DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2018*

Indexing file metadata using a distributed search engine for searching files on a public cloud storage

SIMON HABTU

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

Visma Labs AB or Visma wanted to conduct experiments to see if file metadata could be indexed for searching files on a public cloud storage. Given that storing files in a public cloud storage is cheaper than the current storage solution, the implementation could save Visma money otherwise spent on expensive storage costs. The thesis is therefore to find and evaluate an approach chosen for indexing file metadata and searching files on a public cloud storage with the chosen distributed search engine Elasticsearch. The architecture of the proposed solution is similar to a file service and was implemented using several containerized services for it to function. The results show that the file service solution is indeed feasible but would need further tuning and more resources to function according to the demands of Visma.

Keywords

public cloud ; distributed search engine ; metadata indexing ; scalability

Sammanfattning

Visma Labs AB eller Visma ville genomföra experiment för att se om filmetadata skulle kunna indexeras för att söka efter filer på ett publikt moln. Med tanke på att lagring av filer på ett publikt moln är billigare än den nuvarande lagringslösningen, kan implementeringen spara Visma pengar som spenderas på dyra lagringskostnader. Denna studie är därför till för att hitta och utvärdera ett tillvägagångssätt valt för att indexera filmetadata och söka filer på ett offentligt molnlagring med den utvalda distribuerade sökmotorn Elasticsearch. Arkitekturen för den föreslagna lösningen har likenelser av en filtjänst och implementerades med flera containeriserade tjänster för att den ska fungera. Resultaten visar att filservicelösningen verkligen är möjlig men skulle behöva ytterligare modifikationer och fler resurser att fungera enligt Vismas krav.

Nyckelord

publika moln ; distribuerad sökmotor ; metadata indexering ; skalbarhet

Contents

Chapter 1

Introduction

There are different solutions for storage in system architectures, each with their own benefits and flaws. Normally, there is almost never a general optimal solution for all system types. As a result this introduces the issue of identifying system specific attributes when determining a suitable storage solution. As organizations develop and new technologies emerge, current implementations tend to out-date quickly [1]. Different ways of storing data come with different challenges for developers as there is often no optimal solution for all use cases. Storing data can be done in several ways, with relational databases being the most common way of storage today.

Since previous years it has become more common for companies to externalize computing resources to cloud solutions [2]. Such solutions may increase costs for computing resources as the cloud providers are the ones managing them. As this can allow companies to focus on their main activity, there are cases where the costs for using public clouds can become too high thus making it infeasible [3].

Migrating to a cheaper storage solution sounds simple but can be an extensive task. The issue occurs when the insertion and retrieval of data is done differently which can require the whole infrastructure of the system to adapt to that specific way of information management [4]. A migration would be done assuming the destination system does not critically affect the performance negatively.

1.1 Background

This chapter will present background relevant information to understand the problem, purpose and goal of the thesis.

1.1.1 Relational databases and BLOBs

A database is defined as an organized collection of data [5]. When discussing databases, the two major types of database models are often mentioned which are *relational* and *non-relational* database models. Relational databases offer properties such as atomicity, consistency, isolation and durability also known as *ACID* [6]. Relational databases are the most commonly used databases and are made up of table containing rows and columns where each row represents an entry with data for every defined column, meaning the data is structured. It is also possible to store binary objects in data fields, these are called Binary Large OBjects or *BLOBs*. Using BLOBs allows for a storage with ACID guaranteed.

Unstructured data on the other hand come i different forms and often do not follow a pre-defined pattern. Images and text files are examples of unstructured data, meaning the information they contain can not be foreseen [7]. Unstructured data can be stored as BLOBs in databases, but the consequences are dramatically growing databases which can generate high costs.

1.1.2 Visma and Proceedo

Visma Labs AB is a company based in Norway which specialize in automating business processes [8]. Under Visma Labs is the product Proceedo, a tool for companies to process purchases with direct access to several supply providers within a single system. Proceedo also handles invoices to simplify accounting. These invoices are saved to a single database hosted on a storage network which allows for a scalable way of database storage while avoiding a single point of failure [9]. To Proceedo the solution is an expensive alternative which is why new solutions are being discussed.

1.2 Problem

The root cause of all issues for this study is the cost of the database in the storage area network. This has made Visma consider extracting all attachment files from the current Oracle Database storage to a public cloud storage. Migrating these files will in its turn forces Visma to manage the external files in an accessible way. Visma has proposed using a document-based search engine called Elasticsearch to find external files. The problem will then be to find out: how can Elasticsearch be used for indexing and retrieving files on a public cloud storage?

1.3 Purpose

This thesis will evaluate the usage of Elasticsearch and how it will help Visma Labs choosing whether if this is a useful approach suited for their demands. Using Elasticsearch for finding files on a public cloud storage would help developing a solution with a more scalable storage.

1.4 Goal

The deliverable of the study is a file service used for indexing and searching files. Implementing the file service will allow measuring the performance of the proposed solution which will help reaching the goals of the thesis. The goal of the thesis is to answer this set questions, which in its turn will answer the problem stated in section 1.2:

- Is it possible to keep Elasticsearch aware of all changes in the public cloud storage?
- How fast can Elasticsearch index and search files on the public cloud storage?
- Is it possible to scale the solution for faster file indexing and search?

The summarized goal would be to have enough measurements to make a fair evaluation of how well the distributed search engine performs when indexing and searching files on a public cloud storage.

1.5 Benefits, Ethics and Sustainability

This section will shortly present the benefits, ethical questions and sustainability considerations of the study.

Benefits

Until now there are only a few to no studies regarding evaluating Elasticsearch as a solution to handling external files. Similar studies are those of content-addressable storage (CAS) systems [10]. This study will manage files on public cloud storages that has roots in the same problem area, which is rapidly growing databases and migrating internal files to external storages. Hopefully it will be proven that using the proposed solution for indexing and search has acceptable performance results, which could benefit organizations in decreased storage costs.

Ethics

Proceedo currently handles client data which can be regarded as personal information. Protecting personal information is vital to the client. During the year 2018, the European Union (EU) will enforce a General Data Protection Regulation or *GDPR* [11]. The regulation will help harmonizing the data protection regulations for the EU, making it easier for non-EU countries to adapt. For Visma this will be considered important to avoid being heavily fined.

Sustainability

The root cause for Visma to consider migrating files to an external storage is the current cost of today's storage. This can be related to *economical sustainability*. Economical sustainability can according to Kungliga Tekniska Högskolan (KTH) be defined as economical growth with/without regard to its impact on the social or environmental consequences [12]. The proposed solution to retrieving externalized files is planned to decrease storage costs for Visma. The contribution will therefore be to economical sustainability where the consequences to social or environmental are minimal or close to none.

1.6 Methodology / Methods

The thesis has two potential ways of being conducted which are as a *qualitative* or *quantitative* study. The thesis will be a performance measurement of the proposed solution to the problem which makes it a quantitative study. This section will describe the different research methods and methodologies presented by A. Håkansson [13] that are to be used for the thesis in a model called the "Portal of Research Methods and Methodologies for Research Projects and Degree Projects". The model will be used to determine aspects for the research methods and methodologies.

Philosophical assumptions

The study that will be conducted is one of a quantitative character. The model mentioned shows that a quantitative study mainly includes the philosophical assumptions of *positivism*. The views of positivists are that reality is given objectively and independent of the researchers influence. It is often used in manners of testing theories and are useful for testing performances within information and communication technologies.
The assumptions of *realism* are not fully regarded as a quantitative philosophical assumptions. It states that realists observe phenomenons to provide credible data and facts, which is why it is relevant for the study.

Research methods

Further looking into the model shows the research methods chosen for the study which are: *experimental*, *fundamental*, *applied*, *empirical*

The results of the study will be derived from the conducted experiments to test relationships between variables. The study also tests the theory of using a distributed search engine to index files on a public cloud which has not been explored enough to answer the research question. Most of the tools needed for the experiment are available which requires software development to some extent. An analysis will then be made on the retrieved results. This motivated the choice of research methods.

Research approaches

The study is done to validate or falsify the theory of a suitable solution for indexing and retrieving files with a varying amount of files. According to the model it is safe to say that a *deductive* research approach is the correct way to tackle the study, due to the falsifiable hypothesis.

Research strategy

The research strategy chosen for the study is an *experimental* one. The thesis will test the hypothesis with an experiment to validate it and will play the major role in deciding whether if the solution is useful. A file service will be implemented which is separated from the Proceedo application to module test a document-based search engine with a public cloud storage to index and retrieve files.

1.7 Stakeholder

Visma Labs AB is a company based in Norway which specialize in automating business processes [8]. Visma has several products and provide software, outsourcing services, purchasing solutions, etc. Visma is the owner of Purchceed-to-Pay solution Proceedo, which includes the entire value chain - from product to pay.

As the company is moving away from private hosted solutions, cloud solutions for computing resources has been extensively discussed. The main reason for this is the necessity to outsource the server maintenance to be able to solely focus on development.

1.8 Delimitations

The thesis focuses on the functionality and performance of the implemented system. The verification process will test the functionality of the implemented system by expecting certain outputs depending on inputs. The performance tests are done to measure the rate at which the system can index and retrieve files to evaluate if it meets the requirements as described by the stakeholder. It will be shortly discussed how a distributed search engine can increase fault tolerance

but it will not be tested nor proven. Security will not be of interest during implementation since it does not help prove the hypothesis.

1.9 Outline

Chapter 2 explains the fundamental technologies that will be used for the methodology.
Chapter 3 presents the research methodologies used to tackle the research.
Chapter 4 describes the chosen approach for solving the problem of the thesis.
Chapter 5 the performance results of the implemented solution are presented.
Chapter 6 discusses the collected results with regard to the problem statement.
Chapter 7 concludes the discussion and summarizes how the thesis answers to the problems stated in this chapter.

Chapter 2

Theoretical background

The theoretical background will cover all the information that is necessary to fully understand the contents of this research. Section 2.2 explains the meaning and purpose of a "cloud". Section 2.3 explains how message-oriented middleware work and section 2.4 explains the architecture and metrics of search engines, which cover the most part of the thesis. Section 2.5 helps understand deployment using containers which is vital for the implementation.

2.1 Relational databases

A database is defines as a way of storing information that can be retrieved [5]. Relational databases are databases that are structured using rows and columns to store data. Tables are used to store information in rows and columns. The information within a table is coherent, meaning that all data is somewhat similar or possess similar characteristics.

A relational database can be summarized as data storage created using a relational model. A Database Management System (DBMS) is used to handle the way the data is stored and retrieved. Specifically for relational databases, Relational Database Management Systems (RDBMS) are used.

Binary Large OBject or *BLOB* represents a data type that is used for storing binary objects in databases[14]. When files in relational databases there are two common practices of how these are stored. The first ap-

proach is to store a binary object as a field value in the database. The other is to store a path to an external file, called an external pointer. The stakeholder has chosen to store binary objects within the relational database which has benefits such as guaranteed atomicity, consistency, isolation and durability also known as *ACID* [6].

Name	Age	Location	Number	Picture
John	24	Oslo	123	(BLOB)
Tobias	25	London	234	(BLOB)
Simon	21	Paris	345	(BLOB)
Peter	47	Riga	456	(BLOB)

Figure 2.1: An example of a table in a RDBMS containing a column for BLOBs.

While these are major benefits, a set back is that this will eventually result in a large monolithic database structure. To prevent bad scaling, Proceedo uses an Oracle SAN storage as a method of database storage as described in section 4.1.1. Figure 2.1 shows an example of how a BLOB is stored within a table row. The example shows how personal information is stored in primitive data types for the first four columns and a BLOB containing a picture.

2.2 Cloud and cloud storage

A cloud is a general term that describes resources being available to users over a remote network. The resources that are made available are virtually on the same host but physically this is often spread out over several locations. Clouds can be either public, private or hybrid. Public clouds are available for everyone's use while private clouds are "on-premises" and are usually available for users within an organization. The hybrid cloud model adopts features of both previously mentioned models and makes resources available in-house or on a public cloud [15].

Cloud computing is an abstraction of computer power that is usually made available as a "pay-as-you-go" service, meaning you only pay

for the resources that you actually use. Buyya et al. use the analogy of electricity used from a wall socket to describe how cloud computing is an abstraction of computational resources the same way as the wall socket is an abstraction of the power plants [16].

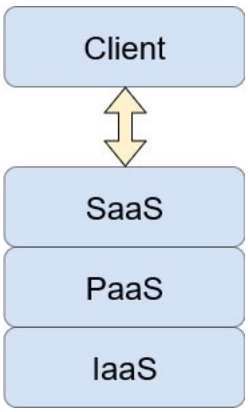

Figure 2.2: Abstraction of the relations between the main types of services.

