongoDB is an open-source NoSQL document-oriented database, written in C++. A MongoDB database consists of a set of databases in which each database contains multiple collections. MongoDB works with dynamic schemas and any collection can contain different types of documents. These documents are represented as a list of key-value-pairs (a JSON structure), and the value can be a primitive value, an array of documents or again a document. The database supports indexes.

To query these objects, the client can set filters on the collections expressed as a list of key-value pairs. It is even possible to query nested fields. The queries are also JSON structured; hence a complex query can take much more space than the same query for a relational database in SQL syntax. MongoDB requires using always the correct type: If you insert an integer value into a document, you have to query for it

also with an integer value. Using its string representation does not yield the same result.

MongoDB supports two types of replication: master-slave and replica sets. In the master-slave replication, the master has full data access and writes every change to its slaves. The slaves can only be used to read data. Replica sets work the same as master-slave replications, but it is possible to elect a new master if the original master went down. Another feature supported by MongoDB is automatic sharding. Using this feature the data can be partitioned over multiple nodes. To avoid data losses, every logical node can consist of multiple physical servers which act as a replica set. MongoDB does not support full single server durability which means you need multiple replications to avoid data losses if one server suffers a power loss or crash [74]. MongoDB is designed to work as a general purpose database, not as a full-text search database. It has capable full-text search tools in testing and prototyping scenarios, but requires a high resource usage (depends on RAM) and does not scale up well, as can be seen in the graph of Figure 7, which shows the correlation between the response time of search commands and number of records made against MongoDB and Elasticsearch.

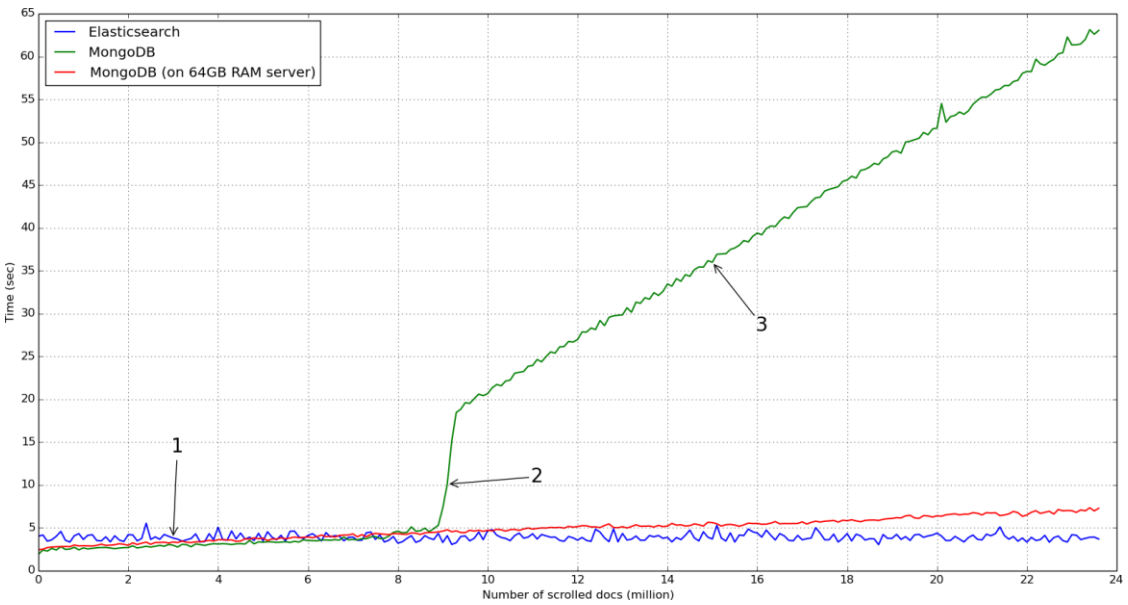


Figure 7. Correlation between response times and number of records for MongoDb and Elasticsearch(lower response time is better) [75].

Another drawback is that MongoDB does not allow full availability due to its single master architecture, provoking a downtime of 10 to 40 seconds to auto-elect a new master in case there is a failure, making it not possible to write to the replica set [76].

MongoDB is a versatile way to cover a variety of use cases in a wide range of applications. However, there are still some complex scenarios, such as a log

management scenario, where the use of this database system may not be the most appropriate.

3.1.3 Summary

From the database technologies in sections 3.1.1 and 3.1.2, Elasticsearch and Cassandra seem to fit the best with the solution due to their unique search and storage capabilities. Elasticsearch fits well with the solution, bringing high performance search capabilities, as well as having competent scalability and availability features. Cassandra is a write optimized, highly available and durable database. In the scope of this thesis the features from Cassandra and Elasticsearch will result in a database framework that is in compliance with the log management system's storage requirements.

3.2 Server Technologies

Besides the need of having a database system capable of handling a high number of inputs and storing data with the assurance that it will not get lost, it is also necessary to have the appropriate tools to interact with the database. It is necessary to develop a custom-made entry point for the log management system, which will handle the requests, validating and forwarding them to a processing pipeline that will ship the data to the storage systems.

This section introduces technologies that fit with the back-end requirements of the log management solution. The technologies described below can comprise the backbone of the CLS, helping with designing and running the system and fetching and providing data.

3.2.1 Apache HTTP Server

Apache is an open-source and free web server software maintained and developed by the Apache Software Foundation. The official name is Apache HTTP Server and it allows website owners to serve content on the web. It is one of the oldest and most reliable web servers, with the first version released in 1995. As a web server, Apache acts as a middleman between the server and client machines, pulling content from the server on each user request and delivering it to the web. Apache is highly customizable, as it has a module-based structure, allowing server administrators to turn additional functionalities on and off. Apache has modules for security, caching, URL rewriting, password authentication, and more.

The biggest challenge of a web server is to serve many different web users at the same time, each of whom is requesting different pages [77]. Apache is thread based rather than event-based, and as such, it spawns a system thread for every

incoming request. The thread based model does not scale well with many long-lived connections that would be needed in order to serve real-time and traffic-heavy applications, as is the case with log management systems that ingest high amounts of data every second. Although the Apache HTTP Server works well for web based applications, its additional layers of modules and security add latency, which can compromise the performance and reliability of a high throughput system, such as a log management system.

3.2.2 Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment. It runs on Chrome's V8 JavaScript engine, the core of Google Chrome, outside of the browser. A Node.js application is run in a single process, without creating a new thread for every request. It provides a set of asynchronous I/O primitives in its standard library, preventing JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm. Its non-blocking nature means that scalable systems are very reasonable to develop in Node. To make the development faster and efficient there are packages available through the Node Package Manager (NPM), a registry used to download and manage dependencies which hosts almost 500 000 open source packages that can be freely used.

Node.js is event-based rather than thread based, as is the case with Apache (see section 3.2.1), and, as such, presents an event loop as a runtime construct. When Node.js needs to perform an I/O operation, like reading from the network or accessing a database, instead of blocking the thread and wasting the CPU cycles waiting, Node.js will resume the operations when the response comes back. In other systems there is a blocking call to start the event-loop. In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script and exits it when there are no more callbacks to perform. This allows it to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency which could be a significant source of bugs.

Node also has a unique advantage because it is written in JavaScript, allowing for developers that were used to write JavaScript for the browser, to transition to writing server-side code without learning a new language. [78], [79].

3.2.3 Elastic Stack (ELK)

Figure 8. The Elastic Stack data flow [80].

The Elastic Stack is a collection of three open source software tools that helps in providing real time insights about data that can be either structured or unstructured. It is possible to search and analyze data using its tools with ease and efficiency. The stack is a complete end-to-end log analysis solution which helps in deep searching, analyzing and visualizing the logs generated from different machines. It has become a popular open source platform for logging [80]. In the following sub-chapters, each open source project part of the Elastic Stack will be presented, as a way to better understand how they can work together to address the log management problem of this thesis.

3.2.3.1 Elasticsearch

The first component of the Elastic Stack is Elasticsearch, the open source, distributed, RESTful, JSON-based search engine and data storage. This database is at the center of the stack as it contains the data, and the technologies that are under the Elastic Stack generally interact with Elasticsearch in one way or another. Interacting with Elasticsearch is not mandatory but since there is a strong synergy between these technologies, they are frequently used together for various purposes [81]. A more detailed explanation of Elasticsearch can be found in Section 3.1.2.1.

3.2.3.2 Logstash

Logstash is an open source data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize it into user defined destinations. Logstash originated and drove innovation in the log collection use case, allowing for the enrichment and transformation of events with the help of input, filter and output plugins, that simplify the ingestion process.

Logstash is an horizontally scalable data processing pipeline with strong Elasticsearch and Kibana synergy [82]. Logstash has a persistent queue feature to protect against data loss during abnormal termination, which will store the message queue on disk. Persistent queues provide durability of data within Logstash. Persistent queues are also useful for Logstash deployments that need large buffers. Instead of

deploying and managing a message broker[19], such as Redis, RabbitMQ, or Apache Kafka, persistent queues can be used to buffer events on disk and remove the message broker [84].

3.2.3.3 Kibana

Kibana is an open source analytics and visualization platform used for log and time-series analytics, application monitoring, and operational intelligence use cases. It is designed to work with Elasticsearch. Kibana's simple, browser-based interface makes it easy to search, view, and interact with data stored in Elasticsearch indices. With Kibana, users can perform advanced data analysis and visualize the data in a variety of charts, tables, and maps, making it easy to interactively navigate through large amounts of log data [85], [86].

The visualization of data is a necessary step in situations where a large amount of data is generated every single moment. Data-Visualization tools, such as Kibana, offer executives and other workers new approaches to dramatically improve their ability to grasp information hiding in their data. Rapid identification of error logs, easy comprehension of data and highly customizable data visuals are some of the advantages. It is one of the most constructive ways of organizing raw data [87].

The Elastic Stack can provide a strong mechanism to perform centralized logging, playing an important role in identifying server or application related problems. It lets users search through all logs in a single place (Kibana) and identify the issues spanning through multiple servers by correlating their logs within a specific time frame. These logs are collected and parsed by Logstash, which then stores them in appropriate storage systems, such as Elasticsearch. The end-to-end stack could help in solving the log management issues included in the scope of this thesis, such as the **BR2: Integration with a performant database for reads**, **BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema** and **BR4.3: the system must provide a visualization platform for the log data** business requirements.

3.2.4 Loggly

Loggly is a paid stack that provides a fully hosted end-to-end log management solution. It is a SaaS log analysis and management tool that takes costumer log data into the cloud to make sense of it and to reveal the most appropriate information to each costumer. Loggly takes streaming log data from whatever sources a company wants to provide. A Loggly costumer streams its log data to the cloud, and, as the data comes in,

[19] A message broker is an intermediary program that allows applications to communicate by sending messages to each other. Message brokers provide temporary message storage when the destination program is busy or not connected[83].

Loggly identifies it, parses it and puts it into a semi-structured representation of the data. Once the data is structured, Loggly presents the customer with a dynamic catalog of their data, available at different levels, such as category level or field level. Loggly can perform anomaly detection and notification, detecting anomalies and sending an alert for users to investigate the situation [88].

Loggly's solution is considered to be versatile, and, while most customers tend to use it for monitoring and troubleshooting, the use cases can get broader. However for this thesis use case, Loggly may not be the best fit for the final solution, as it operates on the cloud, contradicting the **R4: the system must only be accessed and developed inside the company's intranet** business restriction. Furthermore, as a SaaS product, Loggly comes with an associated cost, which is a disadvantage discussed in section 1.3.5. These limitations, along with the fixed number of functionalities and lack of independence to build custom components, discourage the deployment of Loggly in the final log management solution.

3.2.5 Summary

This section introduced technologies essential to the central functioning of the log management system. Node.js (3.2.2) can work as the backbone for the CLS, taking advantage of its JavaScript event-based asynchronous environment. The Elastic Stack (3.2.3) presents a collection of three open source software tools that can help with the centralized logging problem in searching, analyzing and visualizing the logs generated from different machines. Loggly (3.2.4), the paid stack that provides a fully hosted end-to-end log management, was also presented as an alternative to a custom-made centralized logging solution.

The technologies presented above are solutions capable of solving the structural problems of the CLS. However, there is a need to make the information stored in the centralized log management system available, a problem discussed in section 1.3.3. Section 3.3 presents two approaches to this problem.

3.3 Interfacing

Nowadays, most applications have the need to fetch data from a server where that data is stored in a database. It is the responsibility of an API to provide an interface to the stored data that fits an application's needs. A good API makes it easier to develop a program by providing all the building blocks for the interaction between components. In sections 3.3.1 and 3.3.2, we present REST and GraphQL API's, respectively, two popular means of providing an interface to fetch and provide data for the thesis' log management system.

3.3.1 REST API

A RESTful API is an API that uses HTTP requests to GET, PUT, POST and DELETE data. It is based on the REpresentational State Transfer (REST) technology, an architectural style and approach to communications used in web services development. The REST technology uses little bandwidth, making it suitable for internet applications. REST is a logical choice for building APIs, allowing the exposure of web services. A REST API takes advantage of HTTP methodologies, where a GET is used to retrieve a resource, a PUT to change the state of a resource, a POST to create the resource and a DELETE to remove it. With REST, networked components are a resource that users request access to, abstracting the implementation details. When retrieving a resource, the result is usually a JSON based response, one of the most popular and lightweight format for transporting data. The presumption is that all calls are stateless, meaning that nothing can be retained by the RESTful service between executions [89].

The implementation of a REST API as a means to insert and obtain data from the log management system fits with the case study requirements and specifications, helping to solve the 1.3.3 problem and **the BR3: Validate the log inputs, checking if values are acceptable according to the defined schema** and **BR4: Implement an interface (REST API) that allows client applications delivering log files to also selectively retrieve log information for usage within an application** business requirements.

3.3.2 GraphQL

GraphQL is a query language for APIs, an execution engine and a specification. It provides a complete and understandable description of the data in the API, giving clients the power to ask for exactly what they need, making it easier to evolve APIs over time and enabling powerful developer tools [90].

GraphQL is a new API standard that provides an efficient, powerful and flexible alternative to REST. It was developed and open-sourced in 2015 by Facebook and is currently being maintained by a large community of companies and developers from all over the world. GraphQL enables declarative data fetching where a client can specify exactly what data it needs from an API. GraphQL services typically respond using JSON, due to being a familiar format for clients and API developers, while also being easy to read and debug. HTTP requests are commonly associated with the REST pattern, which map endpoints to "resources" as its core concept. In contrast, GraphQL's conceptual model is an entity graph. As a result, entities in GraphQL are not identified by URLs. Instead, a GraphQL server operates on a single URL/endpoint, and all GraphQL requests for a given service should be directed at this endpoint. Figure 9 shows a query directed against the Github API with GraphQL, and, as can be seen, the

query has the same shape as the result. This is because the server knows what fields the client is asking for.

Figure 9. Querying the Github API with Graphql. The example query asks for a Github user's details, and, in the same request, its correspondent repositories and last commits to that repository.

GraphQL uses a strong type system to define the capabilities of an API. All the types that are exposed in an API are written down in a schema using the GraphQL Schema Definition Language (SDL). This schema serves as the contract between the client and the server to define how a client can access the data. GraphQL is database agnostic and provides a database abstraction layer, meaning that it is not restricted to a specific database type, and can effectively be used in any context where an API is used.

GraphQL has a rapidly growing community, and despite being such a young technology, it has already been widely adopted, being used in production by companies such as GitHub, Twitter, Yelp and Shopify [91], [92]. As an alternative to REST, GraphQL also fits with the log management use case and can be used as a proof of concept for ams, to show how it can cope with the need for more flexibility and efficiency. If implemented in the final solution, it can be used to illustrate its major differences when compared to REST, when it comes to fetching data from an API.

3.3.3 Summary

Sections 3.3.1 and 3.3.2 presented two solutions able to provide an API for the log management solution. Both solutions can be fetched via an HTTP request with an URL, and in response JSON data can be returned. In REST, each request usually calls exactly one route handler function, while in GraphQL, one query can call many resolvers to construct a nested response with multiple resources (see Figure 10).

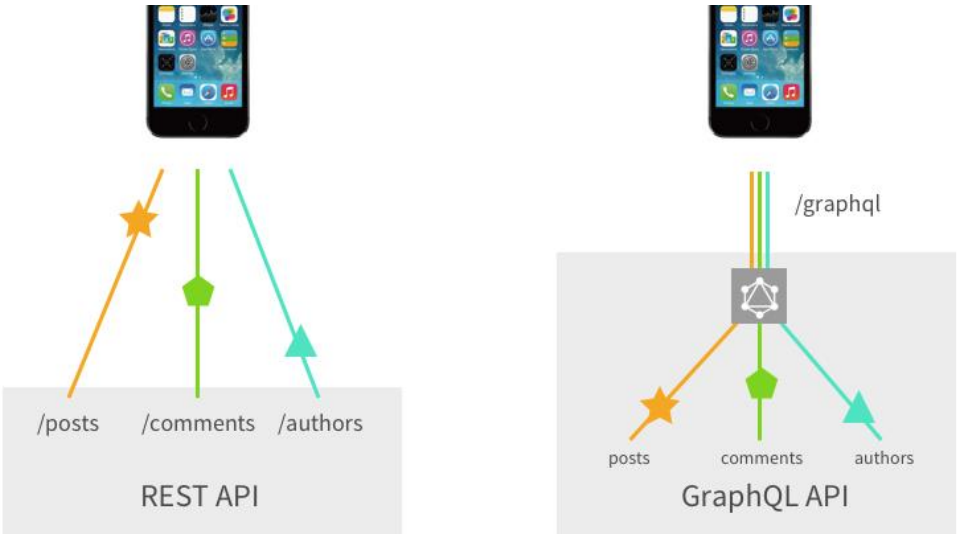


Figure 10. An interpretation of fetching resources with multiple REST roundtrips vs. one GraphQL request[93].

In REST, the shape and size of the resource is determined by the server, while is GraphQL, the server declares what resources are available, and the client asks for what it needs at the time. Overall, both REST and GraphQL APIs are different interpretations to call functions over a network.

GraphQL takes a different approach to API design and brings with it many benefits and advantages, but also unique challenges and differences. Neither GraphQL, REST, nor any other approach is purely better or worse, but each has its own unique design considerations, constraints, and tradeoffs.

Errors and bugs can appear during the APIs development, and it is essential that these are corrected before deployment. To test, document and manage the CLS, development tools should be used to ensure that the development process goes without many issues. Section 3.4 introduces development tools that help with the CLS development.

3.4 Development tools

Nowadays there are tools available to speed development, improving workflow effectiveness and productivity without risking reliability and performance. These tools help with administration, management and testing of applications and systems, in a simple and cost effective manner.

API development is usually also accompanied by some form of development tool. As an API grows in usage, there is a need for documentation, so that the end users can figure out what they can do with them. An API can be documented in many ways. If the goal is to make things easy for readers and to deliver information in a good way, then a specification is crucial to better understand its functionality. There are

tools that allow for the documentation of APIs in easy and intuitive manners, making the job of the developers simpler, as well as allowing for a faster comprehension of the API for its consumers.

This sections introduces useful development tools for managing databases (see section 3.4.2) and for API documentation and testing purposes (see sections 3.4.1 and 3.4.3).

3.4.1 Swagger

Swagger is a set of open-source tools built around the OpenAPI Specification(OAS)[20] that helps the design, building, documentation and consumption of REST APIs [95].

One of Swaggers open-source tools is called SwaggerHub, a platform where it is possible to define an API using OAS. In this tools it is possible to host all the API definitions in a single place, store common API components (such as data models or responses), later referencing them from API definitions. It is also possible to generate server and client code based on API definitions [96].

Swagger uses a common language that everyone can understand, being easily comprehensible for both developers and non-developers. Also, Swagger is easily adjustable and can be successfully used for API testing and bug fixing. The Swagger framework is supported by corporations such as Google and Microsoft. Also big companies like Yelp and Netflix have already used it in their work.

Swagger is a good fit for solving the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement, as it provides a set of tools that helps to easily generate and visualize client code, as well as being one of the most popular frameworks for this purpose.

3.4.2 Toad

Toad for Oracle is a set of database management tools from Quest Software built around an advanced SQL-PL/SQL editor[97]. It can be used for Application development, Database development, Business intelligence, and deploying Oracle-based applications and Web services. Using Toad, developers can build and test scripts, PL/SQL packages, procedures, triggers and functions. It allows for the creation and editing of database tables, views, indexes, constraints and users. Toad for Oracle is the toolset that ams uses to view and manage the Oracle data at the company. For this reason it will be an important toolset to use in the testing and configuration tasks of the Oracle Database system.

[20] The OpenAPI Specification defines a standard, language-agnostic interface to RESTful APIs, allowing its readers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection [94].

