Masters in Informatics Engineering
Internship
Final Report

Soft Real Time Processing Pipeline for Healthcare Related Events

Jaime Filipe Carvalho Pereira Correia
jaimec@student.dei.uc.pt

Supervisors:

Joel Perdiz Arrais, University of Coimbra
jpa@dei.uc.pt

Marco Tinoco, MedicineOne
marco@medicineone.net

1st September 2016

FCTUC DEPARTMENT
OF INFORMATICS ENGINEERING
FACULTY OF SCIENCES AND TECHNOLOGY
UNIVERSITY OF COIMBRA

"I think the big mistake in schools is trying to teach children anything, and by using fear as the basic motivation. Fear of getting failing grades, fear of not staying with your class, etc. Interest can produce learning on a scale compared to fear as a nuclear explosion to a firecracker."

Stanley Kubrick

UNIVERSITY OF COIMBRA

Abstract

Faculty of Sciences and Technology
Department of Informatics Engineering
Masters in Informatics Engineering

Soft Real Time Processing Pipeline for Healthcare Related Events

by Jaime Filipe Carvalho Pereira Correia

Data analytics and business intelligence approaches have proven their potential for assisting decision making and in some cases even achieve complete automation. The healthcare industry, public and private, generates and stores large amounts of data. Such data is valuable not only for internal usage but also for commercialization as data centric services, relevant to the surrounding markets and organization, such as the pharmaceutical complex.
MedicineOne is a Portuguese technological company that develops and commercializes electronic health record (EHR) and clinic management systems (CMSs). Aware of their privileged position, the company decided to build a soft real time platform to support healthcare event aggregation, analysis and visualization. Building this new system will outfit the company with the necessary technological infrastructure and expertise to start providing new services to their clients, empowering them with insight from their operational data. The extracted intelligence creates conditions to streamline internal processes and improve resource allocation. A direct consequence is better healthcare.
Companies operating abroad in homologous positions have proven that this business model is viable. There is a clear demand from the pharmaceutical industry for ways to measure, in a short time-frame, the success of their products and services and how they rank among competitors. Access to healthcare events as they happen would give institutions such as governments and pharmacies an understanding of the state of the healthcare system as well as the health of the general population. It would also make it possible to accurately determine the demand for drugs and diagnostic tests, increasing efficiency.
During this project, the intern was responsible for studying the state of the art and become proficient enough to build a scalable soft real time data ingestion platform capable of receiving and aggregating events form CMS / EHR and make them available for processing and querying.

KEYWORDS: business intelligence, aggregation, real time, soft real time, stream processing, batch processing, in memory, database, noSQL, visualization

Contents

List of Figures

List of Tables

Glossary

availability The degree to which a system or component is operational and accessible when required for use. Often expressed as a probability. Availability is usually expressed as a ratio of the time that the service is actually available for use by the business to the agreed service hours[1]. 5

container A lightweight alternative to full machine virtualization that involves encapsulating an application, effectively giving it its own operating environment, such as operative and file systems. 44, 48, 54

containerizing Wrapping / packaging an application in a container. 44

MATE Codename of the project. 5, 6, 18, 36–38, 40–42, 45, 56, 57, 59–62, 65, 66, 68

quorum The minimum number of members of an assembly or society that must be present at any of its meetings to make the proceedings of that meeting valid. 50

real time In this document, real time is an abuse of language used to refer to soft real time in non technical contexts. 4, 19–22, 24, 26, 36, 62

reliability The probability that software will not cause the failure of a system for a specified time under specified conditions[1]. 5

REST RESTful application program interface (API) is an API that uses hypertext transfer protocol (HTTP) requests to GET, PUT, POST and DELETE data. 62

soft real time A real time system where the utility of the results produced by a task decreases over time after the deadline expires, as defined by K. G. Shin et al.[2]. viii, 1, 2, 5, 6, 10, 20–23, 36, 40, 45, 47, 62, 65, 68

Acronyms

API application program interface. viii, 26, 35, 45, 56, 61, 62

BI business intelligence. 5, 20, 22

CMS clinic management system. ii, 6, 8, 40, 44, 56, 60

CQL cassandra query language. 58

DC/OS Data Center Operating System[3]. 45, 46, 48–50, 52–54, 60–62, 64, 67, 68

DNS domain name system. 44, 49

DWH data warehouse. 5, 19, 20

EHR electronic health record. ii, 18

ETL extract, transform and load. 6, 20, 21, 25, 45, 59, 60, 64

ETLR extract, transform, load and refresh. 20

GUI graphical user interface. 62, 63

HTTP hypertext transfer protocol. viii, 54

IoT internet of things. 25

IT information technology. 38

JIT just in time. 16, 21

JSON javaScript object notation. 54, 56, 57, 60, 64

KPI key performance indicator. 2

NoSQL not only structured query language (SQL). 44, 60

OLAP online analytical processing. 2, 19, 21, 25, 41–43, 45, 47, 60, 65, 67

OLTP online transaction processing. 20

RDBMS relational database management system. 21

ROI return of investment. 37

RTDWH real time data warehouse. 5

SMART specific, measurable, assignable, realistic and time related. 27, 29

SOA service oriented architecture. 37, 38, 50, 56, 66

SQL structured query language. ix, 44

SSH secure socket shell. 51

WAG wild altogether guess. 13

WCF windows communication foundation. 56

Chapter 1

Introduction

1.1 Purpose

This document describes the requirement analysis, planning, implementation and testing of a soft real time event processing and storage solution at MedicineOne as part of a curricular internship.

On the following chapters, the project is described, firstly in terms of planning and employed methodologies and secondly by reporting the results of its constituent steps / components.

The document aims to make the process and technological decisions known to the reader and provide the necessary indicators so that they can recreate or evaluate them.

Finally, it will serve as an introductory point to new developers, joining the project, looking to understand the code base, documentation, state of the art and reasoning behind the decisions made along the way.

1.2 MedicineOne

The internship was hosted by MedicineOne, a Portuguese technological company that specializes in developing software for the healthcare market.

MedicineOne develops and commercializes a complete clinical management system with a strong market presence. The multiple solutions and deployments coursing with data create the ideal conditions for intelligence extraction.

FIGURE 1.1: MedicineOne logo (http://medicineone.net).

With very large deployments, some of them centralized and under the control of the company, events such as drug prescriptions can be ingested in soft real time and analyzed to create new intelligence based services and products. An example would be the ability to empower pharmacies with access to a current view of what drugs are being prescribed on their area of coverage.

Being in a favorable position, MedicineOne idealized the project and submitted it as a internship proposal for research and development.

1.3 Project Objective

The objective of the internship was to, starting from a product idea, analyze, plan, implement and validate a soft real time event aggregation, processing and visualization system for healthcare related events, such as drug therapy prescriptions. Generally it can be described as a soft real time online analytical processing (OLAP) system with especial focus on arbitrary aggregation and querying.

The objective of the system is providing insight into healthcare metrics such as drug and diagnostic test prescription frequencies as well as geographical and chronological distribution of certain events like pathology diagnoses. More specifically, create a platform atop of which higher level function can be implemented. Some examples are machine learning, notifications or complex event processing.

The information is already being generated and stored by the current deployments of MedicineOne's products. Through means of anonymization and aggregation, it becomes possible to treat it statistically and generate a set of metrics that will interest the healthcare professionals and surrounding industry. Even though the exact business model is not clear at the time of development, such a system can add value both to healthcare organizations and the pharmacy and related industries. And example of how it could do so would be providing key performance indicators (KPIs) and metrics that can contribute to a more efficient processes or market strategies.

Starting from the back-end components of the system, the objective was developing it as much as possible going towards the front-end as time and resources allowed. The success threshold is defined by the Annex A, containing a system requirements specification document, which means the project would be considered successful if all Medium and High Priority requirements were satisfied. This means that the Front-end components were considered optional. The rational behind that decision is that they were added to the scope of the project after the internship had begun. Also, it was very likely that there wouldn't be enough time given the size and breadth of the back-end.

For the company, the final objective is turning the result of the internship into a market ready product. There are preliminary plans to commercialize it in two different formats. The first is deployed on premise under the control of a client organization so they can explore their internal data. The other option is selling access to the platform as a service, putting focus on performance and scalability to accommodate a growing number of sources as well as a growing number of clients using the service.

1.4 Document Structure

The present document is organized as follows;

- **Introduction:** Describes the report and introduces the reader to the project and its context.
- **Project:** Contains the general description of the project, detailing its scope and objectives as well as the contributions made by the intern.
- **Plan and Methodology:** Overviews the general plan for the internship and employed project management methodologies, from life cycle to risk management.
- **State of the Art:** Presents the state of the art, not only from a technical standpoint, but also from a product perspective, presenting and briefly exploring similar solutions.
- **Requirements:** Describes the process of requirement analysis and turning the idea into a set of well defined requirements, presenting an introduction to the most relevant ones.
- **Risk:** Section dedicated to listing identified risks as well as how they were prioritized and dealt with.
- **Evaluation:** Various possible technologies and components were found and are discussed on the State of the Art. This chapter compares homologous solutions.
- **Architecture:** Contains the results of the architectural design step and respective backing information, such as drivers. Also, contains the necessary information to correctly interpret the architectural views and implement them.
- **Implementation:** Describes the implementation step of the project including programming, configuration and deployment of components as well as documentation of each of them for review and replication purposes.
- **Results:** This section presents and discusses the results and evaluates the success of the project.
- **Conclusion:** Section with the final reflections and conclusions.
- **References:** List of cited or referenced sources.