While the term resources can have several meanings, the very commonly used term of "services" encapsulates its usages well:

- **Software as a Service**
  A SaaS is often a web-based application that is subscribed for rather than purchased [17].
- **Platform as a Service**
  Provides a computing environment and is on a higher level than IaaS [18].
- **Infrastructure as a Service**
  IaaS often provides simple resources such as hardware (storage, network, virtual machines etc.) [19].

2.2.1 Software as a Service

Software as a Service or *SaaS* is the highest layer in the cloud model as shown in Figure 2.2. The strongest characteristic of the SaaS is that it almost never uses a client to communicate with the system. The interface usually uses remote invocations of the system functions. The

system functions are then invoked on the server side of the application. This setup simplifies the usage of service for the user in terms of portability and user friendliness.

2.2.2 Platform as a Service

Platform as a Service or *PaaS* is on a lower level than SaaS as shown in Figure 2.2 and allows for developers to deploy applications on the premises of the cloud provider. The provided APIs of the platform are only configurable by the cloud provider and limits the capabilities of the developer. A PaaS is basically an abstraction of the platform on which the application is running. This means that one does not have to worry about scalability or load balancing issues for example. To summarize, the PaaS simplifies deployment and spares developers the time thinking about underlying issues. The consequences can be that developers are limited to the functions that are provided by the cloud provider.

2.2.3 Infrastructure as a Service

Infrastructure as a Service or *IaaS* is the lowest level of abstraction in cloud computing. It usually provides resources such as storage, networking or computational resources as a part of an application or system. The exact location of where the data is handled is unknown to both users and developers. IaaS allows for good elasticity of services by providing functionalities for quick scaling-up or scaling-out. The storages can for example be in form of simple file storage or file systems. Computational resources often provide virtual machines with a desired operating system to deploy applications on.

The SaaS and PaaS will not be very relevant for this research. The IaaS, shown in Figure 2.2, includes cloud storage which plays a major role when implementing the new solution since the problem that the thesis will solve is accessing external files on a public cloud storage by indexing them.

2.3 Message-oriented middleware

In distributed systems, the possibilities of scaling are often better than those of a monolithic architecture. Remote Procedure Calls or *RPC* is used as a fundamental part of communication in these systems, which has limited functionalities.

A message-oriented middleware or *MOM* allow clients send and receive messages from other clients of the MOM. The following section is a summarized version of MOM's and message brokers as explained by Steglich et al [20].

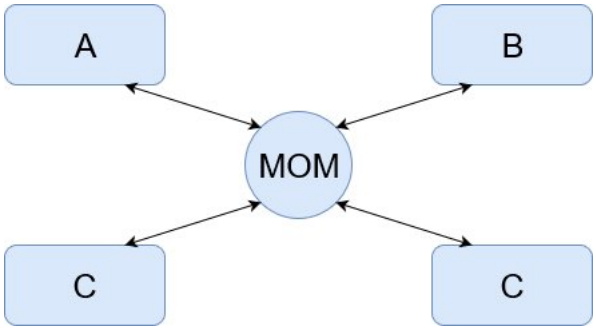

Figure 2.3: Example of how a MOM can be deployed with four applications.

MOMs use an asynchronous interaction model, meaning that sending or retrieving messages are non blocking calls. Asynchronous calls means that the sending client does not have to wait for a message to be delivered and the receiving client can decide whether it wants to read the message or not. Figure 2.3 shows how several applications connected to the MOM can communicate without being connected directly to each other.

Adding a MOM as a communication layer between applications gives properties such as *low coupling*, *reliability*, *scalability* and *availability*. MOMs add a layer between the senders and receivers which lowers the coupling of the system, meaning that they are independent of each other. Message loss is prevented throughout the system by using *store and forward* messaging, meaning that lost messages have already been stored and can be re-sent. This guarantees that the messages is delivered exactly once. By allowing consumers to read messages at their

own pace MOMs allow applications to scale without regard to other applications in the system since they do not have to be adapted to each other. Systems using MOMs do not require all applications to available at the same time since a failure in one application does not disrupt other applications, which gives higher availability.

2.3.1 Message queues

Message queues are a fundamental part of the MOM which allows the middleware to store messages on its platform, this is a vital part of the asynchronous interaction model.

Figure 2.4: A queue is used to store messages that have not been read yet.

The queue is where the senders send their messages to be consumed by the receivers. The usual order of the messages is *First-In-First-Out* or *FIFO*, meaning that the first message in is the first message to be consumed. A queue usually has a name which the consumers bind to and several other attributes that can be configured. Figure 2.4 demonstrates how producers add messages to the queue while the consumers read the messages from the queue.

2.3.2 Messaging models

There are two major messaging models used when discussing MOMs. The first messaging model is the *point-top-point* messaging model which allows for applications to asynchronously send messages to their peers. The second messaging model is the *publish/subscribe* messaging model. It allows *publishers* to publish their messages to a specific topic which the *subscribers* subscribe to.

Figure 2.5 shows how the two presented messaging model work. The publish/subscribe can potentially have several publishers publishing to multiple topics and multiple subscribers can consume messages from multiple topics. The relation between the publisher and

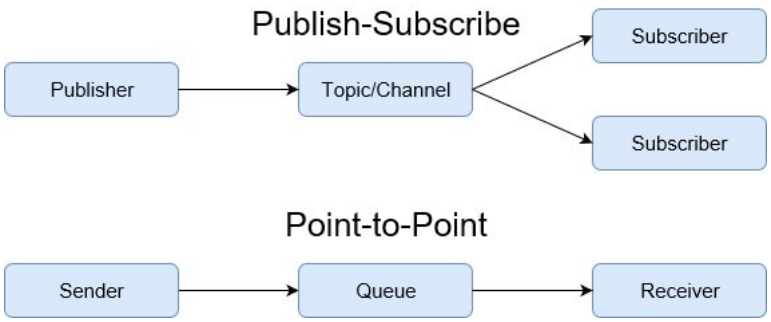

Figure 2.5: The publish/subscribe and point-to-point messaging models in comparison.

topic is a *many-to-many* relationship, this goes for the relation between subscribers and the topics as well.

2.3.3 Message brokers

A message broker is built on top of a MOM and often works as an intermediate step between communicating services. The protocol used for messaging is usually the Advanced Messaging Queue Protocol (AMQP) [21, 22]. It is used in distributed systems to handle message communication.

Message brokers decrease coupling between services and can asynchronously distribute messages which increases the overall throughput. The messages that are not processed immediately are placed in a message queue to be processed when possible. Each queue allows for multiple subscribers to read messages from it. When a message is read from the queue it is deleted, meaning the subscriber has consumed the message. The queue gets its messages from the publisher which is the producer of messages.

2.4 Search engines

A *search engine* can be described as an application of information retrieval techniques to large collections of data. Search engines can be used for purposes such as web-based, desktop or enterprise search for example. This section is presented as interpreted by Croft et al [23].

The different kinds of search engines available can be summarized as following:

- *Web-based search engine* - These often have the ability to crawl or capture huge amounts of data with response times of less than a second.
- *Enterprise search engine* - Processes a variety of information sources within the enterprise to provide information relevant to the company. This is often used in purposes of *data mining*.
- *Desktop search engine* - Allow fast incorporation of new documents, web pages and emails for example that are created by the user. Desktop search engines often provide an intuitive interface.
- *Open source search engine* - Search engines used for a variety of commercial applications. These are designed differently depending on the application domain.

All search engines go through the process of *indexing* and *query processing*. Indexing a document means creating a data structure called *index* which improves the search speed by only searching in relevant indexes instead of all documents available.

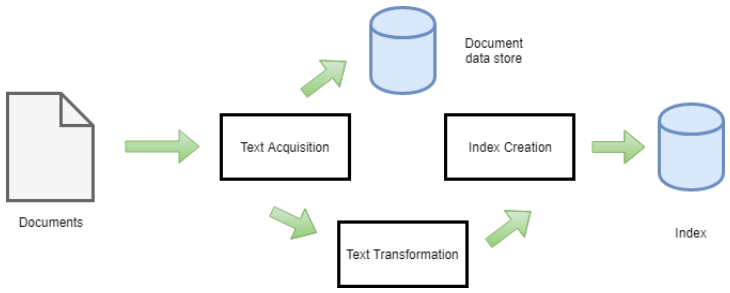

Figure 2.6: Indexing as represented by Croft et al [23].

Croft et al visualize indexing as a process of three major steps as shown in Figure 2.6; *text acquisition*, *text transformation* and *index creation*. The text acquisition makes the document available by creating a document data store with content and metadata. The text transformation is what creates *index terms* from the document to be used in the search process. The output of the text transformation is used by the index creation step to create an index, allowing for faster search of the document.

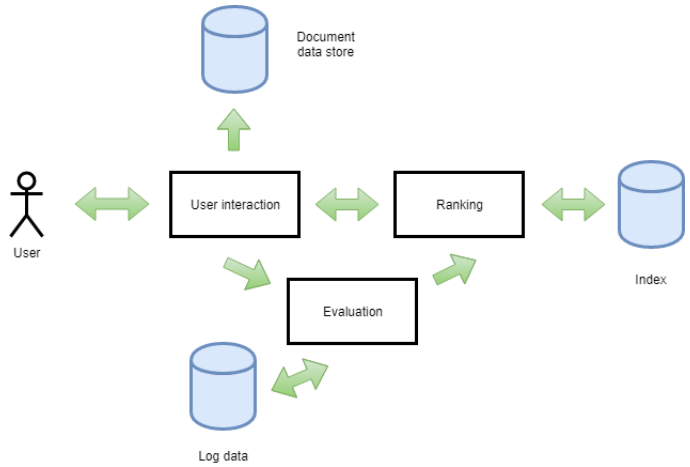

Figure 2.7: Query processing as represented by Croft et al [23].

The query process as presented in Figure 2.7 is visualized as a process of three steps; *user interaction*, *ranking* and *evaluation*. The user interaction step means allowing the user to submit its valid query and transforming the query to index terms. It is also responsible for presenting the results to the user in a structured manner. The ranking step generates results based on document ranking score. This is an important part of the search since it decides what is relevant to the user. The evaluation is done for possible tuning of the ranking from log data.

2.4.1 Search metrics

These are a few of the different types of search engines but also very major ones. Even though these types of search engines differ, they have several features in common. A search engine can have its performance measured in different metrics. Some important metrics of these are *response time*, *query throughput* and *indexing speed*. The characteristics of these metrics and others are briefly explained as following:

- *Response time* - The time it takes to submit a request and receiving a result.
- *Query throughput* - The amount of queries that can be processed per time unit.
- *Indexing speed* - The rate at which a document can be indexed and available for search.

- *Scalability* - Describes how well a search engine performs during high loads relative to smaller loads of requests.
- *Adaptability* - A measurement of how well a search engine can be customized to fit the application.

Response time, query throughput and indexing speed are metrics simpler to measure than scalability and adaptability which need further specification to be measured.

2.4.2 Distributed search engines

Distributed search engines allow processing of queries over several servers or *nodes* which describes server instances. Until now it has been usual to use expensive and powerful hardware to handle the process of indexing and querying. Since the price for commodity hardware is very affordable it has become normal to *scale out* instead of *scaling up*, also known as *horizontal* scaling [24].

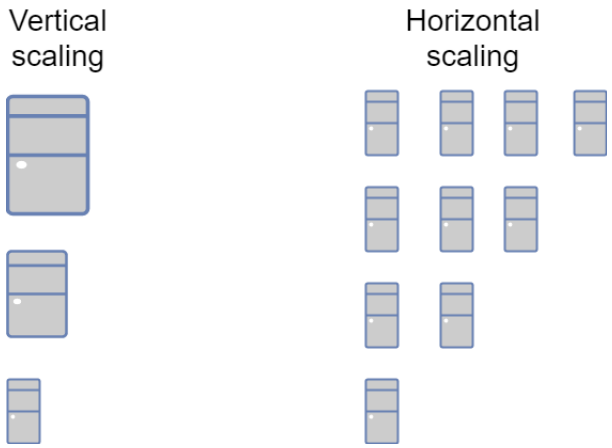

Figure 2.8: Vertical vs. horizontal scaling.

Figure 2.8 shows the difference of scaling vertical and horizontally. This allows for a more parallel search processing which can help solve issues such as scalability by distributing tasks among several nodes in a cluster.

2.4.3 Elasticsearch

Elasticsearch is a scalable distributed full-text search engine [25]. It was developed by the Elastic team and is distributed as an open-source software under the Apache License 2.0 [26]. Elasticsearch is based on Apache Lucene, a full-featured search engine library written in Java, this is also an open-source software distributed under the Apache License 2.0 [27].

