Master's Degree in Informatics Engineering

Thesis - Intermediate Report

Observing and Controlling Performance in Microservices

Author:
André Pascoal Bento

Supervisor:
Prof. Filipe João Boavista Mendonça Machado Araújo

Co-Supervisor:
Prof. Jorge Cardoso

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

January 2019

This page is intentionally left blank.

Abstract

Nowadays, we find ourselves in a world in which the increasing technological evolution demands more and more of the computational systems and specially, of the people who develop and maintain them. With this growth, certain characteristics such as the complexity and the distribution of the systems increase in stride, so that it becomes quite difficult to manage them and to perceive their operation in general. It is in this problem that this work aims to provide solutions. The main objective of the solutions presented in this document are to improve the way in which the administrators of this kind of systems, face and solve the problems that are occurring in their systems.

Keywords

Microservices, Cloud Computing, Observability, Monitoring.

This page is intentionally left blank.

Resumo

Hoje em dia encontramos-nos num mundo em que a crescente evolução tecnológica exige cada vez mais dos sistemas computacionais e especialmente das pessoas que os desenvolvem e mantêm. Com este crescimento, certas características como a complexidade e a distribuição dos sistemas aumentam a passos largos, de modo a que se torna bastante difícil de os gerir e de perceber o seu funcionamento em geral. É neste problema que este trabalho visa prestar soluções. As soluções apresentadas neste documento, têm como principal objetivo melhorar a forma como os administradores deste género de sistemas, encaram e resolvem os problemas que estão a ocorrer nos seus sistemas.

Palavras-Chave

Micro-serviços, Computação na nuvem, Observabilidade, Monitorização.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

API Application Programming Interface. 7, 16

CPU Central Processing Unit. 14

DEI Department of Informatics Engineering. 1, 3, 11

DevOps Development and Operations. 1, 13, 21, 22

GDB Graph Database. 11, 15, 30, 32

HTTP Hypertext Transfer Protocol. 7, 9, 10

QA Quality Attribute. 26–29, 32

RPC Remote Procedure Call. 9

TSDB Time Series Database. 11, 17, 30, 32

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

This document represents the *Master Thesis* in *Informatics Engineering*, of the student *André Pascoal Bento* during the school year of 2018/2019, taking place in the *Department of Informatics Engineering (DEI)* of the *University of Coimbra*.

1.1 Context

In today's world, IT(Information Technology) and Development and Operations (DevOps) teams have lots of problems when is needed to identify the root causes that creates issues in their cloud or distributed based systems. Moreover, debugging this type of systems has not been an easy task, as it involves a hard and tedious work of diving through logs and system input and output registries and, in the most cases, it reveals like a big "find a needle in the barn" problem. Taking this into consideration, the work presented in this thesis, aims to perform an investigation around this kind of needs and present some solutions to the presented problem.

TODO: Polish and insert one or two more paragraph explaining the main context.

1.2 Motivation

The motivation behind this whole work resides on improving the today's knowing about cloud and distributed based system analysis. The analysis in this kind of systems are very tricky due to their properties and the way they are designed. With this, problems regarding the system operation usually appear, and they must be identified and resolved, otherwise they will tend to interfere with the normal functionality of this systems.

- Explain and present the motivation about the theme behind this thesis.

- "Find a new way and approach toward the process of analysis and monitor of distributed systems"

1.3 Goals

The main goals of this thesis relies on study the ways that these kind of systems are analysed today by understanding the current methodologies and ways to do it. After that, what we expect to do is to improve this kind of approaches and elaborate a new way to do it. With the defined approaches and questions that we want to solve, we aim to design and implement a new tool capable of prove the deductions exposed and perform the system analysis based on some input.

TODO: Finish.

1.4 Document Structure

This section presents the document structure in this report, with a brief explanation of the contents in every section. The current document contains a total of five chapter, including this one, 1 - Introduction. The remaining four of them, are exposed bellow.

In chapter 2 - Methodology are presented the elements involved in this work, with their contributions, has well as the work plan, with "foreseen" and "real" work plans comparison and analysis.

In chapter 3 - State of the Art the current state of the art for this kind of problem is presented. This chapter is subdivided in two sub-sections. The first one, 3.1 - Concepts introduces the reader to the core concepts to know as a requirement for a full understanding of the topics discussed in this thesis. The second one, 3.2 - Technologies presents the result of a research for current technologies, that may be able to help solving this kind of problem.

In chapter 4 - Research Objectives and Approach presents how we faced the problem, and what were the main difficulties that we found when handling this kind of problem, as well as the objectives involved to solve the issues that are presented.

In the last chapter, 5 - Solution, a solution for the problem is presented. This chapter is subdivided in three sub-sections with the objective to clearly explain the involved solution. The first one, 5.1 - Functional Requirements expose the functional requirements whit their corresponding levels, and a brief explanation to every single one of them. The second one, 5.2 - Quality Attributes contains the gathered non-functional requirements that were used to build the solution architecture. The last one, 5.4 - Architecture, presents the solution architecture using some representational diagrams and ends with a analysis and checkup if the presented architecture meets up the restrictions involved in the non-functional requirements.

Chapter 2

Methodology

In this chapter are present the methodology of work carried out in this project. At first, every member involved in the project will be mentioned as well as their individual contribution for this project. At second, the adopted approach and process organisation of the collaborators involved will be explained. And at last, we present the work plan as well as the work performed, including the foreseen and real work plans for the whole year of work.

The main people involved in this project were myself, André Pascoal Bento, student at the Master course of Informatics Engineering at the Department of Informatics Engineering (DEI), who carried out the investigation and development of the project. In second, Prof. Filipe Araújo, assistant professor at the University of Coimbra, who contributed with his vast knowledge and guidance on topics about distributed systems and cloud computing. In third, Prof. Jorge Cardoso, Chief Architect for Intelligent CloudOps at Huawei Technologies, who contributed with his vision and great contact with the topics addressed in this work. In fourth, Eng. Jaime Correia, doctoral student at the DEI, who contributed with his vast technical knowledge regarding the topics of tracing and monitoring microservices.

As this work stands for an investigation, it was necessary to perform an exploratory work there were no clear development methodology adopted. Instead of a development methodology, meetings were scheduled in the beginning to happen every two weeks. In this meetings, the people mentioned in the previous paragraph were gathered together to discuss the work carried out in the last two weeks and topics like the information gathered, the analysis about some existing tools, ideas and solutions were discussed between all. In the end, although there wasn't defined some development methodology, the meeting that were carried out were more than enough to keep the productivity and good work.

For the work plan and starting by some numbers, the time spent in each semester of the year by week are sixteen hours for the first semester, and forty hours for the second one. In the end, it was spent a total of 304 hours for the first semester, starting in 11.09.2018 and ending in 21.01.2019 (19 weeks times 16 hours per week), and it is expected to be spent a total of 840 hours for the second semester, starting in 04.02.2019 and ending in 30.06.2019 (21 weeks times 40 hours per week).

In the beginning, there was a work plan for the two semesters presented in the the thesis proposition. For record it is presented in the figures 2.1 and 2.2.

For effects of analysis, the real work plan that was carried out for the first semester is presented in the figure 2.3.

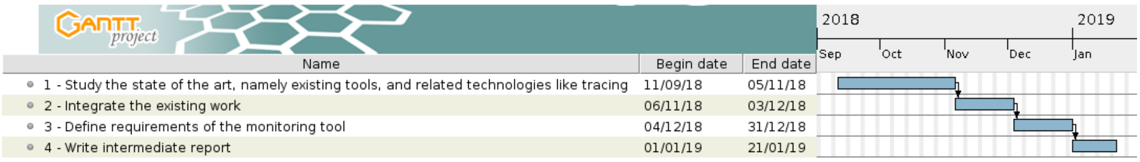


Figure 2.1: Proposed work plan for the first semester.

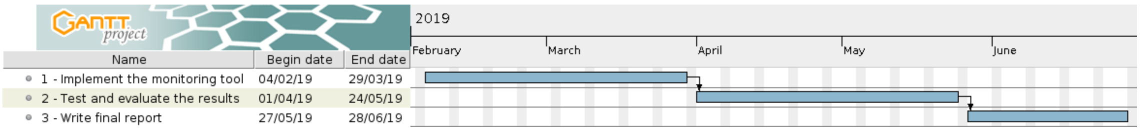


Figure 2.2: Proposed work plan for the second semester.

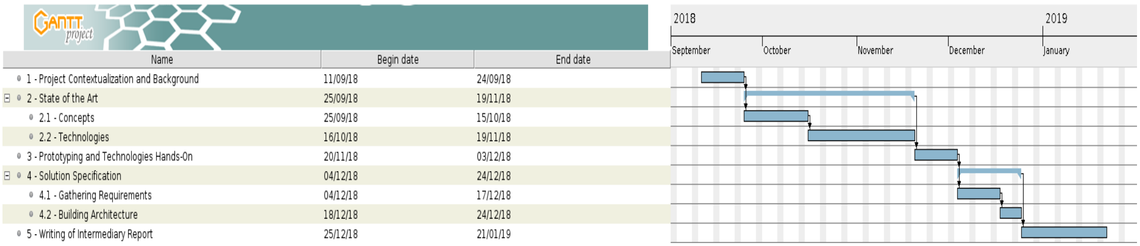


Figure 2.3: Real work plan for the first semester.

As we can see, the "foreseen" work plan for the first semester has suffered some changes, when comparing it to the real work plan. The predicted task 1 - Study the state of the art(...), was branched into two 1 - Project Contextualisation and Background and 2 - State of the Art, and took some more time to do because of the non-concrete and lack of documentation in the technologies related to the subject of this thesis. The predicted task 2 - Integrate the existing work, was replaced by 3 - Prototyping and Technologies Hands-On. This replacement was done because of the interest in test the technologies gathered in the state of the art and see some results with them, enhancing our investigation work and allowing us to get a better visualisation of the data that we had back then. The remaining tasks took almost the predicted time to do.