3.4.3 Postman

Postman is an API development tool that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP requests, as well as read their responses. Postman has the ability to make various types of HTTP requests (GET, POST, PUT, PATCH), saving environments for later use and allowing the conversion of APIs to code for various languages. Along with these functionalities, Postman helps to see the status codes, time taken for response and other performance parameters. In result, developers get a more efficient and less tedious workflow [98].

For the log management thesis scope, the Postman software can be used to send and receive requests, and with these requests, test if the interfacing components (present in section 3.3) of the CLS comply with the company's business requirements.

3.4.4 Summary

Section 3.4 presented development tools that help with improving workflow effectiveness and productivity. Swagger, the tool used for API documentation, can help in solving the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement. Toad for Oracle, the toolset used by ams, will help in viewing and managing the Oracle data at the company. Lastly, Postman, the API development tool, can help with testing the CLS API endpoints.

3.5 Deployment solutions

In order to have a stable, scalable and easy to manage development and deployment environment, companies often resort to Platform as a Service (PaaS) solutions, which provide resources that enable developers to test and deliver their enterprise applications. These PaaS solutions can include infrastructure, middleware, development tools, business intelligence services, database systems, and other related development services. They are designed to support the building, testing, deployment, management and updating of applications (the complete web application lifecycle) [99]. As such, an environment that helps to develop and deploy software is a must have for companies that rely on the continuous availability, and performance of their applications, while also being capable to easily modify/re-deploy existing software with little to none effort. In the following subsections we present two PaaS solutions that provide a fast and scalable way to host the log management system applications.

In section 3.5.1 we introduce Openshift, the company's application platform of choice.

3.5.1 OpenShift

Red Hat OpenShift is a multifaceted, open-source container application platform from Red Hat Inc. for the development, deployment and management of applications. OpenShift provides developers with an integrated development environment for building and deploying Docker-formatted containers, and then managing them with the open source Kubernetes[21] container orchestration platform. OpenShift provides the OpenShift Container Platform as a private platform as a service (PaaS) for organizations such as ams, which uses it to deploy and manage OpenShift on their own on-premises hardware. The OpenShift Container Platform is implemented as an assemblage of Docker-based application containers managed via Kubernetes orchestration, all running on an operating system foundation of Red Hat Enterprise Linux.

As container use increases in app development and production, services such as OpenShift offer ways to manage and automate a large amount of containers. Doing so frees up developers from the manual management of containers. OpenShift's options promote continuous app development and common tools for development operations teams. This approach is a cornerstone of DevOps efforts [101]. Due to its features, ams chose OpenShift as the PaaS application platform for developing, deploying and managing applications. For this reason, and due to the **R1: the system must be deployed in Openshift's container application platform** business restriction, the application platform will be used for deploying the CLS custom components.

3.5.2 Microsoft Azure PaaS

Microsoft offers a cloud computing solution, named Microsoft Azure. Microsoft Azure is a cloud computing platform that provides SaaS, PaaS, IaaS (Infrastructure as a Service), and supports numerous programming languages and tools, which include but are not limited to ASP.NET, PHP, and Node.js. The Azure's PaaS environment features a Portal that acts as the command console, allowing for the administration and monitoring of the infrastructure. The Azure platform includes a complete infrastructure of servers, storage, networking, and middleware developing tools, allowing its clients to host their applications without being concerned about the maintenance of the servers and its operating systems. The Azure PaaS is designed for organizations that have high operational costs and want to lessen administration needs, as well as the ones that want to reduce complexity while increasing scalability and improving the quality of service.

[21] Kubernetes is a portable, extensible open source platform for managing containerized workloads and services, facilitating both declarative configuration and automation [100].

Instead of being only a cloud computing solution, the Azure infrastructure can also be extended to run on premise, providing companies with an additional security layer.[102]

3.5.3 Summary

There is no denying that both PaaS' deployment solutions offer flexible, stable and profitable results. For this thesis use case OpenShift will be the platform utilized for deployment, considering that it is an already on premise solution and due to the **R1: the system must be deployed in Openshift's container application platform** business restriction. However, it is important to be aware that there are further PaaS application platforms available which come with additional costs, such as Microsoft Azure's solution, which could fit well with other use cases.

3.6 State of the art - Summary

This chapter provided an overview of technologies and solutions that address the problems stated in section 1.3. We explored solutions of relational (section 3.1.1) and non-relational (section 3.1.2) databases to address storage issues, focusing on the non-relational ones due to being the preferred choice for large data sets, as is the case with this thesis use case. We also introduced technologies that may form the backbone of the centralized log management system (section 3.2), such as server software and software stacks designed to help solve the thesis use case. Application programming interface solutions were presented (section 3.3), as a way to provide an interface to the stored data that fits with the system needs. Section 3.4 introduced development tools that can improve workflow productivity, useful for managing the Oracle database (section 3.4.2) and for testing and documenting API's (sections 3.4.1 and 3.4.3). Finally, section 3.5 presented deployment platforms which can be used for deploying and managing applications and systems, such as the log management system to develop.

We focused on the technologies that will eventually be selected to implement the proposed solution (addressed in section 4.1). However, other relevant technologies/solutions were described in order to better contextualize our work decisions.

4 Solution

In this chapter, the technologies and approaches taken to solve ams' requirements and comply with the restrictions will be presented. The features of the selected technologies will help to comply with the business requirements, providing a scalable and performant log management system, capable of handling the company's log workload. Section 4.1 explains why each technology was chosen and its role in the Log Management System. In section 4.2 an in-depth explanation of the main implementation aspects in each component will be described.

4.1 Architecture

To set up a log solution that can address every business requirement, it is crucial to choose the right technologies. These tools will make it possible to overcome the business requirements such as the need of persistent and reliable write and read performant databases, as well as the implementation of a query interface for data input and consumption.

After specifying the problem statement (see section 1.3) of the log management system and analyzing several well-known technologies used for log data storage (section 3.1), web development and data transfer (sections 3.2, 3.3 and 3.4), it is now possible to define which technologies fit with the needs of the log management system.

The log management solution is based on the approach described in section 1.4 and divided in three layers, one for the database technologies, one for the back end/ server side technologies and another for the interfaces of the system. Every technology of each category has its own importance to the final solution, either to deal with interfacing purposes, store log data or handle the backbone of the log management system.

All the presented technologies were introduced in chapter 3, and, in the following sub-section we describe how the selected technologies can fit into the ams' case study, business requirements and restrictions.

4.1.1 Storage systems

The database technologies of the log management system must provide good write performance, as well as high read capabilities. Additionally, considering ams' use case, which deals with high volumes of data and is constantly growing in demand, it is important to ensure that the databases can scale up and are kept online and operational, otherwise the company may lose business. Databases, such as Cassandra

(section 3.1.2.2) and Elasticsearch (section 3.1.2.1) provide the necessary mechanisms to ensure that the storage system is scalable and elastic, features that help in improving the availability and performance when demand changes.

A storage system that adds searchable references, such as indexes, to the logs being stored allows for quick data search and retrieval. Elasticsearch was designed for this specific use case, providing full-text search and analysis of big data volumes without compromising performance. Besides its powerful search capabilities, Elasticsearch also provides scalability and high availability. These qualities make Elasticsearch the right choice to handle the search tasks of the Log Management system, while also addressing the **BR1.2: the system shall implement a database with better write performance than the currently used by ams (oracle)** and **BR2.1: the system must provide a better read performance than the currently used by ams' Oracle system** business requirements.

Elasticsearch may work well as a search engine, but it lacks the robustness in relation to data durability. To handle the eventual data loss that Elasticsearch may experience, it is best to use it alongside a database that has a stronger focus on constraints, correctness and robustness [103], [104]. For this log management scenario, Cassandra has proven suitable for applications that cannot afford to lose data, due to being horizontally scalable, highly optimized for write throughput, never modifying data on disk and offering continuous availability and uptime. From a performance comparison between Oracle and Cassandra (see Appendix A), the latter proved more performant under high stress loads. For these reasons, Cassandra will be the database of choice to address the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log** business requirement.

Elasticsearch and Cassandra are part of the solution because they offer reliable functionalities which are vital for the log management system's operation. Besides these two technologies, Oracle (section 3.1.1.1), the company's database of choice, will also be implemented in the final solution, taking into account the **R2: the system must maintain an Oracle persistent database** restriction. The reason for this is the fact that Oracle is already in operation at ams. Furthermore, ams' Oracle database contains internal set-points concerning the behavior of the log alerting and rotation jobs to implement, which is part of the **R5: the system's alerting and rotation jobs business logic must stay stored in Oracle Database** restriction. This database will also serve as an alternative backup storage solution to Cassandra, allowing for developers to search data with a tool they are familiar with.

4.1.2 Back end/ server side

The back-end technologies for this project should be lightweight, efficient, scalable, easy to integrate with each other and provide enough resources to implement the major set of business requirements:

- BR1.1: the system shall be able to handle 100 log inputs per second without losing any log;
- BR3.1: the system must specify a schema to be used by all log inputs;
- BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema;
- BR4.1:the system must provide means to modify, visualize and access log data;
- BR4.2 the system must provide an API to insert and consume data;
- BR7.1: the system must alert each application stakeholder by email;
- BR6.1: the system must comply with ams' defined log persistence schedule;

To develop the necessary back-end applications (alerting job, rotation job and REST/GraphQL API) it is important that the environment where these applications will be developed offers scalability, high performance and seamless integration between the technologies. Node.js (section 3.2.2) is known for its speed and event-driven architecture that helps in the development of a throughput and scalable application, being very handy in terms of high volume of IO bound requests. Being lightweight and fast, Node.js makes for a great server side environment by using JavaScript to build the log management back end portion. As such, a REST API will be developed with Node.js, which will work as the log data entry point for the log management system. There will be an endpoint for data collection, which will validate the inputs according to a predefined schema, and forward them to Logstash (section), the data pipeline system that works as a message broker, thus solving the **BR4.2 the system must provide an API to insert and consume data** and **BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema** business requirements. In the REST API there will be another endpoint which will serve as the data consumption source, allowing for developers to query and filter for logs. Alongside the REST API, there will also exist a GraphQL endpoint in the Node.js API server. The GraphQL endpoint will be used to query the database, providing the same functionality as the data fetching REST endpoint, which also addresses the same business requirements.

Besides the API server with the RESTful and GraphQL endpoints, Node.js will also be used to develop a custom alerting and log rotation mechanisms, allowing us to solve the **BR7.1: the system must alert each application stakeholder by email** and **BR6.1: the system must comply with ams' defined log persistence schedule by**

making use of database drivers and packages adequate for alerting purposes business requirements. The purpose of the alerting job and rotation job is to report developers when errors occur and to delete dated logs, respectively.

4.1.2.1 Log pipelining

The data being ingested by the REST API must be redirected to three different storage systems, one for search (Elasticsearch) and the rest as a back-up for permanent and reliable storage (Cassandra and Oracle). To ensure that data is not lost during this process and is being reliably sent to all destinations with the correct format, a message queue-like data pipeline must be deployed between the REST API and the database systems. Logstash, as a scalable data collection engine with real-time pipelining capabilities that ensure that no data is lost (by providing a persistent queue feature and retry the policies), can help solve the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log** and **BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema** business requirements.

Besides the integration with Elasticsearch, Logstash can also normalize the log data to other storage systems like Cassandra and Oracle, with the help of community plugins. This makes Logstash the perfect technology for handling workload peaks and making data persistent, while also reducing errors that may occur when other parts of the system go offline [105].

4.1.2.2 Deployment

When talking about back end/server side, it is imperative to comply with the **R1: the system must be deployed in Openshift's container application platform** restriction. To address this restriction, the final solution, with the exception of the storage systems, will be deployed using Openshift, an open source container application platform that helps with the development, deployment and management of applications. The storage systems were not deployed in Openshift because, during the CLS's implementation, the ams' Openshift platform was not supporting persistent storage, and thus, made it unrealizable to store the logs permanently in it. To get around this issue, the storage systems, apart from Oracle Database (which was already at production in ams), will be deployed in a reserved Windows Server, therefore allowing for permanent storage and complying with the **R3: the system's read and write databases must be deployed in a Windows Server** restriction.

OpenShift allows the creation of applications using source code from a local or remote Git repository, and, since all Node.js components of the Log Management System's (such as the REST API) are stored in GitHub repositories, the only action

necessary is to specify the repository URL. OpenShift then automatically builds the source code into a new application image and deploys that image with an adjacent service to provide a load balanced access to the deployment that runs the image.

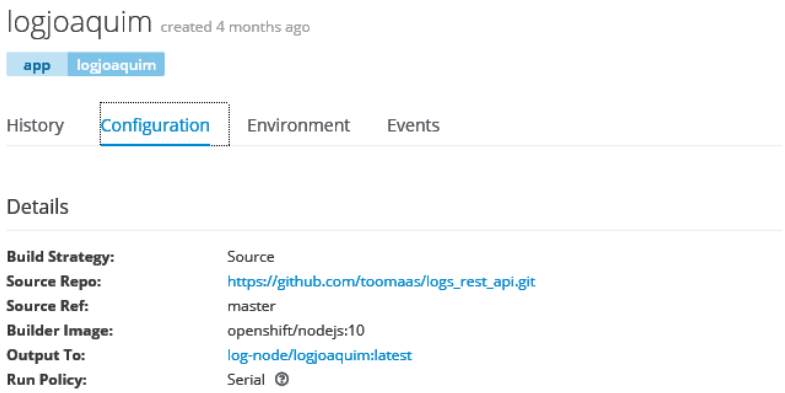


Figure 11. The logjoaquim Openshift application, which runs the REST API source code of the Log Management System.

Figure 11 shows the details of the deployed REST API service in the OpenShift interface, referenced as the `logjoaquim` application, where it is possible to view the source repository, as well as the internally accessible URL. With the platform's functionalities, OpenShift will allow developers to easily manage the development environment for building and deploying applications such as the one developed in this thesis.

4.1.2.3 Summary

Figure 12 summarizes the back end and database (section 4.1.1) technologies, describing the flow that a log takes in the Log Management System at the input state.

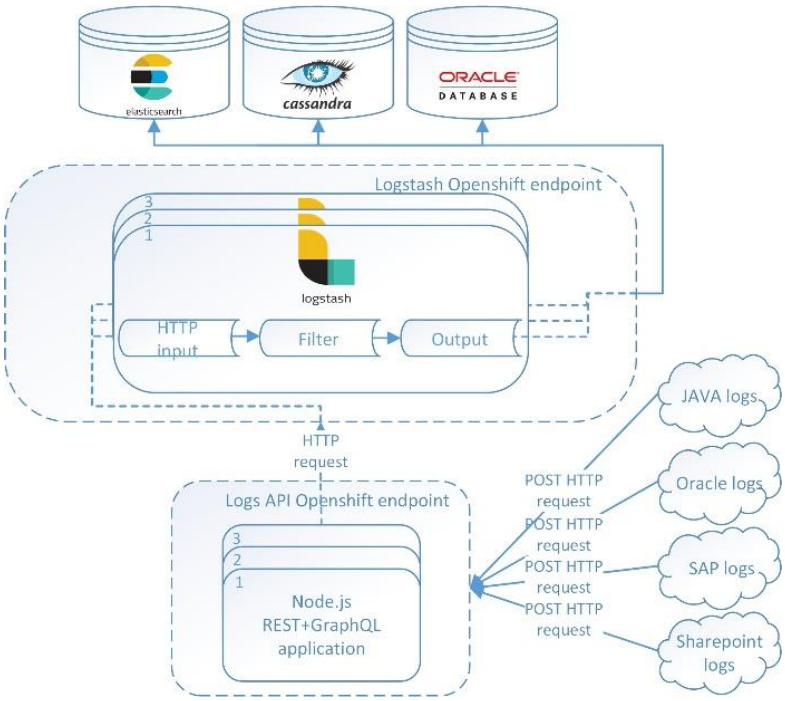


Figure 12. Log data input flow architecture

Looking at Figure 12 and taking into consideration the previous chapters, the log input flow has the REST API as its entry point. Its job is to receive logs from ams' systems and validate them according to a schema common to all applications. These logs are then forwarded to Logstash, which, with its pipelining capabilities, makes sure that the logs are safely transported to the Log Management System's storage systems.

The integration of Node.js, Logstash and Openshift will be beneficial for the back end portion of the log management solution, as these technologies ensure that the solution is scalable and fault tolerant, issues that have to be addressed when dealing with critical/important business aspects, such as the scope of this project, which is essential to the welfare of ams DevOps.

4.1.3 Front end and interfaces

As stated in section 4.1.2, the Node.js API server will have two endpoints (REST and GraphQL) for data fetching, which is accomplished by sending HTTP requests to get data back. To consume the API, developers will first have to know how to use it and what to expect. Documentation is the key to solve this problem, as stated by Doglio[106]. Giving the ability to try out the API before implementing it, and providing a detailed list of parameters and explanatory examples, decreases the amount of time spent onboarding new users. Swagger (section 3.4.1) provides tools to easily define APIs using the OAS[20] standard, making the documentation process easier to generate and maintain, while also allowing for a better experience for the ones consuming the

API. As Swagger is one of the most popular open source API documentation tools, it will be implemented as a solution to the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement, giving users the necessary documentation to consume the API and the ability to try it out.

The interaction with the API (REST and GraphQL) can be seen as an interface for the CLS, thus solving the **BR4.1:the system must provide means to modify, visualize and access log data** and **BR4.2 the system must provide an API to insert and consume data** business requirements.

Besides this "developer/programmer oriented" interface, there are also graphical user interfaces (GUI) that provide visually rich ways to navigate around the log data. This is where Kibana contributes to the solution, by providing an analytics and visualization platform that works with the Elastic stack. Kibana will allow for not only developers/programmers, but also system managers and administrators to search, view and interact with data stored in Elasticsearch. Kibana's simple, browser-based interface enables its users to generate reports, monitor data across the Elastic Stack and visualize and embed dynamic dashboards that display the log data in real time. With the implementation of Kibana in the log management solution, we aim to solve the **BR4.1:the system must provide means to modify, visualize and access log data** and **BR4.3: the system must provide a visualization platform for the log data** business requirements, allowing for a wider range of user types to interact with the log management system.

To conclude, the front end and interface technologies, the Log Management System will have three different entry points, showed in Figure 13.

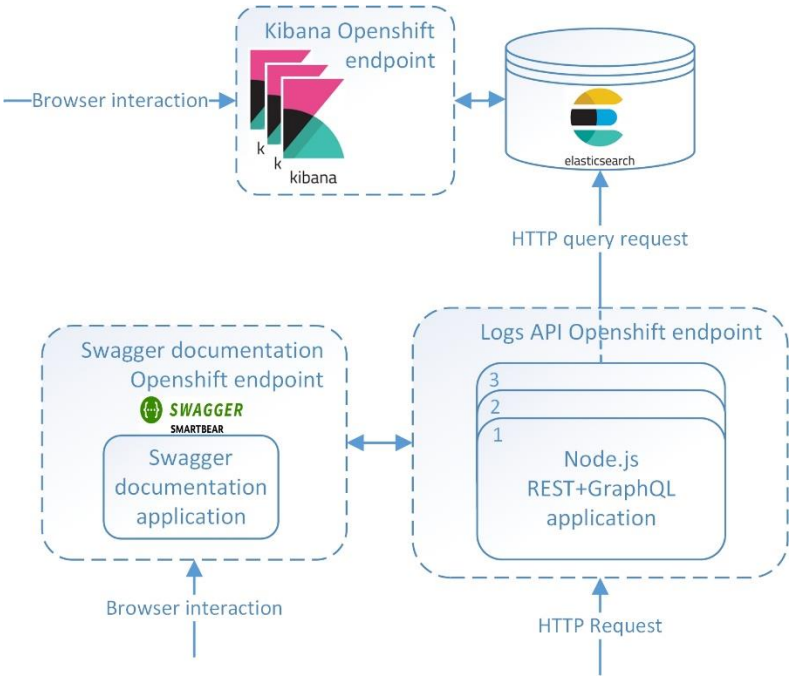


Figure 13. The three entry points of the log management system.

The CLS front end and interfaces represented in Figure 13 can be summarized as follows:

The first entry point is an API server (with REST and a GraphQL endpoints), which will be developed in Node.js, and allow for the input and retrieval of logs. Next, Kibana (section 3.2.3.3) will be deployed as a solution for **BR4.3: the system must provide a visualization platform for the log data** business requirement, providing a visual platform to monitor the logs. The last interface of the log management system aims to solve the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement and will be represented by a Swagger documentation server that will be working as an interface to test the REST API and GraphQL endpoints and provide developers with details on how to work with these interfaces.

4.2 Implementation

This section is reserved to explain the main aspects of each tool developed and deployed in the log management system. To understand the role that each tool has in the final solution, a system architecture was designed, as can be seen in Figure 14. As stated in section 4.1, the solution is divided in three layers, the storage, back end and the front end technologies.