Elasticsearch uses a RESTful-API with HTTP-requests to execute queries which makes it suitable for a distributed system. Elasticsearch is usually set up in a cluster of servers or nodes but can be set up as a single node as well. To understand Elasticsearch there are a few terms that need explanation [28]:

- *Node* - describes an instance of Elasticsearch. Several nodes can be run on a single server and can also be distributed in a cluster of servers.
- *Document* - represents a single entry in Elasticsearch. Each document is indexed as a JSON-object to be made searchable.
- *Index* - contains documents with similar characteristics and can be seen as something in between a relational database and its table.
- *Shards and replicas* - each index can be infinitely large which is why Elasticsearch offers sharding which divides the index and distributes them throughout the cluster. Indexes can also be replicated to allow parallel search in the replicas while increasing fault tolerance and availability.

Architecture

An Elasticsearch cluster consists of master nodes and slave nodes. A master node can be configured to contain data as well. The master node is where all requests are received before distributing the operation further to all shards across the cluster.

Figure 2.9 shows how a cluster has been set up. In this case the master node is a data node as well, meaning it receives all request that are to be distributed while also containing indexed documents. Shards

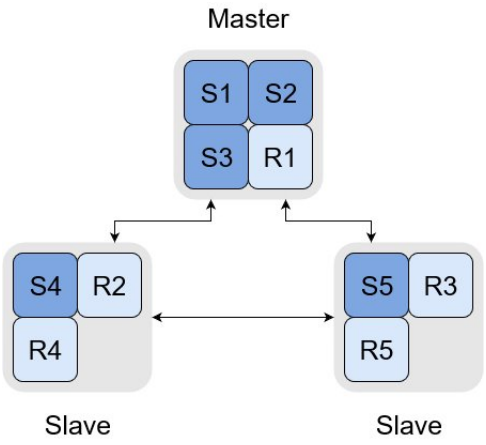

Figure 2.9: Example of an Elasticsearch cluster with one master node and two slave nodes. The cluster contains five shards with one replica each.

of indexes will be distributed throughout the cluster dynamically by Elasticsearch. The example assumes that there is only one index which has been sharded. The presented picture shows a cluster with five shards and one replica each. The distribution of shards is automatic in Elasticsearch but it can be specified before and after setting up the cluster.

2.5 Operating-system-level virtualization

Operating-system-level virtualization also known as *containerization* enable isolation of the software from the host operating system [29]. An instance of the virtualization is referred to as a *container* and is instantiated within the kernel of the operating system.

Previously virtual machines or *VMs* have been to avoid running only a single application on a physical server. The host operating system would have a hypervisor running on top of the operating system. It was later discovered that the hypervisor could be ran on the "bare-metal" server without the operating system. Each VM has its own resources such as CPU, memory and network configurations. Figure 2.10 shows how VMs are ran on physical servers using the hypervisor. The VM still needs a operating system within it to run, meaning that

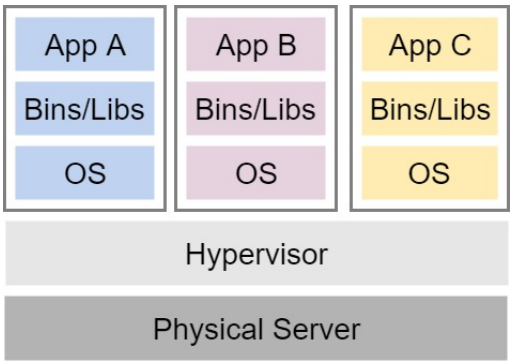

Figure 2.10: A visualization of the usage of VMs.

licensing costs still remain. This increased the necessity for using software containers instead of VMs.

Linux containers or *LXC* use control groups to isolate computing resources between applications. LXC uses namespaces to be isolate applications from the operating system which separates the process trees and different accesses. To summarize, LXC provides are a flexible way of virtualizing the application instead of an operating system such as for VMs. This saves licensing costs and allows for isolated applications.

Chapter 3

Methodology

The research methodology is useful for planning, designing and conducting the research. This chapter is a summary of a study presented by Anne Håkansson.[13].

3.1 Research strategies

The research methodology are guidelines followed for carrying out the research. A research strategy will be chosen from the presented ones to help conducting the thesis.

Experimental research strategies try to verify or falsify the stated hypothesis of the research. All independent and dependent variables which may affect the result are of interest and are also analyzed to prove their correlations to each other.

Ex post facto research is done after the data is already collected. It is done to verify or falsify a hypothesis without changing any variables. Ex post facto researches are also used to study behaviours and can therefore use qualitative methods.

Surveys is a descriptive research method which studies the attitudes of a population to find correlations between variables by examining frequencies. The survey study can be *cross-sectional*, meaning that a population is assessed at a single point of time. It can also be *longitudinal*, meaning that a population is assessed over a period of time. The characteristics of the survey research allows it to be used with qualita-

tive and quantitative methods.

Case study is an empirical study and investigates cases where boundaries between phenomena and contexts are not obvious. The strategy requires empirical investigations on evidence from several sources. Case studies can be based on evidence from both qualitative and quantitative studies and therefore both methods can be used.

Action researches are performed to provide general approaches in problematic situations. An action research help improving the way problems are addressed by observing taken actions that are evaluated and criticized. Action research studies specific settings with limited data sets, which makes qualitative methods suitable.

Exploratory research methods try to find as many relationships between variables as possible and often provides general findings. It does not provide results to specific problems but instead it finds key issues for hypotheses. Exploratory research strategies use qualitative data collection methods.

Grounded theory analyzes the collected data to find new theories based on the analysis.

Ethnographic research is research done to study people, often divided into groups where the people have something in common such as culture to find phenomena within these.

This research will use an experimental research strategy and will try to verify or falsify the hypothesis of using Elasticsearch as a scalable way of indexing and searching files.

3.2 Data collection methods

A data collection method is a technique for collecting data for the research. This section present the most common ones.

Experiments collect large datasets for variables.

Questionnare collect data through either; quantifying data (alternative questions) or qualifying data (reviewing questions).

Case studies analyze cases with single or a small number of participants. This is used with the case study research method.

Observation observes behaviour for specific situations (participation) and culture (ethnography).

Interviews can be structured, semi-structured or unstructured. They give a deep understanding of the problem from the participants' view.

Language and Text analyzes and tries to interpret documents to find meanings.

Experiments will be used as a data collection method for this research. The indexing and search functions of the implementation will be tested to collect performance results that will be further analyzed.

3.3 Data analysis methods

Data analysis methods are used to analyze the collected data. It describes the process of analyzing data that will be used to support decisions. These decisions will allow for drawing conclusions.

Statistics uses statistical methods to analyze data by calculating results. It also includes evaluating the results to find significance to the study.

Computational Mathematics has a purpose of modelling and simulating calculations for algorithms, numerical- and symbolic methods.

Coding turns qualitative data into quantitative data and observes it by naming concepts and strategies to apply statistics to it.

Analytic Induction and *Grounded Theory* are iterative methods that continue until no case dismisses the hypothesis. Analytic induction is considered complete when the hypothesis and grounded theory end with a validated theory.

Narrative Analysis relates to literary discussion and analysis. *Hermeneutic* and *Semiotic* are used for analyzing texts and documents which supports traceability in requirements and interfaces.

Statistics will be used as a data analysis method for this research. The collected data are performance results and by analyzing these the hypothesis can be verified or falsified.

3.4 Quality Assurance

Quality assurance validates and verifies the material of the research. The validation and verification differs for quantitative and qualitative studies since the quantitative uses a deductive approach while the qualitative uses an inductive approach. Quantitative researches apply; *validity*, *reliability*, *replicability* and *ethics*. Qualitative researches apply; *validity*, *dependability*, *confirmability*, *transferability* and *ethics*. The following terms are explained in context of a quantitative research [13]:

- Validity - makes sure that the instruments used for testing are measuring what is expected of the research.
- Reliability - describes the stability of the measurements and provides consistency for the results.
- Replicability - provides the ability for other researchers to repeat the tests and retrieve the same results, which requires a carefully documented test setup.
- Ethics - is independent of research type (quantitative or qualitative), describes the moral principles when conducting the research. Ethics protects participants, handles confidentiality and avoids coercion.

For a qualitative study, the terms are explained as following:

- Validity - a guarantee of trustworthiness, makes sure that the research has been conducted according to rules.
- Dependability - corresponds to reliability and judges the correctness of the conclusion by auditing it.

- Confirmability - confirms that no personal assessments have affected the results of the research.
- Transferability - to create a well described research to be used as an information source for other researchers.

Since this research is of quantitative character validity, reliability, replicability and ethics will be discussed to validate and verify the quality of the research.

3.5 Software development methodology

A software development methodology describes the frameworks used to structure, plan and control the development process of a software. These frameworks differ in many ways and all have their strengths and weaknesses, meaning there is no optimal way to guarantee a sound methodology fitting all software projects. The considerations for choosing a framework are both technical and organizational [30]. The three most basic software development process frameworks are: *agile*, *waterfall* and *spiral*. For this research an agile methodology is used since this is what the stakeholder use within the developing teams.

Agile software development is a methodology used to tackle software development projects. There are several frameworks for agile software development processes such as Crystal Methods, Dynamic Systems Development Model (DSDM) and Scrum [31]. Agile methodologies came to attention when the "Manifesto for software development" was published, which describes principles behind an agile methodology [32]. The manifesto has twelve principles describing the agile development model which can be summarized as following [32]:

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently ranging from weeks to months.
4. Developers and others must work with each other daily throughout the project.

5. Build projects around motivated individuals. Give them a supportive environment and trust them.
6. The most effective method of conveying information is face-to-face.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development, therefore the pace should be constant.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity is essential.
11. Good architectures, requirements and design emerge from self-organizing teams.
12. The team must regularly reflect on how to become more effective and adjust accordingly.

The development process for this research took inspiration of the manifesto described. Meetings were held to continuously gather new requirements after the initial requirements had been submitted. The meetings were held twice every week to present and review the progress of the past few days, often leading to altered requirements.

Chapter 4

Procedure

This section covers the implementation of the file service as well as description for configurations. Choices that were made for the implementation are based on the research method to guarantee that the research question can be answered. The purpose of the implementation is to deliver a file service able to perform tests with reference to the specified metrics for the research.

4.1 Persistence in Proceedo

Proceedo handles invoices to simplify accounting. These invoices are saved to a data storage network with Oracle's SAN storage which allows for a scalable way of database storage while avoiding a single point of failure [9]. The solution is an expensive alternative which is why the stakeholder is considering new solutions.
Figure 4.1 shows how the current architecture of Proceedo is built. The interesting part for this research is the "Persistence" part and more specifically the "Main DB". The main database contains several hundred tables necessary for the Proceedo application. One of the tables is a file table containing binary objects or BLOBs [14]. These binary objects contribute to the table being responsible for a fourth of the used database space. This research will develop a way to externalize the invoice files to a separate storage. The retrieval of the files will therefore be separated from the database structure, meaning that all calls for inserting and searching invoices will be done to a separate file service.

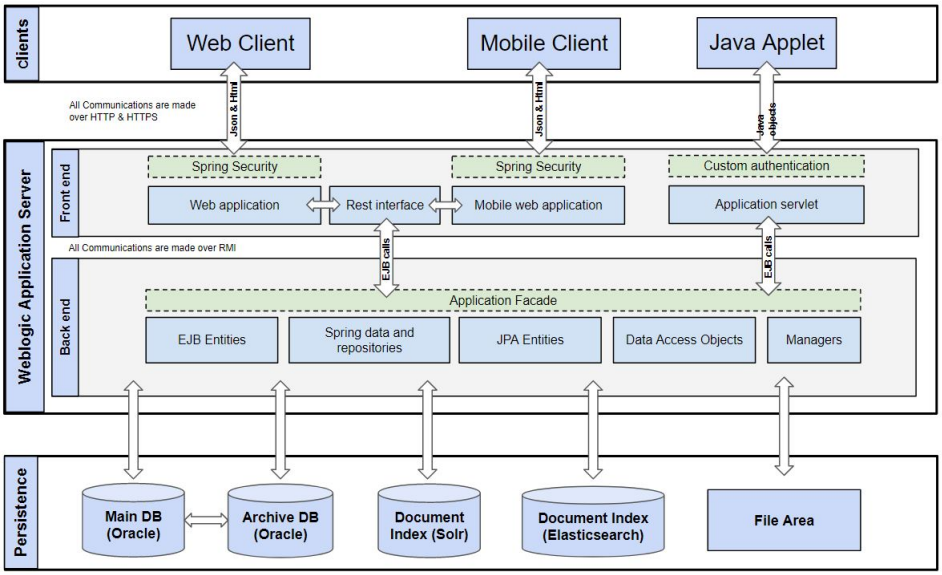

Figure 4.1: The current architecture of the Proceedo application.

4.1.1 Oracle Database and SAN storage

Proceedo currently use a solution which retrieves the files from the database based on its metadata. The database that is used is an Oracle Database which is a relational database. The database is hosted by Oracle and is ran on an Oracle SAN storage.

A *storage area network* (SAN) is a network with the primary purpose of transferring data between computers and storage elements. The network is what attaches servers and storage devices, often called the "network behind servers" [9]. Using an Oracle SAN storage is currently not a highly profitable solution due to the high costs of storage.