XXX TODO: Finally, it was generated a foreseen work plan for the second semester even knowing that, with almost one hundred percent of certainty the work plan will change when we perform the real work, it is presented in the figure 2.4. This work plan was generated taken into account the effort needed to implement the defined solution presented in the chapter 5 - Solution, and the effort needed to perform the investigation of some algorithms for the solution.

images/foreseen_work_plan_semester_2.pdf

Figure 2.4: Foreseen work plan for the second semester.

All the figures to expose the work plans have been created by an open-source tool called GanttProject[30] that produce Gantt charts, a kind of diagram used to illustrate the progress of the different stages of a project.

This page is intentionally left blank.

Chapter 3

State of the Art

In the following chapter, we will be discussing the core concepts regarding the project subject and the most modern techniques and technology that's available for the purpose today. All the information that will be presented was the result of a work of investigation through published articles, knowledge exchange and web searching.

The main purpose of the section 3.1 - Concepts is to introduce and provide a brief explanation about the core concepts. In the second section 3.2 - Technologies, all the important technologies are analysed and discussed using tables, diagrams and clear text.

3.1 Concepts

The following concepts represents the baseline to understand the work related to this project.

3.1.1 Microservices

Microservices is "an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities"[28].

This kind of style have a very long history, and have been being introduced and evolving since the first contact with topics like distributed computing, Application Programming Interface (API) and containers. "A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another"[11].

The core concept of microservices stands in isolation, or by other words, what everyone wants to achieve when building a software with microservices in mind, is to share less things between the services and deal with correlated failures. In this sense, a service is a small part of the entire system (e.g. Get messages microservice), and represents a tiny feature of the whole service (e.g. Chat Service). To do this, normally every microservice is encapsulated inside a container (e.g. Docker container[12]), and each runs in its own process and communicates with the other using lightweight mechanisms, often an Hypertext Transfer Protocol (HTTP) resource API.

"If you can't manage building a monolith inside a single process, what makes you think putting network in the middle is going to help?", Simon Brown[6]

In the other side, and for comparison purposes, we have another very well known architectural style, the monolithic. This style have a logically modular architecture, and the services are packaged and deployed in a single application using a single code base. To compare both architectural styles presented before we have the figure 3.1 that shows and provides a more clear insight about the differences between them.

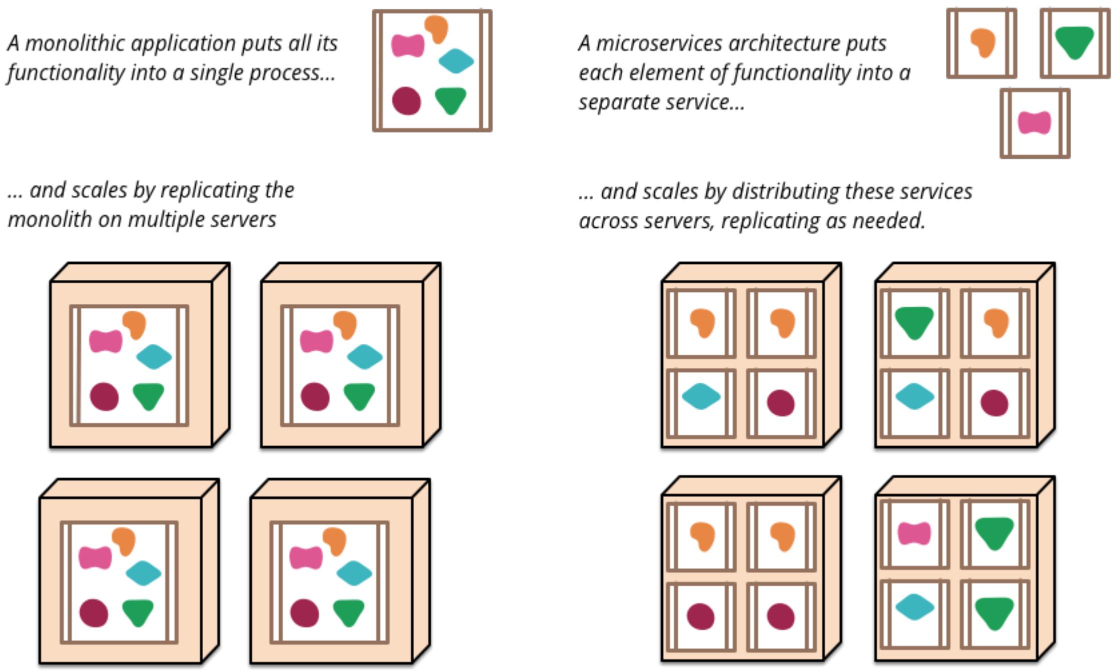


Figure 3.1: Monolithic and Microservices architectural styles.

Every style presented has its owns pros and throwbacks and its usages benefit from case to case. A brief example of pros and throwbacks of each one is: for very large teams developing a big and complex service that needs to scale, its good to use the microservices architectural style, because they can tackle the problem of complexity by decomposing application into a set of manageable services which are much faster to develop by individual members, and therefore its much easier to understand and maintain, however its harder to assemble and perform the deployment of the whole service composed by tinny granular parts. In the monolithic architecture, it is normally simpler to deploy, because we just have to copy a single packaged application to a server and run it, however when adding features to the application and it starts to grow in complexity, it gets harder to fully understand and made changes fast and correctly.

3.1.2 Observability and Controlling Performance

Observing is "to be or become aware of, especially through careful and directed attention; to notice"[31].

The presented definition represents the meaning of the word Observing and it is reflected exactly as it is in the project context. For example, observe the interaction between some microservices to notice a fault.

When we want to understand the working and behaviour of a system, we need to watch it very closely and pay special attention to all details and data it provides. This kind of details and data may be in multiple structured text formats, and it can contain lots of information regarding the interaction between microservices and the corresponding access to them. Therefore, we may work with this information as a starting point to perceive the characteristics of the system and build a tool that is able to be aware of it.

TODO: Finish.

3.1.3 Traces and Spans

First things first, we can think in a trace as a group of spans. A trace is a representation of a data/execution path through the system and a span represents the logical unit of work in the system. The span has an operation name, the start time of the operation, its duration and some annotations regarding the operation itself. An example of a span can be an HTTP call or an Remote Procedure Call (RPC) call. For a more clear insight of how spans are related with each other and with time, we have the figure 3.2.

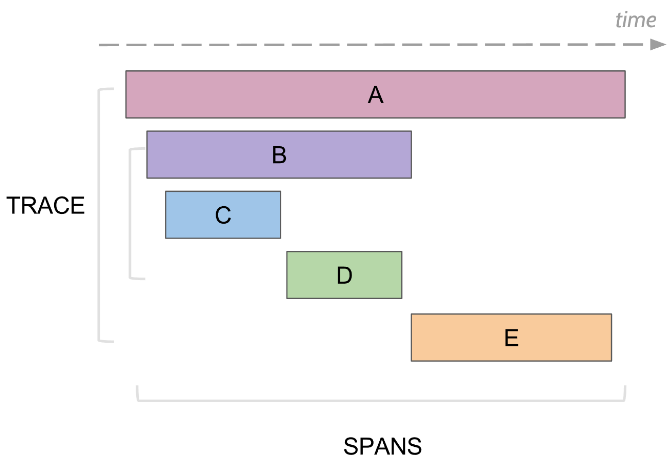


Figure 3.2: Traces and spans disposition over time.

As we can see in the figure 3.2, the spans spread over time, overlapping each other, since nothing prevents the occurrence of multiple calls in very close times. The creators of OpenTracing have made a data model specification that says, "with a couple of spans, we might be able to generate a span tree and model a graph of a portion of the system" [24]. This is because they represents causal relationships in the system. As presented by the guys who defined this specification, and again for a more clear insight, the span tree can be like the one presented in the figure 3.3.

```
          [Span A]  ←←←(the root span)
              |
      +------+------+
      |             |
  [Span B]      [Span C] ←←←(Span C is a `ChildOf` Span A)
      |             |
  [Span D]      +---+-------+
                |           |
            [Span E]    [Span F] >>> [Span G] >>> [Span H]
                                        ↑
                                        ↑
                                        ↑
                          (Span G `FollowsFrom` Span F)
```

Figure 3.3: Span Tree example.

In the figure 3.3 it's represented a span tree with a trace made up of eight spans. Every span must be a child of some other span, unless it is the root span. With the information presented in the span tree, we can generate a multi-directed graph of the system (explained in the subsection 3.1.4 - Graphs).

This type of data is extracted and can be obtained, as trace files or by transfer protocols ex. HTTP, from technologies like Kubernetes[9], OpenStack[23], and other cloud or distributed management system technologies that implements some kind of system or code instrumentation using, for example, OpenTracing[25] or OpenCensus[13].

In the end and as explained before, traces and spans contains some vital system details as they are the result of instrumentation of a part or the whole system and therefore, this kind of data can be used as a starting point resource information to analyse the system.

3.1.4 Graphs

As it was briefly explained before, we might be able to model a graph of the system using a couple of spans. Normally, in discrete mathematics and more specifically in graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related"[34].

Taking the very common sense of the term, a graph is an ordered pair G = (V, E), where G is the graph itself, V are the vertices/nodes and E are the edges. The figure 3.4 gives us, a simple visual representation, of what a graph really is for a more clear understanding. There we can see a graph composed by a total of five nodes that contains some labels in it and, in this case, five relationships between them.