Chapter 2

Project

2.1 Motivation

MedicineOne develops and commercializes a complete clinical management system, the current iteration, MedicineOne 8, is still fundamentally a solution thought and designed for deployment on premise. Even though they provide hosted solutions, the data still lives isolated inside a logically and physically portioned relational database.

Given the importance of data driven approaches, with an ever growing list of studies and success examples, every company with that possibility wants to start exploring and extracting value from their data. Surrounding markets understand the value offered by products that can give them insight into the current state of the healthcare environment.

Besides the immediate and obvious benefits for the healthcare providers themselves, such a solution can add value to the pharmaceutical and other tightly connected industries, by giving them relevant, up to date, market and demand metrics. Being made available in real time can improve the actionability of the data as well as greatly reduce the time to take action leading to competitive advantage. A great example would be to monitor in real time how effective a new drug direct marketing campaign is, making it possible to adjust as necessary and monitor the effects.

With a large client base and large volumes of data, MedicineOne is in a privileged position to develop and market this service. A centralized system to aggregate the relevant information from all the individual deployments and empower its clients with the ability to explore and visualize this data in real time.

2.2 Overview

The project, code named MATE for ease of reference stems from the need to make use of the available data. Using empirical information extracted from real data to assist decision making can bring benefits to public and private organizations alike, however, historically, extracting business intelligence (BI) from data, and maintaining a data warehouse (DWH) are costly and high latency operations. From the moment an event takes place, it used to take at least days before the data could be analyzed using classic BI and DWH approaches. Presently, with the advancements in real time data warehouse (RTDWH) and stream processing, it is possible to analyze and visualize data in soft real time[4], creating the necessary tools for faster decision making, lower response times and increased efficiency.

MedicineOne wants a system that will aggregate relevant information, such as prescriptions and pathology diagnostics, from the various deployments of their MedicineOne 8 clinical management system. The system is supposed to ingest and process events fast enough to be able to present relevant information to the users in soft real time. Due to its area of application, the system needs to be reliable and tolerant to technical and human failures, putting great emphasis on availability and reliability.

From a high level of abstraction, the system needs to gather the events, store them and execute the necessary computations such as transforming them or generating new data. The existing data also needs to be made available for query with low latencies.

The system is aimed at the healthcare industry, possibly to be made available for subscription as a service, where the end users can extract insight from historic data or quickly get an overview of the current state of affairs, being able to for example assess the effects of recently implemented policies.

2.3 Product

The MATE system is meant to serve as the base for a commercial product, mentioned and described to some extent in various parts of this report, like chapter chapter 1. However, that aspect was not part of the scope and is meant to contextualize and help elicit and prioritize requirements. It also had the purpose of validating the usefulness of the resulting system. For that purpose when studying the state of the art, products similar to the one intended where analyzed in terms of commercial viability.

The product development was at a very early stage in the host company, it's exact features and commercialization plan being uncertain. To avoid restricting the evolution of the product and it's adaptability to market requirements, special care was taken in making sure that the more general solutions and approaches were given preference.

2.4 Scope

The purpose of this section if defining the scope internship, with special focus on what is expected of the intern. MedicineOne wants a system that can consume real time events from the various deployments of their CMS, for purposes of storage, aggregation, and processing to support analytical and visualization applications.

Initially the project was divided in two parts, visualization and data aggregation, assigned to two different internships. Due to external circumstances, the visualization internship was terminated and its scope merged into this one. However, the tasks resulting from this addition were kept as low priority due to the high likelihood of not having enough time to successfully complete them.

In order to avoid ambiguity when crediting contributions, a preoccupation brought by the supervisor, the second intern, in charge of the front-end was internally reallocated to another project at MedicineOne. To deal with the added workload both parties, intern and host company, agreed that the added requirements would not be included in the success threshold definition. In light of this, a more requirement oriented approach was taken, putting most of the effort into developing functionality with the intent of creating a prototype as close as possible to what would be expected of a system capable of sustaining an actual product. The selected approach was start from the back-end, and progress towards the front-end components, and do as much as possible in the available time.

This was reflected by the requirements in the form of prioritization, all requirements relating to the front-end, even though elicited and planed for, were considered low priority and only covered if they would not negatively impact the development of the back-end.

As a conclusion the intern was responsible for implementing a soft real time data storage and processing system, optimized for dimensional time series. On top of this, the system was also expected to implement mechanisms to be able to push data to the front-end, making it capable of displaying events and metrics to the users in real time (push). It was also the intern's responsibility to, at least partially, modify the MedicineOne 8 Server in order to make it capable of sending out events to a MATE deployment.

From a more technical perspective, the product MedicineOne wants to create will be an aggregation of a few different components, listed below and illustrated by Figure 2.1.

- MedicineOne 8, their CMS;
- MedicineOne 8 Mate Connector, responsible for sending events to the real time pipeline;
- Real time pipeline, responsible for ingesting events, aggregating and making them available for processing and query;
- Processing modules, responsible for data processing tasks, like extract, transform and load (ETL) or machine learning;

- Front-end, responsible for making the information available to the end clients;

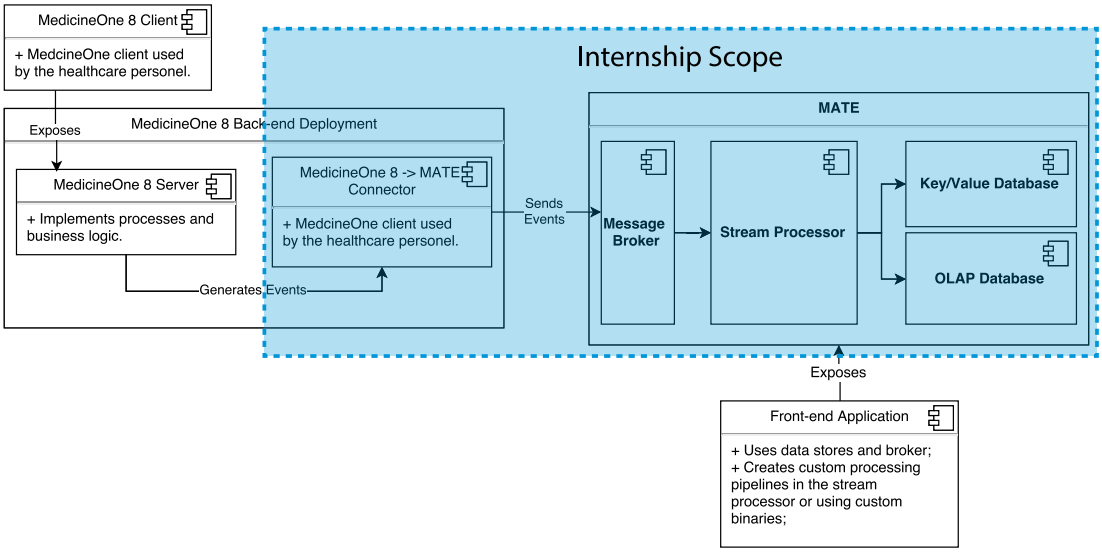


FIGURE 2.1: Visual representation of the scope.

For disambiguation purposes, follows a list of concerns that are outside of the scope of this project;

- Securing the infrastructure, it is assumed to be secured with conventional methods, such as firewalls and a distinction between internal machines and internet facing machines;
- Data science or machine learning, while the system tries to be general to support such applications, implementing or studying them, is not part of the scope and outside of the interns area of knowledge;
- Product and market development, the project is purely technical;

2.5 Contributions

The following contributions were made towards the advancement of the project:

- First Semester:
  - Study of the state of the art, both in technology and similar products;
  - Requirement elicitation, analysis and documentation;
  - Risk management plan and analysis;
  - Evaluation of existing technologies;
  - Architectural design;
- Second Semester:

- Study and selection of infrastructure and deployment management systems;
- Selection of technologies and platforms;
- Configuration and composition of the selected technologies and platforms;
- Automation and execution of deployment;
- Development of a event generation service in the existing CMS;
- Development of event processing components;
- Evaluation of the results;

Chapter 3

Plan and Methodology

Being a rather large project for a single developer, good planning and the choice of an adequate life cycle were fundamental for the success of the project and quality of the produced artifacts. The project was divided into two distinct stages, with different levels of dedication. During the first semester the project was given 16 hours of dedication weekly and 40 during the second.

TABLE 3.1: Time budget for the project.

	First Semester	Second Semester	Total
Start Date:	29/9/15	2/6/2016	
End Date:	22/1/16	17/6/16	
Weekly Effort (hours):	16	40	
Business Days:	84	95	179
Weeks:	16	19	35
Total Effort (hours):	256	760	1016
Effort Percentage:	25%	75%	100%

This distribution of effort was a conditioning factor when it comes to task distribution and life cycle. Trying to get development work done during the first semester would not be productive since it only corresponds to 25% of the time. This aspect becomes even more relevant when considered that the first semester was shared with other courses.

On that basis the tasks scheduled for the first semester were:

- Familiarization with the domain;
- Research of the state of the Art;
- Requirements elicitation and analysis;
- Planning and life-cycle selection;
- Architectural design;
- Technology selection;
- Production of this report;

In summary, the first semester was used for planning and document production, creating the supporting structures to guide implementation during the second semester. While some life-cycle methodologies encourage less formal documentation, the decision was made, that since the project was starting from an

unstructured idea, at least the requirements and the architecture should be formalized.

The second semester was dedicated to more practical tasks, detailed below.