Since all attachment files related to e.g. invoices are stored in the database, a single table for the files containing the binary objects take up approximately 20 percent (around 3TB) of the used database space. This table is the major reason to the scaling of the database and therefore it is considered to move the binary files to an external space. The stakeholder is currently in a process where most solutions are preferred to be on public cloud services to guarantee availability and decrease maintenance, which is why it was proposed to migrate the files of the database to a public cloud storage.

4.2 Software requirements

During the meetings with the stakeholder requirements were gathered to be able to plan the implementation of the system. The requirements were divided into functional and non-functional requirements that the stakeholder specified for the system. With concern to the quality assurance factors presented in section 3.4, these were specified as following:

- Functional requirements:
  - Elasticsearch should be used for indexing/retrieval of file metadata.
  - Elasticsearch should be made aware of all changes in the public cloud storage.
  - The system should be able to provide stable performance results.
- Non-functional requirements:
  - The implemented system should be scalable to improve indexing and search performance for the measured metrics.
  - The implemented system should be replicable for demonstration purposes.
  - Since the handled data is confidential, the system is to be implemented within bounds of the stakeholder's network and should not be available for public use.

The requirements will be used to guarantee that the implemented system meets the demands of the stakeholder with a certain level of quality.

4.3 Choice of cloud provider

The *service licence agreement* (SLA) states what service qualities the client and the cloud provider agree upon. The stakeholder is firstly looking for a cloud service provider with low storage costs. Other attributes that can be compared between cloud providers are latency and throughput. Availability can be compared in form of up-time but will be insignificant due to the almost identical guarantees of the cloud

providers that are to be compared.

The current solution for storing files is not suitable. A solution to this would be finding a suitable cloud storage for the externalized files. Public cloud storages can offer several features such as quality, availability and responsibilities [33].

These important features are included in a service licence agreement which is vital when choosing cloud storage provider. The stakeholder is restricted to using a public cloud storage where the storage is within Sweden due to legal aspects, therefore Amazon Simple Storage Service a.k.a Amazon S3 was chosen as a cloud storage provider [34]. Amazon has not yet released Sweden as a region for their web services but since the implementation is currently only investigating the matter the stakeholder had no further concerns regarding this [35]. Since none of the other larger cloud providers (IBM, Microsoft or Google) currently have or will have Sweden available as a region they will not be considered. It is also insignificant to the research question.

4.4 System components

To allow indexing of unstructured data such as files from a relational database, these will firstly have to be downloaded in a generalized manner. The second step is to index the files based on their metadata and upload them to a cloud storage. Finally, these will have to be made searchable, meaning that they can be searched for to later be retrieved. The system that is to be implemented will therefore be divided into two main parts; indexing and search. The components used for the system were chosen based on the stakeholder's previous knowledge of the components.

4.4.1 Requirements analysis

Two components have previously been discussed in the research; Elasticsearch and Amazon S3. Elasticsearch will be used to index the files that have been stored on the public cloud storage. This assumes that Elasticsearch is always aware of file uploads to the cloud storage. This relates to the first question stated in section 1.4 and also one of the functional requirements in section 4.2.

Amazon S3 allows for notifications to be sent whenever changes occur a storage unit [36]. Currently, Amazon S3 provides event notifications for uploaded files and removal of files. The events can be used to notify the Elasticsearch service about a new uploaded file to be indexed. Notifications for removal of objects are not important for the research since it is only indexing and search that will be measured for performance. Handling events for removal of objects will therefore not be implemented.

Replication of the research will be feasible by simplifying the deployment of necessary software. Section 4.4.2 will discuss *software containers* and how these simplify the deployment phase.

4.4.2 Docker

Docker is a software container platform used for managing and deploying containerized applications using LXC containers [37, 29]. Applications that are containerized can be deployed and managed easily and uniformly with the abstraction that Docker provides. Some terms that need explanation [38]:

- *Dockerfile* - contains configuration details about resources needed to create the customized application. A description of how the image should look.
- *Docker Image* - a snapshot of an application created by using a Dockerfile. These are stored in the Docker Registry.
- *Docker Container* - the standard isolated unit in which an application is packaged with all necessary libraries. The Docker Engine reads a Docker Image to run the Docker Container.
- *Docker Engine* - the container runtime containing orchestration, networking and security that is installed on the host.
- *Docker Registry* - the service containing all stored Docker Images. Docker Images can be named using tags to manage serveral versions of the same image.

Figure 4.2 shows Docker containers that are ran on underlying infrastructure independent of the underlying operating system thanks

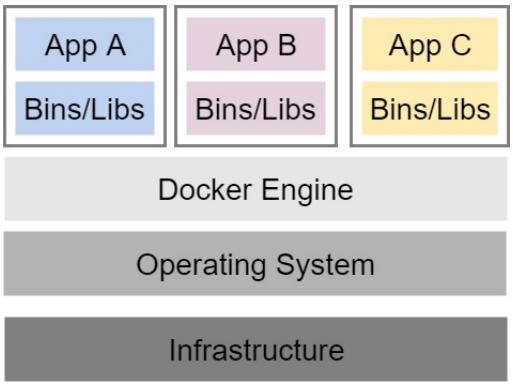

Figure 4.2: A visualization of how the Docker Engine runs on top of an operating system.

to the Docker Engine. Docker containers increase agility by allowing developers to quickly patch service-based applications. According to Docker this has proven to increase the deployment rate to be 13 times faster [38]. Portability is also increased as Docker containers can be ran anywhere a Docker Engine can be installed, which is currently a wide range of operating systems. Developers also gain better control of the applications. Docker Containers can be deployed anywhere with exact configurations using Dockerfiles and Docker images.

Throughout the research, containerized services will be used as long as it satisfies the requirements of the desired system. Containerization of these services will help isolating the parts of the application during performance tests, since only a few of the implemented components will be tested. Docker allows users to configure the containers uniformly among services, this helps creating a reproducible environment.

The used software presented in sections 4.4.3, 4.4.4 and 4.4.5 all have separate Dockerfiles used to build images configured specifically for them, excluding configurations described in section 4.7.4.

Docker Compose is also distributed by Docker Inc and extends Docker. It simplifies the deployment of multi-container Docker applications with easily manageable configuration [39]. Docker Compose will be used to set up and tear down all Elasticsearch nodes since they will be

varying in both amounts and configurations during the performance tests.

4.4.3 Minio

Minio is an Amazon S3-compatible private cloud storage suitable for unstructured data [40]. Choosing Amazon S3 as a cloud storage provider includes conducting the tests on it as well. Using Amazon S3 is not free and since the research is not funded, Minio will be used as an alternative. The architecture of Minio is very similar to Amazon S3 and provides an API with the same functionalities. This will allow a self-hosted cloud storage provider. Minio allows setting up a cluster of nodes in its architecture to provide robustness. A brief explanation of Minio terms is necessary for the research:

- *Object* - an entry of data, usually a file.
- *Bucket* - a unit within a node with a collection of data. Usually used to partition data.
- *Node* - describes a running instance of Minio. A storage unit for buckets.

Minio has features for allowing bucket notifications on events [41]. All events from Minio are received as JSON-objects which contain many fields of information such as source, event type etc. However, the information needed for the stakeholder to find their documents is not included in those fields

4.4.4 RabbitMQ

RabbitMQ is a message broker that uses the AMQP protocol to pass messages [22, 42]. What makes RabbitMQ unique is that it is available on all major operating systems and allows the messages to be read in any way as long as they are sent over HTTP or TCP, which decreases dependencies even more.

The messaging model that RabbitMQ uses allows producers to send their messages to an *exchange* and not directly to the queue. The exchange decides what to do with the messages, meaning it could send them to specific queues for example or discard them.

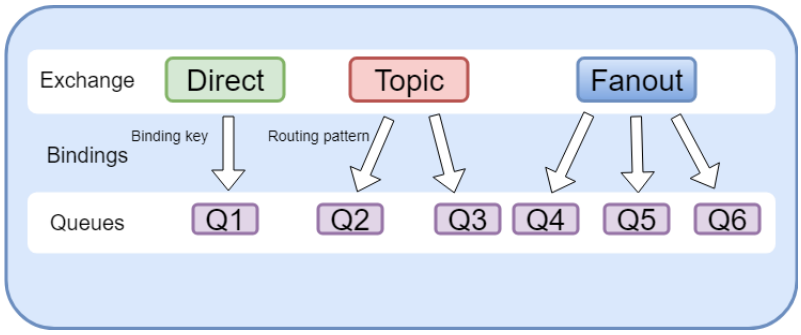

Figure 4.3: The three types of exchanges that RabbitMQ offers.

Figure 4.3 shows how the exchanges in RabbitMQ work. A *direct*-exchange binds to the queues it wants to send messages to (point-to-point). The *topic*-queue routes messages depending on their topic, this based on information provided from the message (publish-subscribe). The last one is the *fanout*-exchange which indiscriminately passes all messages to all known queues.

For this research the simplest functionality was chosen for the exchange, fanout. The fanout-exchange send the incoming messages to all known queues. In this research we will only need one queue so this setting is fine.

The purpose of using RabbitMQ for the research is adding an extra step before indexing the event to include additional metadata fields. The client for RabbitMQ can be written in Java which is known for its benefits in portability and is further described in section 4.7.2.

4.4.5 Kibana

Kibana is a complementary software distributed by Elastic to provide visualization of Elasticsearch [43]. Kibana visualizes the architecture of the Elasticsearch cluster as well as the withheld data with further information such as index and search rate.

Kibana offers an interface for interacting with the connected Elasticsearch cluster which will be used to verify the basic functionalities of the cluster. As it also visualizes the health (CPU, memory, I/O and storage usages) of the cluster it will be used to monitor the Elasticsearch cluster during the tests.

4.5 Test setup

This section mentions the different modules of the system that will be part of the file service. The modules will be tested together that will help prove that their are functional enough to take on their assigned roles for the system architecture. A test setup will also be done to test the performance of the current solution, the Oracle database.

The metrics that are to be tested for the full system are *indexing* and *search* time. These metrics will be compared against the current solution that the stakeholder uses to answer research question.

4.5.1 Indexing

To make the files searchable they have to be indexed into Elasticsearch. The procedure has several steps to it but can be simplified into a visual representation as presented in this section. Indexing speed is interpreted as described in section 2.4.1.

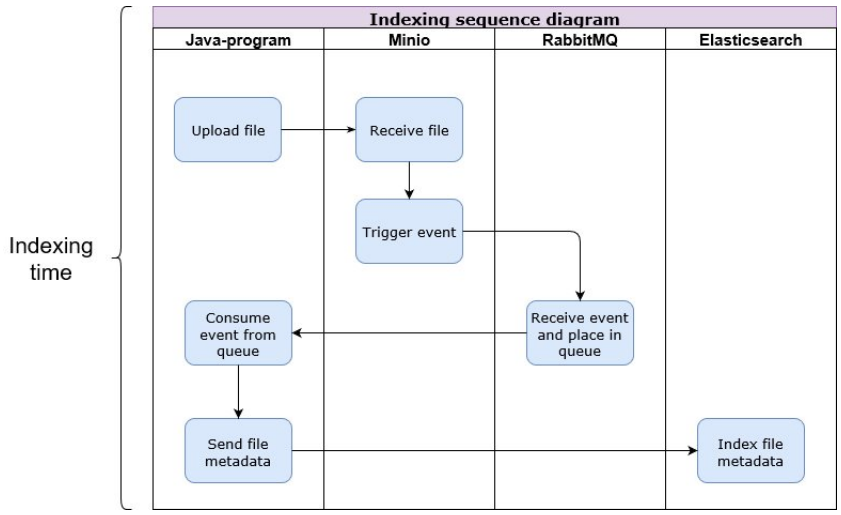

Figure 4.4: Sequence diagram representing the indexing of files.

Figure 4.4 presents the process of indexing files into Elasticsearch. This is done by first uploading the file to Minio which causes a triggered event sent to RabbitMQ. A Java-client will consume the produced event from the specified RabbitMQ queue and post the index as a HTTP-request to Elasticsearch with metadata included.

4.5.2 Search

Retrieving a file from the Minio-storage is done by specifying the bucket and object name. Elasticsearch will contain information enough to find the files on the cloud storage. The search will be done by sending a search request to the Elasticsearch node and retrieving the necessary information for finding them in Minio. Since the time taken downloading the file is irrelevant, the file will only be found on the cloud storage for the search to be considered complete.