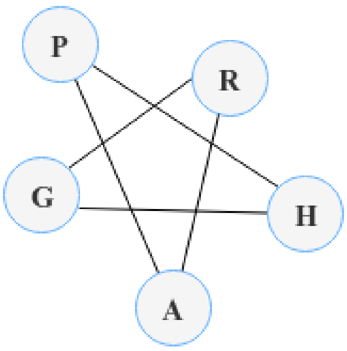


Figure 3.4: Graph visual representation.

This graph is the representation of the following information:

V = {'G', 'R', 'A', 'P', 'H'}

E = {{'G', 'R'}, {'R', 'A'}, {'A', 'P'}, {'P', 'H'}, {'H', 'G'}}

There are multiple types of graphs. In this term they can be undirected, where the set of edges don't have any orientation between a pair of nodes like in this example, or be directional, where the set of edges have one and only one orientation between a pair of nodes, or be a multigraph, where in multiple edges are more than one connection between a pair of node that represents the same relationship, and so forth.

Graphs can have many use cases, has they can model the representation of a lot of real life practical problems in the fields like physics, biology, social and information systems and for the purpose of this thesis, they are considered first class citizens.

3.1.5 Graph Database

A Graph Database (GDB) is "a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data"[35].

The composition of a GDB is based on the mathematics graph theory, and therefore this databases uses three main components called nodes, edges, and properties. This main components are defined and explained in the following list:

- Node: Are the entities in the graph. They can hold any number of attributes (key-value pairs) called properties. Nodes can be tagged with labels, representing their different roles in your domain. Node labels may also serve to attach metadata (such as index or constraint information) to certain nodes.
- Edge (or Relationships): provide directed, named, semantically-relevant connections between two node entities (e.g. André STUDIES_IN Department of Informatics Engineering (DEI)). A relationship always has a direction, a type, a start node, and an end node. Like a Node it can attach metadata.
- Property: can be any kind of metadata attached to a certain Node or a certain Edge.

3.1.6 Time Series Database

A Time Series Database (TSDB) is "is a database optimised for time-stamped or time series data like arrays of numbers indexed by time (a date time or a date time range)"[38].

This kind of databases are natively implemented using specialised database algorithms to enhance it's performance and efficiency due to the widely variance of access possible. The way this databases use to work on efficiency is to treat time as a discrete quantity rather than as a continuous mathematical dimension. Usually a TSDB allows operations like create, enumerate, update, organise and destroy various time series entries.

The TSDB and the GDB, presented in this subsection and in the subsection before respectively, are at the time, the most wanted and fastest growing kind of databases due to their use cases in the trending fields of Cloud and Distributed based Systems and in the *Internet of Things (IoT)*. The figure 3.5 presents the growing of this databases in the last two years. As we can see in the figure presented, the TSDB and GDB are distancing from the remaining databases in terms of popularity starting from the same spot in December of 2016. The predictions are that this databases will not stop increasing popularity, until this kind of systems described before start loosing it too.

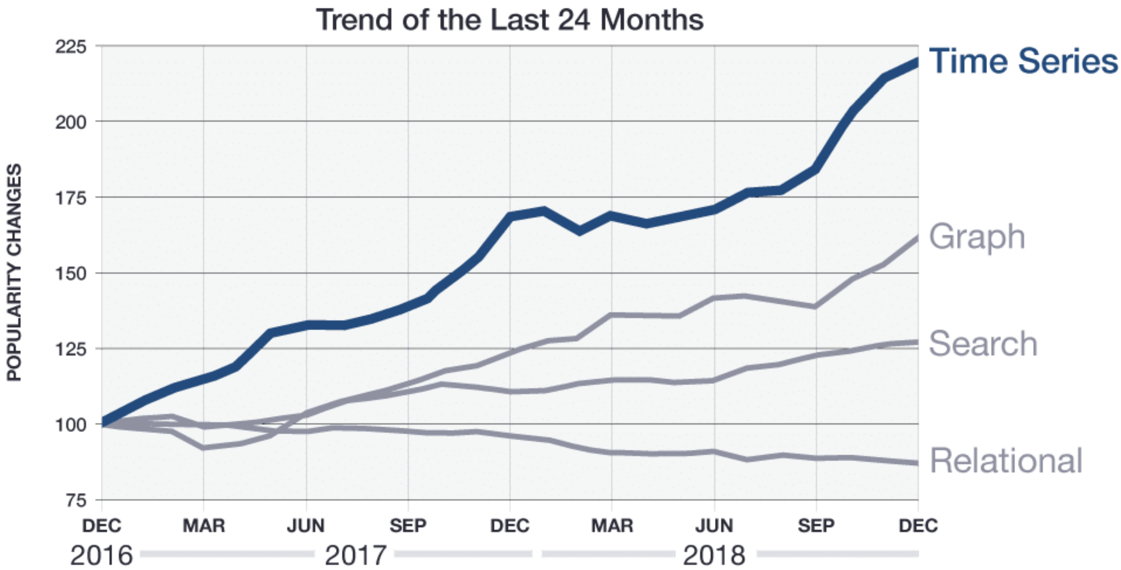


Figure 3.5: Fastest Growing Databases.[17]

3.2 Technologies

In this section are presented the technologies and tools that were researched, as well as the corresponding discussion considering the main objectives for this project. With the concepts presented in the section 3.1 in mind, the research were focused in a group of main topics regarding the problem we have in hands. The main topics were 3.2.1 - Monitoring and Tracing Tools, 3.2.2 - Graph Manipulation and Processing Tools, 3.2.3 - Graph Database Tools and 3.2.4 - Time-Series Database Tools.

3.2.1 Monitoring and Tracing Tools

This sub-section presents what are the most used and knowed monitoring and tracing tools. This tools are mainly oriented for monitoring and tracing microservices-based distributed systems. The table 3.1 presents them.

Table 3.1: Graph tracing and monitoring tools comparison.

Name	Jaeger	Zipkin
Repository	Jaeger GitHub [18]	Zipkin GitHub [27]
Brief description	It is a distributed tracing and monitoring system released as open source by Uber Technologies, that is used for monitoring and troubleshooting microservices-based distributed systems.	It is a distributed tracing and monitoring system. It helps gather timing data needed to troubleshoot latency problems in microservice architectures. It manages both the collection and lookup of this data. Zipkin's design is based on the Google Dapper paper.
Pros	OpenSource. Docker-ready. Can be used with some Zipkin functionalities, as it has a collector dedicated to it. Dynamic sampling rate. Browser UI.	OpenSource. Docker-ready. Allows lots of span transport ways (HTTP, Kafka, Scribe, AMQP). Browser UI.
Cons	Only supports two span transport ways (UDP and HTTP).	Fixed sampling rate.
Used mainly by	Uber	Lightstep

As we can see, this kind of tools are very similar and very good for monitoring and tracing a system as they provide a bunch of pros like being opensource, dockerized, support for some well known technologies for span transport and aggregate the spans in a good representational browser user-interface. However they are always focused on span and trace lookup and presentation, and do not provide a more interesting analysis of the system, for example to determine if there is any problem related to some microservice presented in the system. This kind of work falls into the user (Development and Operations (DevOps)) and he needs to perform the investigation and analyse the traces and spans with the objective of find anything wrong with them.

This kind of tools can be a good starting point for the problem that we face, because they already do some work for us like grouping the data generated by the system and provide a good representation for them.

3.2.2 Graph Manipulation and Processing Tools

Considering that we have data to be processed and manipulated, we have to be sure that we can handle it for analysis. For this purpose, and knowing that the data is a representation and an abstraction of a graph, we needed to study the frameworks available this task. The table 3.2 presents the main technologies available at the time for graph manipulation and processing.

Table 3.2: Graph manipulation and processing tools comparison.

Name	Apache Giraph [1]	Ligra [29]	NetworkX [22]
Description	It's an iterative graph processing system built for high scalability. For example, it is currently used at Facebook to analyse the social graph formed by users and their connections.	A library collection for graph creation and manipulation, and for analysing networks.	A Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
Licence[37]	Free Apache 2	MIT	BSD - New License
Supported languages	Java and Scala.	C and C++.	Python.
Pros	Distributed and very scalable. Excellent performance (Can process one trillion edges using 200 modest machines in 4 minutes).	Can handle very large graphs. Exploit large memory and multi-core CPU's (Vertically scalable).	Good support and very easy to install with Python. Lots of graph algorithms already implemented and tested. Mature project.
Cons	Uses the "Think-Like-a-Vertex" programming model that often forces into using sub-optimal algorithms and is quite limited and sacrifices performance for scaling out. Can't perform many complex graph analysis tasks because it primarily supports Bulk synchronous parallel.	Lack of documentation and therefore, very hard to use. Don't have many usage in the community.	Not scalable (single-machine). High learning curve due to the maturity of the project. Begins to slow down when there's a high amount of data (400.000 plus nodes).

With the information presented in the previous table, we can have a notion that this three frameworks don't work and perform in the same level in many ways.

One thing to consider when comparing them is the scalability and performance that each can provide, for instance, in this component the first one, Apache Giraph is the winner since it is implemented with the distributed systems paradigm in mind and can scale to multiple-machines to get the job done in no time, considering high amounts of data. In other way, we have the third framework, called NetworkX, that is different from the previous as it works in a single-machine and doesn't have the ability to scale to multiple-machines. This can be a very problematic feature if we are dealing with very high amounts of data and we have to process it in short amounts of time. The last framework, called Ligra, works in a single-machine environment like the previous one, but it can scale vertically as it has the benefit of use and exploit multi-core CPU's.

The second, and also most important thing to consider, is the support and quantity of implemented graph algorithms in the framework, and in this field the tables turned, and the NetworkX has a lot of advantage as it have lots of implemented graph algorithms defined and studied in graph and networking theory. The remaining frameworks don't have very support either because they don't have it documented or because the implementation and architecture that where considered don't allow to implement it.

For a more clear insight of the position of the presented technologies in the previous table, we have the figure 3.6 [10].