- Automating deployment of platforms;
- Studying orchestration and scheduling platforms;
- Streamlining deployment of Apache Mesos;
- Deployment of Mesosphere DC/OS 1.6;
- Porting existing platforms to DC/OS;
- Developing MedicineOne 8 to MATE connector and event generator;
- Developing data processing topologies;
- Testing and benchmarking;

3.1 System Development Cycle

Being the first of its kind in the organization, the system development cycle[1] required sufficient flexibility to explore what was in fact viable while keeping as close as possible to its initial objectives. That coupled with the fact the intern had no previous experience with soft real time event processing made an agile system development cycle methodology with emphasis on incremental development the best fit. It was important to make the project easy to gradually steer in the desired direction while being forgiving to mistakes and making it possible to fix them without having to re-plan everything.

Incremental planing and implementation with focus on delaying decisions, gave the project the necessary agility to accommodate change and correct mistakes. With that in mind as a limiting factor, a few different development cycles methodologies were explored before making a decision.

3.1.1 Scrum

Scrum is a management and control process devised to reduce complexity and focus on building software that creates values for businesses. Management and teams are able to get their hands around the requirements and technologies, and deliver working software, incrementally and empirically[5].

The key concepts of Scrum, as defined by the Scrum Alliance[7], necessary to understand this report, are:

- **Sprint:** A spring is the fundamental work unit of scrum, defined by a timebox of one month or less during which a deployment ready, usable, and potentially releasable product Increment is created. The sprint is by nature immutable, meaning that its goal will not change while it is being undertaken. May be canceled when necessary.

- **Product backlog:** The Product Backlog is an ordered, prioritized list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability,

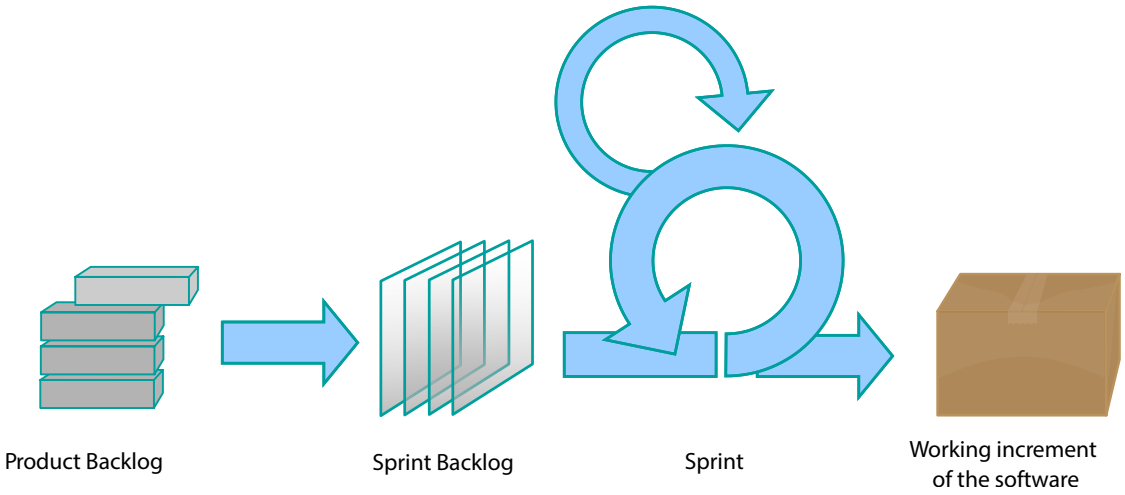


FIGURE 3.1: A visual depiction the key concepts in Scrum[6] (http://wikimedia.org).

and ordering. It lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases.

- **Sprint backlog:** The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a "Done" Increment. The Sprint Backlog makes visible all the work that the Development Team identifies as necessary to meet the Sprint Goal.

- **Increment:** Is the sum of all completed items of the Product Backlog, an increment is generated as the result of each sprint and it must be complete, according to the goal set for the sprint. By definition, it must also be usable, deployment ready in other words, regardless of whether the Product Owner decided to deploy it or not.

- **Product Owner:** Is responsible for maximizing the value of the product and the efficiency of the development team. Even though it varies, their usual responsibilities are managing the product backlog and properly prioritize them tasks in it, and making sure the product stays consistent with its goals.

From scrum it imported the concepts of sprint, product backlog and sprint backlog. However, to make it more agile, sprints were purposely kept short and task oriented as advised by just in time delivery methodologies like *kanban*[8].

3.1.2 Kanban

Kanban is the name commonly given to a software development management process based on Toyota's JIT production management system[9]. The most characteristic principle, is having a board divided into sections each representing a stage of the process where the tasks are posted and moved as they advance in completion.

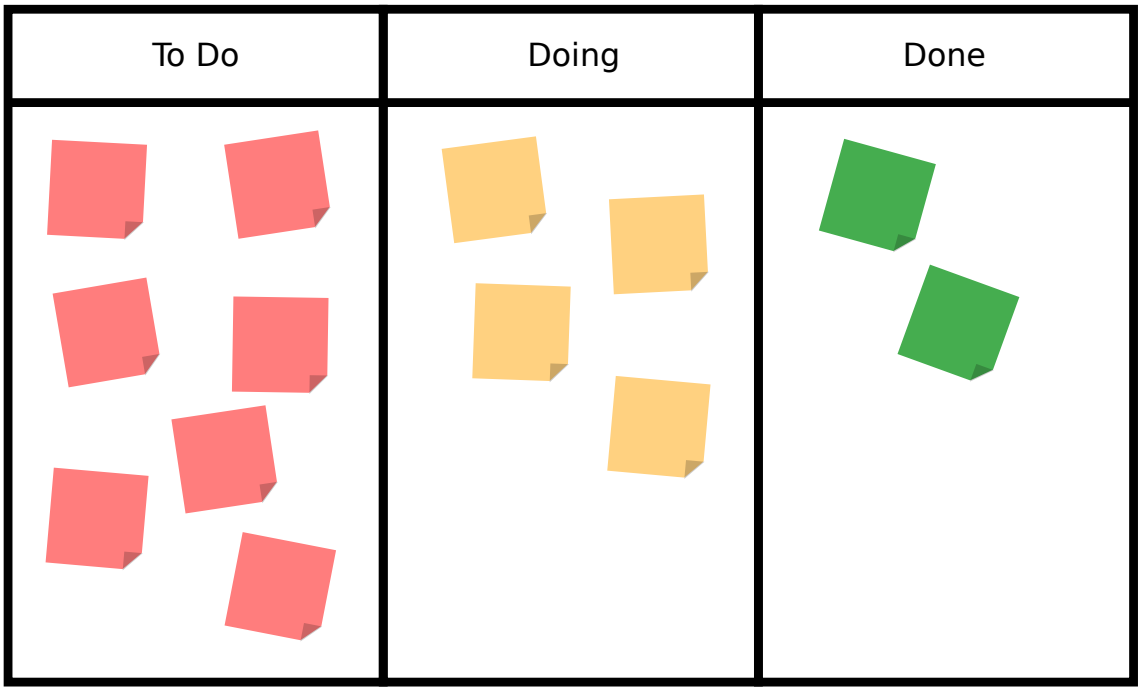


FIGURE 3.2: Illustration of a *Kanban* board.

When applied to software development it tries to increase focus by reducing tasks to atomic user stories, a small story describing the desired functionality, and making the work-flow visual and team oriented. By having a board with tasks, as user stories, and their current stage in process, it helps reduce bottlenecks in cases where the development process is multi staged. If a team is overworked and bottle-necking the process, the problem becomes evidently and visually apparent.

3.1.3 Selected Methodology

The decision was made with a few considerations in mind;

- The intern would be working on an isolated branch project;
- Most of the contacts with other organizational structures would happen at requirement elicitation;
- The lack of experts on the area made technical review a non issue to an extent;
- Most validation would happen on results and properties of the resulting system;

- A lot of the effort would be directed to research and technology selection;

As there was no need to formally communicate a lot of technical results on a regular basis, a lighter methodology with emphasis on production was preferred. However, due to the technical breath of the project, dwelling into a multitude of domains, it was necessary to still have a formal way of breaking down and managing the activities. The objective being help maintain the effort task oriented and make prioritization decisions with a good overview of what was the current state of completion.

To fit all these requirements, a hybrid methodology gave the best chances of success. Incorporating most of *kanban's* principles while at the same time borrowing the notions of iterative development, product and sprint backlog from SCRUM.

During development, at the beginning of each sprint, a functional requirement, described as a user story, was selected from the *kanban* board, and worked on for the duration of the sprint, producing a version of the system that fulfills it. During a spring, each user story went through the following stages, represented as columns in the *kanban*;

- To Do;
- Research & Prototyping;
- Implementation;
- Validation;
- Done.

Sprints lasted around 1 work week (40 hours).

3.2 Chronogram

This section shows the estimated duration and scheduling of the constituents of the project. All shown durations are the result of a three point wild altogether guess (WAG) estimation process. While not ideal, this was the only option since both the hosting organization and the intern lacked the necessary expertise to make better estimations.