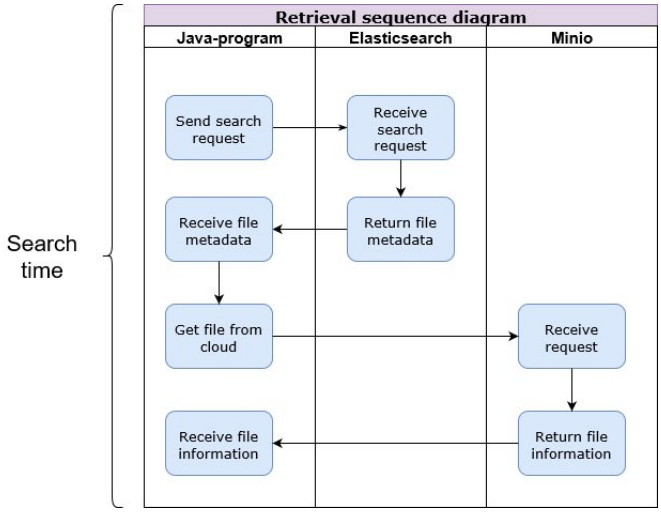

Figure 4.5: Sequence diagram representing the retrieval of files.

Time taken for *searching* is therefore the time taken before receiving a reply from Elasticsearch plus the time taken finding the file in Minio. Figure 4.5 visually explains the process of retrieving files from the cloud storage.

4.5.3 Test data

The data used for the tests is provided by the stakeholder. The Oracle database that is currently being used, as explained in Section 4.1.1, has a replica used for testing. The table which contains the files is called *APL_FILE* and is an actual replica of the file table that the stakeholder is planning to migrate. The table has several fields and some of these will be chosen as metadata for the files to be indexed. The data was collected using software that downloaded and named the file according to a chosen naming convention. The naming convention is used

to transfer the metadata with the files to be able to index them accordingly on the Elasticsearch server. The file name will contain the metadata fields and will look as following:

A_B_C_D_E.G

The fields A, B, C, D and E are metadata fields retrieved from the database. G is the file extension name which also needs to be exported.

Simple SQL statements were used to get basic statistics of the table. The files are ranging between approximately 3 B to 27 MB in the test table. For simplification, the average file size of the database files was calculated and was 77,379 bytes. We will round the average file size to 77 kB.

4.5.4 Performance metrics

The performance of the implementation will be a time measure of the search and indexing time. The results of the tests are supposed to support that Elasticsearch can indeed be used to index unstructured data such as files on a public cloud storage.

4.6 Setting up the services

The services given for the experiments are; Minio, RabbitMQ, Elasticsearch and Kibana. All mentioned services are available for deployment through Docker as containers. This simplifies the deployment phase while also allowing the tests to be reproduced with similar characteristics. The configurations of the services including the Docker configurations will be further explained for each service.

4.6.1 Minio

One Minio node was setup for the tests. Minio was configured by specifying a configuration file. The configuration set static authentication credentials to the Minio service to avoid the randomized credentials which require the connecting services to reconfigure for each deployment. Using a Dockerfile, a customized Docker image fitting the purposes for this test was built.

Enabling bucket notifications required using a provided Minio client.

To enable bucket notifications the the built-in Simple Queue Service (SQS) must be enabled for specific events. [44, 45]. An event is triggered when one of the pre-defined operations are performed. Since Minio is used instead of Amazon S3 only the events supported by both are considered, which are: *s3:ObjectCreated:Put* and *s3:ObjectRemoved:Delete*. As stated in section 4.4.3 only added objects will need to trigger notifications for the tests. Therefore only *s3:ObjectCreated:Put* will be enabled. The messages are sent using the AMQP protocol.

4.6.2 RabbitMQ

A single node of the RabbitMQ service was started within a Docker container. No further configurations were done for the service to function. The purpose of the RabbitMQ service is to receive the published event notifications.

4.6.3 Interaction between Minio and RabbitMQ

The Minio node is connected to the RabbitMQ service as a way of publishing event notifications. RabbitMQ receives the events in form of JSON-objects as shown in appendix A that are pushed to the queue. The queue allows for multiple subscribers to consume the messages, which include information regarding the PUT-operations performed on the Minio node. Information such as the *object key* can be extracted from the messages which is enough metadata to later retrieve the object from the Minio storage using a GET-operation.

To summarize, this first part of the system activates event notifications from the Minio node that can be published to the RabbitMQ queue and thereafter read by the RabbitMQ Java client which is further explained in section 4.7.2.

4.6.4 Elasticsearch

Elasticsearch will be used for indexing the files on the storage. The messages provided to RabbitMQ from Minio lack information necessary to do searches on metadata relevant to a user of the full system. Elasticsearch allows for adding extra fields to any index entry using its client, which is where metadata from the file name will be added and indexed properly.

The Elasticsearch service is the one that does the actual work of the implementation. The process of indexing within Elasticsearch is demanding and time consuming, while searching is less time consuming. Elasticsearch is deployed using Docker Compose to allow simple manageability for scaling up the service but also for scaling it out. Memory swapping is disabled for all Elasticsearch nodes as recommended by the documentation to increase indexing speed [46].

Scaling up

According to their documentation, Elasticsearch can be scaled up by increasing its heap size [47]. The documentation implies that allocating more memory for the heap will increase the performance of the entire Elasticsearch node. Too large heap spaces are not recommended, but less than 50% of the total memory can be given to the heap. Scaling up the Elasticsearch node for this research will be done by increasing the provisioned memory with 50% of it allocated for the heap. This means that a node with a 2 GB heap size needs at least 4 GB memory.

Scaling out

The Elasticsearch documentation gives information of how scaling out the service can increase the indexing and search performance by adding several nodes to the Elasticsearch cluster [48]. By scaling out the service, shards can be distributed among nodes and gain more resources for processing requests. The stakeholder had hardware with capability of scaling out the cluster to a total of three nodes.

4.7 Java Clients

The Java clients used for the system acted as middle-ware and for simulation of the incoming files. They were complementary steps necessary for a fully functional system as required. The services described in previous sections are the servers that the Java clients connected to. The Minio client is used for storing and retrieving files from the Minio node. The RabbitMQ client is used to consume messages from the queue to which the Minio publishes events. The Elasticsearch client

is used to index metadata to the Elasticsearch server and making the files available for search.

4.7.1 The file uploader

The Minio client was used to load files from the memory and upload them to the storage node. Minio provides a Java client for this purpose and this is what was used to perform PUT and GET operations. Amazon published a guide to considerations for increased performance of uploading files to their Amazon S3 storage which were applicable to Minio as well [49]. These hints, which stated that reads and writes are blocking operations, were considered when implementing the Minio client. Blocking operations only allow sequential execution in directories which nullifies the effects of implemented parallelism in the Minio client.

However, if the files were put in different directories (called *prefixes* in S3 since a bucket is flat and has no actual structure) the parallelism increases the performance of writes to the Minio storage. This helps the system avoid bottlenecks in file uploads which is the first step in the system process.

The client for uploading files to the Minio storage is named *file uploader*. These clients can be parallelized as long as they upload files with different prefixes to the same bucket, which are file names separated with backslashes. The file uploaders are parallelized which increases the rate of produced notifications to the RabbitMQ. For retrieving files found on Elasticsearch, a *file searcher* is implemented. The file searcher can be parallelized to retrieve multiple files from the storage at once.

4.7.2 The event receiver

The RabbitMQ client was implemented using Java. The client was implemented as a consumer which reads messages published from the declared queue. The incoming JSON-objects are modified to add metadata before indexing them to the Elasticsearch server using the Elasticsearch client.

The compound of these two clients is named the *event receiver*. An event receiver is used to read from a queue, add data and index it in Elasticsearch. The event receiver can be parallelized, meaning that it reads faster from the RabbitMQ queue while also allowing faster indexing to the Elasticsearch server. This is recommended to enhance the indexing speed on Elasticsearch [46].

4.7.3 The file searcher

File searching will be done by searching in Elasticsearch and use the retrieved metadata to find the object in Minio. The component responsible for the task is called the *file searcher*. Since it is not stated anywhere in the Elasticsearch documentation that parallelizing the search increases its performance, this will not be implemented [50].

4.7.4 Common configuration settings

The services will be run on a computer provided by Visma. As explained in section 4.4.2 Docker provides containers for deployment of applications. All Docker containers except the ones running Elasticsearch are ran on the same hardware, which is described in appendix B. Elasticsearch will be run on the hardware specified in appendix C. For this to be possible, all containers were configured to listen to the ports of the hosting PC and the ports of the containers were exposed to the ones of the hosting PC. The memory given to all service containers was 4 GB each by default except for the Elasticsearch containers which varied for different tests.

4.8 The file service

Previous sections describe all steps needed to allow indexing and searching files on Amazon S3 using Elasticsearch. The file service has the capability of indexing files from the test database to Minio, but also allows searching for files on Minio through Elasticsearch.

Figure 4.6 shows how the file service has been implemented. The image gives a step-wise description of how the process of indexing and searching is performed. The numbers in black represent the steps taken to index a document while the blue ones describe the search process:

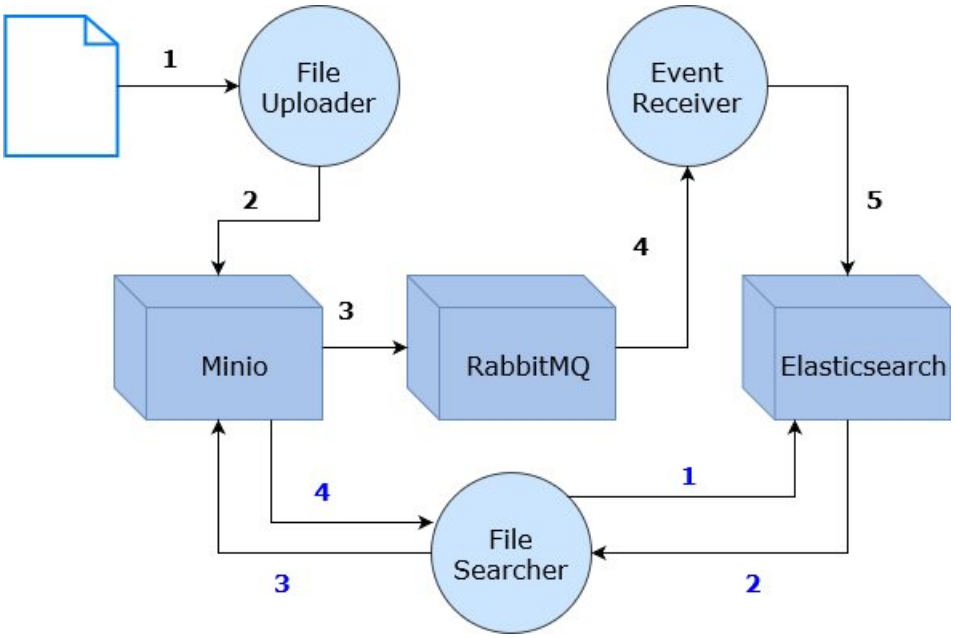

Figure 4.6: The implemented file service with numbered arrows visualizing the flow of events.

The process of indexing

1. A file is retrieved from the database and sent to the file uploader.
2. The file uploader uploads the file to Minio (the cloud storage).
3. The uploaded file triggers an event notification (AMQP message) to RabbitMQ (the message broker).
4. The event receiver consumes the event from RabbitMQ and creates an index request.
5. The index request it sent to Elasticsearch to index the object key and file metadata.

The process of searching

1. The file searcher uses file metadata and creates a search request that is sent to Elasticsearch.
2. Elasticsearch responds with the file information, including the object key.

3. The file searcher invokes a call with the given object key to find the file on Minio.
4. Minio responds with the file if it is found.

Chapter 5

Results

This chapter presents the results of the test cases ran on Proceedo's current solution and the implemented file service. The setup of the file service was previously described to recreate the tests. The purpose of the test cases are further explained to understand why they are of relevance for the thesis.

5.1 Evaluating the file service

The file service was created from the functional an non functional requirements stated in section 4.2 and will be evaluated considering these. Elasticsearch has successfully been implemented to index and retrieve file metadata, making the implementation able to find files on the cloud storage.

Elasticsearch is not aware of all types of changes, but is made aware of all changes of interest for the research. This was verified by successfully searching for all indexed files and getting the index count in Kibana. The message broker will store events for the events occurring in the storage and can therefore guarantee that the messages are delivered to the event receiver.

The file service was deployed using Docker containers which makes the tests highly reproducible in any environment with the Docker Engine installed.

Ethical requirements were followed by only running the services on

the premises of the stakeholder to decrease chances of leaking confidential information.

5.2 Test cases

The chosen test cases were designed to test two types of performance; *indexing*- and *search speed*. The terms were previously explained in section 4.5.1 and 4.5.2. The sections also visualize the whole process of indexing and searching for a file. The whole data set consisted of 4,096 files which averaged on 77 kB per file.

The time was measure using Java's System library which has the *nanoTime()*-function. The recorded time for indexing was a lot slower than for searching and will therefore be presented in *seconds*, while the search is presented in *milliseconds*. All tests were ran 10 times each to verify consistency which was one of the requirements for the implementation.