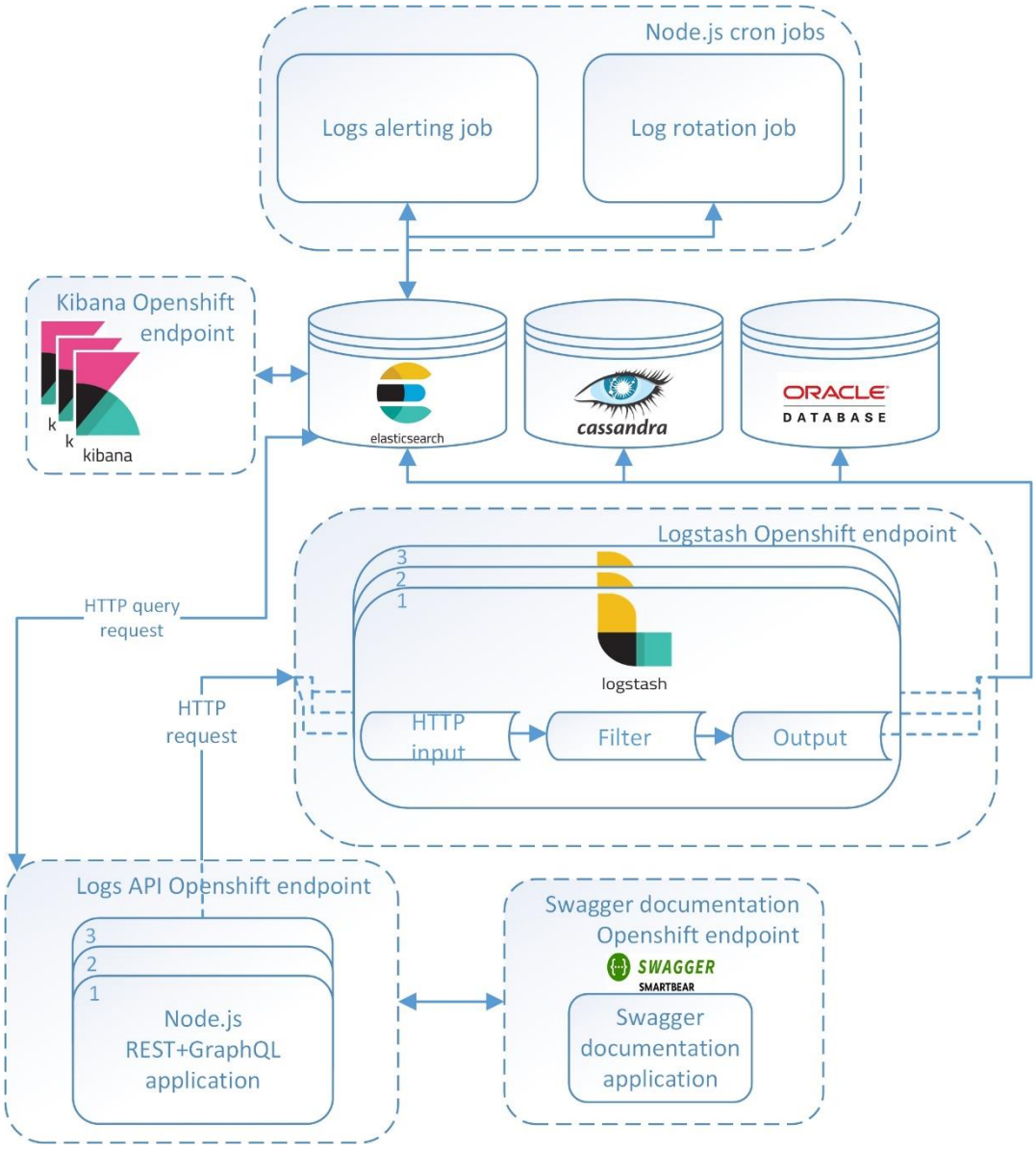


Figure 14. Architecture sustaining the log management system.

Instead of dividing the structure of this chapter the same as was done in section 4.1(i.e. by category), each component of the log management system will be explained separately followed by its most relevant characteristics.

4.2.1 Openshift deployment

As stated in sections 4.2.3, 4.2.4, 4.2.5, 4.2.6 and 4.2.7, and as a means to comply with the **R1: the system must be deployed in Openshift's container application platform** business restriction, Openshift (introduced in section 3.5.1) was used to build, deploy and run the REST API, GraphQL API, Logstash, Kibana and Swagger applications,

respectively. To manage these applications, the Openshift OKD[22] (Origin Community Distribution) was used. It is important to notice that the deployment and management of the applications in Openshift was performed by ams' supervisor, due to ams security policies.

Although Openshift had restrictions in terms of permanent storage (covered in section 4.1.2), which prevented us to also deploy the storage systems, there were no restrictions when deploying the applications stated in the beginning of this chapter. For the API server and the Swagger documentation server, which were written in Node.js, and had their source code stored in a GitHub repository, it was necessary to share the repositories link and Openshift automatically built a Source Image based on the source code. As for Logstash and Kibana, these applications were deployed based on existing images from the Docker Hub library.

4.2.2 Database management systems

Log data will have to be stored in different storage systems to guarantee performant search capabilities and permanent storage. As a whole, Elasticsearch, Cassandra and Oracle address these needs, but in order to get the most out of these systems, they first need to be configured to handle the log data coming from ams's applications. The next sub-sections explain the need and implementation of a logs schema, as well as the main aspects of the configurations applied in the CLS's database management systems.

4.2.2.1 Schema

Before we can insert data in the storage systems, a schema must be designed. This schema must address every log structure of ams' applications (Oracle, Sharepoint, SAP, as well as legacy applications). The schema was designed with the guidance of ams' supervisor, which helped in making sure that it could handle and represent the logs from ams' applications in a unified manner. The final log management system's schema was comprised of only one table and its representation can be seen in Figure 15.

[22] OKD is the open source upstream community edition of Red Hat's OpenShift container platform. OKD is a solution to manage, deploy and operate containerized applications that includes an web interface, automated build tools, routing capabilities and monitoring and logging aggregation features[107].

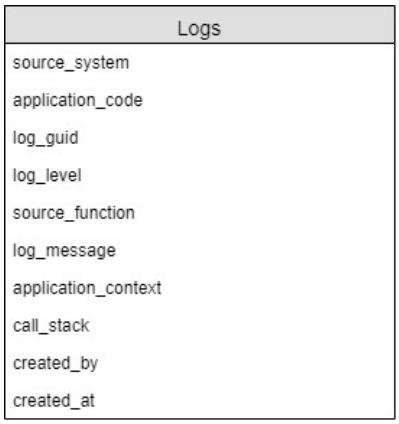


Figure 15. Logs general schema, representing the logical view of the data to store.

This schema is the skeleton structure that will represent the logical view of the Oracle and Cassandra databases. Besides this, the REST API and Logstash's log input validation logic will be based on the same schema. The fields in Figure 15 represent the structure of a log in the log management system:

- `source_system`: is one of ams' systems where the log originated from (Oracle, SAP, SharePoint, to name the most frequent systems);
- `application_code`: is the code name of the application where the log originated from (LDS, MZE, LDSI, as examples of application names from Oracle's system);
- `log_guid`: is a globally unique identifier that references the parent log that originated the current one. It can be used to give context on why the log entry was created;
- `log_level`: is the mean of categorizing the entries in the database by urgency, indicating the severity of the log entry (INFO, DEBUG, ERROR, FATAL_ERROR or WARNING);
- `source_function`: is the name of the function within the application where the entry was generated;
- `log_message`: provides a quick description on what occurred in the log entry;
- `application_context`: is the field used to enter any information pertinent to the log entry that is not covered in other fields. This field can store large sized documents so its size should not be limited (other than by disk space).
- `call stack`: is a report of the active subroutines at the time the log entry was created

- `created_by`: is the user code name that is responsible for the execution of the routines that originated the log entry;

- `created_at`: is the date and time that the log entry was created. The date and time is represented in UTC (Coordinated Universal Time) time zone with the following format: YYYY-MM-DD HH:mm:ss[23].

Since the logs general schema does not have a unique identifier (ID) field, each database system will have the responsibility of creating its own distinguishable field. Besides the `created_at` field, the datatype of every other field of the log management system are variable-length character strings. Regarding data retrieval in databases, it is important to note that data should be stored according to the way it is commonly queried, otherwise the database systems will have to inefficiently filter data, causing an increase in the needed computing resources. According to ams' supervisors, developers search log data filtering by `source_system` and `application_code` fields, which, in ams' use case, is the easiest way to reduce the number of rows returned. This means that related logs will be grouped together for better access, with the help of additional data structures, such as table indexes[24].

The schema will be set in Oracle and Cassandra databases, as well as used for validating the log inputs in the REST API. Elasticsearch will not need a schema definition beforehand, as it dynamically maps its fields to the correct type.

4.2.2.2 Database technology - Oracle

As mentioned in section 4.1.1, the Oracle database will be implemented in the final solution as a backup database and as a means to obtain information regarding the operational logic of the log alerting and rotation jobs, complying with the **R2: the system must maintain an Oracle persistent database** and **R5: the system's alerting and rotation jobs heuristics must stay stored in Oracle Database** restrictions.

As the Oracle database system is already in use at ams, there was no need to worry about database scalability, consistency and durability, as these are properties that are already guaranteed by the database system itself. The only configuration needed to perform was the schema definition. To interact with the Oracle system, a database user account and an empty schema were created. We used the Toad for Oracle (section 3.4.2) application to manage the Oracle database system.

The Oracle database schema was defined taking into account the general logs schema (section 4.2.2.1), which can be seen in Figure 15. Even though there will be

[23] The logs date and time format is YYYY-MM-DD HH:mm:ss, where the digit groupings represent: YYYY – year, MM – month, DD – day, HH – hour, mm – minutes, ss – seconds.
[24] An index is a data structure which is used to quickly locate and access data in a database table. Indexes are created using database columns.

little search queries made to the Oracle database system (i.e. the Oracle database will not be the main log retrieval source of the CLS), the table structure was configured in such a way that allows for quick data retrieval operations. With these considerations in mind the Oracle logs table was created, as shown in Figure 16:

Column Name	ID	PK	Index Pos	Null?	Data Type
ID	1	1	1	N	NUMBER
SOURCE_SYSTEM	2		1	N	VARCHAR2 (50 Byte)
APPLICATION_CODE	3		2	N	VARCHAR2 (50 Byte)
LOG_GUID	4		1	N	VARCHAR2 (128 Byte)
LOG_LEVEL	5			N	VARCHAR2 (50 Byte)
SOURCE_FUNCTION	6			Y	VARCHAR2 (100 Byte)
LOG_MESSAGE	7			N	CLOB
APPLICATION_CONTEXT	8			Y	CLOB
CALL_STACK	9			Y	CLOB
CREATED_BY	10			N	VARCHAR2 (50 Byte)
CREATED_AT	11			N	TIMESTAMP(6)

Figure 16. Oracle database logs table schema

The CLOB (character large object) data type is set in the `SOURCE_FUNCTION`, `LOG_MESSAGE` and `APPLICATION_CONTEXT` columns, as these columns can sometimes contain large documents. With this data type these columns can store up to 4GB of text. As for the remaining columns, with exception of the `CREATED_AT` columns which is set as a timestamp, the VARCHAR2 data type was set, as it allows to save on space used by the table.

To speed the data access of the Oracle database, we took into consideration the most commonly used fields to query data according to ams' supervisor. With this information two index objects were created, as can be seen in Figure 17.

Img	Index Name	Unique	Logging	Degree	Columns	Order	Index Owner
	INDEX1	N	YES	1	SOURCE_SYSTEM, APPLICATION_CODE	Asc	JALD
	INDEX2	N	YES	1	LOG_GUID	Asc	JALD
	KEY1	Y	YES	1	ID	Asc	JALD

Figure 17. Oracle database's index table

One index was created on the `SOURCE_SYSTEM` and `APPLICATION_CODE` columns and another was created on the `LOG_GUID` column, which is a column that is commonly used to understand the context of a log entry. With these two indexes the number of disk I/O is reduced, meaning that the system no longer has to perform a full table scan to fetch results from a query that includes the indexed columns.

Since the logs general schema (seen in Figure 15) does not have a unique identifier column, the `ID` column was created to uniquely distinguish each log entry in the Oracle logs table. The value of this column is automatically incremented with the

help of an Oracle trigger[25], which fires each time an INSERT statement is executed against the logs table. The trigger statement can be seen in Figure 18.

```
CREATE OR REPLACE TRIGGER JALD.JALD_LOGS_ELK_TRG
BEFORE INSERT
ON JALD.JALD_LOGS_ELK
REFERENCING NEW AS New OLD AS Old
FOR EACH ROW
BEGIN
-- For Toad:  Highlight column ID
  :new.ID := JALD.JALD_LOGS_ELK_SEQ.nextval;
END JALD_LOGS_ELK_TRG;
/
```

Figure 18. The trigger statement that is set to fire and increment the ID column before a log is inserted in the logs table.

With the above configurations done, the Oracle database is now ready to accept log inputs and is also optimized for data search. This data will be accessed through drivers that allow the creation of JDBC connections to the Oracle database.

Besides working as a backup storage solution, the Oracle Database will also contain information necessary for the operation of the log alerting (section 4.2.8) and rotation jobs (section 4.2.9), available through the `AIC_OWNER.TEST_LOG_THESIS_NOTIFI` and `AIM_OWNER.AIM_V_LOG_LEVEL_RETENTIONS` views[26], respectively.

4.2.2.3 Database technology - Cassandra

Cassandra will be the database of choice to solve the **BR1: Database with performance for writings** business requirement, by being a write optimized, durable, and horizontally scalable database management system.

In the current development environment the Cassandra database system is deployed in a dedicated Windows Server, hosted in Austria, complying with the **R3: the system's read and write databases must be deployed in a Windows Server** business requirement. Even though Cassandra is a NoSQL database, it requires users to first define a schema before inserting data. To interact with the database system, cqlsh[27], the included command line client for Cassandra will be used to execute CQL

[25] A trigger is a PL/SQL unit that is stored in the database and automatically executes/fires in response to a specified event, such as a statement executed against a table (eg. INSERT, UPDATE or DELETE), a system event (such as a startup or shutdown) or a user event (such as login or logout) [108].

[26] A view in Oracle is a representation a SQL statement that is stored in memory so that it can be easily re-used. A view contains no data itself. The tables upon which a view is based are called base tables. A view can be used as a security mechanism, where permissions to access data are set on the view instead of the underlying tables.

[27] The Apache Cassandra installation includes the cqlsh utility, a python-based command line client for executing Cassandra Query Language (CQL) commands [109].

commands interactively. To create a table in Cassandra, it first needs a keyspace, which is the equivalent to a database in a relational system. When specifying a keyspace, the *replica placement strategy*, *replication factor* and *durable writes* settings can be configured. These settings determine how many copies of the data are kept in a given data center, impacting consistency, availability and request speed of the system. For the current development environment, which is composed of a single dedicated windows server, the keyspace was set as follows (Figure 19):

```
CREATE KEYSPACE jald WITH replication = {'class': 'SimpleStrategy', 'replication_factor':
'1'}  AND durable_writes = true;
```

Figure 19. Current logs keyspace specification

The replication strategy `'SimpleStrategy'` in the logs keyspace specification in Figure 19 is the best fitted strategy for the current Cassandra deployment, as it does not span multiple data centers. The `'replication_factor'` tells Cassandra how many copies of each piece of data are to be kept in the cluster, and, since we are running a single instance of Cassandra, there is no point in keeping more than one copy of the data. In a production deployment, a `'NetworkTopologyStrategy'` replication strategy and a replication factor of at least 3 would ensure that the system is capable of handling multiple data centers and that no data is lost.

With the keyspace configured, the logs table was configured, and its description in CQL is the following (Figure 20):

```
CREATE TABLE jald.jald_logs (
   application_code text,
   source_system text,
   id uuid,
   application_context text,
   call_stack text,
   created_at timestamp,
   created_by text,
   log_guid text,
   log_level text,
   log_message text,
   source_function text,
   PRIMARY KEY ((application_code, source_system), id)) WITH CLUSTERING ORDER BY (id ASC);
CREATE INDEX loglevel ON jald.jald_logs (log_level);
CREATE INDEX srcsys ON jald.jald_logs (source_system);
```

Figure 20. Logs table definition in Cassandra

The Cassandra table structure is based in the logs general schema (see section 4.2.2.1). For each entry to be unique, a `uuid` data type field was included (the id column). The value of this column must be generated outside of Cassandra, as the database system does not have enough capabilities to create uniquely identifiable values. This issue will be addressed in the Logstash implementation (see section 4.2.5).

Another essential feature that makes Cassandra an efficient system when storing and distributing data is the primary key definition. A Cassandra primary key has two components: the partition key (`application_code` and `source_system` columns) and the clustering columns (*id* column). Besides showing the uniqueness of the record in the database, the partition key also determines data locality, grouping related rows together for efficient storage and lookup. The clustering columns specifies the order that data is arranged inside the partition. To recap, the compound primary key in the Cassandra logs table (seen in Figure 20) is defined so that logs with the same application code but from different source systems get stored separately and each partition will be ordered by `id`. This storing structure was proposed by ams' supervisor, as these fields are the most commonly used for filtering data when retrieving logs.

Besides partitioning, indexes also provide means to access data in Cassandra using attributes other than the partition key. The benefit is fast and efficient lookup of data matching a given condition. Cassandra actually does not allow users to conditionally query a column that has no index, as it does not know where the data may be located, thus making the query inefficient and requiring a lot of computing resources. If a user tries to execute a query with a field that has no index or is not part of the primary partition key, the system responds with an error/warning, as can be seen in Figure 21.

```
cqlsh:jald> Select * from jald_logs WHERE log_level='DEBUG';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this
query as it might involve data filtering and thus may have unpredictable performance. If
you want to execute this query despite the performance unpredictability, use ALLOW FILTER-
ING"
```

Figure 21. Error Message when executing a CQL query that requires extra filtering.

To prevent this issue, indexes were created on the `log_level` and `source_system` columns, allowing users to filter data with these columns, as these columns are the most frequently used ones for data filtering. If we execute the query in Figure 21 after setting the referred indexes, Cassandra can now return values without needing to perform extra filtering, as it uses the secondary index on the `log_level` column to find the matching rows (seen in Figure 22).

```
cqlsh:jald> Select * from jald_logs WHERE log_level='DEBUG' limit 2;

application_code | source_system | id                                   | created_at                      | created_by  | log_guid |
log_level | log_message                          | source_function
-----------------+---------------+--------------------------------------+---------------------------------+-------------+----------
+-----------+--------------------------------------+--------------------------------------
             MZE |        ORACLE | 0000b10a-8483-4404-bf6a-f92773f28f5a | 2018-01-24 21:01:45.000000+0000 | MAZEF_OWNER | 69100945 |
DEBUG | Start: MZA TSC Id [3741115], Site [PHI] |     MZE_REPLICATION.SET_MZA_TRANSFERED
             MZE |        ORACLE | 00017909-5c18-475c-b5bc-889a41106661 |                                 |             | 2018-01-25
00:02:06.000000+0000 | MAZEF_OWNER | 69112317 |     DEBUG |                      Start: Site [PHI] | MZE_REPLICATION.GET_TIME-
ZONE_FROM_SITE
```

Figure 22. CQL statement that selects logs with 'DEBUG' severity level.

Overall, the keyspace and partitioning configurations set in Cassandra makes it optimized for data storage, and, if necessary, for data retrieval. The decision to partition the logs this way was done taking into account the most frequent fields used for querying data, as suggested by ams' supervisor.

4.2.2.4 Database Technology - Elasticsearch

Elasticsearch will be the database system responsible for the log search tasks of the Log Management System, as it provides full-text search capabilities and analysis of big data volumes without compromising performance.

Similar to Cassandra, and due to the **R3: the system's read and write databases must be deployed in a Windows Server** development restriction (stated in section 1.3.7), Elasticsearch is deployed in a dedicated company Windows Server. While this may not be ideal considering possible future workloads and management, it is practical for learning and testing the software components used in this section of the solution.

An Elasticsearch instance requires very little configuration. Most settings can be changed on a running cluster. Nevertheless, there are still some configurations needed to be set before deploying an Elasticsearch node. The most important ones are the node-specific settings, such as `node.name` and path variables. We also need to set configurations for a node to be able to join a cluster, such as `cluster.name` and `network.host` [110]. These configurations can be seen in Figure 23:

```
# ---------------------------------- Cluster -----------------------------------
# Use a descriptive name for your cluster:
cluster.name: logs-cluster
# ------------------------------------ Node ------------------------------------
# Use a descriptive name for the node:
node.name: windows_server
# ---------------------------------- Paths -----------------------------------
# Path to directory where to store the data (separate multiple locations by comma):
path.data: /path/to/data
# ---------------------------------- Network -----------------------------------
# Set the bind address to a specific IP (IPv4 or IPv6):
network.host: 10.11.112.38
http.port: 9201
# --------------------------------- Discovery ----------------------------------
# Pass an initial list of hosts to perform discovery when new node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"]
discovery.zen.ping.unicast.hosts: ["10.11.112.38:9301", "10.63.20.29"]
```

Figure 23. Elasticsearch node and cluster configurations.