To improve the quality of the results, three estimations were made for each task, representing the Optimistic (O), Pessimistic (P) and Most Likely (M) scenarios. The final estimations(E) were calculated according to the following formula:

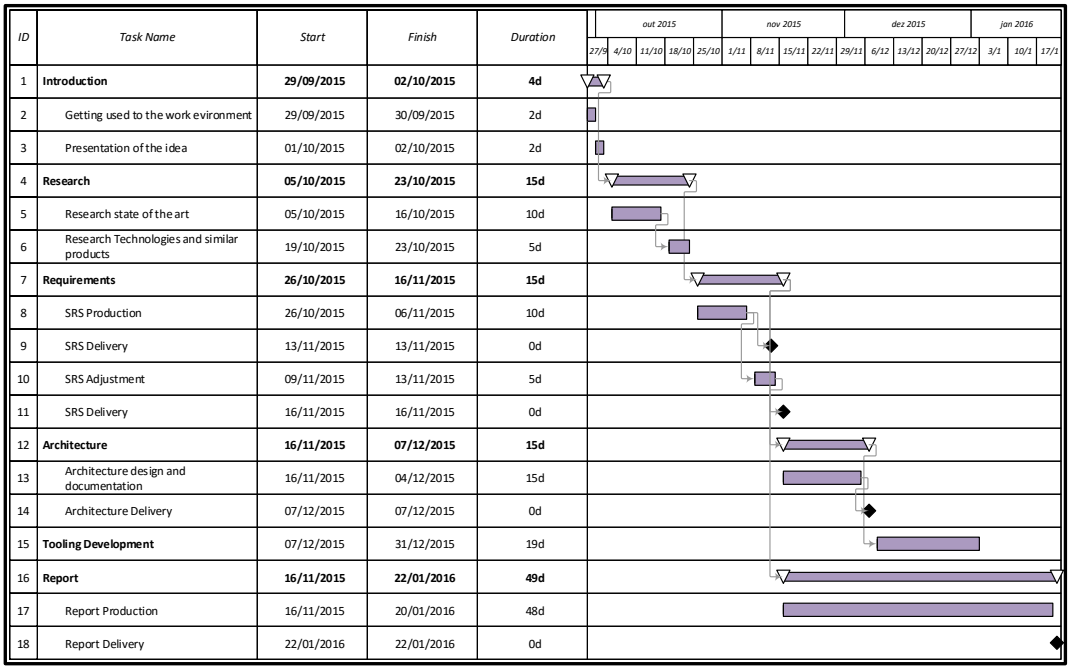
$$E = \frac{O + 4M + P}{6} \tag{3.1}$$

$$\sigma_E = \frac{P - O}{6} \tag{3.2}$$

The weights used in this estimation were chosen for being widely accepted by the industry. They empirically produce good results, and are based on assumption that this type of data typically follows well known distributions[10].

3.2.1 First Semester

The chronogram on figure 3.3 shows the plan for the first semester, with allocated tasks mainly concerning research, requirements analysis, architecture and documentation efforts.



ID	Task Name	Start	Finish	Duration
1	**Introduction**	**29/09/2015**	**02/10/2015**	**4d**
2	Getting used to the work evironment	29/09/2015	30/09/2015	2d
3	Presentation of the idea	01/10/2015	02/10/2015	2d
4	**Research**	**05/10/2015**	**23/10/2015**	**15d**
5	Research state of the art	05/10/2015	16/10/2015	10d
6	Research Technologies and similar products	19/10/2015	23/10/2015	5d
7	**Requirements**	**26/10/2015**	**16/11/2015**	**15d**
8	SRS Production	26/10/2015	06/11/2015	10d
9	SRS Delivery	13/11/2015	13/11/2015	0d
10	SRS Adjustment	09/11/2015	13/11/2015	5d
11	SRS Delivery	16/11/2015	16/11/2015	0d
12	**Architecture**	**16/11/2015**	**07/12/2015**	**15d**
13	Architecture design and documentation	16/11/2015	04/12/2015	15d
14	Architecture Delivery	07/12/2015	07/12/2015	0d
15	**Tooling Development**	07/12/2015	31/12/2015	19d
16	**Report**	**16/11/2015**	**22/01/2016**	**49d**
17	Report Production	16/11/2015	20/01/2016	48d
18	Report Delivery	22/01/2016	22/01/2016	0d

FIGURE 3.3: Chronogram of the 1st Semester.

Not everything went according to plan and only about 65% of the planned effort was actually put in. This delayed some tasks on the chronogram and forced the Tooling Development task to be reallocated to the second semester. However, the remaining tasks, the most important ones, concerning planning and documentation, were still concluded with success laying out a solid foundation for the work to develop on the second semester.

The research phase, even though longer than planned, was satisfactory and gave the developer the necessary know-how and awareness of the employed techniques and existing technologies to achieve the goal of the project. This included real time event ingestion, aggregation and visualization as well as an understanding of the limits of what is currently viable. Due to the approach taken during the research phase, using readily available articles, use cases and benchmarks as the main sources of information, most choices lack a more formal, peer reviewed publication to support them. However, empirical data makes them very likely to hold true.

The requirements were successfully elicited from João Miguel, chairman of MedicineOne as well as the project owner. This was done through a couple of formal meetings and a few more informal conversations for validation. After vetting them for viability, requirements were then distilled into a SRS document that was submitted to a revision and correction process until they were accepted,

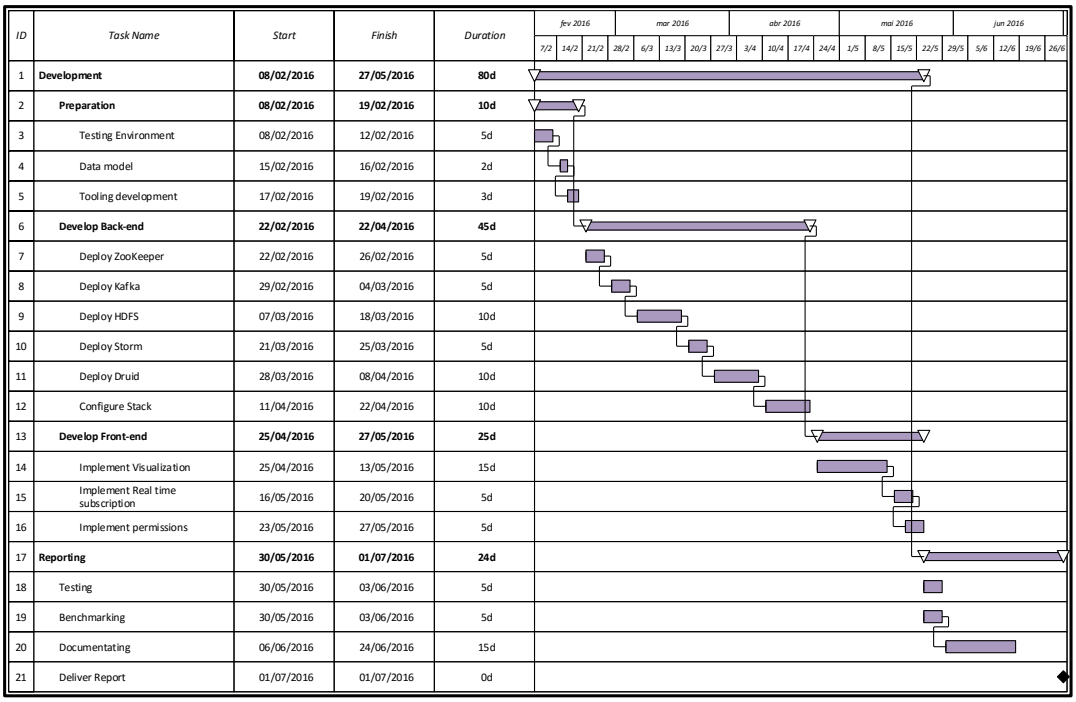
setting the first formal definition of the scope of the project as well as the priorities of each functionality.

After making sure that the project was well understood and its scope well defined, the architecture was designed through an incremental process, creating a proposal and validating it against the SRS until all the drivers were satisfied.

In the end it was presented to supervisor and approved.

3.2.2 Second Semester

The development cycle methodology of the project discourages making decisions too early. However, based on the architecture proposal, the list of tasks and their estimated durations it was conjectured that the second semester would loosely follow what is depicted by the gantt chart in Figure 3.4.



ID	Task Name	Start	Finish	Duration
1	**Development**	**08/02/2016**	**27/05/2016**	**80d**
2	**Preparation**	**08/02/2016**	**19/02/2016**	**10d**
3	Testing Environment	08/02/2016	12/02/2016	5d
4	Data model	15/02/2016	16/02/2016	2d
5	Tooling development	17/02/2016	19/02/2016	3d
6	**Develop Back-end**	**22/02/2016**	**22/04/2016**	**45d**
7	Deploy ZooKeeper	22/02/2016	26/02/2016	5d
8	Deploy Kafka	29/02/2016	04/03/2016	5d
9	Deploy HDFS	07/03/2016	18/03/2016	10d
10	Deploy Storm	21/03/2016	25/03/2016	5d
11	Deploy Druid	28/03/2016	08/04/2016	10d
12	Configure Stack	11/04/2016	22/04/2016	10d
13	**Develop Front-end**	**25/04/2016**	**27/05/2016**	**25d**
14	Implement Visualization	25/04/2016	13/05/2016	15d
15	Implement Real time subscription	16/05/2016	20/05/2016	5d
16	Implement permissions	23/05/2016	27/05/2016	5d
17	**Reporting**	**30/05/2016**	**01/07/2016**	**24d**
18	Testing	30/05/2016	03/06/2016	5d
19	Benchmarking	30/05/2016	03/06/2016	5d
20	Documentating	06/06/2016	24/06/2016	15d
21	Deliver Report	01/07/2016	01/07/2016	0d

FIGURE 3.4: Planned chronogram of the 2nd Semester.

The plan was starting on some preparations deemed necessary, namely defining the data model and creating mock event originators. This was meant to avoid getting tangled into legacy integration too early on, and help prepare the testing environment. This step would facilitate the following ones increasing the productivity of the developer.