5.2.1 Oracle Database

The first set of test cases were done on the Oracle Database which is how the stakeholder is currently persisting files. The database was tested for its performance to insert and search for specific files. This is equivalent to indexing and searching files with the new implementation done for the research. The results will be compared to the ones of the file service.

5.2.2 The file service

The second set of test cases tested the performance of the file service. The first tests were done to verify that multi-threading the file uploader, event receiver and file searcher could increase the indexing and search time of the entire file system. Secondly, tests were made to measure the indexing and search performance when the Elasticsearch cluster was scaled up and scaled out, hopefully increasing it.

All files are loaded to the memory before being uploaded to the Minio bucket and starting the indexing process. This helps evaluate the pro-

cess of indexing several files consecutively. The next two test cases gave results regarding the search performance of the file service.

5.3 Test results

The results of the test cases are presented and described below using graphs. The tests are all benchmarks of elapsed time for the indexing and search. These will be discussed further in later chapters.

5.3.1 Multithreading

Results for the multithreaded indexing and search performances.

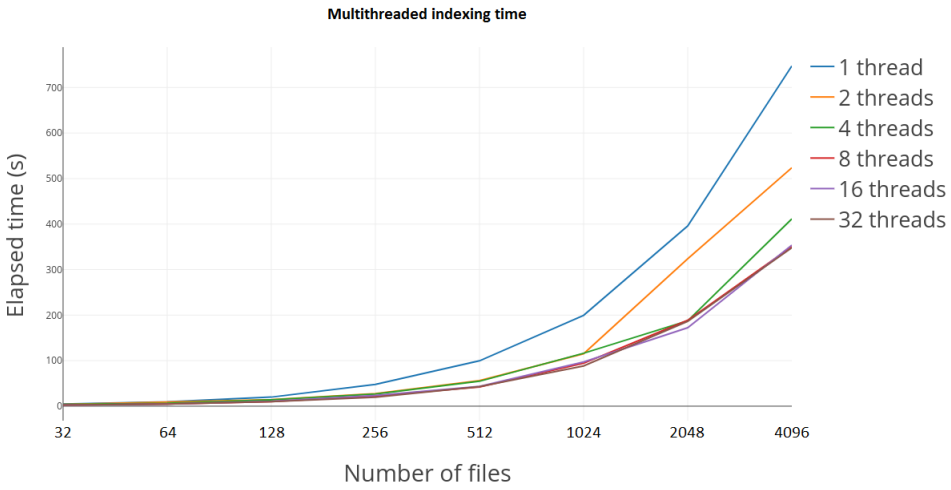

Figure 5.1: Results of the average indexing time run with a varying number of event receiver and file uploader threads and file amounts.

Multi-threaded indexing was tested as presented in Figure 5.1. Each indexing test for file amounts was ran on a different amount of threads for every test. The results show a slight decrease of elapsed time with the increasing amount of threads.

Figure 5.2 shows the performance of searching multi-threaded. Each test searched for a varying amount of files among the 4,096 already in-

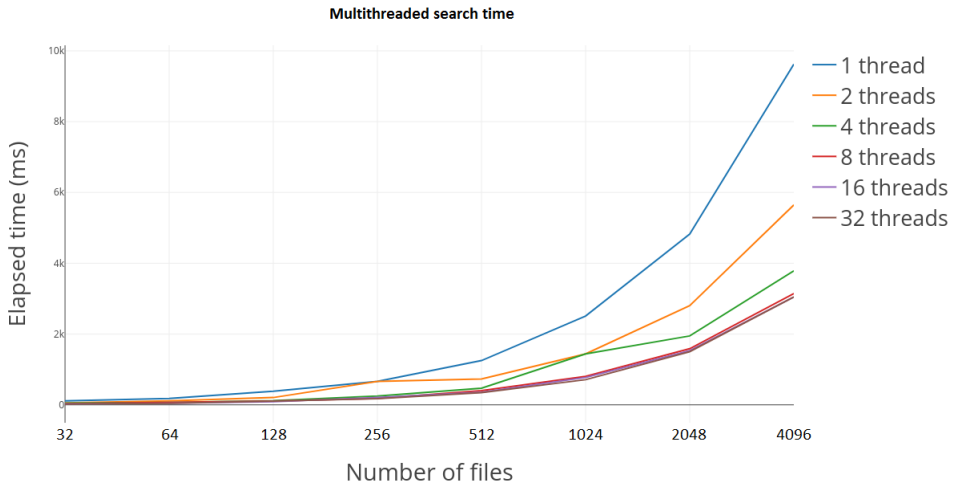

Figure 5.2: Results of the average multi-threaded search times with varying amounts of files and file searcher threads.

dexed files. The results show a decrease search time with the increasing amount of file searcher threads.

5.3.2 Comparing the current solution against the implemented file service

Results of the difference in indexing/insertion and search performance for the current solution in Proceedo using the Oracle Database versus the implemented file service. Elasticsearch was set up using a single node and was given a 1 GB heap size for indexing and searching. 16 file uploaders, 16 event receivers and 32 file searchers were used for following tests.

Figure 5.3 shows a comparison of insertion and indexing times for the current solution versus the implemented file service.

Figure 5.4 shows a comparison of searching times for the current solution versus the implemented file service.

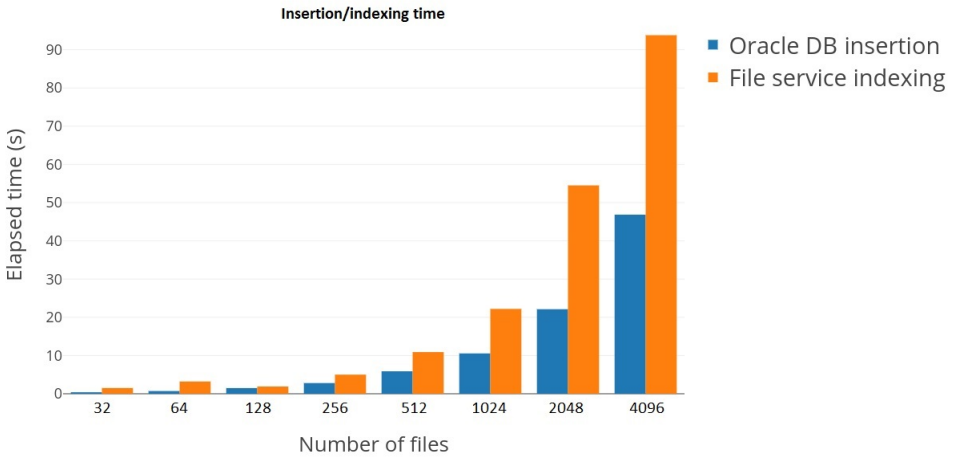

Figure 5.3: A comparison of inserting and indexing files in the current solution versus the implemented file service.

5.3.3 Scaling up Elasticsearch

Section 4.6.4 explains how Elasticsearch can be scaled up. This section gives results of how the file service performed when a single Elasticsearch was provided a larger heap size. 16 file uploaders, 16 event receivers and 32 file searchers were used for following tests.

Figure 5.5 shows results of how the file service performed for the indexing process when given different heap sizes.

Figure 5.6 shows results of how the file service performed for the search process when given different heap sizes.

5.3.4 Scaling out Elasticsearch

Section 4.6.4 explains how Elasticsearch can be scaled out by adding more nodes. This section shows how the file service performed when being scaled out from one to three nodes. 16 file uploaders, 16 event receivers and 32 file searchers were used for following tests.

Figure 5.7 presents results of indexing performance for varying file amounts and amount of nodes.

Figure 5.8 presents results of search performance for varying file amounts and amount of nodes.

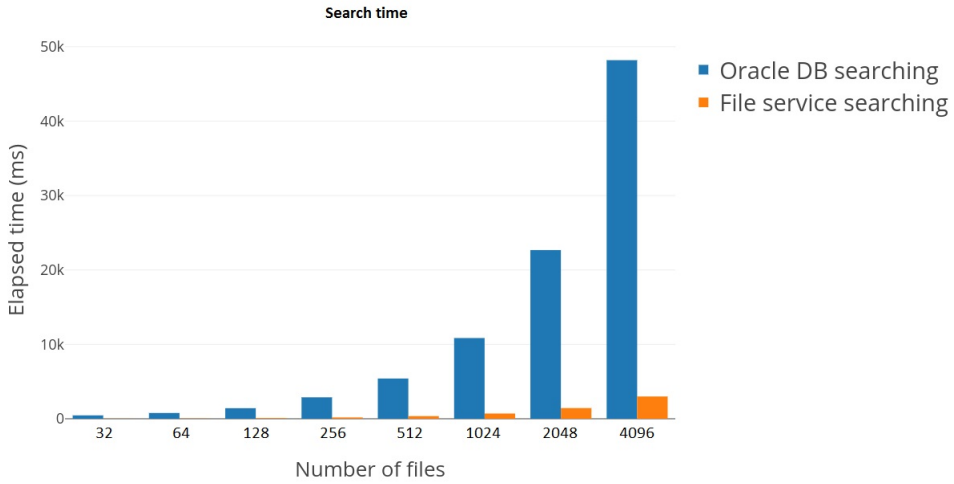

Figure 5.4: A comparison of searching files in the current solution versus the implemented file service.

Summary

The results have been summarized to show the overall differences in performance for all different test cases.

Figure 5.9 shows a summary of the increase/decrease in performance with the file service implemented instead of the current solution. The comparison is done using the best performing setup of the file service, which is using a single Elasticsearch node with 1 GB heap space.

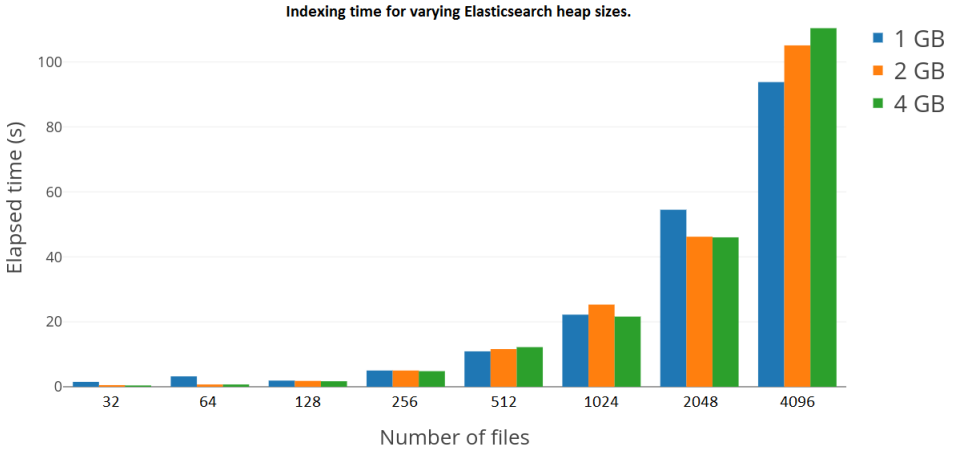

Figure 5.5: Results of indexing performances for varying file amounts and heap sizes.

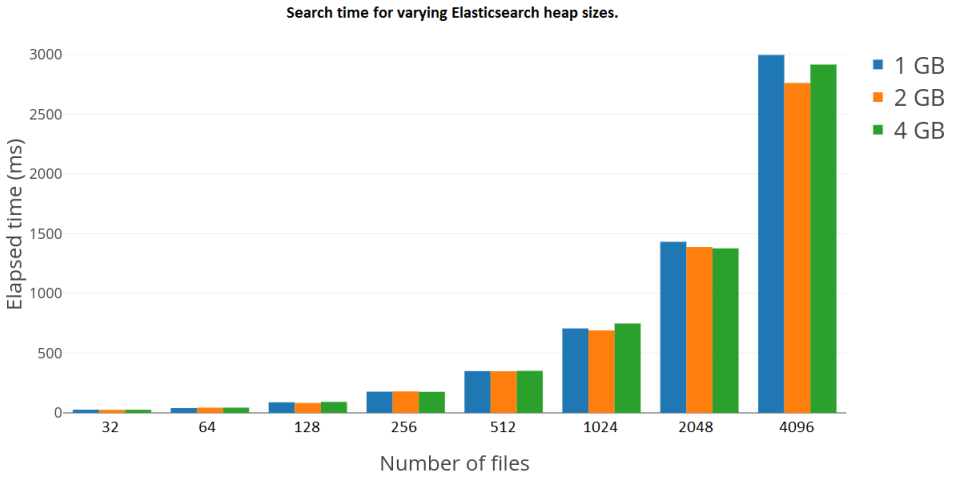

Figure 5.6: Results of search performances for varying file amounts and heap sizes.