With the node configurations done, the system is ready to run and start receiving logs. In Elasticsearch documents will be stored in indexes, which are collections of documents that have similar characteristics. An index is identified by a name and this name is used to refer to the index when performing indexing, search, update and delete operations against the documents in it. For the purposes of the Log

Management system, an index will be created per day, making it easier to delete old data or only search specific date ranges. Each index will be subdivided in 5 shards and replicated 1 time, this will help in preventing node failures and provide high availability and scalability in a network environment [111]. Contrary to Cassandra and Oracle, Elasticsearch does not require any schema, as its engine automatically maps the inputs and stores the documents correctly. This means that after the first log is inserted, Elasticsearch will define if a field is a text, number or date and determine the format of date values. For the logs of the Log Management System, the Elasticsearch mapping is shown in Figure 24:

```
{
  "mapping": {
    "doc": {
      "properties": {
        "@timestamp": { "type": "date" },
        "@version": { "type": "keyword" },
        "application_code": { "type": "text" },
        "application_context": { "type": "text" },
        "call_stack": { "type": "text" },
        "created_at": { "type": "date" },
        "created_by": { "type": "text" },
        "host": { "type": "text" },
        "log_guid": { "type": "text" },
        "log_level": { "type": "text" },
        "log_message": { "type": "text" },
        "source_function": { "type": "text" },
        "source_system": { "type": "text" }
      }
    }
  }
}
```

Figure 24. Elasticsearch logs mapping specification.

A document in Elasticsearch is expressed in JSON, which is a ubiquitous internet data interchange format. A log entry from the Log Management System is stored in Elasticsearch as shown in Figure 25:

```
{
  "_index": "logstash-2019.05.29",
  "_type": "doc",
  "_id": "UKtmBGsBICXP_wkLf7fC",
  "_version": 1,
  "_score": null,
  "_source": {
    "source_system": "ORACLE",
    "application_code": "LDS",
    "created_by": "LDS_OWNER",
    "log_guid": "69043502",
    "source_function": "TEST_REPLICATION.MAIN",
    "@timestamp": "2019-05-29T16:22:34.562Z",
    "@version": "1",
    "log_level": "ERROR",
    "created_at": "2019-05-29T16:22:34",
    "host": "10.128.0.1",
    "log_message": "Start: Begin",
    "call_stack": "",
    "application_context": ""
  },
  "fields": {
    "created_at": [ "2019-05-29T16:22:34.000Z" ],
    "@timestamp": [ "2019-05-29T16:22:34.562Z" ]
  },
  "sort": [ 1559146954000 ]
}
```


Figure 25. Elasticsearch's representation of a document

When Elasticsearch stores a document, it automatically assigns a uniquely identifiable value in the `_id` field, which is a field that is created by default and is used to uniquely distinguish the log entry. The values of the log entry get stored as text in the `_source` object as key value pairs and are all searchable, thanks to the Elasticsearch full text feature.

Elasticsearch exposes REST APIs that can be called directly to configure and access Elasticsearch features. Each request consists of a HTTP verb (GET, POST, PUT and DELETE as the most common verbs), an URL endpoint and an optional body, which in most cases is a JSON object. As an example of a request, in Figure 26 a search to the Elasticsearch endpoint is made using the Query DSL[28] within the request body, and it will return the first 20 logs from the application MZE at the Oracle source system.

[28] Elasticsearch provides a full query DSL (Domain Specific Language) based on JSON to define queries.

```
curl -X GET "elasticsearch_endpoint:9201/logstash*/_search" -H 'Content-Type:
application/json' -d'
{
    "query": {
        "bool": {
            "must": {
                "match": {
                    "source_system": "Oracle"
                }
            },
            {
                "match": {
                    "application_code": "MZE"
                }
            }
        }
    },
    "size": 20,
    "sort": {
        "created_at": {
            "order": "asc"
        }
    }
}
```

Figure 26. Example of Elasticsearch request using curl

The requests made to Elasticsearch from the Log Management System's components, such as the REST API and GraphQL endpoint, will have a structure similar to the one in Figure 26. These components' implementation will be addressed in sections 4.2.3 and 4.2.4.

4.2.3 REST API

A REST API was developed with Node.js, working as a data consumption source and as the log data entry point for the log management system. For readability purposes, we will refer the data consumption source as the GET endpoint and the log data entry point as the POST endpoint, which are the existent endpoints in the REST API.

This component aims to solve the **BR3: Validate the log inputs, checking if values are acceptable according to the defined schema** and **BR4: Implement an interface (REST API) that allows client applications delivering log files to also selectively retrieve log information for usage within an application** business requirements.

As previously mentioned, (in section 4.1.2), the REST API server was developed on Node.js and structured with the Express[29] web framework. Express is the right choice for a server when it comes to creating and exposing APIs to communicate as a client with the server application. The application code is organized having in mind the separation of concerns idea [113], as seen in the following figure (Figure 27):

[29] Express.js, a popular web framework for Node.js, https://expressjs.com [112].

```
src
│   server.js       # App entry point
└──elasticsearch   # Database connection/configuration
└──graphql         # GraphQL endpoint logic
└──routes
    └──api         # Express route controllers for all the endpoints of the REST API
└──package.json    # List of packages the project depends on
```

Figure 27. Node.js project architecture

Although there is a GraphQL endpoint present in the above structure (Figure 27), it does not interfere with the REST API's logic. Besides working as an alternative to the REST API, GraphQL is also an API, being this the reason why the endpoint logic is located in the same project structure as the REST API.

The two REST API endpoints (seen in Figure 28) can be accessed using HTTP requests, specifying the URI (or path) and a specific HTTP request method (GET or POST).

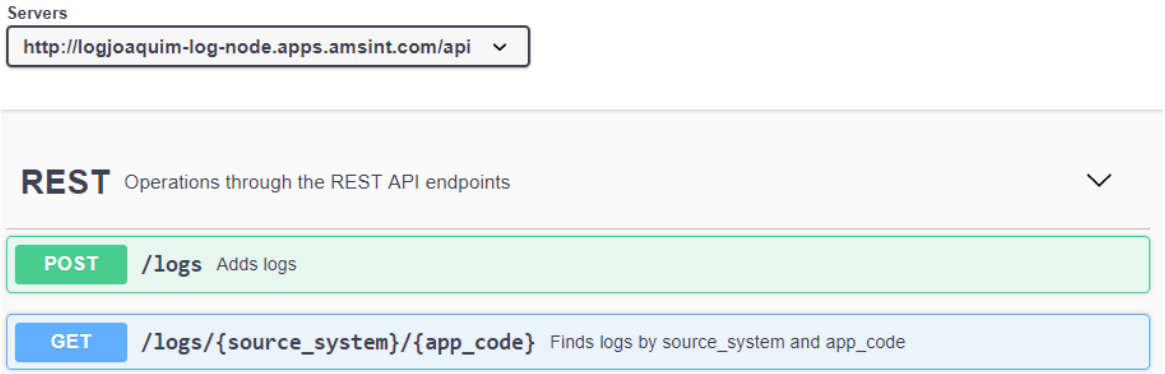


Figure 28. REST API server paths and request methods.

Each endpoint has different route handler functions, which are executed when the route is matched. The following sub-sections (4.2.3.1 and 4.2.3.2) describe the GET and POST endpoints implementation.

4.2.3.1 GET endpoint

A request to the logs GET endpoint requires the user to insert a `source_system` value and an `app_code` value as path variables to be able to access the route, having the results returned in JSON format. Besides these required variables, a request to the logs fetching route can also have additional filtering fields present as query parameters:

- `log_level`, used to filter by log severity;
- `created_at_gte`, to filter logs that have a greater than or equal to the created_at timestamp value;

- `created_at_lte`, to filter logs that have a lower than or equal to the created_at timestamp value;
- `log_guid`, to filter logs related to a unique identifier.
- `from`, the offset from the first result that the user wants to fetch.
- `size`, maximum number of logs in the response.

Pagination of results can be done by using the `from` and `size` parameters. `from` defaults to position 0 and `size` defaults to 500 if not defined.

The GET endpoint route handler will parse the path and query parameters from the HTTP request and execute a search in Elasticsearch based on these filtering values. The response to the client is always JSON based. An example of executing a `cURL` HTTP request to the logs GET endpoint to get the first 5 “ERROR” level logs from the ”MZE” application inside the “SHAREPOINT” source system dated after 2019-02-27 can be seen in Figure 29:

```
  curl -X GET "http://logjoaquim-log-
node.apps.amsint.com/api/logs/sharepoint/mze?log_level=ERROR&created_at_gte=2019-02-
27&size=5"
```

Figure 29.GET endpoint request example using cURL.

A successful JSON response of a request to the GET endpoint (Figure 30), such as the one in Figure 29, is returned to the client with the total number of matches and an array with log objects up to the number set in the size parameter:

```
{
    "total hits": 243427,
    "data": [
        {
            "source_function": "TEST_REPLICATION.MAIN",
            "source_system": "SHAREPOINT",
            "log_guid": "69044095",
            "log_level": "ERROR",
            "application_code": "MZE",
            "application_context": "",
            "created_at": "2019-03-01T12:10:20",
            "log_message": "Start: Begin",
            "call_stack": "",
            "created_by": "MZE_OWNER"
        },
        ...
        ...
        {
            "source_function": "TEST_REPLICATION.MAIN",
            "source_system": " SHAREPOINT ",
            "log_guid": "69043890",
            "log_level": "ERROR",
            "application_code": "MZE",
            "application_context": "",
            "created_at": "2019-03-01T12:10:21",
            "log_message": "Start: Begin",
            "call_stack": "",
            "created_by": "MZE_OWNER"
        }
    ]
}
```


Figure 30. Successful response of a request to the GET endpoint.

4.2.3.2 POST endpoint

A log is inserted in the Log Management system through the POST endpoint. This route allows more than one log to be inserted on the same call. Log data is written in the request body as an array of JSON objects, and each object in the array is a set of key/value pairs representing a log entry. Figure 31 is an example of a cURL HTTP request to the POST endpoint:

```
curl -X POST http://logjoaquim-log-node.apps.amsint.com/api/logs -H 'Content-Type:
application/json'
  -d '[
  {
    "source_function": "TEST_REPLICATION.SET_STATE_END_TS",
    "source_system": "ORACLE",
    "created_at": "2019-07-08 11:21:22",
    "log_guid": "69043197",
    "log_message": "Start: Set state_end_ts_utc to [24.01.18 10:47:26,396404] for tester
name [EX12.1], state_start_ts_utc [24.01.18 10:47:07,307689], id [4876379]",
    "call_stack": "",
    "application_code": "LDSI",
    "application_context": "",
    "created_by": "LDSI_OWNER",
    "log_level": "WARNING"
  }
]'
```

Figure 31. Example HTTP request to the POST endpoint.

When the POST route handler gets a request, it will validate the request body input against a schema that is defined according to the logs general schema (see section 4.2.2.1). Joi[114], a Javascript package for object schema descriptions and validation, was used to validate the inputs of the logs general schema. This package allowed us to create a blueprint of a Javascript object to ensure that the data, which will be processed and ultimately stored in the storage systems, complies with the structure and requirements of the logs general schema. In Figure 32 we can see the schema of what we expect an object to resemble in the HTTP POST request:

```
const schema = Joi.array().items(Joi.object().keys({
            source_system: Joi.string().regex(/^[a-zA-Z]*$/).required(),
            application_code: Joi.string().regex(/^[a-zA-Z]*$/).required(),
            log_guid: Joi.string().required(),
            log_level: Joi.string().required(),
            source_function: Joi.string().required(),
            log_message: Joi.string().allow('').required(),
            application_context: Joi.string().allow('').required(),
            call_stack: Joi.string().allow('').required(),
            created_by: Joi.string().required(),
            created_at: Joi.date().format('YYYY-MM-DD HH:mm:ss').raw().required
        }))
```

Figure 32. Joi object schema validator for the logs input endpoint.

With the schema defined, the `Joi.validate()` method is called and it may either throw an error if the validation fails, or allow for the continuance of the route handler function, which will then forward the validated logs to Logstash (see section 4.2.5).

4.2.3.3 Summary

The REST API was deployed as a Docker container in the OpenShift (see section 3.5.1) platform, and as such, it is important to ensure that the service is kept running without exceptions or abrupt shutdowns. To prevent these occurrences, the REST API was developed to maintain its execution even when a failed API call is requested, providing error messages that helps the user understand the nature of the error so that he/she can design the appropriate solution. The JSON object seen in Figure 33 is an example of an error response from the REST API of a badly formatted date field in a HTTP call to the GET endpoint.

```
{
  "error": {
    "name": "parse_exception",
    "details": "failed to parse date field [2019-20-05] with format [yyyy-MM-dd||yyyy-MM-dd'T'HH:mm:ssZ||yyyy-MM-dd'T'HH:mm:ss].Cannot parse \"2019-20-05\": Value 20 for monthOfYear must be in the range [1,12]"
  }
}
```

Figure 33. Example response of a failed API call.

The returned error object structure seen in Figure 33 is the format that will be returned to the user when using the API. The expected server responses are documented in the Swagger component, detailed in section 4.2.7.

4.2.4 GraphQL

Alongside the REST API, there is also a GraphQL endpoint in the Node.js API server. The GraphQL endpoint will only be used to query the database, providing the same functionality as the data fetching REST endpoint (GET), which also addresses **the BR4.1: the system must provide means to modify, visualize and access log data business requirement**. Additionally, GraphQL gives clients the power to ask for exactly what fields they need in the response.

A GraphQL service is created by defining types[30], then providing functions for each field on each type. For ams' use case, the GraphQL service will return logs stored in Elasticsearch. For data filtering, the GraphQL service will also allow users to filter data with the same set of parameters as those used in the REST API GET endpoint (see section 4.2.3.1).

The GraphQL endpoint was created with an Express GraphQL middleware instance, which required three configuration properties, seen in Figure 34:

[30] A GraphQL Object Type is a representation of a kind of object that can be fetched from a service, and what fields it has. Examples of types can be seen in Figure 35.

```
app.use('/graphql',graphQLHttp({
  schema:schema,
  rootValue:rootValues,
  graphiql:true
}))
```

Figure 34. The Express GraphQL middleware configuration properties.

The GraphQL schema (as seen in Figure 35) is used to describe the APIs type system. It includes the complete set of data and defines how a client can access that data. Each time the client makes an API call, that call is validated against the schema. Only if the validation is successful the action is executed. Otherwise an error is returned. For the Log Management system's use case, we defined the *Log* object type and also defined that a user can Query for logs by issuing the `logsSearch` query action and use as arguments the parameters of the `LogsSearchInput` input object for result filtering, which are the same set of parameters that the REST API GET endpoint uses (see section 4.2.3.1).

```
type Log {
  source_function: String!
  source_system: String!
  log_guid: String!
  log_level: String!
  application_code: String!
  application_context: String!
  created_at: String!
  log_message: String!
  call_stack: String!
  created_by: String!
}
input LogsSearchInput{
  source_system: String!
  application_code: String!
  log_level: String
  created_at_gte: String
  created_at_lte: String
  log_guid: String
  from: Int
  size: Int
}
type Query {
  logsSearch(searchInput : LogsSearchInput) : [Log]
}
```

Figure 35. The GraphQL schema definition.

Next a root resolver was created. A resolver contains the mapping of actions to functions. In our use case the root resolver contains only one action: `logsSearch`. The resolver function assigned to the `logsSearch` action returns logs from the Elasticsearch database and uses the arguments of the `LogsSearchInput` input object as the query filtering fields. It is worth noting that, similarly to the REST GET endpoint, the user is required to enter a value for the `source_system` and `application_code` fields when

requesting logs from the GraphQL endpoint. To achieve this, exclamation marks are represented in the `LogsSearchInput` input object definition, meaning that the `source_system` and `application_code` fields are non-nullable, and thus, the GraphQL service guarantees values in those fields.

The last GraphQL configuration seen in Figure 34 is the `graphiql` property. Setting this property to true enables GraphiQL. GraphiQL is a graphical interactive browser that allows users to directly write queries in the browser and try out the endpoint. Users are also provided with the schema definition from the GraphQL server and can explore what options are available, as seen in Figure 36.

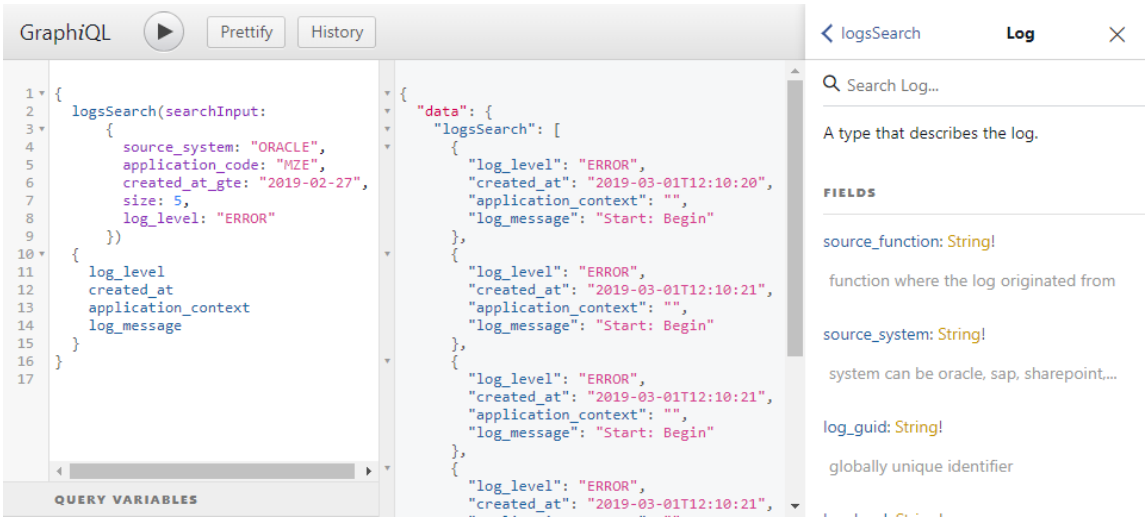


Figure 36.The GraphiQL interface, composed by a section to write queries (left), one for the queries results (middle) and one for documentation (right).

With the configurations done, the GraphQL endpoint was up and running in the node.js API server.

A GraphQL request differs from a REST request. While both of these technologies involve sending an HTTP request and receiving a result, there are core differences between the two. In REST, the core idea is the resource, which is identified by a URL and retrieved by sending a HTTP request to that URL. In ams' use case, the resource might be referred to as the "logs endpoint" and accessed with a GET `/logs/:source_system/:app_code` request. On the other hand in GraphQL, a resource is not coupled to the way it is retrieved, and, to actually access a particular log, the `logsSearch` Query type had to be created (seen in Figure 35). Another difference is that in REST, the shape and size of the resource is determined by the server, while in GraphQL, the server declares what resources are available, and the client asks for what it needs at the time. Figure 37 and Figure 29 illustrate the differences between a request that is asking for the same number of logs to the GraphQL and REST endpoints, respectively. The log data returned by these requests is expected to be the same.

```
curl -X POST "http://logjoaquim-log-node.apps.amsint.com/graphql"
     -H "Content-Type: application/json"
    -d '{ "query":
"{logsSearch(searchInput:
  {
    application_code: \"MZE\",
    log_level: \"ERROR\",
    source_system: \"ORACLE\",
    created_at_gte: \"2019-02-27\",
    size: 5
  })
{ log_level, application_context, created_at, log_message }}\"}'
```

Figure 37. Example cURL request to the GraphQL endpoint.

The result of the GraphQL request in Figure 37 can be seen in the middle section of Figure 36, which only shows four fields of each log in the response (`log_level`, `application_context`, `created_at` and `log_message` fields), as requested in the GraphQL query.

The result of the REST request can be seen in Figure 30.

4.2.5 Logs pipelining - Logstash

Logstash is the data processing pipeline that is placed between the Logs API and the storage systems. It ingest logs coming from the REST API, transforms them, and then sends them to the three storage systems of the Log Management System, and at the same time, makes sure that this data does not get lost by taking advantage of its retry policies and persistent queues capabilities, improving reliability and resiliency (see section0).

Logstash was initially configured in a reserved windows server for testing reasons, before it got deployed in OpenShift (see section 3.5.1) as a Docker container. At its current state, Openshift is holding three Logstash replicas, allowing for a higher throughput rate.

To get Logstash running in the log management system, first we had to define, program and configure the pipeline. A Logstash pipeline is created based on a configuration file.