The lion share of the remaining time was scheduled for sprints implementing user stories and making sure the project would stay on track. Even though a rather inflexible list of tasks is presented, with defined technologies and platforms, all of them are used for illustration purposes and subject to change if the need were to arise. An example would be finding a better solution, or finding out that one of them does not perform as expected. The ability to makes changes

during the development is one of the main reasons why an agile just in time (JIT) methodology was chosen.

The second semester had considerable deviations from plan due to both internal and external circumstances. In reality the distribution was as shown in Figure 3.5.

ID	Task Name	Start	Finish	Duration
1	**Development**	**08/02/2016**	**02/08/2016**	**127d**
2	**Preparation**	**08/02/2016**	**19/02/2016**	**10d**
3	Testing Environment	08/02/2016	12/02/2016	5d
4	Data model	15/02/2016	16/02/2016	2d
5	Tooling development	17/02/2016	19/02/2016	3d
6	**Develop Back-end**	**22/02/2016**	**02/08/2016**	**117d**
7	Deploy ZooKeeper	22/02/2016	26/02/2016	5d
8	Deploy Kafka	29/02/2016	04/03/2016	5d
9	Deploy HDFS	07/03/2016	18/03/2016	10d
10	Deploy Storm	21/03/2016	25/03/2016	5d
11	Deploy Druid	28/03/2016	08/04/2016	10d
12	Automate Mesos Deployment	11/04/2016	29/04/2016	15d
13	Deploy Kafka on Mesos	02/05/2016	06/05/2016	5d
14	Deploy Cassandra on Mesos	09/05/2016	20/05/2016	10d
15	Deploy Storm on Mesos	23/05/2016	03/06/2016	10d
16	Write Processing Topologies	06/06/2016	17/06/2016	10d
17	Replace Mesos with DC/OS 1.7	20/06/2016	08/07/2016	15d
18	Update Kafka and Cassandra	11/07/2016	12/07/2016	2d
19	Containerize Storm 1.0	13/07/2016	26/07/2016	10d
20	Containerize Druid	27/07/2016	02/08/2016	5d
21	**Reporting**	**03/08/2016**	**01/09/2016**	**21d**
22	Testing	03/08/2016	09/08/2016	5d
23	Benchmarking	03/08/2016	09/08/2016	5d
24	Documentating	10/08/2016	30/08/2016	15d
25	Deliver Report	01/09/2016	01/09/2016	0d

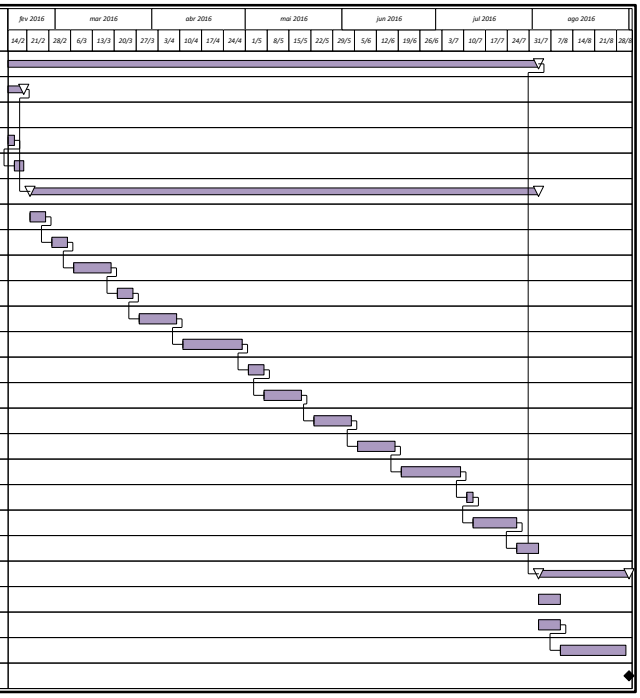


FIGURE 3.5: Chronogram of the 2nd Semester.

Reasoning for the deviation are explored in detail in Chapter 8.

3.3 Change Management

Since the project was experimental for the participants, for the reasons described on the previous subsections, it was assumed chance would be a constant presence throughout the development process.

The changes were expected to come from two main originators. First one would be a selected component not being able to fulfill a requirement and having to be replaced. Another possible source would be the project owner, who would possibly to want to introduce a few changes or corrections.

Since the human and time resources were known to be limited to what the internship context allowed, changes went through a proper process to make sure how costly they would be. Their acceptance depended on the result of said evaluation. If they would enlarge or change significantly the scope of the project, they were not accepted and instead listed as future work. On the other hand, if they did not add significant overhead and were within the scope of the project, they were prioritized by the project owner and added to the product backlog.

3.4 Risk Management

Risk management, is the process of identification, assessment, and prioritization of potential problems and uncertainty, with the intent of minimizing or avoiding their negative impact.

The employed process, discourages focusing on documentation and extraneous activities that do not produce functionality, however, to reduce unexpected issues and reduce risk, the intern decided to use a very light risk management process.

At the end of each stage, sprint, or *kanban* card a brief analysis was conducted to determine if any new risks had arisen. Any risks found were pinned on a separate *kanban* board, and each went through analysis to determine its probability, impact. Additionally, mitigation plans were created when possible or risks were just accepted when it was unavoidable.

Risks that went through this process, as well as the resulting plan, were documented in Chapter 6.

More formal approaches lose their appeal in a single developer context, since there is no need to communicate the risks and plans to an entire team or different levels of management. In this particular case it is used merely as a personal tracking mechanism and a way to deal with uncertainty in a very dynamic and recent technological ecosystem.

3.5 Development Environment

To potentiate scalability, distributed, replicable, components are preferable whenever possible, creating the necessity to emulate a multiple machine development and deployment environment. The default approach is to use virtualization to simulate the entire environment, however this slows down deployment and rebuilds. The current proposal of components and technologies to be used are all open source and run on Linux, therefore containers become a viable, more attractive solution. Their very low overhead, fast spin-up times and flexibility of deployment and network topology add value to the development. Currently docker containers are very popular and come with a set of tools that make it straightforward and easily automatable to build an application or set of services, pack them into a container, distribute and get running with the proper interconnections. On top of that it supports a layered container representation that integrates a git like version management system, making it efficient and fast to ship changes across a network.

Depending on how this rather recent technology evolves, it might also be possible to use it for the actual deployment of the final product, since containers are notoriously easy to schedule and orchestrate over a physical cluster with the help of project like Docker Swarm[11] or Kubernetes[12]

Fundamental chances to the development environment were made during the second semester, but as they were not part of the initial plan, they are not discussed in this chapter. Details are present in chapters 7 and 8.t

Chapter 4

State of the Art

Understanding the state of the art technologies, products and practices is necessary to navigate a new field. To help understand technical decisions and guide readers who are not familiar with this particular subject, this chapter contains an overview of the current state of technology and products.

4.1 Existing Products and Applications

Practice Fusion is the company behind the largest EHR in the United States of American. Founded in 2005, they pioneered an innovative approach to EHRs[13], a free, cloud based product. In terms of business model, the company generates revenue though analytics products, centered around the data available in the system. One of the materializations of this model is Insight[14], a web application that gives users access to low latency healthcare analytical data. Keeping with a growing tendency, Practice Fusion made the public beta available to anyone for free.

The now private service, part of a larger, invite only, platform called Practice Fusion Pharma[15], is only available to their partners. This platform is functionally homologous to the intended application of MATE. It stores the same type of events; drug therapy prescriptions and pathology diagnoses. Finally, it presented a similar business model, based on selling analytics based products to the pharmaceutical industry.

From a market perspective this proves that there is a demand and the business model can be successful, making a platform like MATE, and products built on top of it, relevant and viable.

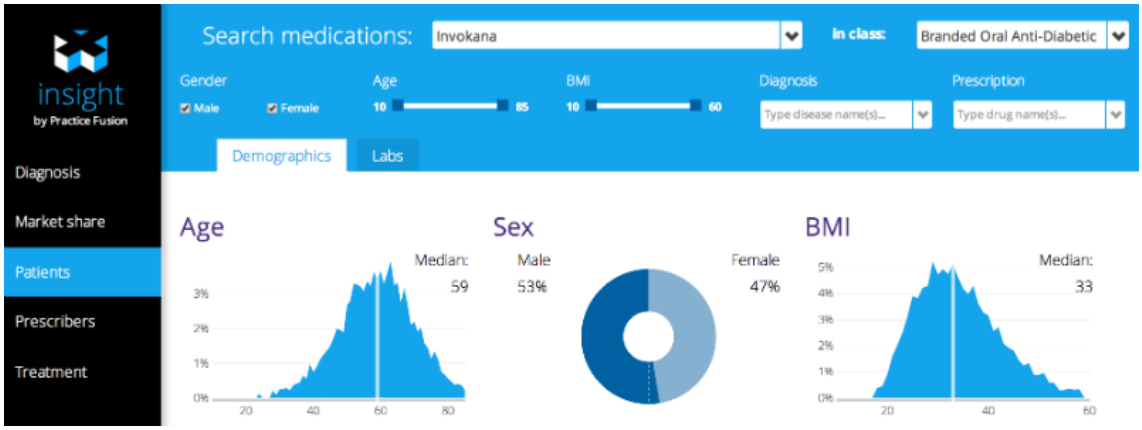


FIGURE 4.1: Screenshot of Practice Fusion Insight (http://www.practicefusion.com).