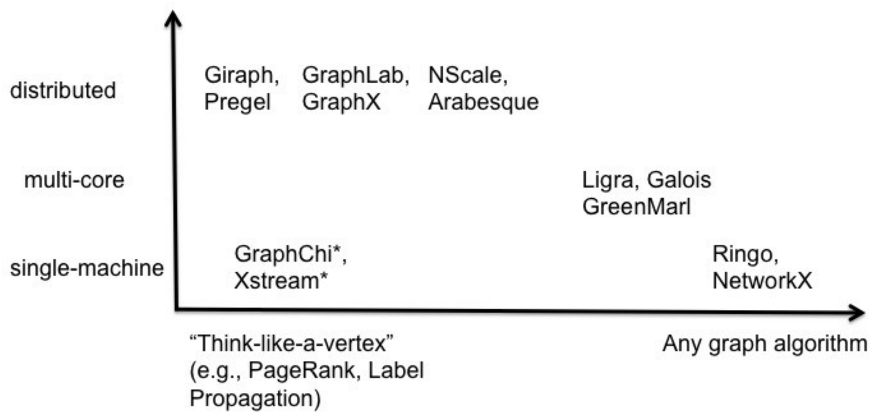


Figure 3.6: Graph manipulation tools comparison, regarding scalability and graph algorithms.

With the presented figure, our perception of what we might choose when considering this tools is more clear, but we have always some trade offs we cannot avoid. The best approach, and if it's possible, is to consider the usage of an hybrid environment where Giraph and NetworkX coexist one with another, as one fills the gaps of the other, but always taking into consideration that a bottleneck will occur between them [20] and that there are almost none implementation where they coexist [20] because of their disparity.

3.2.3 Graph Database Tools

Manipulating and process graph data is not enough, we need to store this data somewhere, and to do this we need a GDB. The results of the research for the best graph databases tools available are presented in the table 3.3.

Table 3.3: Graph databases comparison.

Name	ArangoDB [2]	Facebook TAO [5]	Neo4J [21]
Description	It's a NoSQL database developed by ArangoDB Inc. that uses a proper query language to access the database.	TAO, "The Associations and Objects", is a proprietary database, developed by Facebook, that stores all the data related to the users in the social network.	It's the most popular open source graph database. Has been developed by Neo4J Inc. and is completely open to the community.
Licence	Free Apache 2	Proprietary	GPLv3 CE
Supported languages	C++ Go Java JavaScript Python Scala	—	Java JavaScript Python Scala
Pros	Multi data-type support (key/value, documents, graphs). Allows the combination of different data access patterns in a single query. Supports cluster deployment.	Very fast(=100ms latency). Accepts millions of calls per second. Distributed.	Supports ACID(Atomicity, Consistency, Isolation, Durability)[33]. High-availability. Has a visual node-link graph explorer. REST API interface. Most popular open source graph database.
Cons	Needed to learn a new query language called AQL(Arango Query Language). High learning curve. Has paid version with high price tag.	Not accessible to use.	Can't be distributed (It needs to be vertically scaled).

As we can notice by the data provided by the presented table, the state of the art about graph databases is not very good. The offers are very limited and all of them lack something when we start to see them in detail. The interest in this databases is increasing as graph technology tend to have many use cases and solve lots of problems nowadays.

Facebook detains the most powerful and robust system for this purpose, but as it is the base of their business because they need to perform large operations in their huge social graph in reduced times, it is a proprietary technology and is only referenced in some articles [5].

The remaining two tools, are very supported by the community because of their license and demand, however based on the stars and forks of their repositories, Neo4J is more well received by the community and tends to become more popular. It doesn't implement horizontal scalability by design and this can be a risk when using it in systems with scalabilty in mind, but there are some authors that report they were able to perform implementations and surpass the scalability issue, however with many snags[14]. ArangoDB supports scalability by default as we can see in the figure 3.7[4], but it has a very hard query language with a high learning curve inherent to it, and it is payed to use some special features like SmartGraphs storage[3] that improves the writing of graph in distributed databases.

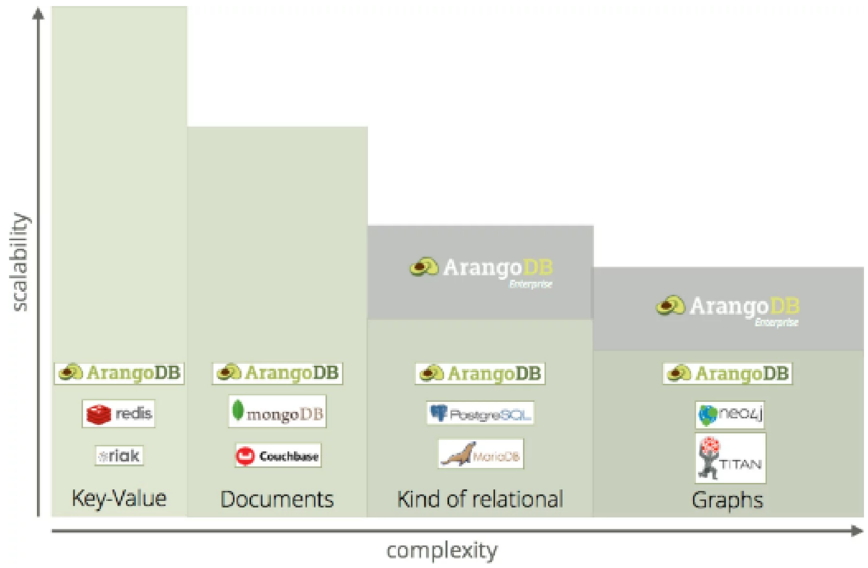


Figure 3.7: ArangoDB vs. Neo4J scalability.

3.2.4 Time-Series Database Tools

As we intend to extract useful data from span trees and graphs, we need to store it somewhere. We already know that the spans and trace data are directly related with time based information, explained in the subsection 3.1.3 - Traces and Spans, so the best way to store the gathered or calculated information from them is in a TSDB.

Thus, this subsection presents the state of the art with this kind of situation in mind. The table 3.4 exposes a comparison between two very well known databases in the world of TSDB.

Table 3.4: Time-series databases comparison.

Name	InfluxDB [16]	OpenTSDB [26]
Description	It is an open-source time series database developed by InfluxData written in Go and optimised for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics.	It is a distributed, scalable Time Series Database (TSDB) written on top of HBase. OpenTSDB was written to address a common need: store, index and serve metrics collected from computer systems (network gear, operating systems, applications) at a large scale, and make this data easily accessible and graphable.
Licence	MIT	GPL
Supported languages	Erlang Go Java JavaScript Lisp Python R Scala	Erlang Go Java Python R Ruby
Pros	Scalable in the enterprise version. Outstanding high performance. Accepts data via HTTP, TCP, and UDP protocols. SQL like query language. Allows real-time analytics.	It's massively scalable. Great for large amounts of time-based events or logs. Accepst data via HTTP and TCP access protocols. Good platform for future analytical research into particular aggregations on event/log data. Doesn't have paid version.
Cons	Enterprise high price tag. Clustering support only available in the enterprise version.	Expensive to try. Not a good choice for general-purpose application data.

Based on the information presented in the referenced table, we can notice that this two databases are very similar on what they offer like the access protocols and scalability capabilities. In the point of licence, both are open source, however the first one, InfluxDB, has an enterprise paid version that is not very well exposed in its documentations and much people don't even notice it, contrarily to OpenTSDB which is completely free. The enterprise version of InfluxDB provides clustering support, high availability and scalability[8], features that OpenTSDB offer for free, however in terms of performance, InfluxDB outperforms OpenTSDB in almost every benchmark by a far distance as we can see in the figures 3.8 and 3.9.

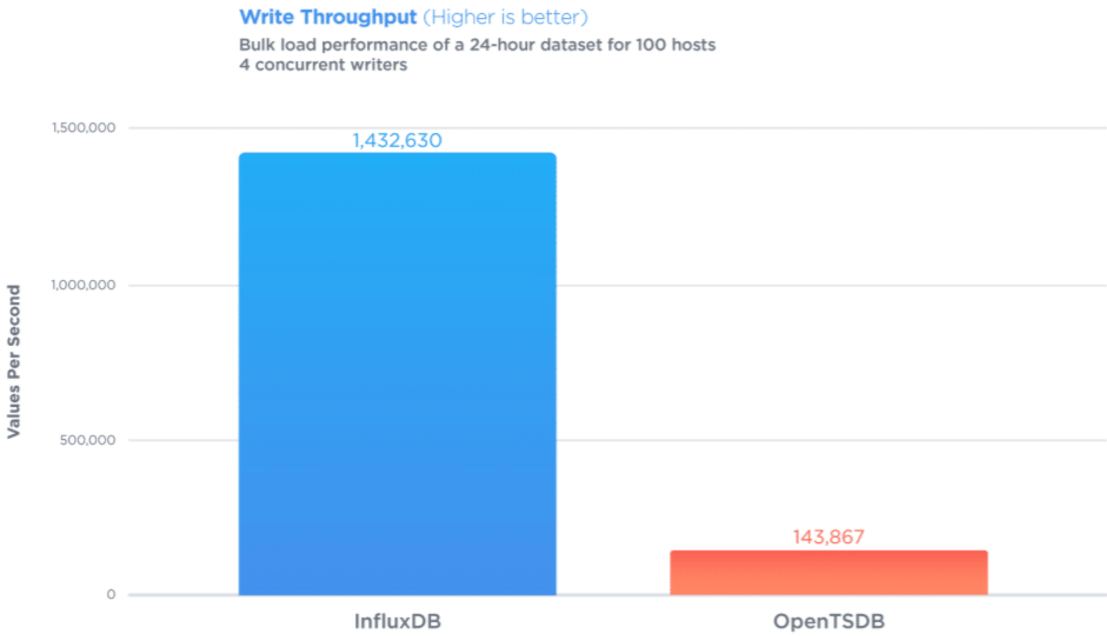


Figure 3.8: InfluxDB vs OpenTSDB write throughput performance[8].