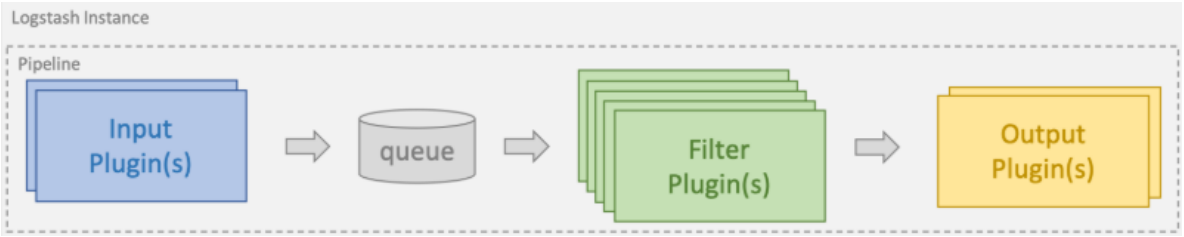


Figure 38. Overview of a Logstash pipeline.

As seen in Figure 38, the Logstash pipeline has three elements: the `input`, `filter` and `output`. The `input` plugins consume data from a source, the `filter` plugins

modify the data as necessary and the `output` plugins write the data to a destination. There are also internal queues between pipeline stages which are used to buffer events.

The Logstash `input` element consists of an HTTP input plugin[115], which lets applications send HTTP requests to the endpoint started by this input and Logstash will convert it into an event for subsequent processing. The inputs generate events, which, in ams' use case, are treated as log entries (each log entry is an event). This plugin makes it possible to receive single or multiline events over HTTP. As referred in section 4.2.3, the REST API will be forwarding the logs that arrive at the Log Management System to Logstash. These logs are sent over HTTP requests to the Logstash endpoint (with the log entries in the request's body), and then are converted into events for subsequent processing. All the data coming in is JSON formatted.

Once the logs are in the pipeline, they are placed on an internal queue. This internal queue is set to 2GB (default is 1GB) and gets persisted on disk in order to improve reliability and resiliency [84]. The logs then need to be transformed to comply with each database structure and with the general schema described in section 4.2.2.1 before being sent to the storage systems,. For this, we make use of the `filter` plugins. The `filter` plugins perform intermediary processing on an event. Filters are applied conditionally depending on the characteristics of the event. The first plugin applied was the `clone` filter plugin, which can be used for creating additional copies of an event [116]. For ams' use case, the `clone` filter plugin triplicated the event from the input into: one for Oracle, one for Cassandra and one for Elasticsearch (Figure 39).

```
  clone{
    # makes three additional copies of the event. The new events come with a “type” field,
with the values of the clone configuration
    clones => ["cassandra_log","elasticsearch_log","oracle_log"]
  }
```

Figure 39. The clone filter plugin configuration.

The triplicated events are inserted into the pipeline as normal events and will be processed by the remaining pipeline configuration starting from the filter that generated them. The event is triplicated because each database output source requires different formats in several fields. Using the `clone` filter made it easy to control how each output is arranged. For example, in Cassandra, there is the necessity of creating and assigning a value to the id column, which cannot be automatically generated at insert time (see Figure 40).

```
if [type] == "cassandra_log"{
    uuid{
      target => "id"
      overwrite => true
    }
}
```

Figure 40. Logstash filter configuration for the Cassandra event.

Once the logs have been processed, the processing threads send them to the appropriate output plugins, which are responsible for formatting and sending the logs to each particular storage system. Outputs are the final stage in the event pipeline. For ams' use case, we will need three different output plugins: `elasticsearch`[117], `jdbc`[118] and `cassandra`[119] output plugins. In this section, each one of the previously triplicated events will be redirected to the appropriate output plugin with the use of "`if`" and "`else`" conditional statements.

For example, the configuration shown inFigure 41 uses the `jdbc` output plugin to send an event to the Oracle database if the type metadata field has a value of `oracle_log`:

```
  if [@metadata][type] == "oracle_log"{
     jdbc{
      connection_string => "jdbc:oracle:thin:@//racqd1-
scan.amsint.com:1521/itasdev.amsint.com"
      statement => [ "INSERT INTO JALD_LOGS_ELK (SOURCE_SYSTEM, APPLICATION_CODE,
LOG_GUID, LOG_LEVEL, SOURCE_FUNCTION, LOG_MESSAGE, APPLICATION_CONTEXT, CALL_STACK,
CREATED_BY, CREATED_AT) VALUES (?,?,?,?,?,?,?,?,?,TO_DATE(?, 'YYYY-MM-DD HH24.mi.ss'))",
"source_system", "application_code", "log_guid", "log_level", "source_function",
"log_message", "application_context", "call_stack", "created_by", "created_at"]
      username => "JALD"
      password => "${OR_PWD}"
      driver_class => "oracle.jdbc.driver.OracleDriver"
    }
   }
```

Figure 41. JDBC output configuration to output to the Oracle database.

The conditional configuration seen in Figure 41 is the same used to output log events to the Elasticsearch database and to the Cassandra database.

As all logs that Logstash receives come from the REST API, it is important to mention that when an exception occurs on an event in the pipeline, the processing of that event halts and a message is returned to the REST API, giving information on what happened.

4.2.6 Kibana

Kibana, the open source analytics and visualization platform designed to work with Elasticsearch, was deployed in Log Management System as a solution to the **BR4.1: the system must provide means to modify, visualize and access log data** and **BR4.3: the system must provide a visualization platform for the log data** business requirements,

working as an alternative to the REST API for obtaining/reading logs. As Kibana is an easy to interact visualization platform, it aims to widen the user type range of the Log Management Solution, allowing for less tech savvy users to view and analyze log data.

As with Logstash and the Node.js API server, Kibana was also deployed in OpenShift (see section 4.2.1) as a Docker container, allowing us to take advantage of the platform's replication and management functionalities.

The Kibana server reads its startup properties from a `kibana.yml` file. Kibana connects with the URL of the Elasticsearch server defined in the `.yml` file, then it search view and interact with the stored data.

Kibana allows its users to interactively explore logs from the Discover page. The Discover page gives access to every indexed document of the Log Management Solution. It allows users to submit search queries, filter the search results and view document data [119]. Since every log in the Log Management solution has a time field, a distribution of documents over a specified time range is displayed in a histogram. Figure 42 shows the Discover page configured to present logs of ams' applications.

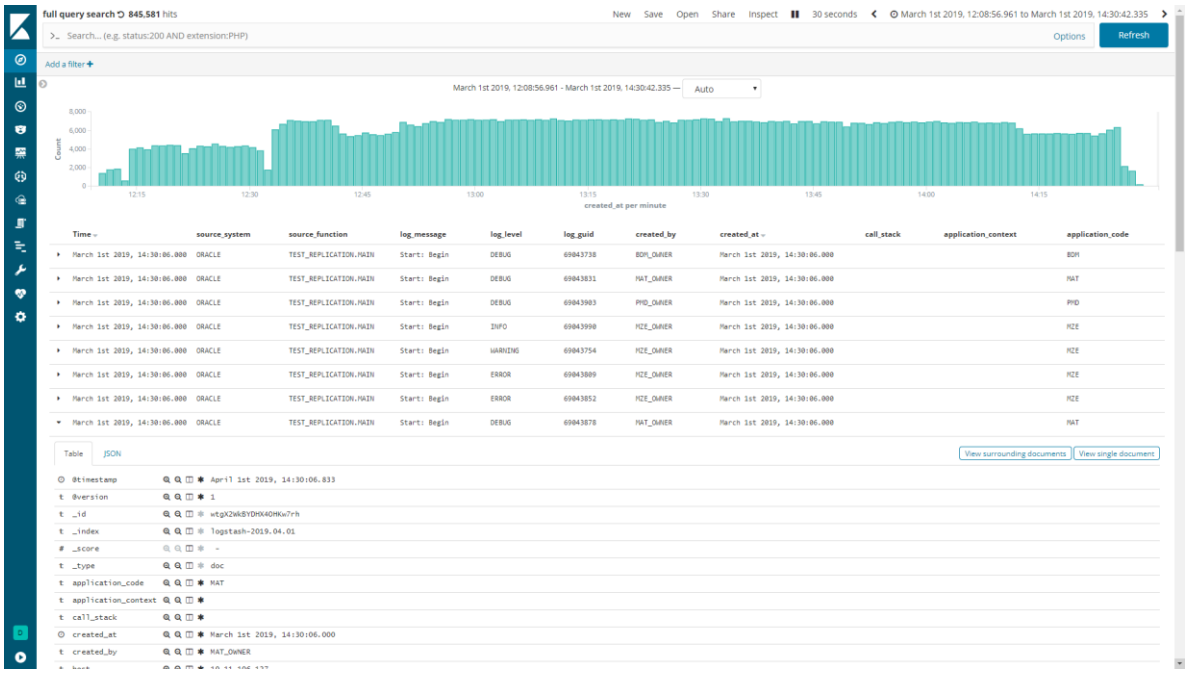


Figure 42. Kibana Discover page.

Kibana also allows the creations of custom visualizations, which let us view data on the Elasticsearch indices. Kibana visualizations are based on Elasticsearch queries. By using a series of Elasticsearch aggregations to extract and process the logs data, it is possible to create charts that show trends or spikes of the ams' applications[121]. One visualization that can be created in Kibana is a Data Table, which is a tabular form of data display, and, as an example for ams' use case, it was used to count how many logs exist for each `log_level` value, as seen in Figure 43.

Log level count

Log level	Count
DEBUG	1,231,574
FATAL_ERROR	507,994
INFO	270,421
WARNING	268,955
ERROR	267,691
	2,546,635

Figure 43. Kibana Data Table with log_level count

Kibana allows the combination of visualizations and searches into a dashboard. The content can be arranged, resized and shared. To demonstrate its utility, a dashboard was created for the Log Management System's application logs, seen in Figure 44.

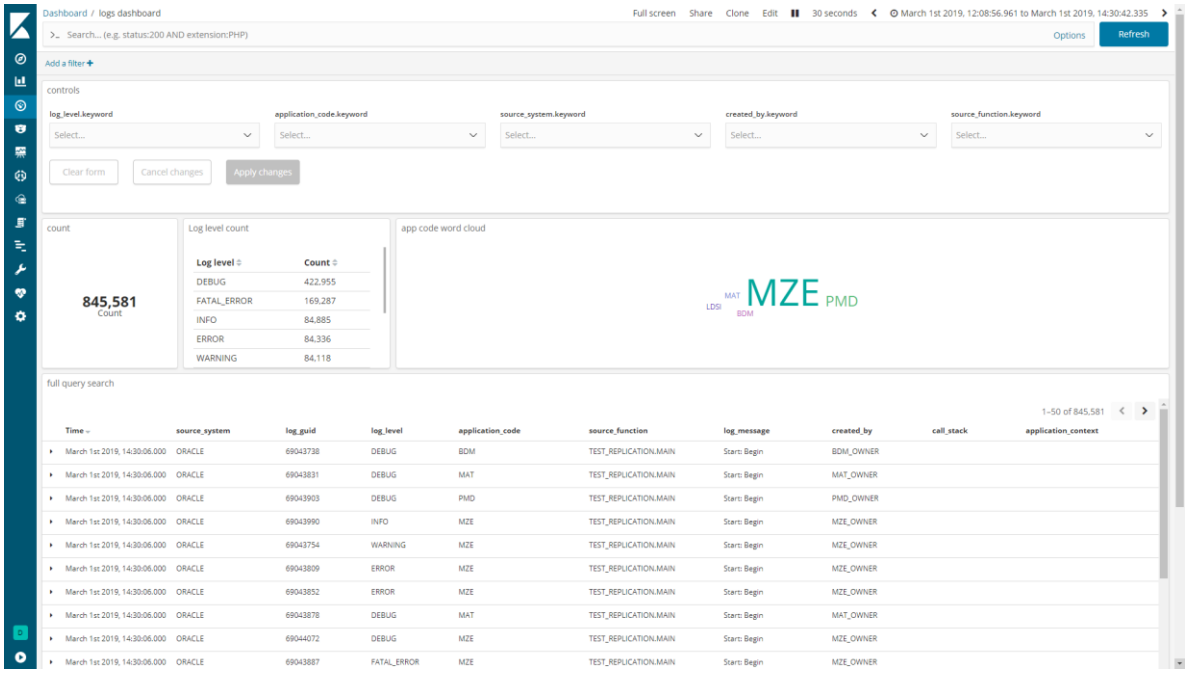


Figure 44. Kibana Dashboard for the Log Management System.

Besides additional visualizations, the above dashboard uses the Discovery page search and Data Table, demonstrated in Figure 42 and Figure 43, respectively.

The created dashboard can be easily embedded in web pages, by using a shareable link (see Figure 45), which can be obtained by clicking on the share button of the dashboard page (seen in Figure 46).

```
<iframe src="http://kibana_host:5601/goto/d7fd7fa0e60440499ca5bf71862a4437?embed=true" height="600" width="800"></iframe>
```

Figure 45. Kibana shareable iframe example of a dashboard.

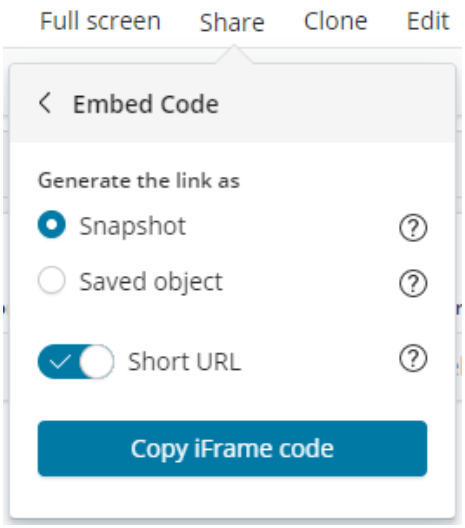


Figure 46. Kibana interface demonstration on how to obtain a shareable iFrame link

The dashboards can work as a monitoring tool for administrators or application managers, which can also integrate them with other monitoring/analysis tools.

4.2.7 Swagger

For the documentation of the REST and GraphQL API endpoints, the Swagger documentation tool was utilized (see section 3.4.1). The Swagger UI provides a display framework that reads an OpenAPI[20] specification document and generates an interactive documentation website. To code and test the OpenAPI specification document the Swagger Editor was used, as it dynamically validates the content and determines whether the specification being created is valid, as seen in Figure 47.

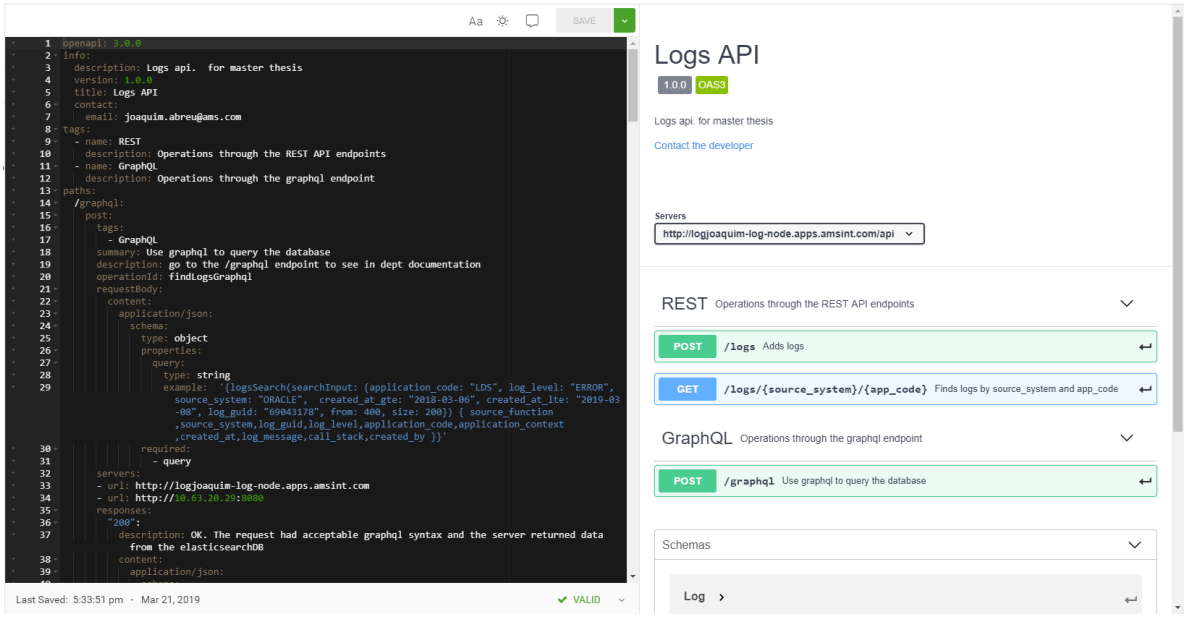


Figure 47. The Swagger editor validates the specification dynamically and shows the final result on the right

The resulting documentation website gives developers the full specifications on how to use the API and lets them test the endpoints on the platform, solving the **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirement. It also gives example values to experiment requests to the endpoints as well as providing a list of expected responses with the respective response code and description.

As for deployment, the Swagger documentation server was set to run as a Node.js server and its source code got saved in Github. Based on this Github repository, an Openshift(section 4.2.1) application was created, thus complying with the **R1: the system must be deployed in Openshift's container application platform** business restriction.

4.2.8 Log alerting job

The log alerting job, specified in **BR7.1: the system must alert each application stakeholder by email** and **BR7.2 the system must update the alerting method according to ams' defined heuristics** business requirements, was implemented and set to run periodically in the reserved Windows server. More specifically, it consists of a node.js cron job that gets executed on a 2 hour schedule. The reserved Windows server has access to the Elasticsearch endpoint, and the alerting job's task is to execute queries against the storage system to fetch data and understand if it is necessary to send any alert to the ams' applications managers.

To know which applications need to be checked for `ERROR` or `FATAL_ERROR` level logs and to whom to alert these errors, ams provided an Oracle view (see footnote 26),

which grants access to the applications that need to be verified and the respective representatives/managers email addresses to alert to. The `AIC_OWNER.TEST_LOG_THESIS_NOTIFI` view returns two columns, one with the application name and the other with an email address to send alerts to, as seen in Figure 48:

APP_CODE	EMAIL
LDS	joaquim.abreu@ams.com
LDS	sergio.viula@ams.com
LDSI	adriano.gama@ams.com
LDSI	joaquim.abreu@ams.com
MZE	joaquim.abreu@ams.com

Figure 48. The AIC_OWNER.TEST_LOG_THESIS_NOTIFI view returns application codes followed by emails to alert to.

To access the `AIC_OWNER.TEST_LOG_THESIS_NOTIFI` view data, such as the one seen in Figure 48 and to comply with the **BR7.2 the system must update the alerting method according to ams' defined heuristics** business requirement, the Log alerting job dynamically retrieves data from the Oracle view by executing PL/SQL statements, seen in Figure 49:

```
//app_codes will get all app_codes existent in the alerting table
    let app_codes = await connection.execute('SELECT DISTINCT APP_CODE FROM
AIC_OWNER.TEST_LOG_THESIS_NOTIFI')
    let app_emails = {}
      for(let row of app_codes.rows){
          let emails = await connection.execute(`SELECT EMAIL FROM
AIC_OWNER.TEST_LOG_THESIS_NOTIFI WHERE APP_CODE LIKE '${row.APP_CODE}'`)
          //saves emails per app_code
        app_emails[[row.APP_CODE]] = emails.rows
      }
```

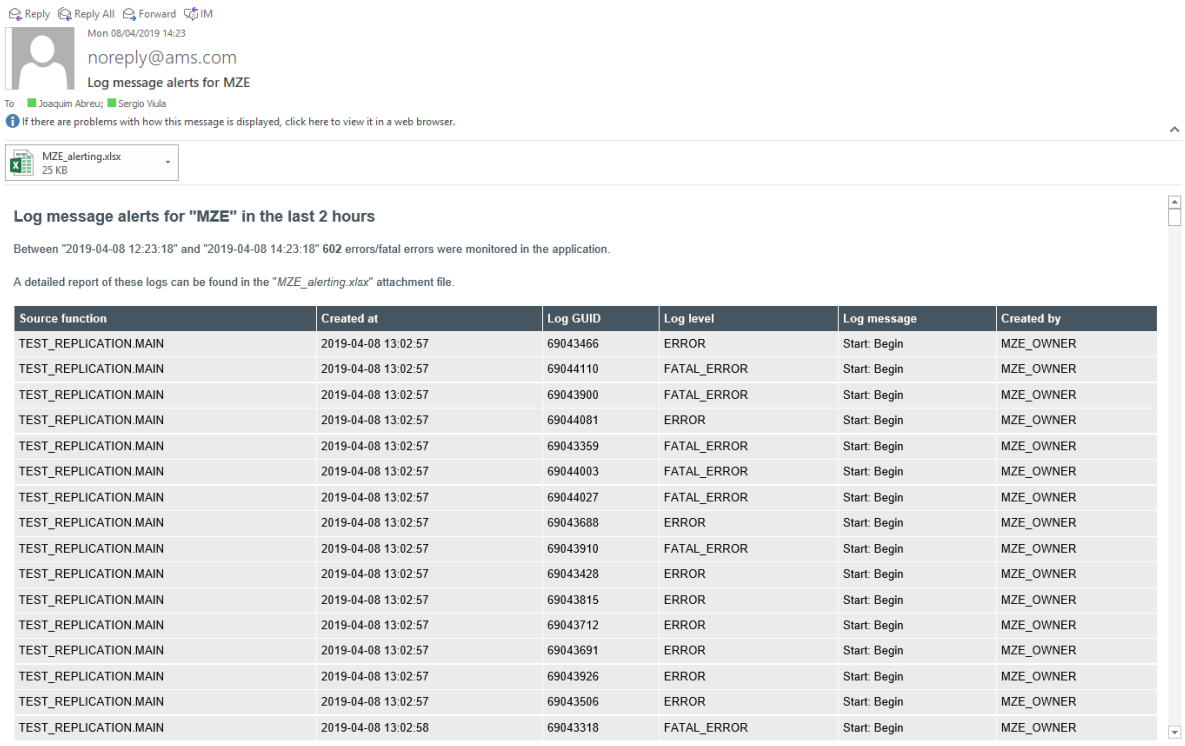
Figure 49. Node.js logic to dynamically obtain emails per application from the Oracle view.