4.2 Analytical Data Processing

Businesses and organizations have always had the need to perform analytical workloads on their data. In the era of information technology information is being produced at ever growing rates, making its analysis a technical challenge. Analytical workloads require a global view of the data as well as the ability to process it at different granularities. The ability to analyse recent data, as close as possible to real time, has always been desirable and the goal of a lot of research.

Classical DWHs for OLAP store data in a relational database, usually in a star schema, shown in Figure 4.2. Factual or event data is on a large table connected to smaller dimensional tables using foreign keys. Most of the analytical workload is write once, read many, so the typical trade offs discarding granular data in favor of rolled up data and writing in batch every so often. Update regularity usually varies from weekly to daily. This type of solution tends to employ views or cube storage to pre-calculate aggregations at various levels of granularity.

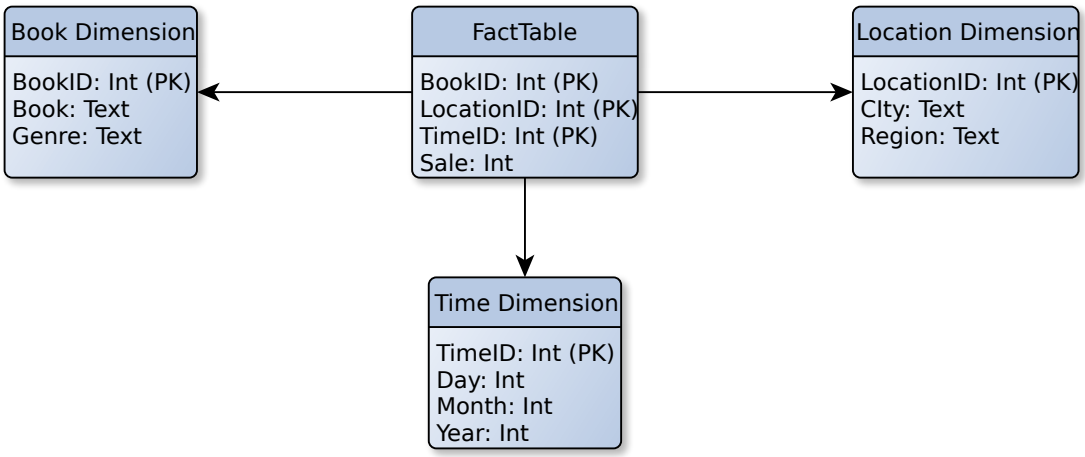


FIGURE 4.2: Sample star schema (http://chrthomsen.github.io/pygrametl)[1].

[2]

To comply with the schema, lading data into the DWH becomes a complex process, commonly called ETL or extract, transform, load and refresh (ETLR). Data is firstly extracted from the original sources, typically online transaction processing (OLTP) databases. After going through a transformation process to comply with the schema, it is inserted into the DWH. Depending on the employed process the DWH might be recreated from scratch every time to avoid consistency and performance issues. As an alternative the ETL step can take over that responsibility and just insert or update data as necessary. In either case the process is very heavy due to the necessity to update or recreate indexes and consistency assertions. Typically, as a result of the inner workings of indexes and views, to get satisfactory load performance it is necessary to drop all indexes, views and cubes before starting the Load or Refresh. Afterwards the will have to be recreated[16]. This process can take a long time, proportional to the data volume and number of dimensions in the star schema, therefore it is obviously not adequate for a real time approach.

The next subsections explore the necessary capabilities to achieve soft real time analytical processing and how they have evolved;

4.2.1 Low Latency

Historically, the first approach to achieve BI on fresh data was decrease the required time for the ETL process. Many techniques were used, the most notable being data base pivoting. Two data warehouse database engines are used in a double buffering, pivoting scheme, while one responds to queries the other one is loaded. Other approaches are higher level of aggregation, faster index types

[2] http://chrthomsen.github.io/pygrametl/

or even optimized operations like the Star Join operation implemented by some databases, optimized to join fact tables with dimensional tables.

Due to technological, financial and conceptual advances, it is now possible, with some trade offs, to process, aggregate and visualize data in soft real time. Most implementations rely on multi staged approaches. A highly indexed and pre-processed store for fast querying of historic data, and a fast layer with small quantities of raw data to cover the time between updates[17].

4.2.2 Fast Aggregation

Due to historically slow access to storage, aggregating large quantities of data was impractically slow. To achieve OLAP capabilities, data had to be pre-aggregated in dimensional structures called cubes, firstly as views in regular relational database management system (RDBMS) and later in specialized storage engines. However, updating existing cubes is too slow and computationally heavy to achieve real time ingestion and querying. To reduce the time between ingestion and availability JIT aggregation strategies are employed leveraging parallel computing tools like Apache Hadoop[18]. A complementary strategy is using faster storage, on the limit keeping the entire data set in main memory, distributed and replicated across a cluster.

Naive implementations will suffer a financial penalty resulting from the higher prices of main memory and the fact that read operations become CPU bound. However, with proper caching and doing pre-computation, when it doesn't cause increased latency, makes it a very powerful technique. Some success examples are InfluxDB[19] and Druid[20].

4.2.3 General Processing

For some applications like data enrichment or machine learning, the traditional data aggregation and indexing processing it not enough. Achieving general processing while not increasing latency / decreasing performance was not trivial.

The biggest impact in this area, kick starting the age of big data, was the usage of mapReduce, namely Apache Hadoop[18] to leverage a cluster of commodity nodes and parallelize not only the ETL but on the extreme, even the data storage itself[21].

This approach is called batch processing, and makes it possible to load and achieve general processing over large batches of data by leveraging large quantities of computational resources. The availability of cloud based computational resources made it even more attractive, however, batch loading, even though powerful, still has high latencies. Processing and loading times went from days to hours, but still far away from what is considered real time due to the overhead of loading data from storage, distributing work and coordination between nodes when necessary.

4.3 Approaches

This section described the technical approaches used to achieve soft real time analytical workloads over historical data and events streams.

4.3.1 Stream Processing

To achieve near real time performance, a different approach emerged, while still leveraging distributed computing, stream processors deal with a continuous stream of data, one event at a time, routing it through a path of computational tasks. Since events are processed individually, or in small batches, the latency is very low, and high input volumes can be dealt with using clustered resources and alternative storage approaches. An example would be keeping data in main memory, more on the subsection dedicated to storage.

While stream processors like Apache Storm[22] achieve very low latencies, sometimes sub second, they are less powerful in the sense that they do not have access to all the data as is the case with batch processing. Usually this can be compensated for by changing the paradigm around data, adopting an immutable approach, accepting that a record should never be altered and is the single source of truth at that point in time. Instead, new information is appended and everything treated as temporal series. When global consistency is required, it can be eventually achieved by using batch processing as a slower, more accurate complementary mechanism.

4.3.2 Storage

As described on the previous sections, storage was for a long time the bottleneck for BI and the possibility of soft real time processing and querying. Under certain specific conditions there were always well documented techniques to deal with this. The following complementary approaches have proven their value;

Columnar and Time Series

Column oriented data stores store data tables as contiguous sections of column data as opposed to rows. Most analytical queries involve operations over very few columns, therefore the locality principle can be explored to get faster loads. Apache HBase[21] is a well known widely used example.

Other data stores leverage this same principle, but segment data chronologically, this is especially useful for dealing with data series, the prime examples are InfluxDB[19] or Amazon RedShift. Commonly these databases deal with growing volumes of data by aging it. For example, older data might get aggregated, saving storage and processing at the cost of granularity loss.

In Memory

In accordance with the von Neumann architecture[23], memory has always been regarded as scarce and expensive. With hardware steadily decreasing in price and being easily available on demand through cloud computing, this is no longer the

case. At this point in time storing an entire database in memory is in many cases a viable option. All the previously mentioned approaches are still valid and benefit greatly from being combined with in memory storage. At first, it might seem like storing data on a volatile medium might not be such a good idea, but the risk of loss can be easily mitigated by using distributed data stores that replicate data segments across many nodes creating a resilient platform. Plus, snapshots of the data can be periodically persisted to disk. A great example of a simple in memory distributed data store is Redis[24].

Sharding

In order to execute analytical workloads in soft real time over growing volumes of data, storage operations need to have controlled complexity, ideally constant. Since this is not always possible, an alternative is using a storage that can scale horizontally and add more hardware as data grows, keeping complexity bounded. Sharding data over a cluster is a very effective way of achieving this goal. In lots of applications, as is the case of the /glsmate platform, data can be treated as a multi dimensional time series and it makes sense to segment it over the temporal dimension. This way, operations can be distributed and each node will always be responsible for a fixed size portion of the data, delivering results in consistent times.

4.3.3 Reference Architecture

With knowledge of the previously explored techniques, soft real time can be achieved by using different complementary technologies together, with the caveat of added complexity.

A widely accepted approach to soft real time is the Lambda Architecture[25] illustrated in Figure 4.3, by Nathan Marz, author of Apache Storm. It can be described as a set of architectural principles that allow generic, scalable, fault tolerant low latency data processing, as long as the data has a temporal dimension.

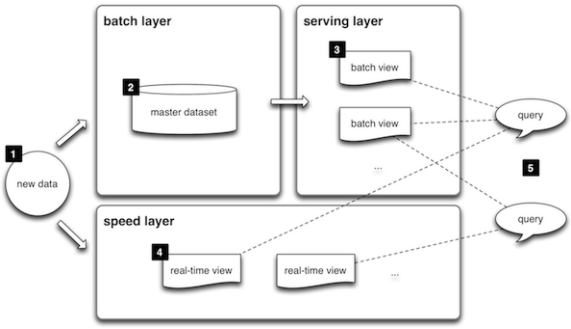


FIGURE 4.3: Sample, two path, lambda architecture. (http://lambda-architecture.net/)

A natural evolution of the database pivoting method previously described, data is batch processed and ingested in regular intervals, and the gap is filled by a stream processing approach. Queries are then mapped and distributed over the

two segments by a serving layer.