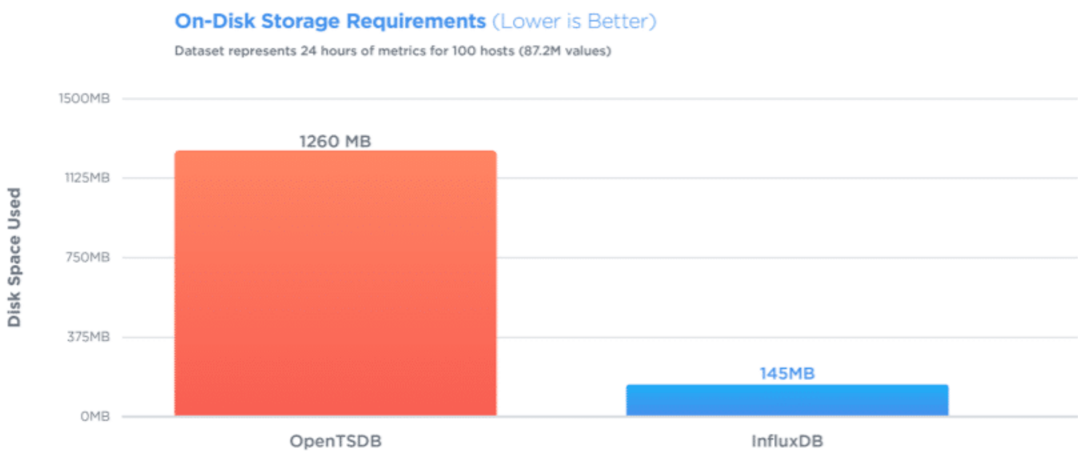


Figure 3.9: InfluxDB vs OpenTSDB storage requirements[8].

This page is intentionally left blank.

Chapter 4

Research Objectives and Approach

This chapter presents the research objectives and approach used in this thesis. We will start to discuss how we faced the problem, and what were the main difficulties that we found when handling this kind of problem as well as the ways we have taken to deal with it.

In the beginning we didn't know were to start or what to solve, as this kind of problem is very ambiguous and there are very few articles and published works in this kind of field. The main solutions existing today are the ones presented in 3.2.1, but they rely on the user perception to do the search and find issues that exist in their platform by performing queries to the spans and trace data. So, with this problem ahead, we started looking for the needs of Sysadmins and/or Development and Operations (DevOps) when they wanted to scan and analyse their system searching for issues.

Searching for issues it's a very vague situation as we didn't have defined what are the kind of issues that exists. To do this, and narrow the problem we were facing, we decided to gathered the personal involved in this work and perform an initial brainstorming. We putted ourselves in the perspective of a DevOps and try to find what are the main difficulties when they perform their search for issues in the system in a *"As a DevOps i want to..."* situation. The kind of questions that were placed were like: *"What are the most common issues?"*, *"What are the variables involved in this kind of issues?"* and *"What are the correlations between this variables and the most common issues?*. From this brainstorm emerged the following eight core questions:

1. What is the neighbourhood of one service?

2. Is there any problem (Which are the associated heuristics)?

3. Is there any faults related to the system design/architecture?

4. What is the root problem, when A, B, C services are slow?

5. How are the requests coming from the client?

6. How endpoints orders distributions are done?

7. What is the behaviour of the instances?

8. What is the length of each queue in a service?

The next step were to work on this questions. We decided to branch them in more concise questions, refine and filter the most relevant to define our objective, and after that, check with someone involved in the DevOps field if the final questions represent some of their needs. First, to handle the information presented in this eight initial questions, we decided to create what we called a "Project Questions Board". This board consists on a Kanban [36] style board present in the project git repository, were everyone involved in the project could access and modify it. The board was defined with four lanes: "To refine", "Interesting", "Refined" and "Final Questions", and the process were to cycle the questions through every lane, generating new ones and filtering others. After this, and to check if the final questions were really some that represented the needs of a DevOps, some colleagues that work directly in the field were contacted and the questions were exposed to them. In the end, the nine questions that were produced in the final lane represented right what are some of their needs. The final questions and the corresponding explanation of the expected work that must be performed to each one are exposed bellow:

1. What is the neighbourhood of one service, based on incoming requests?

Implies generate a graph, based on the spans and traces, using the outgoing connections, from a certain node, that are correlated with the incoming connection(s).

2. What is the neighbourhood of one service, based on outgoing requests?

Implies generate a graph, based on the spans and traces, using the incoming connections, from a certain node, that are correlated with the outgoing connection(s).

3. How endpoints orders distributions are done, when using a specific endpoint?

Implies generate a graph, based on the spans and traces, then calculate the degree of a certain node that represents the endpoint, to finally check if the it is an isolated, a leaf or a dominating (high or low depending on the degree of the other degrees) endpoint.

4. Which endpoints are the most popular?

Implies to retrieve the most popular service, based on the spans and traces, and get the services with more incoming connections sorted by the number of incoming connections.

5. Is there any problem related to the response time?

Implies to get the response time of every trace (difference between end and start time of every span in the trace) and then calculate and store some measurements like the average time, the maximum time, the minimum time and variance. After having some stored values, the system must perform calculations and check if there is too much disparity between them to determine if there is a problem in the response time.

6. Is there any problem related to the morphology?

Implies generate multiple graphs, based on a certain group of spans and traces that are contained in a certain time interval. Then we need to store the graphs gradually using some graph storing mechanism to perform the difference of subsequent stored graphs. This result of the difference between graphs must be stored in a time-series storing mechanism, to be accessed later and determine if there were hard changes that could lead to morphology problems in the system thought time.

7. Is there any problem related to the entire workflow of (one or more) requests?

Implies to generate the graph of the system, identify the path of some request(s) in the system and then perform the calculation to verify and identify if there were cycles presented in the graph involved in the path of the request(s). The results of this calculations must be stored in a time-series storing mechanism, to be accessed later and determine if this cycles are normal, or if they represent a problem related to the request(s) work-flow, based on the kind of request.

8. Is there any problem related to the occupation/load?

Implies to get the occupation/load of every trace (difference between end and start time in the trace) and then calculate and store some measurements like the average time, the maximum time, the minimum time and variance. After having some stored values, the system must perform calculations and check if there is too much disparity between them to determine if there is a problem in the occupation/load.

9. Is there any problem, related to the number/profile of the client requests?

Implies calculate the number of accesses to the system, based on the spans and traces annotated with client requests of a certain time interval, and store the calculations for every node. After having some stored values, the system must determine the level regions of access based in the available data (profile of requests, ex.: high, moderate and low), and check if there were to much requests outside of the defined level regions.

This final questions, with a slight reformulation, could be exported to high level of abstraction functional requirements of the monitoring tool that we want to develop. After having this questions, we decided to study the current state of art to check how things are done nowadays regarding this subject and we found that some tools perform the process of convert spans and traces to a graph, that represents the system at that current time interval, however they do not perform any kind of analysis and study over the span tree and the graph after that[19].

Considering this, what we decided to do was to develop a simple prototype tool to test some state of the art tools. What we were able to achieve was to do the reconstruction of the graph, using our own data (this data was provided by Prof. Jorge Cardoso, representing an approximate two hour collection of spans and traces, about 400.000 spans, generated by one of their clusters). At this point, and since we have already held the hands-on of some tools at the moment, we were ready to start and think about the solution we need to build. Therefore, we decided to specify the solution, and considered to build a monitoring tool named by ourselves, *Graphy*.

In a very briefly explanation, we want that *Graphy* be able to calculate relevant metrics from the span trees and the generated graphs, and to work with this kind of metrics to perform the system analysis and answer the questions exposed above. To perform some of this work, it will be resourcing to machine learning algorithms that we will need to study in parallel with the implementation, as we cannot predict what we might encounter when retrieving the metrics at the time. The machine learning algorithms are to process the metrics and perform some deductions regarding the system behaviour over the time.

This page is intentionally left blank.

Chapter 5

Solution

In this chapter we will be presenting and discussing the solution to be implemented regarding the research objectives of this work. To present the solution and explain it, we will cover some aspects and topics that are considered when defining a software based solution. This topics are the well known functional requirements 5.1, the quality attributes or non-functional requirements 5.2 and finally, the architecture 5.4 that is produced based on all the previous topics.

5.1 Functional Requirements

The functional requirements are in software and systems engineering, by definition, a declaration of the intended function of a system and its components. For the purpose of this specification, we decided to present a brief description of each functional requirement, composed by an id, the corresponding name and its priority.

For this example the notation priority was based on the urgency that we have to implement a certain functionality. For this notation, we decided to use three levels: High, Medium and Low priorities.

Therefore, the functional requirements for the proposed solution, obtained from the questions presented in the chapter 4, are briefly specified in the table 5.1 sorted by priority levels.

As the table 5.1 shows us, the functional requirements can be grouped in three main groups due to their priority levels. The first three (FR-1 to FR-3) are presented with high level of priority, because they don't represent a very high level of difficulty to implement, however they provide some base to the implementation of the following four. This four functional requirements (FR-4 to FR-7) represent a much higher degree of difficulty has there are much more questions to solve involving research and the implementation (explained in the chapter 4 - Research Objectives and Approach). The final two (FR-8 to FR-9) have a low priority level, because they do not represent to much relevance of functionality to our core issues.

Table 5.1: Functional requirements specification.