For each `APP_CODE` retrieved by executing the PL/SQL query against the provided Oracle view (seen in Figure 49), the Log alerting job will create a request to the Elasticsearch endpoint, to understand if any `ERROR` or `FATAL_ERROR` level logs happened in the specified time window. Figure 50 shows the Elasticsearch query that is executed for each distinct `APP_CODE` value found in the `AIC_OWNER.TEST_LOG_THESIS_NOTIFI` view:

```
alertQuery = [...alertQuery, { "index": "logstash*" },
      { "query": { "bool": { "should": [{ "bool": { "must": [{ "range": { "created_at": {
"gte": `now-${gte_time}h` } } }, { "match": { "log_level": "ERROR" } }, { "match": {
"application_code": `${row.APP_CODE}` } }] } }, { "bool": { "must": [{ "range": {
"created_at": { "gte": `now-${gte_time}h` } } }, { "match": { "log_level": "FATAL_ERROR" }
}, { "match": { "application_code": `${row.APP_CODE}` } }] } }] } }, "_source": {
"includes": include_fields }, "size": "3000", "sort": { "created_at": { "order": "asc" } }
}]
```

Figure 50. Elasticsearch query that returns recent ERROR or FATAL_ERROR level logs for a specific application.

In case any `ERROR` or `FATAL_ERROR` are returned for the last `gte_time` hours (the default configuration is set to search for the last 2 hours), the Log alerting job's task is to alert by email the respective `APP_CODE` representatives on the matter. The email's body is an html page with a simple description of the `ERROR` and `FATAL_ERROR` logs that occurred in the predefined time window. The complete log descriptions are then attached in an `.xlsx` file and the email is sent to the ams' SMTP server, which then takes care of delivering the emails to the correct recipients. An example of an email sent by the Log alerting Job can be seen in Figure 51.


Reply Reply All Forward IM

Mon 08/04/2019 14:23

noreply@ams.com

Log message alerts for MZE

To Joaquim Abreu; Sergio Viula

If there are problems with how this message is displayed, click here to view it in a web browser.

MZE_alerting.xlsx
25 KB

Log message alerts for "MZE" in the last 2 hours

Between "2019-04-08 12:23:18" and "2019-04-08 14:23:18" **602** errors/fatal errors were monitored in the application.

A detailed report of these logs can be found in the "*MZE_alerting.xlsx*" attachment file.

Source function	Created at	Log GUID	Log level	Log message	Created by
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043466	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69044110	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043900	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69044081	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043359	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69044003	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69044027	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043688	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043910	FATAL_ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043428	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043815	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043712	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043691	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043926	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:57	69043506	ERROR	Start: Begin	MZE_OWNER
TEST_REPLICATION.MAIN	2019-04-08 13:02:58	69043318	FATAL_ERROR	Start: Begin	MZE_OWNER

Figure 51. Example of email sent to application managers alerting ERROR and FATAL_ERROR level logs.

Currently the log rotation job is a scheduled task that runs every two hours in the reserved windows server. The schedule can be dynamically changed by specifying a command line argument when starting the job, should the ams' administrators have the need to run the alerting job more or less often.

4.2.9 Log rotation job

The log rotation job, specified in the **BR6.1: the system must comply with ams' defined log persistence schedule** business requirement was implemented to address the requirements of each application regarding no longer needed/dated logs, meaning that logs older than a specified threshold are deleted from the Elasticsearch database.

Similarly to the Log alerting job (section 4.2.8), the log rotation job was also implemented in the reserved Windows server as a scheduled task, which is set to run once every 24 hours.

The log rotation job bases its operation according to ams' defined log persistence heuristics. This information is accessible through the `AIM_OWNER.AIM_V_LOG_LEVEL_RETENTIONS` Oracle view, which describes how long a specified application and log level must stay in the Elasticsearch database. As an example, Figure 52, shows the log persistence rules that must be followed in the log rotation job, the first row specifies that `DEBUG` level logs (`LOG_LEVEL`) from the LDS application (`CODE_SHORT`) must be erased if the Elasticsearch document `created_at` time value exceeds two days or 48 hours (`MAX_LOG_ENTRY_RETENTION`):

CODE_SHORT	LOG_LEVEL	MAX_LOG_ENTRY_RETENTION
LDS	DEBUG	2
LDSI	INFO	1
LDSI	WARNING	2
LDSI	ERROR	6
LDSI	FATAL_ERROR	20

Figure 52. The AIM_OWNER.AIM_V_LOG_LEVEL_RETENTIONS contains the rules that must be followed for the log retention job.

The logs erasure is only executed against the Elasticsearch endpoint, as this is the main storage system used to fetch data, and as such, needs to be kept clean of unnecessary logs to keep its performance up, as discussed in 4.1.1. The Log rotation job will not delete logs from Cassandra and Oracle, since these database systems will work as the permanent storage solutions for the Log Management System, complying with the **R2: the system must maintain an Oracle persistent database** restriction and **BR1: Database with performance for writings** business requirement.

For each rule present in the `AIM_OWNER.AIM_V_LOG_LEVEL_RETENTIONS` view, an Elasticsearch DSL delete query is executed against the Elasticsearch endpoint, as seen in Figure 53.

```
deleteQuery = [...deleteQuery, {
        'bool': {
          'must': [
          { 'range': { 'created_at': { 'lt': `now-${row.MAX_LOG_ENTRY_RETENTION}d` } } },
          { 'match': { 'log_level': row.LOG_LEVEL } },
          { 'match': { 'application_code': row.CODE_SHORT } }
         ]
        }
      }]
```

Figure 53. Elasticsearch delete statement that deletes logs according to its application retention policy.

It is important to notice that, if no rule is specified for a particular application and log level, the log data will stay in the Elasticsearch database indefinitely.

4.2.10 Summary

This section described the main aspects and thought process of the CLS implementation. The implementation of this system was accomplished taking into account the business requirements and restrictions imposed by ams, listed in 1.3.6 and 1.3.7 respectively, from the need to deploy the storage systems in a reserved Windows server, to making sure that the logs are being correctly stored according to the defined schema.

The CLS was installed on May 23rd, 2019, and, to test its correct functioning, the system received simulated log samples taken from production databases during three days (24th, 25th and 26th of May). The following section (section 4.3) details the tests made on the CLS after its deployment in the company's ecosystem.

4.3 Database Results

Once the deployment process (mentioned in chapter 4.2.1) was completed, insertion stress tests were performed from the 24th to the 26th of May, in order to prove the system's responsiveness to data inputs. The main goal of the tests was to analyze the write performance of the system in an environment as similar as possible to a production environment.

The simulated log entries were being forwarded from Funchal's ams application developers to the CLS's REST API entry point. Each entry in the API was an HTTP request which contained a log in the request body, similar to the one seen in Figure 31. All tests were executed inside ams' intranet, as this is a controlled environment which eliminates most of the noise on the results collected.

One simulation was run per day, and lasted around 1 hour. At the ended of the three day simulation, an introspection was made on Elasticsearch to verify the

throughput rate and indices size. As a result of the introspection, this data can be seen in Figure 54 and Figure 56, respectively.

```
health status index               uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   logstash-2019.05.26 zjrDVwvkTj-Nwa5rDdNQQA   5   1    1534464            0      227mb          227mb
yellow open   logstash-2019.05.25 VBSEhpLaTgatnUwQCei3Mg   5   1    1319004            0    197.9mb        197.9mb
yellow open   logstash-2019.05.24 4mgqMjHjRgmVIHPLhnbbYg   5   1    1242942            0    157.6mb        157.6mb
```

Figure 54. Elasticsearch indices introspection.

Figure 54 shows the database size and document count per index/day. On the 24th of May of 2019 a total of 1242942 documents were inserted in the system, which in store size equals to 157.6 MB of data. On the 25th of May of 2019 a total of 1319004 documents were inserted in the system, which in store size equals to 197.9 MB of data. Finally, on the 26th of May of 2019 a total of 1534464 documents were inserted in the system, which in store size equals to 227 MB.

```
Keyspace : jald
    Table: jald_logs
    SSTable count: 3
    Space used (live): 96.1 MiB
    Space used (total): 96.1 MiB
    Space used by snapshots (total): 23.9 MiB
    Off heap memory used (total): 1.44 MiB
    SSTable Compression Ratio: 0.25665508920626273
    Number of partitions (estimate): 1013896
    Local write latency: 0.085 ms
    Bloom filter space used: 1.19 MiB
    Bloom filter off heap memory used: 1.19 MiB
    Index summary off heap memory used: 213.91 KiB
    Compression metadata off heap memory used: 35.42 KiB
    Compacted partition minimum bytes: 87
    Compacted partition maximum bytes: 1331
    Compacted partition mean bytes: 321
```

Figure 55. Cassandra keyspace size on the 24th of May of 2019.

Figure 55 shows the Cassandra `jald` keyspace size on the 24th of May of 2019. As figure shows, for a total of 1242942 logs, the used space was of 96.1 MB. This metric is important to compare the data size between Cassandra and Elasticsearch storage systems.

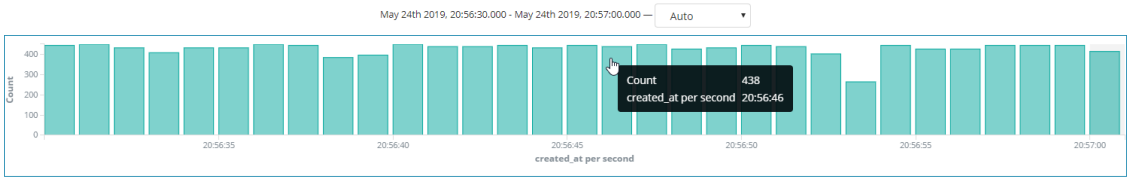


Figure 56. Elasticsearch throughput rate per second on the 24th of May of 2019.

From Figure 56's bar chart we can analyze the number of logs being inserted per second on the 24th of May of 2019. As the chart shows, an average of 438 logs were being indexed per second.

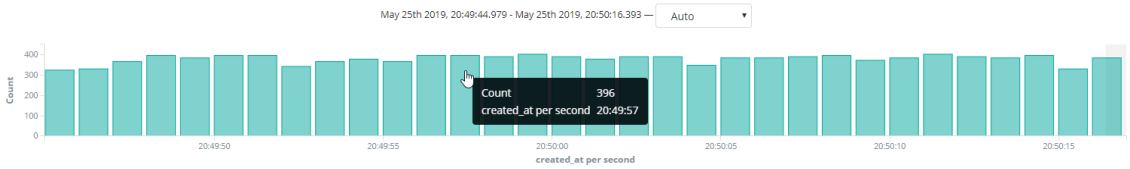


Figure 57. Elasticsearch throughput rate per second on the 25th of May of 2019.

From Figure 57's bar chart we can analyze the number of logs being inserted per second on the 26th of May of 2019. As the chart shows, an average of 396 logs were being indexed per second.

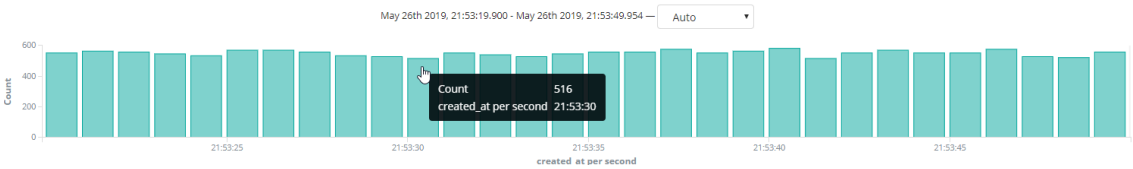


Figure 58. Elasticsearch throughput rate per second on the 26th of May of 2019.

Finally, from Figure 58's bar chart we can analyze the number of logs being inserted per second on the 26th of May of 2019. As the chart shows, an average of 516 logs were being indexed per second.These results will help us verify whether the decisions previously taken were the most appropriate and if the system can meet with the proposed business requirements.

5 Discussion

Considering the results presented before, it is now possible to discuss if the solution meets the requirements and evaluate if the decisions taken were the best. In this chapter we will discuss the developed solution, as well as expose a comparison between the results and the requirements (discussed in chapter 4.3).

5.1 Write performance

It is safe to assume that write operations will necessarily be the most common in the CLS, as logs will be constantly generated in the company's ecosystem. For this reason, it is important to evaluate how the system behaves when ingesting data and compare it to the current company requirements.

From the write performance results presented before (seen in Figure 56, Figure 57 and Figure 58), it is possible to observe that the CLS's input rate is higher than the rate stated in **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log** business requirement. This proves that the system can handle with a considerable amount of traffic without losing any logs in the process.

The amount of logs ingested in the simulation were also higher than the daily average logs ingested by the current system, thus reinforcing the **BR1: Database with performance for writings** business requirement.

5.2 Storage

Storing logs in a permanent and reliable manner was one of the main issues presented in this thesis scope, as stated in **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log**, **BR3: Validate the log inputs, checking if values are acceptable according to the defined schema**, and **BR6.1: the system must comply with ams' defined log persistence schedule** business requirements.

According to the results seen in Figure 54 and Figure 55, and comparing the total space used, we can confirm that Cassandra is the most suitable solution for permanent storage. The difference between the total space used in Elasticsearch and Cassandra for one day is of about 60 MB, meaning that Cassandra is better optimized for storing large amounts of data. Furthermore, this value is expected to grow once the proposed system is deployed into production, and receives logs every day.

The log rotation job (described in 4.2.9) solves the problems of having extra unnecessary space occupied by Elasticsearch's storage system, and resolves the **BR6:**

Implement a log rotation job to address the requirements of each application regarding no longer needed/dated logs business requirement.

The decision of storing the logs only on Elasticsearch would cost us an increase of needed storage space and lead to an eventual data loss (as stated in 4.1.1). Using Cassandra and Oracle as permanent storage solutions solved these issues, but also brought an increase in cost of maintenance of the system. Nonetheless, the decision to duplicate data was the most appropriate, as this approach complies with the **BR1: Database with performance for writings** and **BR2: Integration with a performant database for reads** business requirements, as well as the **R2: the system must maintain an Oracle persistent database** business restriction.

5.3 Log ingestion

Determining if logs are appropriate for ingestion and making sure they are properly inserted in the storage systems is essential for the CLS ingestion process (as seen in Figure 12).

The ingestion process was only made possible with the definition of the logs general schema, described in 4.2.2.1. If we defined a schema for each application it would make the whole system unfeasible and impractical, forcing the CLS users to know beforehand what logs and in what format they would be. Defining the general logs schema helped in specifying a universal log format for inserting and fetching documents.

This schema was useful for validating the logs being inserted in the REST API's entry point, and, if an error occurred, the system was capable of providing detailed information on what went wrong. These decisions ensured that the **BR3: Validate the log inputs, checking if values are acceptable according to the defined schema** and **BR4.2 the system must provide an API to insert and consume data** business requirements were being ensured by the CLS.

Besides the essential logs validation on insertion, the log ingestion process could not be made possible without the features of Logstash (see 4.2.5), the logs pipelining solution deployed in the project. It is important to report that during the tests described in section 4.3, no data was lost in the log ingestion process. If any error occurred in the ingestion process, Logstash would protect the system from data loss, as it has persistent queues enabled (see explanation in section 3.2.3.2). The persistent queues feature, also integrates well with the `http` input plugin used in the pipeline, due to its request-response protocol and acknowledgement capability.

In case there is a necessity of adding new fields to the logs schema, system maintainers would need a considerable effort to apply these changes. The new field would need to be added in the API server, Logstash and in the storage systems.

The reason for such maintenance effort is due to the nature of the CLS, which is comprised of several services, all detached from each other. Such issue could be solved with a common configurations service, which would be used by every other CLS service to fetch the current specification.

Overall, the log ingestion solution developed and deployed ensures that the **BR1.1: the system shall be able to handle 100 log inputs per second without losing any log**, **BR3.1: the system must specify a schema to be used by all log inputs** and **BR3.2 the system must ensure that the data is being correctly inserted according to the defined schema** business requirements are being dealt with.

5.4 Interfacing

Having in mind the many different user types that will eventually use the CLS, four different means of interacting with the project were developed: Kibana (see section 4.2.6), the REST and GraphQL APIs (see section 4.2.3 and 4.2.4, respectively) and the Swagger documentation server (see section 4.2.7).

The main objective of providing several interfacing solutions was to satisfy the needs of the many system's user types, from developers to system managers. Giving many solutions to interact with the logs is a beneficial feature, as each interfacing solution is optimized for a specific task. For example, if the goal is to provide a system manager with an overview of the logs that are entering the system, then a Kibana dashboard is the most appropriate solution. If, on the other hand, a developer wants to develop an application using the logs from the CLS, then it is best to take advantage of the Swagger documentation and the API server's functionalities, whether it is REST or GraphQL.

We can affirm that with the available interacting solutions it is possible to comply with the **BR4.3: the system must provide a visualization platform for the log data**, **BR4.1: the system must provide means to modify, visualize and access log data** and **BR5.1: the system must provide developers with documentation for the available API interfaces** business requirements.

5.5 Alerting

Even though it is not an essential piece to the CLS's core functioning, the logs alerting job (see section 4.2.8) main task is to assure system managers and developers that an issue is not forgotten. Currently it is alerting by email on logs that belong to ams'

defined heuristics, providing an `.xlsx` attachment document with the complete list of such logs. These features are in accordance with the **BR7: Implement a basic and extendable alerting mechanism** business requirement.

Keeping the alerting heuristics in ams' control, reduced the alerting job configurability and made it almost effortless for ams' system managers to configure new alerting rules without system downtime. As for the scheduling of the alerting job, even though it is possible to configure how often the job runs, it requires managers to reset the process and configure the new desired frequency.

The email alerting feature complies with the **BR7.1: the system must alert each application stakeholder by email** business requirement. Emails were used as it is the main method of exchanging messages in the company. The alerting job relies on the ams' SMTP server to send its emails to the respective application stakeholders. This dependency is expected to always be available, but in case some unexpected issue occurs between the SMTP server and the alerting job, the latter will become inoperative. Additional alerting notification options, such as Slack[122] or even a webhook[31] output would make the alerting job more flexible and fit with more use cases.

5.6 Summary

It is difficult to measure how well the system will cope in a fully in production environment. We considered that the results obtained were acceptable, as they showed that the CLS is already capable of handling the 100 logs per second minimum defined by ams. Also, and based on these findings, it is safe to say that the CLS complies with the business requirements and restrictions of this thesis use case.

The technologies initially proposed for the CLS were the ones used for its development. Many other alternatives could be used instead (as seen throughout section 3), but the ones proposed behaved as expected for the purposes they were assigned for.

The decision to use Elasticsearch as a database/search engine was due to its searching, analysis and scalability features, but also due to it being an unknown technology for both thesis involved parties. Through the development process and after gathering database results (section 4.3), Elasticsearch proved to be appropriate for the use case.

As for Node.js (section 3.2.2) being used as the runtime environment for the custom developed components in the CLS, it proved to be easy to integrate with other

[31] A Webhook is an HTTP callback that is triggered by an event[123], such as a rule from the logs alerting job.

applications, thanks to the NPM repository which provided the necessary project dependencies to work with. The only issues that emerged during the development with Node.js were in terms of code maintenance, where in some situations, due to its asynchronous nature and due to heavily relying on callbacks, the code could be difficult to understand and maintain. Despite this issue, Node.js brought more advantages than setbacks to the project development.