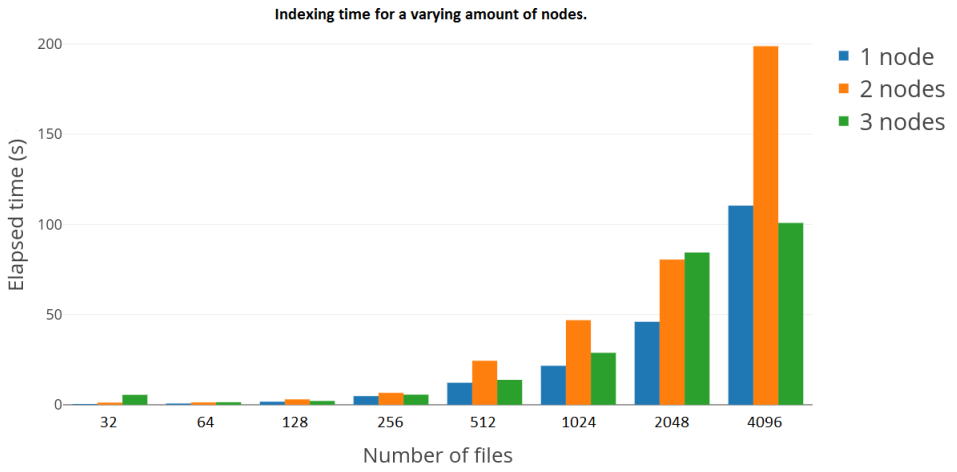

Figure 5.7: Results of indexing performances for varying file amounts and number of nodes.

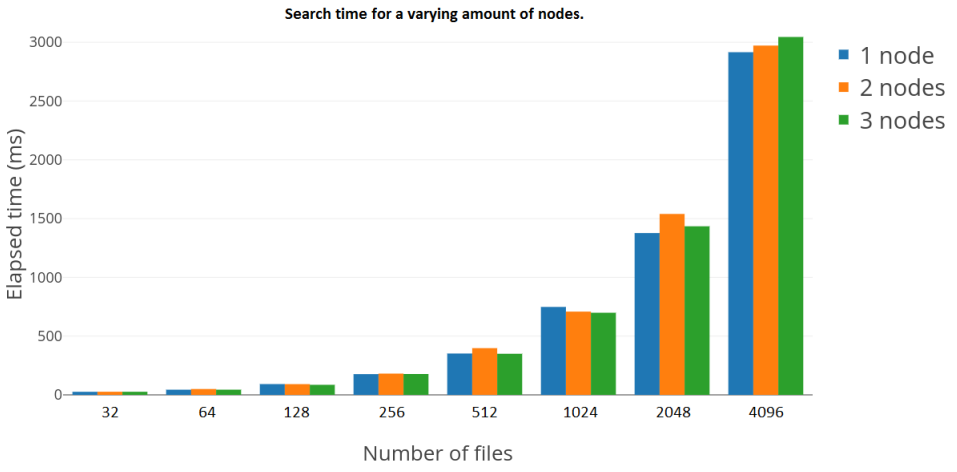

Figure 5.8: Results of search performances for varying file amounts and number of nodes.

Difference	Indexing(%)	Search(%)
32 files	-74	1605
64 files	-78	1820
128 files	-22	1505
256 files	-44	1520
512 files	-46	1445
1024 files	-52	1435
2048 files	-59	1483
4096 files	-50	1509
Average:	-53	1540

Figure 5.9: Summary of performance difference in percentage when using the file service instead of the current solution.

Chapter 6

Discussion

This chapter will discuss the results presented in the previous chapter. An analysis on the results will be done. Considerations regarding the apparent flaws of the system architecture will be brought up and criticized.

6.1 Multithreading

The multithreading showed an increase in indexing- and search performance. The results were used to conduct the other tests to increase tune them for even better performance. The proof in section 5.3.1 showed that multithreading could be used as an improvement for both indexing and searching. In both figure 5.1 and 5.2 there was a clear convergence of decreased indexing and search time, which was for about 16 threads. The amount of threads used for further testing was therefore chosen to be 16 for the amount of file uploaders, event receivers and file searchers.

6.2 Indexing performance

The indexing performance decreased when using the file service. Comparing the indexing performance of the file service to the insertion performance of the old solution showed that the file service is slower when persisting files. Figure 5.3 shows that persisting files in the file service was 74 % slower than the current solution, for the worst case of indexing 32 files. The best case only had a 22 % decrease which was

for indexing 128 files. The average decrease of indexing performance when using the file service was 53 % as shown in figure 5.9.

Indexing rate is calculated by dividing the the amount of files over the elapsed indexing time. The average indexing performance for all tested file amounts gave an indexing rate of about 44 files per second. Since the notifications for new file indexes are sent to a message broker, these can be handled asynchronously assuming the message broker has enough space to store all unprocessed messages. This can allow the indexing to be processed with a steady pace.

6.3 Search performance

The search performance was significantly increased when using the file service instead of the Oracle Database which is shows in figure 5.4. The worst case gave an increase in search performance with 15 times which was for 128 files. The best case gave an 18 times increased search performance which was for 64 files. The average case was around 15 times faster than the old solution.

The search rate for the average case was 1367 found files per second. The old solution gave a search rate of 85 files per seconds, meaning an increase with 1282 files per second.

6.4 Scalability

The research had the hypothesis of increasing the indexing performance when scaling Elasticsearch up or out. To test the scalability of Elasticsearch it was scaled up by increasing its heap size and scaled out by increasing the amount of nodes.

6.4.1 Indexing

The performance of scaling up Elasticsearch showed no increase in indexing time but in fact showed signs of slower indexing performance for some cases. Figure 5.5 shows that the Elasticsearch node with the largest heap size only showed a significant decrease in time for 2/8 test

cases, which was for 1024 and 2048 files. The tests prove that increasing the heap size up to 4 GB is not a way to prove vertical scalability for Elasticsearch. This could be an indication of the tests not being big enough, meaning the heap size should have been increased even further. Since this research only had limited resources larger tests were not feasible.

When scaling out the service there was no further improvement in indexing performance. Figure 5.7 shows that using one node showed better performance results than when using two or three nodes. The distribution of shards and replicas may have affected this. This shows that scaling out was not a feasible alternative to gain scalability for this implementation. This may have been dependent on the setup of the Elasticsearch cluster.

6.4.2 Search

The search performance showed no significant increase in performance when the Elasticsearch node was scaled up. Figure 5.6 showed that the case where the heap size was 2 GB showed best results when looking over the different cases. When increasing the heap size to 4 GB the search time increases for the case with 4096 files.

Scaling out Elasticsearch did not prove to enhance the search performance either. Figure 5.8 shows no decrease in search time when increasing the amount of nodes. Each of the nodes had a 4 GB heap space which is the largest scale of the scaled up services. The cluster had a maximum of 12 GB heap size and a total memory allocation of 24 GB across all nodes.

6.5 File service evaluation

The file service is a module that can be attached to replace the existing solution for persistence. The file service is evaluated according to section 5.1 and will be criticized.

6.5.1 Infrastructure

The implemented file service uses several services to function. This creates a chain of dependencies between services. All services are deployed as Docker containers which simplifies the deployment phase as explained in section 4.4.2. Deploying multiple containers for the same service is simpler than otherwise when using Docker. The infrastructure of the file service had no guarantees of retaining data when suffering from service failures. A crashed message broker could for example cause files to be uploaded without triggering notifications for indexing files which can not be re-created. Security which was delimited was not considered at all when conducting the tests, which would be vital for further developing the file service since the handled information will be confidential.

6.5.2 Shard and replica distribution

Elasticsearch was tested nearly "out-of-the-box", meaning no other settings except for the ones mentioned in section 4.6.4 were altered. Shards and replicas are automatically distributed by Elasticsearch without any configurations. The cases where Elasticsearch was scaled up had no change in shard and replica distributions. Scaling out the service on the other hand resulted in different compositions for the shards and replicas. For one node there were five shards and five replicas on that node. When using two nodes, five shards were on the first node and all replicas were on the second node. For three nodes, four shards were on the first node with one replica. The second node had five replicas and the third node contained one shard and four replicas.

The tests show that two nodes gave worse performance than a single node. One cause can be the lack of memory. The other reason might be that replicating the shards when indexing is more time consuming when they are on separate nodes, since the results for using three nodes gave better results than two nodes but still equal to the results for when using a single node.

6.6 Memory deficiency

The hardware used for running the test cases are specified in appendix B and C. They did not allow for several instances of each service due to lack of memory. The hardware running Minio, RabbitMQ, Kibana and the Java clients had 12 GB memory and provided a maximum of 2GB for each of the three service containers, resulting in 8GB maximum allocated memory. This gave no more room for running the Elasticsearch service which is why two computers were used. The other hardware was only responsible for running the Elasticsearch service since more memory was needed for scaling it up and out.

Testing the Minio and RabbitMQ services separately gave performances higher than when adding Elasticsearch at the end of the chain. Minor tests on Elasticsearch which are excluded from the thesis such as indexing the same file infinitely gave approximately the same results as the results given in section 5. The suspicions were strengthened when reading the tips presented for production environment configurations written by the Elastic team [47].

Elastic describe that using a heap size of less than 4GB is not recommended and could give bad performance results. The same guide advices to not set the heap size larger than 50% of the total allocated memory. When providing the Elasticsearch node with 4GB memory and a heap size with half the memory size as recommended no improvement was seen. The reason for scaling up Elasticsearch was to try decreasing indexing and search time with a larger heap size. Related work proved that increasing allocated memory increased performance, but since both heap size and memory was increased for this research this might be a case where the heap size was always too large [51]. To summarize, memory given to the scaled up service was always double the heap size which could have been too little.

Chapter 7

Conclusions

The research successfully answered the problem of the thesis. All results were not very satisfying but proved that the proposed solution was indeed possible. The implementation was lacking resources such as memory. This regarded in results that were less useful for the stakeholder. The file service showed results of worse indexing performance but a significantly improved search performance. Since the implementation was only a prototype for how indexing and searching using Elasticsearch on a public cloud storage could be solved, this could have been expected. The performance of the file service is highly dependent on how Elasticsearch is set up, which is why it had its scalability tested.

The service was scaled vertically and horizontally to try increasing its performance, which was not successful. Scaling the file service is possible but might require a cluster with plenty of more memory than provided for the research to show satisfying improvements of performance. The results for scaling Elasticsearch in this research showed no apparent improvement in indexing or search performance.

The file service was only used for indexing and retrieving but could have been used for several more purposes such as removing data and logging. One of the goals was to find out if it was possible to make Elasticsearch aware of all changes in the cloud storage relevant for indexing and search which it was when setting the bucket notification triggers to your specific needs.

Future work

The research came to a conclusion stating that the approach is valid with enough resources. Future research in this are should consider issues such as setting up services correctly with a purpose to maximize the performance results. This thesis tested the performance of the experimental approach that was put together from what was given. Scaling the service could be done with more memory provided to a point where the improvements are visible. Elasticsearch is often set up on a cluster of computers so it would be interesting to see how well Elasticsearch performs when nodes are distributed over a network to see at which point the bandwidth is becoming a limitation for the node communication.

Certain functionalities were missing which does not make the file service aware of all changes occurring in the public cloud storage but only for added objects. It would be interesting to see how other types of bucket notifications which are triggered by other events than added objects are implemented. Allowing events for removal of objects for example seems like a fundamental function that already should have been implemented, which there was no time left over for. The main problem was indexing and searching for files in the cloud storage, meaning that adding removal of objects would not help solving the problem of this thesis.

Bibliography

[1] Jesús Bisbal et al. "Legacy information system migration: A brief review of problems, solutions and research issues". In: *Computer Sciencs Department, Trinity College, Dublin, Ireland* (1999), pp. 1–18. URL: http://www.cs.tcd.ie/publications/tech-reports/reports.99/TCD-CS-1999-38.pdf.

[2] RightScale Inc. *Cloud Computing Trends: 2017 State of the Cloud Survey*. 2017. URL: https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey (visited on 06/17/2018).

[3] 451 Research. "Can private cloud be cheaper than public cloud ?" In: June (2017). URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vrealize-suite/vmware-paper1-can-private-cloud-be-cheaper-than-public-cloud.pdf?src=so_5703fb3d92c20&cid=70134000001M5td.

[4] IBM. *IBM Knowledge Center - Challenges when migrating applications*. 2018. URL: https://www.ibm.com/support/knowledgecenter/en/SSEP7J%7B%5C_%7D10.2.1/com.ibm.swg.ba.cognos.ug%7B%5C_%7Dmfdm.10.2.1.doc/c%7B%5C_%7Dmfdm%7B%5C_%7Dchal%7B%5C_%7Dmig.html (visited on 06/19/2018).

[5] Oracle Corporation. *A Relational Database Overview*. 2017. URL: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html (visited on 06/08/2018).

[6] Russell Sears, Catharine Van Ingen, and Jim Gray. "To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?" In: (2007), pp. 1–11. arXiv: 0701168 [cs]. URL: http://arxiv.org/abs/cs/0701168.

[7] RightScale Inc. *Cloud Computing Trends: 2017 State of the Cloud Survey*. 2017. URL: https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey (visited on 06/17/2018).