ID	Name	Priority
FR-1	The system must know what are the neighbours of a certain service, based on the service incoming requests.	High
FR-2	The system must know what are the neighbours of a certain service, based on the service outgoing requests.	High
FR-3	The system must know which endpoints are the most popular.	High
FR-4	The system must be able to identify if there is any problem related to the response time.	Medium
FR-5	The system must be able to identify if there is any problem related to its morphology.	Medium
FR-6	The system must be able to identify if there is any problem related to the entire work-flow of one or more requests.	Medium
FR-7	The system must know how endpoints orders distributions are done when using a specific endpoint.	Medium
FR-8	The system must be able to identify if there is any problem related to the occupation/load.	Low
FR-9	The system must be able to identify if there is any problem related to the number/profile of the client requests.	Low

5.2 Quality Attributes

When designing a system, one of the most important things to consider is to specify and describe well all the quality attributes or non-functional requirements. These kind of requirements are usually Architecturally Significant Requirements and they are the ones that require more of the architects attention, as they reflect directly all the architecture decisions and aspects. Sometimes, they are also named the ”itilities” because most of them share this suffix in the word.

To specify them, we usually use a representation called a utility tree, in which we insert the Quality Attribute (QA) by a certain order of priority, for the architecture and for the business inherent to each one, and in order to consider the trade-offs and decide the weight of each in the produced architecture. The codification of the order of priority is the following:

- H. High
- M. Medium
- L. Low

To describe them, we must pay attention and try to include six important things in the definition: the **stimulus source**, the **stimulus**, the **environment**, the **artifact**, the **response** and the **measure of the response**.

The figure 5.1 contains all the raised QA for this project exposed in a utility tree format, sorted alphabetically by their general QA, and after by the architectural impact.

As we can see by the information presented in the utility tree, there are the following QA's: Interoperability, Performance, Scalability, Traceability and Testability. A briefly explanation for every QA is exposed bellow:

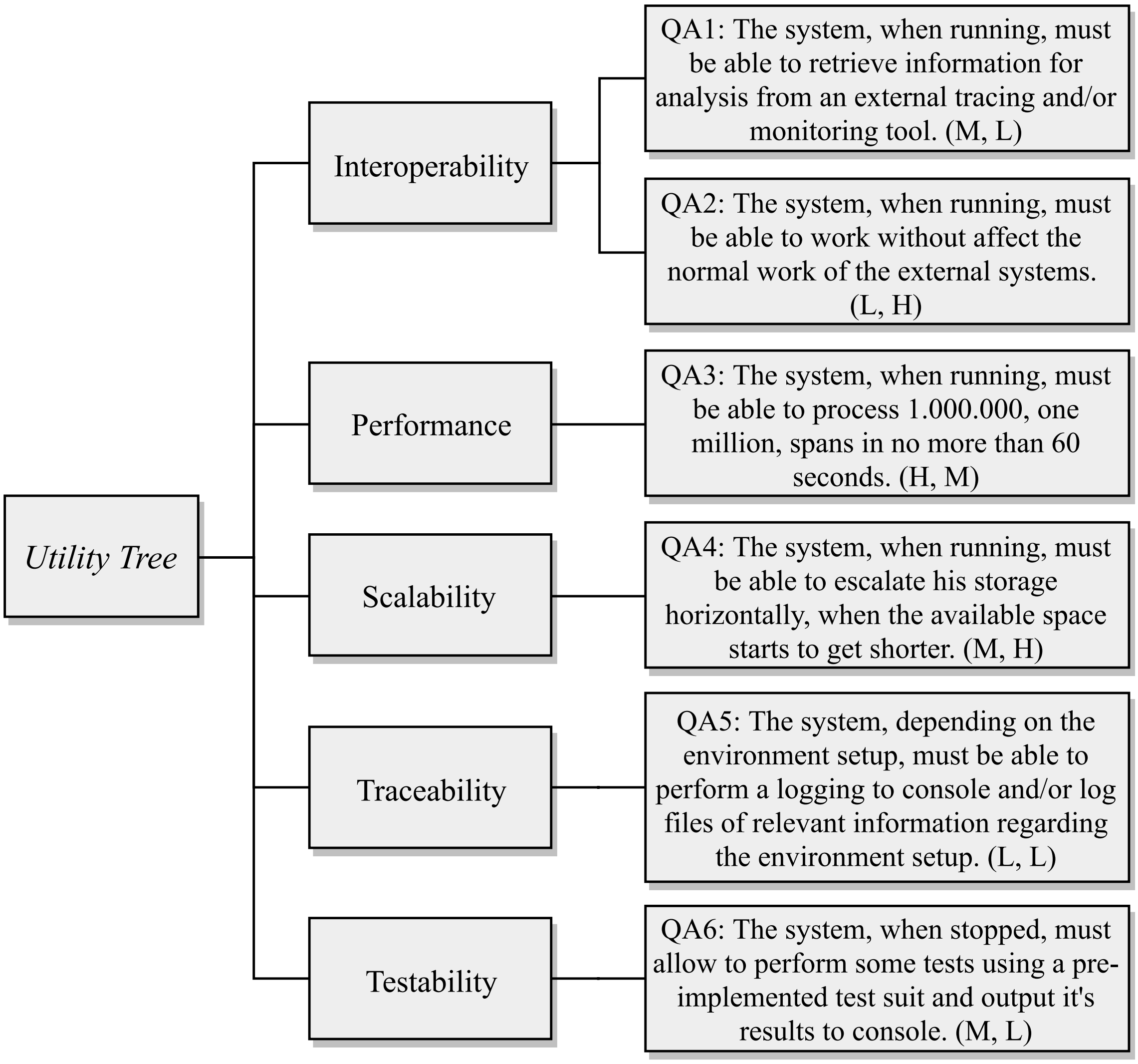


Figure 5.1: Utility tree.

QA1 (Interoperability): Since the wanted solution is a monitoring tool, it has to access an external system in order to obtain the necessary data to analyse and therefore, access to an internal monitoring tool presented in the external system. As this is considered the starting point to obtain our data, we considered a Medium level for the architecture and a Low for the business.

QA2 (Interoperability): Since the solution will be accessing an external system or outputs generated by it, all interactions with it must not cause conflicts. This is very important in the business perspective, because if our solution is not co habitable with other systems it may be completely rejected. For the architectural perspective it doesn't represent a big impact.

QA3 (Performance): We defined this QA taking into account the number of spans produced in an hour, by the system were we gathered our data. As it produces approximately 200.000 spans in an hour, and to ease our research work when using the tool, we decided to set the target for our solution as 1.000.000 spans in about a minute. This QA will have an high architectural impact, as it can define a certain technology for graph processing. For the business perspective, we considered a Medium level, as it presents some interest.

QA4 (Scalability): Due to the amount of data needed to process and store over time, our system has to be able to store the data into multiple machines because it may start running out of space. We decided to give a medium level of architectural impact as it can change the solution in terms of storage components. For the business this has an high impact, as it need more machines to run the solution if this QA is fully considered against the remaining.

QA5 (Traceability): This QA was considered due to the simple fact that we need to bee able to see what's the system is doing, when it's processing the data. For this QA we decided to give low levels for both architecture and business, as it does not represent relevance to any of them.

QA6 (Testability): As the system will be analysing external systems, we need to be sure that it analyses it in a correct way, and to be sure of this we need to test our solution. We considered a medium level for the architectural impact for this QA, because its implementation needs to analyse some components from within the system. For the business we considered a low level, because the main interest to test the system is ours in order check if it is working correctly.

5.3 Technical Restrictions

In this section we present the technical restrictions involved in this solution.

Normally, when specifying a solution in software engineering, after presenting the functional requirements and non-functional requirements, comes the specification of business restrictions. However, in this project none were raised due to the simple fact that this work is focused on the exploration and research and that there isn't any formal client defined.

To define the technical restrictions, we used an id and its corresponding description. The raised technical restrictions are presented in the table 5.2 followed by an explanation.

Table 5.2: Technical restrictions specification.

ID	**Description**
TR-1	Use OpenTSDB as a Time-Series database.

We raise only one technical restriction, as we can see in the table 5.2. This technical restriction was considered because prof. Jorge Cardoso suggested it, due to the simple fact that OpenTSDB is the database that they are currently using in their projects where a time-series database is needed. However, this project does not have a concrete and formally defined client, it is good to use a technology used by the people that will use the tool to ease their work and possibly introduce changes.

5.4 Architecture

In this section, the architecture will be presented based on all the previous topics with resource to the defined Simon Brown's C4 Model[7]. This approach of defining an architecture uses the following four diagrams to do it: 1 - Context Diagram, 2 - Container Diagram, 3 - Component Diagram and 4 - Code Diagram. For the definition of this architecture, only the first three representations will be considered. Every representation will be exposed with a briefly explanation of the decisions taken to draw that specific diagram. After presenting the representations and the corresponding explanations, we will cycle thought all the QA, in order to explain where it is reflected and the considerations taken to produce the current architecture.

5.4.1 Context Diagram

In this subsection the context diagram is presented. This kind of diagram allows us to see "the big picture" of the system as it represents the system as a "big box" and its interactions with the users and other systems. The figure 5.2 presents the context diagram for this solution.