Overall, the decisions taken when developing the Centralized Log Management System ensure that the requirements and restrictions, listed in 1.3.6 and 1.3.7 respectively, are being met.

6 Conclusion

In this thesis we addressed the problem of decentralized and unstructured log data. For this we proposed a Centralized Log Management Solution. This system would standardize the collection, persistence and analysis of the company's applications logs.

In this chapter we aim to present the conclusions about this thesis, presenting an overview of all the process and providing some final thoughts on it.

6.1 Contributions

The main goal of the Centralized Log Management System was to explore and present to ams a new stack of technologies that can reliably ingest, manage and provide access to a high number of logs in a centralized architecture.

The CLS was made possible using a wide range of technologies, such as the Elastic Stack (introduced in 3.2.3), Node.js, Cassandra, Swagger and the company's Oracle Database System and its infrastructure.

As a result of this project, ams got a proof of concept of a centralized solution that aims to accelerate problem solving, automate error notifications, all while ensuring best practices and simplifying the various activities and tasks in the log management process. The CLS offers Elasticsearch's search engine that is exposed through Kibana and REST and GraphQL APIs, log persistence and duplication with Cassandra and Oracle storage systems, a reliable ingestion solution with Logstash and Alerting and Rotation Jobs for Error notifications and log archival, respectively.

The project was developed taking into account the business requirements and restrictions set by ams (section 1.3.6 and 1.3.7, respectively), which had as key requirements the ingestion of logs from several company applications and the exposure of such logs through an Interfacing system.

The CLS was deployed in ams internal environment through Openshift, which contained a Swagger documentation endpoint that can be delivered to future developers and will allow them to use the system.

6.2 Future work

In order to improve the quality of the CLS, it is expected, as future work, to add features that will turn the system more robust and secure, helping in making the

transition from the current company's unstructured system to the designed centralized solution.

Future work will include the definition of a security concept for secure communication with the logging server, should the service be accessed from outside the intranet.

Keeping the technologies stack updated, in order to benefit from their latest features and security constraints is also important, as, during the project's development, new Elasticsearch functionalities became available, which would have facilitated the log rotation task.

Future work will also include the synchronization of the database systems, as well making Cassandra and Oracle databases searchable through the API server. There are also plans to add the feature of logs exporting, such as exporting data in `.csv` format.

6.3 Lessons learned

This was the first experience in a professional environment, and, overall it was very satisfying and allowed me to put in practice many of the knowledge acquired during my academic course at the University of Madeira. The experience also expanded my knowledge, as many of the technologies and architectural decisions applied in the project were not part of the undergraduate and master's education program.

During the development process, some issues related to the architectural design occurred, but the support received from my advisor Prof. Filipe Quintal and ams, particularly from my IT supervisor was essential to overcome these issues and develop the project according to the business requirements and restrictions.

After concluding the Centralized Log Management System's project, I feel more prepared to face new Software development challenges and got a better sense of the business procedures when developing a product in a professional environment, from the weekly reunions with supervisors to the development restrictions of the technologies.

References

[1] «Log rotation». [Online]. Disponível em: https://www.cl.cam.ac.uk/~jw35/courses/apache/html/x1670.htm. [Acedido: 04-Ago-2020]

[2] K. Kent e M. P. Souppaya, «Guide to computer security log management», National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-92, 2006 [Online]. Disponível em: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf. [Acedido: 04-Fev-2019]

[3] J. J Bernard, *Handbook of Research on Web Log Analysis*. IGI Global, 2008.

[4] «Batch-oriented», *TheFreeDictionary.com*. [Online]. Disponível em: https://encyclopedia2.thefreedictionary.com/Batch-oriented. [Acedido: 13-Mar-2019]

[5] «The History of Monitoring Tools - Sumo Logic Blog», *Sumo Logic*, 02-Abr-2018. [Online]. Disponível em: https://www.sumologic.com/blog/devops/monitoring-tools-history/. [Acedido: 13-Mar-2019]

[6] «UNIX Tutorial - Introduction». [Online]. Disponível em: http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html. [Acedido: 26-Mar-2019]

[7] «About Unix». [Online]. Disponível em: https://kb.iu.edu/d/agat. [Acedido: 26-Mar-2019]

[8] «How Log Analysis Has Evolved», *Sumo Logic*, 02-Abr-2018. [Online]. Disponível em: https://www.sumologic.com/blog/log-management-analysis/log-analysis-evolution/. [Acedido: 13-Mar-2019]

[9] «Brief introduction of log file formats». [Online]. Disponível em: https://www.loganalyzer.net/log-analysis/log-file-format.html. [Acedido: 13-Mar-2019]

[10] «Logging vs Tracing vs Monitoring», 30-Nov-2017. [Online]. Disponível em: https://WinderResearch.com/logging-vs-tracing-vs-monitoring/. [Acedido: 04-Fev-2019]

[11] V.-P. Ranganath, «Logging, Monitoring, and Observability», *Venkatesh-Prasad Ranganath*, 02-Jul-2018. [Online]. Disponível em: https://medium.com/@rvprasad/logging-monitoring-and-observability-219c043b5c81. [Acedido: 04-Fev-2019]

[12] «Syslog Tutorial: How It Works, Examples, Best Practices, and More», *Stackify*, 30-Jun-2017. [Online]. Disponível em: https://stackify.com/syslog-101/. [Acedido: 14-Mar-2019]

[13] «What is distributed tracing? - Definition from WhatIs.com», *SearchITOperations*. [Online]. Disponível em: https://searchitoperations.techtarget.com/definition/distributed-tracing. [Acedido: 08-Fev-2019]

[14] «Logging is not Tracing». [Online]. Disponível em: http://geekswithblogs.net/akraus1/archive/2009/06/21/132968.aspx. [Acedido: 04-Fev-2019]

[15] «1b24d4fa-08e0-265d-c5df-dd2e5430a672.pdf». [Online]. Disponível em: https://ams.com/documents/20143/83556/Roadshow+presentation+ams+Q2+2018+July+2018+f.pdf/1b24d4fa-08e0-265d-c5df-dd2e5430a672. [Acedido: 17-Out-2018]

[16] «ams Start - ams». [Online]. Disponível em: https://ams.com/. [Acedido: 15-Out-2018]

[17] «Distributed Logging Architecture for Microservices - DZone Cloud», *dzone.com*. [Online]. Disponível em: https://dzone.com/articles/distributed-logging-architecture-for-microservices. [Acedido: 05-Fev-2019]

[18] «Cloud Data Archiving | Long-term Object Storage | Amazon Glacier», *Amazon Web Services, Inc.* [Online]. Disponível em: https://aws.amazon.com/glacier/. [Acedido: 11-Fev-2019]

[19] «Centralized Logging Architecture ·». [Online]. Disponível em: http://jasonwilder.com/blog/2013/07/16/centralized-logging-architecture/. [Acedido: 08-Fev-2019]
[20] P. Zaitsev, «Why MySQL Could Be Slow With Large Tables?», *Percona Database Performance Blog*, 09-Jun-2006. [Online]. Disponível em: https://www.percona.com/blog/2006/06/09/why-mysql-could-be-slow-with-large-tables/. [Acedido: 10-Set-2019]
[21] «Building a centralized logging system. Choosing the architecture. · DevOps experience». [Online]. Disponível em: http://artemstar.com/2017/03/11/logging-system-pipeline/. [Acedido: 15-Mar-2019]
[22] «Alerts», *Log Analysis | Log Monitoring by Loggly*. [Online]. Disponível em: https://www.loggly.com/docs/adding-alerts/. [Acedido: 15-Mar-2019]
[23] «Alerting & Monitoring», *Log Analysis | Log Monitoring by Loggly*. [Online]. Disponível em: https://www.loggly.com/docs/alerts-overview/. [Acedido: 15-Mar-2019]
[24] «Europe: Ireland Pricing», *Sumo Logic*. [Online]. Disponível em: https://www.sumologic.com/pricing/europe/. [Acedido: 10-Set-2019]
[25] «Why is centralized logging so expensive? - Quora». [Online]. Disponível em: https://www.quora.com/Why-is-centralized-logging-so-expensive. [Acedido: 15-Mar-2019]
[26] A. Mavin, P. Wilkinson, A. Harwood, e M. Novak, «Easy Approach to Requirements Syntax (EARS)», em *2009 17th IEEE International Requirements Engineering Conference*, 2009, pp. 317–322, doi: 10.1109/RE.2009.9.
[27] K. Wiegers e J. Beatty, *Software Requirements*. Pearson Education, 2013.
[28] O. Lomič, «Log management in distributed environment», p. 57, 2017.
[29] «Log Analysis | Log Management by Loggly», *Log Analysis | Log Monitoring by Loggly*. [Online]. Disponível em: https://www.loggly.com/. [Acedido: 27-Mar-2019]
[30] «TIBCO LogLogic® Log Management Intelligence | TIBCO Software». [Online]. Disponível em: https://www.tibco.com/resources/datasheet/tibco-loglogic-log-management-intelligence. [Acedido: 04-Fev-2019]
[31] «Logging as a Service Helps You Focus on the Big Picture», *IT Infrastructure Advice, Discussion, Community - Network Computing*, 11-Ago-2017. [Online]. Disponível em: https://www.networkcomputing.com/cloud-infrastructure/logging-service-helps-you-focus-big-picture. [Acedido: 04-Fev-2019]
[32] A. W. S. Startups, «On the Importance of Logs», *AWS Startup Collection*, 14-Mai-2014. [Online]. Disponível em: https://medium.com/aws-activate-startup-blog/on-the-importance-of-logs-72afccab49d7. [Acedido: 04-Fev-2019]
[33] «SIEM, AIOps, Application Management, Log Management, Machine Learning, and Compliance», *Splunk*. [Online]. Disponível em: https://www.splunk.com. [Acedido: 27-Mar-2019]
[34] Datadog, «Datadog log management & analytics | Datadog», *Datadog log management & analytics*. [Online]. Disponível em: https://www.datadoghq.com/log-management/. [Acedido: 27-Mar-2019]
[35] «Logz.io: Secure & Scalable Log Management with Cloud-Based ELK», *Logz.io*. [Online]. Disponível em: https://logz.io/. [Acedido: 27-Mar-2019]
[36] «ELK Stack: Elasticsearch, Logstash, Kibana | Elastic». [Online]. Disponível em: https://www.elastic.co/elk-stack. [Acedido: 27-Mar-2019]
[37] «JSON». [Online]. Disponível em: https://www.json.org/. [Acedido: 28-Mar-2019]
[38] «Amazon Web Services (AWS) - Cloud Computing Services», *Amazon Web Services, Inc.* [Online]. Disponível em: https://aws.amazon.com/. [Acedido: 23-Abr-2019]
[39] «Logging Setup», *Log Analysis | Log Monitoring by Loggly*. [Online]. Disponível em: https://www.loggly.com/docs/logging-setup/. [Acedido: 28-Mar-2019]

[40] Datadog, «Real time Interactive Dashboards | Datadog», *Real time Interactive Dashboards*. [Online]. Disponível em: https://www.datadoghq.com/product/platform/dashboards/. [Acedido: 03-Ago-2020]
[41] «Cloud-based service and log management for collecting, analyzing and visualizing machine data | Splunk Cloud», *Splunk*. [Online]. Disponível em: https://www.splunk.com/en_us/software/splunk-cloud.html. [Acedido: 27-Mar-2019]
[42] V. Tiwari, «Part 1: Building a Centralized Logging Application», *Hacker Noon*, 21-Jan-2018. [Online]. Disponível em: https://hackernoon.com/part-1-building-a-centralized-logging-application-5a537033da0a. [Acedido: 29-Out-2018]
[43] «SQL vs. NoSQL: What's the difference?», *Hiring | Upwork*, 09-Mai-2016. [Online]. Disponível em: https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/. [Acedido: 24-Abr-2019]
[44] «What is a Relational Database? – Amazon Web Services (AWS)», *Amazon Web Services, Inc.* [Online]. Disponível em: https://aws.amazon.com/relational-database/. [Acedido: 24-Abr-2019]
[45] «Imperative vs Declarative Programming», *TylerMcGinnis.com*. [Online]. Disponível em: https://tylermcginnis.com/imperative-vs-declarative-programming/. [Acedido: 24-Abr-2019]
[46] «What is Oracle Database (Oracle DB)? - Definition from Techopedia», *Techopedia.com*. [Online]. Disponível em: https://www.techopedia.com/definition/8711/oracle-database. [Acedido: 26-Abr-2019]
[47] «What is Oracle database (RDBMS)? | DaDBm». [Online]. Disponível em: http://www.dadbm.com/what-is-oracle-database-rdbms/. [Acedido: 26-Abr-2019]
[48] «custstory-cocacola-en-177613.pdf». [Online]. Disponível em: http://www.oracle.com/us/solutions/custstory-cocacola-en-177613.pdf. [Acedido: 29-Abr-2019]
[49] «Database Concepts». [Online]. Disponível em: https://docs.oracle.com/cd/E11882_01/server.112/e40540/glossary.htm#CBAIDJHG. [Acedido: 26-Abr-2019]
[50] «Database Concepts». [Online]. Disponível em: https://docs.oracle.com/database/121/CNCPT/intro.htm#CNCPT001. [Acedido: 29-Abr-2019]
[51] «MySQL». [Online]. Disponível em: https://www.mysql.com/. [Acedido: 02-Mai-2019]
[52] «MySQL :: MySQL 8.0 Reference Manual :: 1.3.1 What is MySQL?» [Online]. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. [Acedido: 30-Abr-2019]
[53] «What is MySQL? - Definition from WhatIs.com», *SearchOracle*. [Online]. Disponível em: https://searchoracle.techtarget.com/definition/MySQL. [Acedido: 30-Abr-2019]
[54] «DuBois_-_MySQL._4Ed.pdf». [Online]. Disponível em: https://www.e-reading.club/bookreader.php/142091/DuBois_-_MySQL._4Ed.pdf. [Acedido: 02-Mai-2019]
[55] «MySQL vs Oracle - Top 7 Most valuable Differences To Learn», *EDUCBA*, 15-Set-2018. [Online]. Disponível em: https://www.educba.com/mysql-vs-oracle/. [Acedido: 02-Mai-2019]
[56] A. B. M. Moniruzzaman e S. A. Hossain, «NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison», *International Journal of Database Theory and Application*, vol. 6, n. 4, p. 14, 2013.
[57] «What is vertical scalability (scaling up)? - Definition from WhatIs.com», *SearchCIO*. [Online]. Disponível em: https://searchcio.techtarget.com/definition/vertical-scalability. [Acedido: 11-Abr-2019]
[58] Xplenty, «The SQL vs NoSQL Difference: MySQL vs MongoDB», *Xplenty Blog*, 28-Set-2017. [Online]. Disponível em: https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2. [Acedido: 11-Abr-2019]

[59] O. Kononenko, O. Baysal, R. Holmes, e M. W. Godfrey, «Mining modern repositories with elasticsearch», em *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, Hyderabad, India, 2014, pp. 328–331, doi: 10.1145/2597073.2597091 [Online]. Disponível em: http://dl.acm.org/citation.cfm?doid=2597073.2597091. [Acedido: 22-Out-2018]
[60] «Basic Concepts | Elasticsearch Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/elasticsearch/reference/6.5/getting-started-concepts.html. [Acedido: 15-Abr-2019]
[61] R. Kuć e M. Rogoziński, *Mastering Elasticsearch - Second Edition*. Packt Publishing Ltd, 2015.
[62] «Basic Concepts - Lucene Tutorial.com». [Online]. Disponível em: http://www.lucenetutorial.com/basic-concepts.html. [Acedido: 15-Abr-2019]
[63] M. S. Divya e S. K. Goyal, «An advanced and quick search technique to handle voluminous data», p. 5.
[64] «What is multi-tenancy? - Definition from WhatIs.com», *WhatIs.com*. [Online]. Disponível em: https://whatis.techtarget.com/definition/multi-tenancy. [Acedido: 16-Abr-2019]
[65] «How open source technologies are transforming the BBC», *Information Age*, 02-Set-2016. [Online]. Disponível em: https://www.information-age.com/open-source-technologies-transforming-bbc-123462182/. [Acedido: 06-Mai-2019]
[66] «What is Apache Cassandra», *DataStax Academy: Free Cassandra Tutorials and Training*. [Online]. Disponível em: https://academy.datastax.com/planet-cassandra/what-is-apache-cassandra. [Acedido: 29-Out-2018]
[67] «Introduction to Apache Cassandra's Architecture - DZone Database», *dzone.com*. [Online]. Disponível em: https://dzone.com/articles/introduction-apache-cassandras. [Acedido: 16-Nov-2018]
[68] M. Brown, *Learning Apache Cassandra*. Packt Publishing Ltd, 2015.
[69] Adrian Cockcroft, «Migrating Netflix from Datacenter Oracle to Global Cassandra», 18:16:59 UTC [Online]. Disponível em: https://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra/8-Data_Center_Ne8lix_could_not. [Acedido: 03-Mai-2019]
[70] «[Analysis & Comparison of Top 15] Time Series Database Explained», *InfluxData*. [Online]. Disponível em: https://www.influxdata.com/time-series-database/. [Acedido: 30-Out-2018]
[71] «InfluxData Documentation». [Online]. Disponível em: https://docs.influxdata.com/. [Acedido: 30-Out-2018]
[72] 02 Aug 2017 Justin W. Flory Feed 230up, «How time-series databases help make sense of sensors», *Opensource.com*. [Online]. Disponível em: https://opensource.com/article/17/8/influxdb-time-series-database-stack. [Acedido: 07-Mai-2019]
[73] «Time-Series Databases and InfluxDB», *Time-Series Databases and InfluxDB*. [Online]. Disponível em: https://www.xaprb.com/blog/2014/03/02/time-series-databases-influxdb/. [Acedido: 07-Mai-2019]
[74] S. V. B, «Introduction to MongoDB!», *Sai Vittal B*, 29-Out-2018. [Online]. Disponível em: https://medium.com/@saivittalb/introduction-to-mongodb-859ed4426994. [Acedido: 03-Mai-2019]
[75] B. Systems, «Moving from MongoDB Full-Text Search to ElasticSearch (for Node.Js Applications)», *Bitcom Systems*. [Online]. Disponível em: http://bitcom.systems/blog/moving-mongo-to-elasticsearch/. [Acedido: 07-Mai-2019]
[76] «Cassandra vs MongoDB in 2018». [Online]. Disponível em: https://blog.panoply.io/cassandra-vs-mongodb. [Acedido: 07-Mai-2019]

[77] G. B, «What is Apache? An In-Depth Overview of Apache Web Server», *Hostinger Tutorials*, 20-Jun-2018. [Online]. Disponível em: https://www.hostinger.in/tutorials/what-is-apache. [Acedido: 26-Set-2019]
[78] N. js Foundation, «About», *Node.js*. [Online]. Disponível em: https://nodejs.org/en/about/. [Acedido: 09-Mai-2019]
[79] «Introduction to Node.js», *Introduction to Node.js*. [Online]. Disponível em: https://nodejs.dev/. [Acedido: 09-Mai-2019]
[80] U. Elahi, «Elastic Stack — A Brief Introduction», *Hacker Noon*, 18-Jul-2018. [Online]. Disponível em: https://hackernoon.com/elastic-stack-a-brief-introduction-794bc7ff7d4f. [Acedido: 29-Mar-2019]
[81] A. Jain, «Full Stack Geek - For WebDevs: Introduction to Elasticsearch and the ELK stack», *Full Stack Geek - For WebDevs*, 23-Mar-2019. [Online]. Disponível em: https://fullstackgeek.blogspot.com/2019/03/introduction-to-elasticsearch-and-elk-stack.html. [Acedido: 10-Mai-2019]
[82] «Logstash Introduction | Logstash Reference [7.0] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/logstash/7.0/introduction.html. [Acedido: 10-Mai-2019]
[83] «What is message queuing? - CloudAMQP». [Online]. Disponível em: https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html. [Acedido: 10-Dez-2018]
[84] «Persistent Queues | Logstash Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/logstash/current/persistent-queues.html#durability-persistent-queues. [Acedido: 21-Jan-2019]
[85] «REST API | Kibana User Guide [master] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/kibana/master/api.html. [Acedido: 31-Jan-2019]
[86] «What is Kibana? – Amazon Web Services», *Amazon Web Services, Inc.* [Online]. Disponível em: https://aws.amazon.com/elasticsearch-service/kibana/. [Acedido: 10-Mai-2019]
[87] R. Khanna, «How to use Elasticsearch, Logstash and Kibana to visualise logs in Python in realtime», *freeCodeCamp.org*, 29-Jan-2019. [Online]. Disponível em: https://medium.freecodecamp.org/how-to-use-elasticsearch-logstash-and-kibana-to-visualise-logs-in-python-in-realtime-acaab281c9de. [Acedido: 10-Mai-2019]
[88] L. Musthaler, «Loggly aims to reveal what matters in log data», *Network World*, 21-Out-2016. [Online]. Disponível em: https://www.networkworld.com/article/3134020/loggly-aims-to-reveal-what-matters-in-log-data.html. [Acedido: 27-Set-2019]
[89] «What is RESTful API? - Definition from WhatIs.com», *SearchMicroservices*. [Online]. Disponível em: https://searchmicroservices.techtarget.com/definition/RESTful-API. [Acedido: 25-Set-2018]
[90] «GraphQL: A query language for APIs.» [Online]. Disponível em: http://graphql.org/. [Acedido: 26-Set-2018]
[91] «Learn GraphQL Fundamentals with Fullstack Tutorial». [Online]. Disponível em: https://www.howtographql.com/basics/0-introduction/. [Acedido: 15-Out-2018]
[92] «GraphQL vs REST - A comparison». [Online]. Disponível em: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/. [Acedido: 10-Mai-2019]
[93] S. Stubailo, «GraphQL vs. REST», *Medium*, 06-Abr-2018. [Online]. Disponível em: https://blog.apollographql.com/graphql-vs-rest-5d425123e34b. [Acedido: 02-Out-2019]
[94] «OpenAPI Specification | Swagger». [Online]. Disponível em: https://swagger.io/specification/. [Acedido: 15-Abr-2019]
[95] «The Best APIs are Built with Swagger Tools | Swagger». [Online]. Disponível em: https://swagger.io/. [Acedido: 15-Abr-2019]
[96] «Tools to Develop, Design & Document APIs | Swagger». [Online]. Disponível em: https://swagger.io/tools/swaggerhub/. [Acedido: 15-Abr-2019]