[8] Visma. *Om Visma*. 2018. URL: https://www.visma.se/om-visma/ (visited on 02/20/2018).

[9] Tate Jon, Fabiano Lucchese, and Richard Moore. "Introduction to Storage Area Networks". In: *Contract* (2006), p. 352. URL: http://itcertivnetworking.riverinainstitute.wikispaces.net/file/view/Red+Book+-+What+is+a+SAN.pdf.

[10] Margaret Rouse. *Definition from WhatIs.com*. 2006. URL: https://searchstorage.techtarget.com/definition/content-addressed-storage (visited on 04/18/2018).

[11] European Union. "Proposal for a Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation)". In: 2015.June (2015), pp. 1–201.

[12] Kungliga Tekniska Högskolan. *Ekonomisk hållbarhet | KTH*. 2018. URL: https://www.kth.se/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/sustainable-development/ekonomisk-hallbarhet-1.431976 (visited on 02/20/2018).

[13] Anne Håkansson. "Portal of Research Methods and Methodologies for Research Projects and Degree Projects". In: *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering FECS'13* (2013), pp. 67–73. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-136960.

[14] Sharpened Productions. *Blob (Binary Large Object) Definition*. 2018. URL: https://techterms.com/definition/blob (visited on 04/16/2018).

[15] Margaret Rouse. *What is cloud storage? - Definition from WhatIs.com*. 2016. URL: http://searchstorage.techtarget.com/definition/cloud-storage (visited on 03/14/2018).

[16] Rajkumar Buyya, James Broberg, and Andrzej Goscinski. *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, Inc., 2011. ISBN: 9780470887998. DOI: 10.1002/9780470940105. arXiv: 1003.4074. URL: https://books.google.se/books?id=S1NvRRd77rQC%7B%5C&%7Dlpg=PT22%7B%5C&%7Dots=HTibo8Xu2h%7B%5C&%7Ddq=Cloud%20Computing%7B%5C%%7D3A%20Principles%20and%20Paradigms%7B%5C&%7Dlr%7B%5C&%7Dpg=PT57%7B%5C#%7Dv=onepage%7B%5C&%7Dq%7B%5C&%7Df=false.

[17] PCMag Digital Group. *SaaS Definition from PC Magazine Encyclopedia*. 2018. URL: https://www.pcmag.com/encyclopedia/term/56112/saas (visited on 03/14/2018).

[18] PCMag Digital Group. *PaaS Definition from PC Magazine Encyclopedia*. 2018. URL: https://www.pcmag.com/encyclopedia/term/61404/paas (visited on 03/14/2018).

[19] PCMag Digital Group. *IaaS Definition from PC Magazine Encyclopedia*. 2018. URL: https://www.pcmag.com/encyclopedia/term/61344/iaas (visited on 03/14/2018).

[20] Stephan Steglich and Stefan Arbanowski. *Middleware for communications*. John Wiley & Sons, Ltd, 2004, pp. 2530–2531. ISBN: 0470862068.

[21] OASIS. *AMQP is the Internet Protocol for Business Messaging | AMQP*. 2018. URL: https://www.amqp.org/about/what (visited on 03/16/2018).

[22] Mahesh Singh. *RabbitMQ: Understanding Message Broker*. URL: https://www.3pillarglobal.com/insights/rabbitmq-understanding-message-broker (visited on 03/16/2018).

[23] W Bruce Croft, Donald Metzler, and Trevor Strohman. "Information retrieval in practice". In: (2009).

[24] David Beaumont. *How to explain vertical and horizontal scaling in the cloud - Cloud computing news*. 2014. URL: https://www.ibm.com/blogs/cloud-computing/2014/04/09/explain-vertical-horizontal-scaling-cloud/ (visited on 04/25/2018).

[25] Elasticsearch BV. *Elasticsearch: RESTful, Distributed Search & Analytics*. 2018. URL: https://www.elastic.co/products/elasticsearch (visited on 02/20/2018).

[26] The Apache Software Foundation. *Apache License, Version 2.0*. URL: http://www.apache.org/licenses/LICENSE-2.0 (visited on 02/20/2018).

[27] The Apache Software Foundation. *Apache Lucene - Apache Lucene Core*. 2016. URL: https://lucene.apache.org/core/ (visited on 06/08/2018).

[28] Elasticsearch BV. *Basic Concepts | Elasticsearch Reference [6.2] | Elastic*. 2018. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/%7B%5C_%7Dbasic%7B%5C_%7Dconcepts.html (visited on 03/15/2018).

[29] Scott Hogg. *Software Containers: Used More Frequently than Most Realize | Network World*. 2014. URL: https://www.networkworld.com/article/2226996/cisco-subnet/software-containers--used-more-frequently-than-most-realize.html (visited on 03/28/2018).

[30] Centers for Medicare & Medicaid Services. "Selecting a development approach". In: *Centers for Medicare & Medicaid Services* (2008), pp. 1–10. ISSN: 09205489. DOI: 10.1016/j.csi.2016.06.003. URL: http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf.

[31] Association of Modern Technologies Professionals. *Software Development Methodologies*. 2018. URL: http://www.itinfo.am/eng/software-development-methodologies/ (visited on 06/05/2018).

[32] Kent Beck et al. "Manifesto for agile software development". In: (2001).

[33] Philipp Wieder et al. *Service Level Agreements for Cloud Computing*. SpringerLink : Bücher. Springer New York, 2011. ISBN: 9781461416142. URL: https://books.google.se/books?id=z306GUfFL5gC.

[34] Amazon Web Services Inc. *Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service*. 2018. URL: https://aws.amazon.com/s3/ (visited on 03/16/2018).

[35] Jeff Barr. *Coming in 2018 – New AWS Region in Sweden | AWS News Blog*. 2017. URL: https://aws.amazon.com/blogs/aws/coming-in-2018-new-aws-region-in-sweden/ (visited on 03/14/2018).

[36] Inc. Amazon Web Services. *Configuring Amazon S3 Event Notifications - Amazon Simple Storage Service*. 2018. URL: https://docs.aws.amazon.com/AmazonS3/latest/dev/NotificationHowTo.html (visited on 06/07/2018).

[37] Docker Inc. *Get Docker | Docker*. 2018. URL: https://www.docker.com/get-docker (visited on 03/28/2018).

[38] Docker. "The Definitive Guide To Docker Containers". In: (2015). URL: https://www.docker.com/sites/default/files/WP-%20Definitive%20Guide%20To%20Containers.pdf.

[39] Docker Inc. *Docker Compose | Docker Documentation*. 2018. URL: https://docs.docker.com/compose/ (visited on 03/28/2018).

[40] Inc. Minio. *Minio Features*. 2018. URL: https://www.minio.io/features.html (visited on 03/16/2018).

[41] Minio Inc. *Bucket Notification Guide*. 2018. URL: https://docs.minio.io/docs/minio-bucket-notification-guide (visited on 03/16/2018).

[42] Pivotal Software Inc. *RabbitMQ - Messaging that just works*. 2018. URL: https://www.rabbitmq.com/%7B%5C#%7Dfeatures (visited on 03/16/2018).

[43] Elasticsearch BV. "Kibana". In: (2018). URL: https://www.elastic.co/products/kibana.

[44] Amazon Web Services Inc. *Amazon Resource Names (ARNs) and AWS Service Namespaces - Amazon Web Services*. 2018. URL: https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html (visited on 06/08/2018).

[45] Amazon Web Services Inc. *Amazon Simple Queue Service (SQS) | Message Queuing for Messaging Applications | AWS*. 2018. URL: https://aws.amazon.com/sqs/ (visited on 06/08/2018).

[46] Elasticseach BV. *Tune for indexing speed | Elasticsearch Reference [6.2] | Elastic*. 2018. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-indexing-speed.html (visited on 05/25/2018).

[47] Elasticseach BV. *Heap: Sizing and Swapping | Elasticsearch: The Definitive Guide [2.x] | Elastic*. 2018. URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html (visited on 05/18/2018).

[48] Elasticsearch BV. *Scale Horizontally | Elasticsearch: The Definitive Guide [2.x] | Elastic*. 2018. URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/%7B%5C_%7Dscale%7B%5C_%7Dhorizontally.html (visited on 06/09/2018).

[49] Amazon Web Services Inc. *Amazon Simple Storage Service*. 2006. URL: https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html (visited on 04/18/2018).

[50] Elasticseach BV. *Tune for search speed | Elasticsearch Reference [6.2] | Elastic*. 2018. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-search-speed.html (visited on 05/25/2018).

[51] Niklas Ekman. "Handling Big Data using a Distributed Search Engine". In: (2017).

Appendix A

Bucket notificiation JSON-object


```
{
        "eventVersion": "2.0",
        "eventSource": "minio:s3",
        "awsRegion": "",
        "eventTime": "2017-03-30T08:00:41Z",
        "eventName": "s3:ObjectCreated:Put",
        "userIdentity": {
                "principalId": "minio"
        },
        "requestParameters": {
                "sourceIPAddress": "127.0.0.1:38062"
        },
        "responseElements": {
                "x-amz-request-id": "14B09A09703FC47B",
                "x-minio-origin-endpoint": "http://192.168.86.115:900
        },
        "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "Config",
                "bucket": {
                        "name": "images",
                        "ownerIdentity": {
                                "principalId": "minio"
                        },
                        "arn": "arn:aws:s3:::images"
```

```
                },
                "object": {
                        "key": "myphoto.jpg",
                        "size": 6474,
                        "eTag": "a3410f4f8788b510d6f19c5067e60a90
                        "sequencer": "14B09A09703FC47B"
                }
        },
        "source": {
                "host": "127.0.0.1",
                "port": "38062",
                "userAgent": "Minio (linux; amd64) minio-go/2.0.3
        }
}
```

Appendix B

System specifications

OS Name	Microsoft Windows 10 Enterprise
Version	10.0.14393 Build 14393
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	PRODEV12
System Manufacturer	Dell Inc.
System Model	Precision WorkStation T3500
System Type	x64-based PC
System SKU	
Processor	Intel(R) Xeon(R) CPU W3540 @ 2.93GHz, 2933 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Dell Inc. A03, 9/9/2009
SMBIOS Version	2.5
Embedded Controller Version	255.255
BIOS Mode	Legacy
BaseBoard Manufacturer	Dell Inc.
BaseBoard Model	Not Available
BaseBoard Name	Base Board
Platform Role	SOHO Server
Secure Boot State	Unsupported
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume2
Locale	United States
Hardware Abstraction Layer	Version = "10.0.14393.2068"
User Name	INTERNAL\simon.habtu
Time Zone	W. Europe Daylight Time
Installed Physical Memory (RAM)	12.0 GB
Total Physical Memory	12.0 GB
Available Physical Memory	3.95 GB
Total Virtual Memory	24.0 GB
Available Virtual Memory	13.7 GB
Page File Space	12.0 GB
Page File	C:\pagefile.sys
Device Guard Virtualization based security	Running
Device Guard Required Security Properties	
Device Guard Available Security Properties	Base Virtualization Support
Device Guard Security Services Configured	
Device Guard Security Services Running	
A hypervisor has been detected. Features required for Hyper-V will not be displayed.	

Figure B.1: System specifications of the host PC.

Appendix C

System specifications for Elasticsearch host

OS Name	Microsoft Windows 10 Enterprise
Version	10.0.17134 Build 17134
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	SE-LDA-SC-0006
System Manufacturer	Dell Inc.
System Model	Precision 5520
System Type	x64-based PC
System SKU	07BF
Processor	Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz, 3001 Mhz, 4 Core(s), 8 Logical ...
BIOS Version/Date	Dell Inc. 1.9.4, 2018-04-23
SMBIOS Version	3.0
Embedded Controller Version	255.255
BIOS Mode	Legacy
BaseBoard Manufacturer	Dell Inc.
BaseBoard Model	Not Available
BaseBoard Name	Base Board
Platform Role	Mobile
Secure Boot State	Unsupported
PCR7 Configuration	Binding Not Possible
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	Sweden
Hardware Abstraction Layer	Version = "10.0.17134.1"
User Name	INTERNAL\Fredrik.Almstrom
Time Zone	W. Europe Daylight Time
Installed Physical Memory (RAM)	32,0 GB
Total Physical Memory	31,6 GB
Available Physical Memory	21,6 GB
Total Virtual Memory	63,6 GB
Available Virtual Memory	51,7 GB
Page File Space	32,0 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualization-based security	Running
Virtualization-based security Re...	
Virtualization-based security Av...	Base Virtualization Support, DMA Protection, Mode Based Execution Control
Virtualization-based security Se...	
Virtualization-based security Se...	
Device Encryption Support	Reasons for failed automatic device encryption: PCR7 binding is not supporte...
A hypervisor has been detecte...	

Figure C.1: System specifications of the host PC running Elasticsearch.

TRITA TRITA-EECS-EX-2018:281