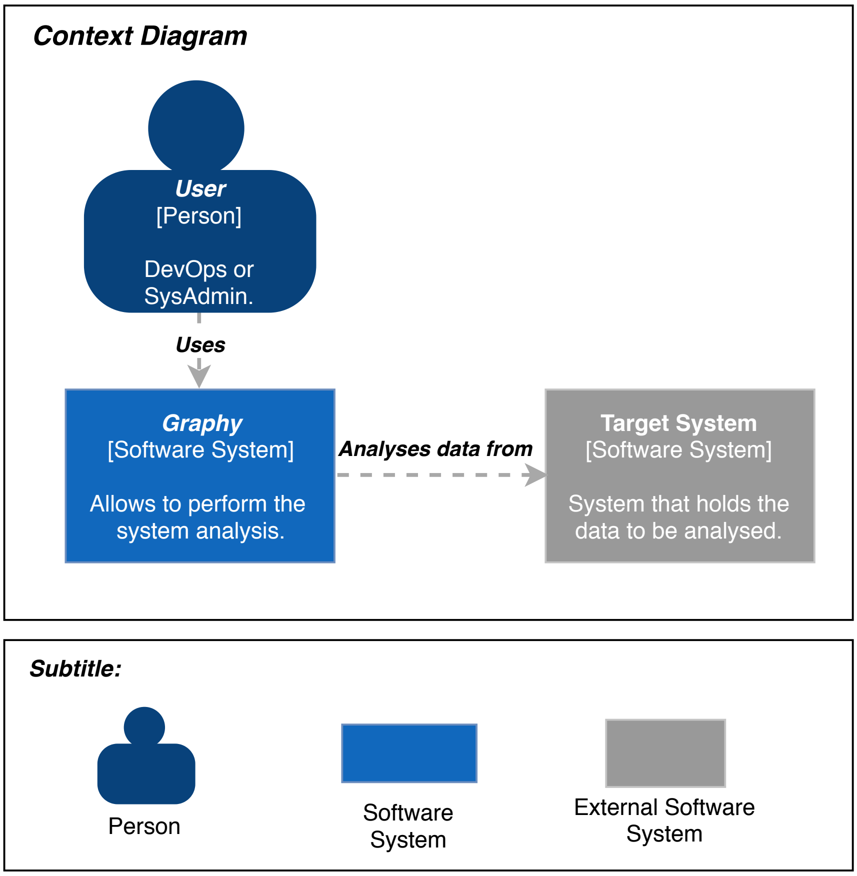


Figure 5.2: Context diagram.

In the presented diagram we can see that the solution, *Graphy*, will receive interactions from the user, as it need someone to control its operations and will analyse data from an external system infrastructure, defined as the target system.

5.4.2 Container Diagram

In this subsection the container diagram is presented. This kind of diagram allows us to perform a "zoom-in" in the context diagram, and get a new overview of this system, therefore in this diagram we can see the high-level shape of the software architecture and how responsibilities are distributed across it. The figure 5.3 presents the container diagram for this solution.

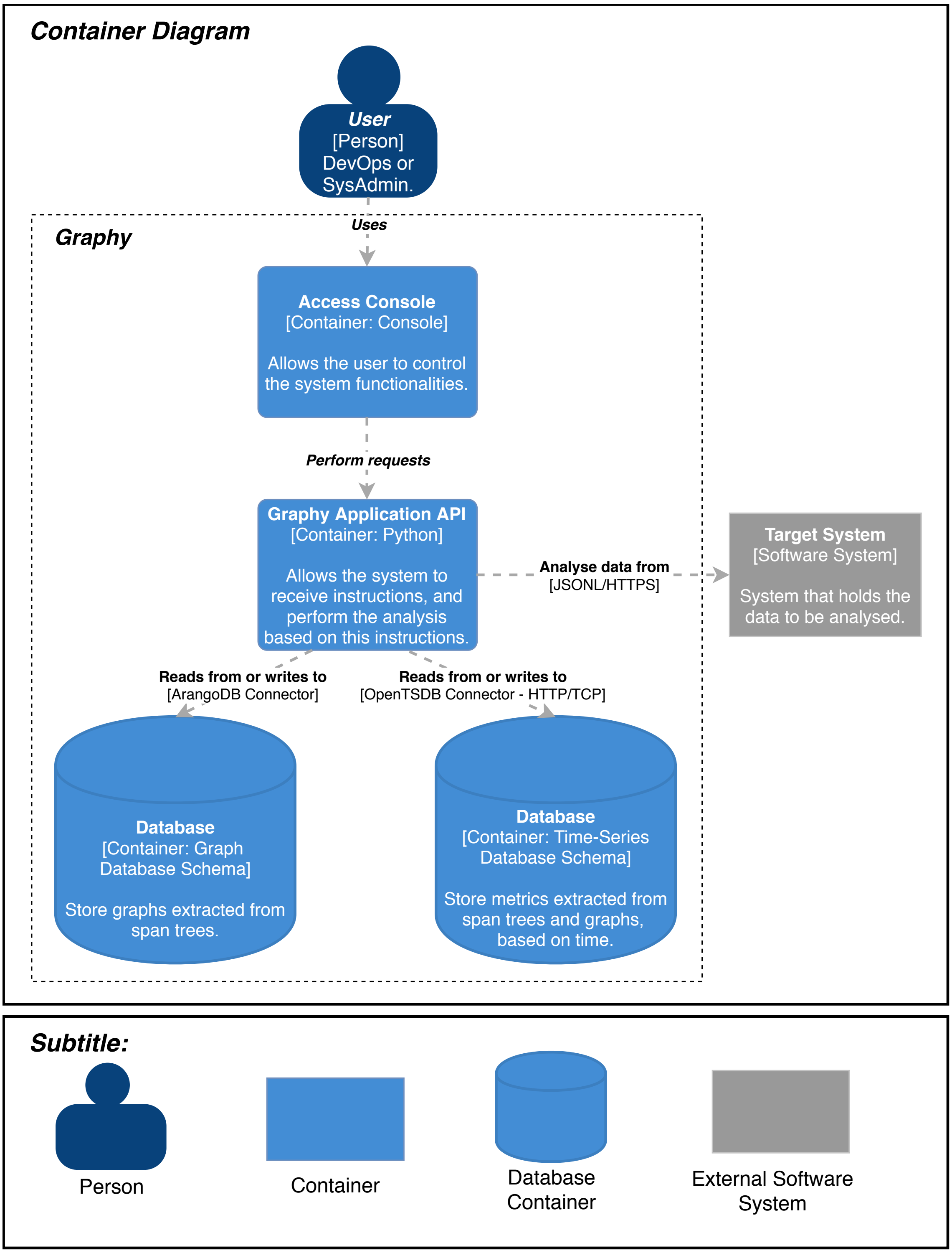


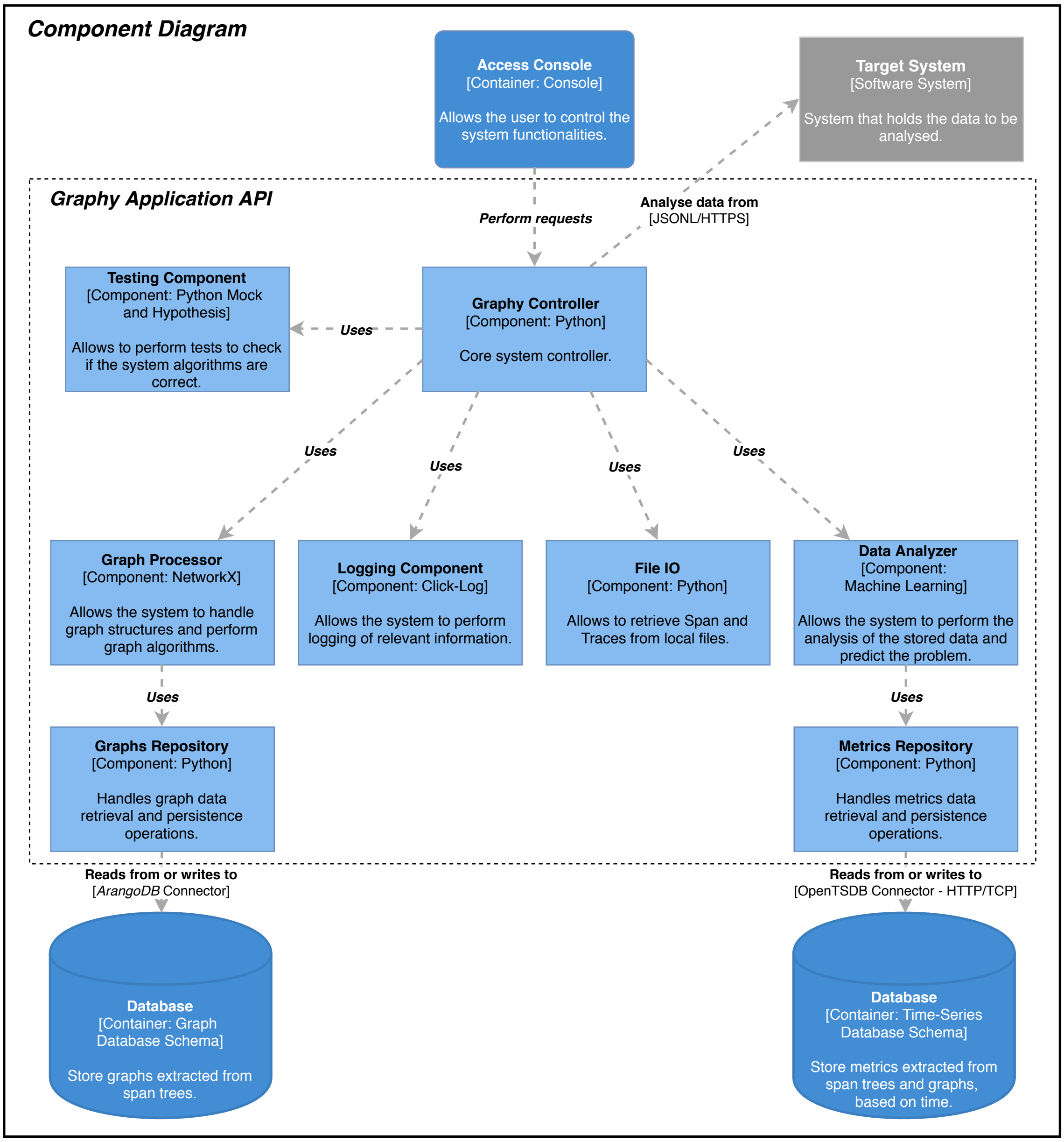
Figure 5.3: Container diagram.