[97] «Best Oracle Developer and Administrator Database Tools | Free Trial». [Online]. Disponível em: https://www.quest.com/products/toad-for-oracle/. [Acedido: 13-Fev-2020]
[98] «Introduction to Postman for API Development», *GeeksforGeeks*, 28-Mai-2018. [Online]. Disponível em: https://www.geeksforgeeks.org/introduction-postman-api-development/. [Acedido: 02-Out-2019]
[99] «What is PaaS? Platform as a Service | Microsoft Azure». [Online]. Disponível em: https://azure.microsoft.com/en-us/overview/what-is-paas/. [Acedido: 23-Abr-2019]
[100] «What is Kubernetes». [Online]. Disponível em: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Acedido: 10-Mai-2019]
[101] «What is Red Hat OpenShift? - Definition from WhatIs.com», *SearchCloudComputing*. [Online]. Disponível em: https://searchcloudcomputing.techtarget.com/definition/Red-Hat-OpenShift. [Acedido: 10-Mai-2019]
[102] «What is Microsoft Azure Platform as a Service (PaaS)?», *SherWeb*, 06-Abr-2018. [Online]. Disponível em: https://www.sherweb.com/blog/cloud-server/what-is-azure-paas/. [Acedido: 03-Out-2019]
[103] «Elasticsearch as a primary database», *Discuss the Elastic Stack*, 16-Mai-2017. [Online]. Disponível em: https://discuss.elastic.co/t/elasticsearch-as-a-primary-database/85733/13. [Acedido: 23-Mai-2019]
[104] «Elasticsearch as a NoSQL Database», *Elastic Blog*, 15-Set-2013. [Online]. Disponível em: /blog/found-elasticsearch-as-nosql. [Acedido: 23-Mai-2019]
[105] «What are Benefits of Message Queues? – AWS», *Amazon Web Services, Inc.* [Online]. Disponível em: https://aws.amazon.com/message-queue/benefits/. [Acedido: 16-Jul-2019]
[106] F. Doglio, *REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development*. Apress, 2018.
[107] 01 Nov 2018 Ricardo GerardiFeed 119up 5 comments, «Getting started with OKD on your Linux desktop», *Opensource.com*. [Online]. Disponível em: https://opensource.com/article/18/11/local-okd-cluster-linux. [Acedido: 25-Jul-2019]
[108] «Using Triggers». [Online]. Disponível em: https://docs.oracle.com/database/121/TDDDG/tdddg_triggers.htm#TDDDG51000. [Acedido: 24-Jun-2019]
[109] «cqlsh». [Online]. Disponível em: https://docs.datastax.com/en/archived/cql/3.1/cql/cql_reference/cqlsh.html. [Acedido: 10-Jan-2019]
[110] «Configuring Elasticsearch | Elasticsearch Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/elasticsearch/reference/6.5/settings.html. [Acedido: 27-Jun-2019]
[111] P. Ragone, «Scaling Elasticsearch», *hipages Engineering*, 12-Jan-2017. [Online]. Disponível em: https://medium.com/hipages-engineering/scaling-elasticsearch-b63fa400ee9e. [Acedido: 28-Jan-2019]
[112] «Express - Node.js web application framework». [Online]. Disponível em: https://expressjs.com/. [Acedido: 28-Out-2019]
[113] «5. Separation of Concerns - Programming JavaScript Applications [Book]». [Online]. Disponível em: https://www.oreilly.com/library/view/programming-javascript-applications/9781491950289/ch05.html. [Acedido: 28-Out-2019]
[114]: *office: Object schema validation. Contribute to hapijs/joi development by creating an account on GitHub*. hapi.js, 2019 [Online]. Disponível em: https://github.com/hapijs/joi. [Acedido: 08-Jul-2019]

[115] «Http input plugin | Logstash Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/logstash/6.5/plugins-inputs-http.html. [Acedido: 18-Jul-2019]
[116] «Clone filter plugin | Logstash Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/logstash/6.5/plugins-filters-clone.html#_description_108. [Acedido: 19-Jul-2019]
[117] «Elasticsearch output plugin | Logstash Reference [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/logstash/6.5/plugins-outputs-elasticsearch.html. [Acedido: 22-Jul-2019]
[118] Karl, *JDBC output for Logstash. Contribute to theangryangel/logstash-output-jdbc development by creating an account on GitHub*. 2019 [Online]. Disponível em: https://github.com/theangryangel/logstash-output-jdbc. [Acedido: 22-Jul-2019]
[119] *Logstash output plug-in for cassandra. Contribute to PerimeterX/logstash-output-cassandra development by creating an account on GitHub*. PerimeterX, 2019 [Online]. Disponível em: https://github.com/PerimeterX/logstash-output-cassandra. [Acedido: 22-Jul-2019]
[120] «Discover | Kibana User Guide [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/kibana/6.5/discover.html. [Acedido: 24-Jul-2019]
[121] «Visualize | Kibana User Guide [6.5] | Elastic». [Online]. Disponível em: https://www.elastic.co/guide/en/kibana/6.5/visualize.html. [Acedido: 24-Jul-2019]
[122] Slack, «Where work happens», *Slack*. [Online]. Disponível em: https://slack.com/. [Acedido: 12-Fev-2020]
[123] Zapier, «What Are Webhooks?», *Zapier*, 29-Mar-2018. [Online]. Disponível em: https://zapier.com/blog/what-are-webhooks/. [Acedido: 12-Fev-2020]
[124] «Deep Diving into cassandra-stress (Part 1)», *Instaclustr*, 19-Ago-2016. [Online]. Disponível em: https://www.instaclustr.com/deep-diving-into-cassandra-stress-part-1/. [Acedido: 26-Nov-2018]
[125] «MS SQL/Oracle Database Benchmark & Performance Testing Tool». [Online]. Disponível em: https://www.quest.com/products/benchmark-factory/. [Acedido: 13-Fev-2020]
[126] «Database Net Services Administrator's Guide». [Online]. Disponível em: https://docs.oracle.com/cd/B28359_01/network.111/b28316/glossary.htm#BGBCJBBD. [Acedido: 15-Jan-2019]
[127] phpMyAdmin contributors, «phpMyAdmin», *phpMyAdmin*. [Online]. Disponível em: https://www.phpmyadmin.net/. [Acedido: 13-Fev-2020]
[128] «MySQL :: MySQL 8.0 Reference Manual :: 4.5.8 mysqlslap — Load Emulation Client». [Online]. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html. [Acedido: 28-Nov-2018]

Appendix

A. Database Benchmark

In order to choose a good database technology, it's important to evaluate how they perform under high stress loads and compare it with other DBMS.

These tests are particularly important in the scenario we are working, since a centralized logging system as the one proposed is expected to receive large amounts of data at a high throughput rate and, at the same time, work as an easy and performant search tool.

These tests should be performed with data that is as close to reality as possible, and in a controlled environment to prevent misleading results and achieve a conclusive outcome.

In this section an assessment of the performance of Apache Cassandra and Oracle databases will be conducted in a closed environment to determine their read and write capabilities in production situations.

A 1. Introduction

One of the requirements of ams is to have a database with good performance for writing and also be one that does not lose data. The discussed results are obtained from a benchmark between the Cassandra Database and Oracle. The choice for testing these two databases, is to verify the performance of the company's database of choice and to compare it with a non-relational database that is known for its high write throughput and scalability.

The purpose of testing a Database technology that is not being used by the company is to verify if there is a better alternative to the Oracle Database. At the end of the analysis, we will be able to decide which storage management system is a better fit for the proposed log framework system.

Since each system uses a query language like SQL, the queries made in each database are similar. Each database was configured in a separate environment. A new environment had to be configured for the case of Cassandra's, on the other hand an Oracle Database environment was already set up by AMS.

A MySQL database benchmark will also be reviewed, working as a reference point to the overall benchmark.

A 2. Environment

Both database technologies needed to be tested under conditions that were as close as possible to one another.

One point to mention is that each database was set in a separate environment. Most of the configuration were done in the Cassandra environment since the Oracle Database technology is already in use on ams, and in Oracle's the only thing needed was to add a separate database, inside this environment to be used in the benchmark.

The Cassandra database was configured to run in a remote Windows Server, physically near to the Oracle Database location. This server was running a single-node Cassandra cluster, configured to listen to remote requests.

A 3. Configuration/Method

Before testing the performance of each database system, a schema had to be defined. The table to store data should be one that is usually used to store logs of events. The schema used in both databases up for testing was the next:

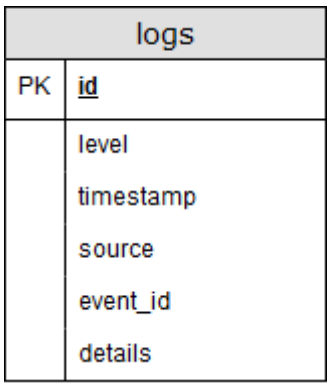


Figure 59. Logs table representation

The fields seen in Figure 59 are normally used to represent a log entry from a system:

- `level` is a mean of categorizing the entries in the database by urgency (information, warning or error);
- `timestamp` is the time when the logged event occurred;
- `source` is the origin of the log entry;
- `event_id` is an auxiliary identification that helps finding the origin;
- `details` is an extended description of the log entry, that is used to understand better the log.

The configuration was completed by defining how the queries would run in each database, we considered writing small scripts that would insert and read log entries in each database and record metrics such as insert and read times, and database size. Fortunately there are open source tools that allowed us to stress test both databases.

In Cassandra's case, there are built in tools to do so, and in Oracle's case, there was a third party tool that allowed for industry standard benchmark testing.

The databases for testing data sets were similar, where only the `id` field was being generated differently in each database. The configuration of either one of test tools allowed the specification of any field and the way these are being generated. For example, it is possible to specify the size of each field, while also being able to limit what kind of data is being inserted. These configurations made it so that the data being inserted in the databases was very similar in each environment, as can be seen in Figure 60 comparison:

```
CASSANDRA
 id                                   | details                           | event_id | level      | source   | tmstp
--------------------------------------+-----------------------------------+----------+------------+----------+---------------------------------
 00000010-2eb2-f7b2-0000-00102eb2f7b2 | FX\x015ME;ei<D7Bnzp\xJ7\#6u\x08... |       82 | \x08\x1\ne | \x0h4}'e | 1970-01-01 00:00:01.743000+0000


ORACLE
 id    | details                           | event_id | level   | source | tmstp
-------+-----------------------------------+----------+---------+--------+----------------------
 22673| dtldepWbZqQWZzeyBEInxHjrqFlSdtld... |       95 | warning | AD     | 03/12/2018  16:19:00
```

Figure 60. Oracle and Cassandra testing data samples

Cassandra

For Cassandra's testing, we used `cassandra-stress`, a Java-based stress testing tool for basic benchmarking and load testing a Cassandra cluster [124]. This tool can read the properties configured in a `YAML` file which will contain the settings to use in the stress test. These settings will the keyspace and table definition, and the query definitions where it's possible to specify the range of values each filed should have and also how the write and read flow should behave. The configuration was set as shown in Figure 61.

```
# Keyspace info
keyspace: stresslogs

keyspace_definition:
  CREATE KEYSPACE stresslogs WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 1};

# Table info
table: logstest

# The CQL for creating a table you wish to stress (optional if it already exists)
table_definition:
  CREATE TABLE logstest (
        id uuid,
        level text,
        tmstp timestamp,
        source text,
        event_id int,
        details text,
        PRIMARY KEY(id)
  )
# Defaults for all columns are size: uniform(4..8), population: uniform(1..100B),
cluster: fixed(1)
columnspec:
  - name: details
    size: uniform(1..200)
    population: uniform(1..1000)     # the range of unique values to select for the field
(default is 100Billion)
  - name: tmstp
    population: uniform(1..2350)
  - name: event_id
    population: uniform(1..100)
# 1 query at a time = fixed(1)
insert:
  partitions: fixed(1)
  batchtype: LOGGED              # type of batch to use

# A list of queries you wish to run against the schema
queries:
   get-logs:
      cql: select * from logstest where id = ?
      fields: samerow
```

Figure 61. Configuration of the cassandra-stress stress load test

The configurations shown in Figure 61 tells us that a one node cluster will be created with a keyspace named `stresslogs`. Inside this keyspace there will be a table called `logstest`, defined using CQL, which has the columns defined for a standard logs table, as can be seen in Figure 59. One important definition is in the `columnspec` section, where the population distribution can be defined for any columns in the table. This section specifies a uniform distribution between 1 and 200 characters for `details` values in generated rows, using the `size` parameter, since this column will have relatively more data compared to the others. The `population` parameter specifies the range of unique values to select for the `details`, `tmstp`, and `event_id` columns. After the column specifications, we define how each batch runs and, in the `insert` section, the `partitions` value directs the test to use the `columnspec` definitions to insert one row in the partition in each batch. The last section contains a query, `get-logs`, that can be run against the defined table.

To run a `cassandra-stress` test with a `.yaml` file, such as the one defined above, the commands in the cassandra-stress.bat should be executed. However, in order to run the tests in a controlled environment several options need to be defined:

- the `user` option is required to state that this is a custom test;
- the value for the `profile` parameter is the path and filename of the `.yaml` file;
- the n parameter specifies the number of batches to run;
- the values supplied for `ops` specify which operations to run and how many of each. These values direct the command to insert or read rows in the database, meaning that, for the tests to be performed, since each insert or read counts as one batch, the number of operations will be equal to the total number of batches (`n`);
- the `–node` parameter tells what node to connect to;
- the `–rate` option contains the threads parameter that specifies the number of clients to run concurrently;
- the `–graph` option provides visual feedback, where a file must be named to build the resulting HTML file. The `title` and `revision` parameters are optional but revision must be used if multiple stress tests are graphed on the same output, as is the case in the tests produced.

The following command (Figure 62) shows using the *user* option and its parameters to run `cassandra-stress` insert tests from a configuration file named `cqlstress_logs.yaml`:

```
cassandra-stress user profile=./cqlstress_logs.yaml ops(insert=1) n=10000 -node 10.11.112.38 -rate threads=4 -graph file=remote-benchmark.html title=remote_tests revision=fixed1write
```

Figure 62. Example of command used to run cassandra-stress insert tests

The next command (Figure 63) shows the use of parameters used to run `cassandra-stress` read tests from the same *cqlstress_logs.yaml* configuration file.

```
cassandra–stress user profile=./cqlstress_logs.yaml ops(get-logs=1) n=10000 no-warmup -node 10.11.112.38 -rate threads=4 -graph file=remoteRead-benchmark.html title=remote_read_tests revision=fixedread3
```

Figure 63. Example of command used to run cassandra-stress read tests

The two commands shown above (in Figure 62 and Figure 63) were executed several times to achieve well-founded results, to better understand Cassandra's capabilities.

Oracle

For the Oracle Database testing, two third party tools were used, a database management toolset and a database testing tool, Toad for Oracle[97] and Benchmark Factory[125], respectively. Toad for Oracle was used to configure the table in the database, whereas Benchmark Factory was utilized for the configuration and testing of data.

Before inserting data from Benchmark Factory to the Oracle Database, a table needed to be set. To manage the database, a TNS[32] connection must first be established in Toad for Oracle, and then the next (Figure 64) PL/SQL script was executed to create the `JALD_LOGS` table:

```
CREATE TABLE JALD.JALD_LOGS
(
      ID        NUMBER                    NOT NULL,
      LVL       VARCHAR2(30 BYTE)         NOT NULL,
      TMSP      DATE                      NOT NULL,
      SRC       VARCHAR2(60 BYTE)         NOT NULL,
      EVENT_ID  VARCHAR2(90 BYTE)         NOT NULL,
      DETAILS   VARCHAR2(300 BYTE)        NOT NULL
)
```

Figure 64. PL/SQL script of the logs table

With the database configured, the next step was to setup Benchmark Factory to insert and read data on the `JALD_LOGS` table. In this tool it is possible to create a Job, which is a workload that is defined to run on a specific database.

As the goal was to insert data as close as possible to the data inserted in Cassandra, the queries prepared to be run on the job were very similar. Next (Figure 65 and Figure 66) are the insertion and read operations made against the database:

```
INSERT INTO JALD_LOGS (LVL,TMSP,SRC,EVENT_ID,DETAILS) VALUES ('$BFRandList
("error":1,"information":1,"warning":1)',SYSDATE,'AD',$BFRand(500),'dtl$BFRandStr(25,ALPHA)');
```

Figure 65. SQL insertion query to be run on Benchmark Factory

```
SELECT * FROM JALD_LOGS WHERE TMSP=TO_DATE ('$BFDate ("11/26/2018","11/26/2018")
$BFRandRange(10,18):$BFRandRange(28,44):$BFRandRange(23,39)', 'MM/DD/YYYY
HH24:MI:SS');
```

Figure 66. SQL read query to be run on Benchmark Factory

To prevent the insertion and reading of the same data, scripts that randomize data were used in the SQL transactions, making the tests more dynamic, while also maintaining its veracity. This scripting feature is known as `BFScripts`, which allow the insertion of randomized data into the load testing process.

MySQL

During the database testing phase a MySQL database was also prepared, but was not tested in a remote environment, so these results cannot be compared to the ones from Oracle or Cassandra. Nevertheless, the inclusion seemed relevant to serve as a reference point and as an additional comparison for the remaining databases.

[32] TNS (Transparent Network Substrate), a proprietary Oracle computer-networking technology [126].

To prepare and test the MySQL database, the tools used were:

- `phpmyadmin`[127], an administration tool for MySQL;
- `mysqlslap`[128], a diagnostic program designed to emulate client load for a MySQL server;

A 4. Results

Each database had to face around one million writes and reads from four concurrent clients to simulate a real environment. Once the recording process was completed, we made an analysis of the database sizes and insertion and read times, representing these in the following table:

	Casssandra-stress	mysqlslap	Benchmark factory(Oracle)
Read operations/s on a 1001100 rows DB	53 ops/s	100 ops/s	49 ops/s
Write operations/s	53 ops/s	397 ops/s	46 ops/s
Database size	100 MBs	264 MBs	81 MBs

Table 1. The results from the load stress tests of each database system

From Table 1 we can analyze the insertion and read times and sizes of the data sets in Cassandra, MySQL and Oracle databases. As the table shows Cassandra and Oracle had similar insertion and read times, even though Cassandra's times were faster by a small margin. As for storage sizes, Oracle's database size is smaller than Cassandra's.

Measuring the insertion time in both databases is the main goal of the benchmark, as one of the main requirements of the proposed project was to suggest a storage system which was capable of handling the write throughput of logs better than the current storage system (Oracle).

A 5. Discussion

Having the results, we are able to make the decision on whether Cassandra's storage system will be able to handle the log traffic required by the company.

Referring to data insertion, Cassandra consumed 100 MB, storing 1 000 000 logs at an average insertion time of 53 operations per seconds. Oracle, on the other hand, consumed 81 MB to store 1 000 000 logs at an average insertion time of 46

operations per second. This means that Oracle took 19% less disk space but was 13.2% slower when compared to Cassandra.

This said, and to comply with the requirements defined by the company, we conclude that the database system chosen was Apache Cassandra, due to it providing a write throughput higher than Oracle.

Further tests would need to be made to better measure Cassandras capabilities, but, at the time the Database benchmark was executed, no additional results were needed to prove the database decision.