The batch segment brings the benefits of batch loading and high availability of a write once read many data store. At the same time, the stream segment compensates for the batch loading delay by keeping the recent data in memory and highly available. This also adds fault tolerance to the system since errors during stream loading will later be corrected by the batch loading. Even in case of total loss at the serving layer, data can be recreated from the raw storage at the batch layer.

This architecture and derivatives are the current state of the art on what concerns real time analytical workloads and is used by small and multi-billion dollar companies alike. Some notable examples are twitter, with Apache Storm[22] and Heron, and NetFlix with SURO[26]

As a last note, it is important to understand how overloaded[4] the term real time is. It can mean anything from real time ingestion, and real time processing to real time visualization, and chronologically range from sub-second to sub-hour.

4.4 Technologies

The following subsections briefly introduce the technologies found and studied during research. Due to their open source nature and immaturity, these are actively changing technologies, for more detailed information, consult the official documentation.

4.4.1 Apache Kafka

Apache Kafka[27] is a distributed high performance publish-subscribed messaging system, implemented as a distributed commit log. One of its core advantages is performance, able to handle megabytes of reads and writes ever second from thousands of clients[27]. Kafka ensures durability by using persistent storage and sharding across nodes[27].

4.4.2 Apache HBase

Apache HBase [21] is a distributed, columnar, scalable data store aimed at big data based on Google's BigTable. Capable of managing billions of rows times millions of columns[21] by leveraging the Hadoop Distributed File System (HDFS), HBase is well suited for raw storage and for batch loading while being very popular, and well supported.

4.4.3 Apache Cassandra

Apache Cassandra[28] is a distributed, high performance, highly scalable database designed to handle large amounts of data on a cluster of commodity nodes. Cassandra was developed by Facebook and later open sourced as an apache top level project. Boasting linear scalability[29], Cassandra provides high availability, with

no single point of failure, as well as a hybrid key-value and column oriented storage format. Made with scalability and performance in mind, you can add new machines without any downtime or interruption. Currently it has over 1500 well known companies and organizations relying on it, for example, CERN, Comcast, eBay, Instagram, Netflix and Reddit. Some of the known production deployments have over 75000 nodes storing over 10PB of data.

4.4.4 Apache Spark

The Apache Spark[30] project calls itself a fast and general distributed platform for big data processing. It can serve as the supporting layer for such diverse tasks as visualization, machine learning or even stream processing.

Since Spark operates on RDDs (Resilient Distributed Datasets), a logical collection of data partitioned across machines[31], it would be well described a micro-batch processors, however, it has been evolving into a more general distributed computing platform.

4.4.5 Apache Storm

Apache Storm[22] is a distributed stream processing system, making it easy to process large streams of data with very low latency. It is used for a variety of purposes such as machine learning, analytics, ETL, etc. Widely considered as the default stream processor, it integrates with most other platforms, namely Druid, Kafka and other stores and compute platforms.

4.4.6 Apache Hadoop

Widely known and used, Apache Hadoop[18] is the default batch processor, map reduce framework and even general distributed computing platform. It boasts support for many applications on top of itself such as Pig, Hive and HBase. Based on Google's MapReduce, even though being slowly replaced by Apache Spark, Hadoop made big data main stream and is still relevant as the batch processor of choice.

4.4.7 InfluxDB

InfluxDB[19] is a time series database built to handle heavy write / read loads, able to serve as the data back-end for large internet of things (IoT) deployments and system monitoring. Capable of managing its own cluster, build in go and completely self contained, influx is a very optimized purpose specific database.

4.4.8 Druid

Druid[20] is a distributed, in memory, data store designed for OLAP on dimensional time series. It boasts low latency data ingestion, fast, sub-second aggregation and well suited for exploratory design.

Druid achieves this by internally employing the same principle behind a lambda architecture, saving the trouble of having to add or implement a custom merging layer, the component responsible for querying both the historic and real time data and merge the results to answer client queries.

4.4.9 Redis

Redis[24] is a popular, in memory, key-value data structure store[24] with typical use cases such as database, message broker, cache and even Publish / Subscribe messenger. Due to its very low level API, functionality and in memory storage it is capable of achieving outstanding performance. Due to supporting replication, with very fast non-blocking first synchronization, auto-reconnection, redis is painless to scale and make resilient.

Chapter 5

Requirements

A more comprehensive and complete system requirements specification document is provided as Annex A, however for ease of understanding, and since some key requirements become architectural drives later in the document, the requirements elicitation and analysis process and the main functional and quality requirements, will be presented. While the SRS in Annex A, describes requirements formally, for development and reporting purposes, requirements will be used in the form of user stories, as part of the Kanban management process.

5.1 Elicitation

At the beginning of the internship, there was no formal definition of the desired system, therefore, going through a requirements elicitation process was inevitable, not only to understand the project but also to properly define its scope. This was especially important, because there were no imposed constraints in terms of approach or used technologies.

Requirements were gathered through two formal meetings with the Project Owner, João Miguel, Chairman of MedicineOne, and a few subsequent informal conversations. During the process the intern tried to clarify the expected operational environment of the system, estimate the load it would be submitted to and manage the Project Owners expectations on what concerned to what could actually be done. Requirements adhered to the specific, measurable, assignable, realistic and time related (SMART) principles, so that they could be used for validation of the final artifacts.

5.2 Functional Requirements

This section contains a list of functional user stories and their respective priorities.

TABLE 5.1: Functional Requirements.

ID	Priority	User Story
SR-F-M1S.01	High	As the MATE back-end system, I want to receive events containing anonymized factual and dimensional information from the MedicineOne servers, relating to drug therapy prescriptions, pathology diagnoses and diagnostic tests requisitions so that I can process them.
SR-F-BE.01	High	As the MATE back-end system, I want to persist all received events so that I can be reliable.
SR-F-BE.02	High	As the client of the MATE back-end system I want to be able to subscribe to events in real time and related metrics so that I can consume them.
SR-F-BE.03	High	As the client of the MATE back-end system I want to be able to get time series of the metrics and events in SR-F-BE.02 so that I can consume them.
SR-F-BE.04	High	As the client of the MATE back-end system I want to be able to get historic data at different levels of granularity, so that I can use it for comparison.
SR-F-BE.05	High	As the client of the MATE back-end system I want to be able to get multidimensional aggregates of the data so that I can consume them.
SR-F-FE-01	Low	As the User I want to be able to authenticate on the MATE front-end using my credentials so that I can use the system according to my permission level.
SR-F-FE-02	Low	As a Operator or Administrator type of User, I want to be able to set the permissions for each user so that I can control their access to information according to type, time intervals, metrics and granularity.
SR-F-FE-03	Low	As the User I want to visualize data in a multitude of formats, line, pie, bar and geographical charts as well as gages so that I can easily assess the information I am being presented.
SR-F-FE-04	Low	As the User I want to overlay real time data over historic data so that I can easily compare.
SR-F-FE-05	Low	As the User I want to have the real time information I am consulting be updated in real time so that I can stay up to date without having to refresh.
SR-F-FE-06	Low	As the User I want to be able to select between the different types of available events so that I can view the one that interests me.
SR-F-FE-07	Low	As the User, for each type of event, I want to be able to select from the different metrics, time series and aggregates available so that I can view the one that interests me.
SR-F-FE-08	Low	As the User I want to be able to see the pathologies, drugs prescriptions and diagnostic tests in a user friendly way so that I don't have to do the translation between different classification systems and identifiers myself.

5.3 Load Estimation

To properly define SMART quality attributes, metrics had to be chosen, and specific values assigned to the requirements so that they can later be verified, for that it was necessary to estimate the workload the system will be submitted to and how it is expected to vary during its life cycle.

From operating information provided by the supervisor at MedicineOne, taken from two of their largest running systems, and assuming that the real volume of data is 4 times greater and will grow 10% each year, as suggested by the Project Owner, the following projections have been made for the next 20 years.

For the estimation the following assumptions were made, events are sent as soon as they happen, no buffering to keep near real time latency and each occupies 2KiloBytes;

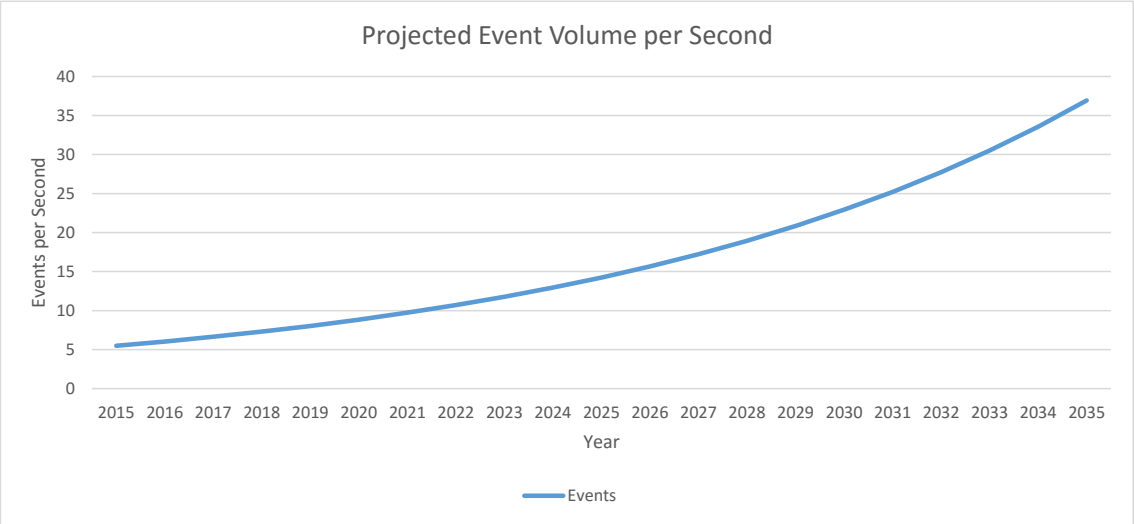


FIGURE 5.1: Event volume estimation, in events / second.