In this figure we can see the main containers involved in the Graphy system. The first one, from top to bottom, is the *Access Console* and this container was considered as it is needed for the user to be able interact with the *Graphy Application API*. The *Graphy Application API* controls the entire Graphy system, that uses a communication protocol to retrieve information from the system it must analyse, and two databases to store the information resulted from the processing, a Graph Database (GDB) and a Time Series

Database (TSDB).

5.4.3 Component Diagram

In this subsection, the last diagram is presented, the component diagram. This kind of diagram gives us a more deeper vision about the system, and therefore, it presents us with its main components. The figure 5.4 presents the component diagram for this solution.



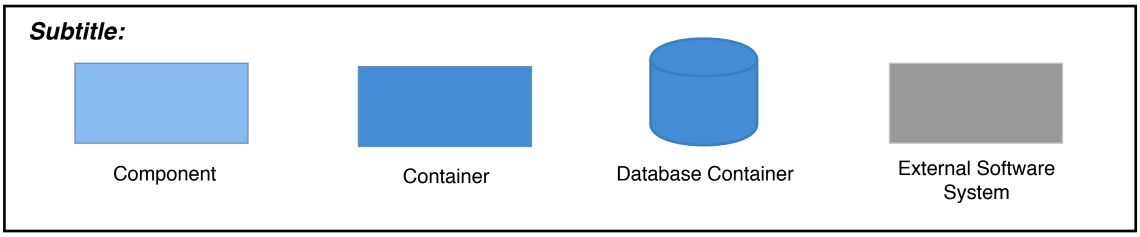


Figure 5.4: Component diagram.

In this last diagram, we have a lower level of abstraction sight of the *Graphy Application API* container, composed by a total of eight components. At its core we have the *Graphy Controller*, that has the responsibility of controlling the remaining components involved in the system, orchestrating the actions requested by the user thought the *Access Console* and performed by the whole system. In the remaining components, we have two that have greater relevance, the *Graph Processor* and the *Data Analyzer*. These components are responsible of processing the data converted from the Spans and Traces data, use the corresponding repository component to store or retrieve information from its database, and perform the analysis of the data. Finally we have the *Testing Component*, which is responsible of test the systems algorithms, the *Logging Component* which is responsible of log the relevant information and the *File IO* which is responsible of handle the files involved in the system processes.

To check the architecture produced, and as said before, we will now cycle thought all the QA and check were they are reflected in the architecture presented for this solution, and explain the trade-off involved and what were our considerations about each one.

QA1 and QA2 are satisfied by the simple fact that the system is able to analyse data from an external system, using a transmission protocol where data is exchanged thought HTTPS in JSONL format. As the data is only read from an exposed endpoint presented in the target system, this will not affect the normal work of the external system as this preserves its independence.

QA3 is satisfied when we decide to use NetworkX as the technology to process our graphs. This technology does not scale horizontally, however it has a very decent performance and it's able to retrieve a certain measure of a graph with about 100.000 nodes, in near 15 seconds[15]. From 1.000.000 spans, in normal conditions, we will never be able to get a graph of this kind of size, as with our experiments, with 100.000 spans we were able to get a graph of almost 20 nodes, and with 200.000 spans we were able to get a graph of almost 30 nodes. In the end, we are considering this time and span quantity to be sure that our tool will give us good times and ease our work of performing the research and implementation of this kind of tool.

QA4 is satisfied because we decided to use two databases that are scalable horizontally by design, the ArangoDb for a GDB and the OpenTSDB for a TSDB, both presented in the 3.2.3 and 3.2.4 subsections respectively. In the end, this QA can not be fully satisfied because we can not scale our system entirely, due to the fact of the technology that we chose to perform the graph processing. However, we have chosen this technology because it can perform much more graph algorithms, as we can see in the figure 3.6, and this is much more relevant for our main purpose.

QA5 is satisfied by the existence of the component *Logging Component*, that allows the system to perform logging of relevant information. For the technology here, we decided to use click-log[32], a python library used for logging purposes as it has all the main capabilities needed here to perform the logging.

QA6, like the previous one is satisfied by the existence of a certain component, the *Testing Component*, which implements all the capabilities and functionalities to perform tests and check if the systems is working correctly.

Finally, for the only technical restriction raised, we can see that it is satisfied by the usage of the OpenTSDB as our TSDB.

With all that was presented before, we conclude that our solution satisfies all the quality attributes, business constraints and technical restrictions, and therefore, we may claim that this architecture fits our needs as a solution, taking into consideration the trade-offs involved and explained before.

This page is intentionally left blank.

References

[1] Apache Software Foundation, *Apache Giraph*. [Online]. Available: http://giraph.apache.org/.

[2] ArangoDB Inc., *ArangoDB Documentation*. [Online]. Available: https://www.arangodb.com/documentation/.

[3] ——, *ArangoDB Enterprise: SmartGraphs*. [Online]. Available: https://www.arangodb.com/why-arangodb/arangodb-enterprise/arangodb-enterprise-smart-graphs/.

[4] ——, *What you can't do with Neo4j*. [Online]. Available: https://www.arangodb.com/why-arangodb/arangodb-vs-neo4j/.

[5] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani, "TAO: Facebook's Distributed Data Store for the Social Graph", [Online]. Available: https://cs.uwaterloo.ca/~brecht/courses/854-Emerging-2014/readings/data-store/tao-facebook-distributed-datastore-atc-2013.pdf.

[6] S. Brown, *Distributed big balls of mud*. [Online]. Available: http://www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html.

[7] ——, *The C4 model for software architecture*. [Online]. Available: https://c4model.com/.

[8] C. Churilo, *InfluxDB Markedly Outperforms OpenTSDB in Time Series Data & Metrics Benchmark*. [Online]. Available: https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics-benchmark/.

[9] Cloud Native Computing Foundation, *What is Kubernetes?* [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[10] A. Deshpande, *Surveying the Landscape of Graph Data Management Systems*. [Online]. Available: https://medium.com/@amolumd/graph-data-management-systems-f679b60dd9e0.

[11] Docker Inc., *What is a Container*. [Online]. Available: https://www.docker.com/resources/what-container.

[12] ——, *Why Docker?* [Online]. Available: https://www.docker.com/why-docker.

[13] Google LLC, *What is OpenCensus?* [Online]. Available: https://opencensus.io/.

[14] K. V. Gundy, *Infographic: Understanding Scalability with Neo4j*. [Online]. Available: https://neo4j.com/blog/neo4j-scalability-infographic/.

[15] T. Hladish, E. Melamud, L. A. Barrera, A. Galvani, and L. A. Meyers, "EpiFire: An open source C++ library and application for contact network epidemiology".

[16] InfluxData, *InfluxDB GitHub.* [Online]. Available: https://github.com/influxdata/influxdb.

[17] ——, *Time Series Database (TSDB) Explained.* [Online]. Available: https://www.influxdata.com/time-series-database/.

[18] JaegerTracing, *Jaeger GitHub.* [Online]. Available: https://github.com/jaegertracing/jaeger.

[19] W. Li, *Anomaly Detection in Zipkin Trace Data*, 2018. [Online]. Available: https://engineering.salesforce.com/anomaly-detection-in-zipkin-trace-data-87c8a2ded8a1.

[20] R. McColl, D. Ediger, J. Poovey, D. Campbell, and D. A. Bader, "A Performance Evaluation of Open Source Graph Frameworks", Georgia, [Online]. Available: http://www.stingergraph.com/data/uploads/papers/ppaa2014.pdf.

[21] Neo4J Inc., *No Title.* [Online]. Available: https://neo4j.com/docs/.

[22] NetworkX developers, *NetworkX.* [Online]. Available: https://networkx.github.io/.

[23] OpenStack, *What is OpenStack?* [Online]. Available: https://www.openstack.org/software/.

[24] OpenTracing, *OpenTracing Data Model Specification.* [Online]. Available: https://github.com/opentracing/specification/blob/master/specification.md.

[25] OpenTracing.io, *What is OpenTracing?* [Online]. Available: https://opentracing.io/docs/overview/what-is-tracing/.

[26] OpenTSDB, *OpenTSDB.* [Online]. Available: https://github.com/OpenTSDB/opentsdb.

[27] OpenZipkin, *Zipkin Repository.* [Online]. Available: https://github.com/openzipkin/zipkin.

[28] C. Richardson, *Microservices Definition.* [Online]. Available: https://microservices.io/.

[29] J. Shun and G. E. Blelloch, "Ligra: A Lightweight Graph Processing Framework for Shared Memory", Pittsburgh, [Online]. Available: https://www.cs.cmu.edu/~jshun/ligra.pdf.

[30] The GanttProject Team, *GanttProject.* [Online]. Available: https://www.ganttproject.biz/.

[31] thefreedictionary.com, *Observing definition.* [Online]. Available: https://www.thefreedictionary.com/observing.

[32] M. Unterwaditzer, *Click-log: Simple and beautiful logging.* [Online]. Available: https://click-log.readthedocs.io/en/stable/.

[33] Wikipedia, *ACID(Computer Science).* [Online]. Available: https://en.wikipedia.org/wiki/ACID_(computer_science).

[34] ——, *Graph.* [Online]. Available: https://en.wikipedia.org/wiki/Graph_(discrete_mathematics).

[35] ——, *Graph database.* [Online]. Available: https://en.wikipedia.org/wiki/Graph_database.

[36] ——, *Kanban Board.* [Online]. Available: https://en.wikipedia.org/wiki/Kanban_board.

[37] ——, *Software License*. [Online]. Available: `https://en.wikipedia.org/wiki/Software_license`.

[38] ——, *Time series database*. [Online]. Available: `https://en.wikipedia.org/wiki/Time_series_database`.