Projected to stay well bellow the thousands of events per second, even after considerable expansion as seen on Figure 5.1, the speed of ingestion should be no problem, leaving a lot of leeway for addition of additional computations at a later date. Also makes it trivial to ensure persistence using a simple synchronous transactional mechanism like acknowledging only after persisting. The low ingestion volume also makes it possibly to implement direct event subscription from the front-end with only minimal temporal aggregation.

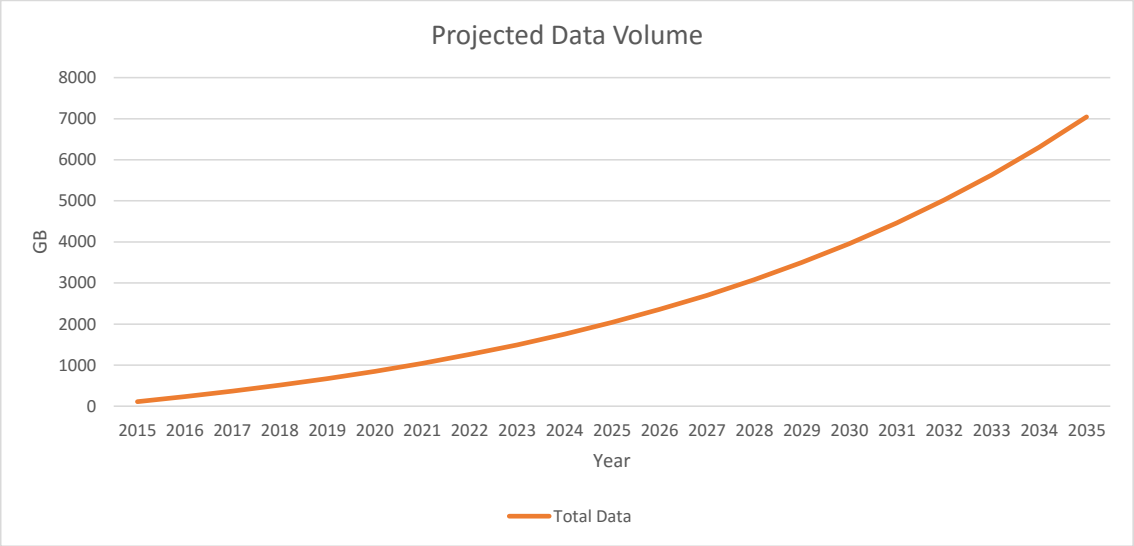


FIGURE 5.2: Storage estimation, in GigaBytes.

Figure 5.2 shows that the storage volume will stay under 10 TeraBytes, in reality it will be probably even less than projected since the 2 kilobyte per event assumption was a slight overestimation. When it comes to raw storage it presents no challenge, it also makes it viable and easy to employ in memory databases for superior performance and near real time on demand aggregation and processing.

Assuming the estimations are good enough, off at most by a small factor (2 or 3 times more), this means that an initial deployment will require no more than a very modest quantity of convenience hardware, and will most likely have a lot of scale out potential, enough to cover the necessities in the expected life cycle of the system (less than 20 years).

5.4 Quality Requirements

From talking to the project owner, and due to the sensitive nature of the data, the system must meet the following quality requirements, varying in priority:

TABLE 5.2: Quality Requirements.

ID	Priority	User Story
SR-PERF-SYS.01	High	The system shall be able to scale up to ingest at least 40 events per second.
SR-PERF-SYS.02	High	The system shall present latencies inferior to 10 seconds between ingestion of an event and it being reflected on the metrics and/or data model.
SR-QOS-SYS.02	Medium	The system shall be horizontally scalable.
SR-QOS-SYS.03	High	The system shall be extensible by means of module addition. New modules shall be able to subscribe to the input even feed and query the existing data.
SR-REL-SYS.01	Medium	The system shall be available to ingest events at least 99% of the time.
SR-REL-SYS.02	Medium	The system shall be available to reply to user queries at least 99% of the time.
SR-REL-SYS.03	High	The system shall implement the necessary redundancy to ensure data persistence in the event of hardware failure.
SR-MAINT-SYS.01	High	The system shall employ general standardized interfaces so that components may be replaced with homologous ones when they exist, for example a standard SQL database.
SR-MAINT-SYS.02	High	Failure or planned maintenance of a distributed component shall not cause system downtime.
SR-SAFE-SYS.02	High	To be safe against human error, the system shall provide an event replay mechanism, leveraging permanent storage.

Chapter 6

Risk

Risk is defined in ISO 31000[1] as the effect of uncertainty on objectives. By definition outside of our control, risks can negatively impact a project, however, just because they are fortuitous in nature, it does not preclude management and mitigation or even complete avoidance of its effects.

" The first step in the risk management process is to acknowledge the reality of risk. Denial is a common tactic that substitutes deliberate ignorance for thoughtful planning. "

Charles Tremper

This section will list all identified and analyzed risks as well as the results of the analysis. Each risk will contain the following properties and respective values:

- Description
- Probability (Low, Medium or High)
- Impact (Low, Medium or High)
- Time-frame
- Consequence
- Plan

For more information about the used risk management method consult Chapter 3.

6.1 Identified Risks

This section lists all identified and analyzed risks. For ease of visualization, table 6.1 depicts the risk exposure matrix.

TABLE 6.1: Risk Exposure Matrix.

Impact \ Probability	Low	Medium	High
High	R01, R03		
Medium	R07	R06	R02, R04, R05
Low			

[1] http://www.iso.org/iso/catalogue_detail?csnumber=43170

R01	**DESCRIPTION:**	Project too complex for the intern.
	PROBABILITY:	Low
	IMPACT:	High
	TIME-FRAME:	1st semester
	CONSEQUENCE:	Failure of the project and internship.
	PLAN:	Research and study of domain during the 1st semester.

R02	**DESCRIPTION:**	Time and effort conflict with other 1st semester courses.
	PROBABILITY:	Hight
	IMPACT:	Medium
	TIME-FRAME:	1st semester (25% of total time).
	CONSEQUENCE:	Considerable delay.
	PLAN:	Shift work to 1st semester, focus on planing.

R03	**DESCRIPTION:**	Legal issues resulting from manipulation of patient data.
	PROBABILITY:	Low
	IMPACT:	High
	TIME-FRAME:	Whole project.
	CONSEQUENCE:	Added workload to comply with norms.
	PLAN:	Anonymizing data at source by removing personal identification information.

R04	**DESCRIPTION:**	Not enough time to develop front-end.
	PROBABILITY:	High
	IMPACT:	Medium
	TIME-FRAME:	2st semester.
	CONSEQUENCE:	Project left at prototype stage at the end of the internship.
	PLAN:	Accept consequences and try to accelerate development by using already existing frameworks and technologies.

R05	**DESCRIPTION:**	Too hard to integrate with the existing, undocumented, unstructured, code base with no regression testing.
	PROBABILITY:	High
	IMPACT:	Medium
	TIME-FRAME:	2st semester.
	CONSEQUENCE:	Delay project and cause regression.
	PLAN:	Develop a mock up event generator, request help from someone familiar with the code base.

R06	**DESCRIPTION:**	Finding out that some of the selected technologies or frameworks are inadequate.
	PROBABILITY:	Medium
	IMPACT:	Medium
	TIME-FRAME:	2st semester.
	CONSEQUENCE:	Having to replace it or in the worst case, develop a custom one.
	PLAN:	Adopting a JIT agile process to delay decision, and be able to rely on incremental development to reduce the cost of change.

R07	**DESCRIPTION:**	Basal open source components losing community support and becoming obsolete and or unsupported.
	PROBABILITY:	Low
	IMPACT:	Medium
	TIME-FRAME:	Life of the project.
	CONSEQUENCE:	Forcing the platform to stagnant resulting in slowed down development and or extra costs.
	PLAN:	Employing encapsulation and separation of concerns to make functional components easy to replace.

6.2 Materialized Risks

Follows a list of all risks that materialized and any notes considered relevant.

- **R02:** Risk 01 materialized, however, it had been predicted and the plan reduced impact to acceptable levels, no major consequences were felt.

- **R04:** When it became evident that effort would have to be directed to operational concerns, development of a front-end was dropped since it was low priority.
- **R05:** The data model as well as application logic of MedicineOne 8 was too complex to integrate by a single person. Since the various teams were busy, an event generation API was developed to be used by the system when integration takes place.
- **R06:** Some technologies proved to be inadequate and it was necessary to update some of them at an advanced stage. To accommodate, the delivery of the project was delayed.

Chapter 7

Architecture

" The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. "

Bass L.; Clements P.; Kazman R.[1]

A software architecture is one of the first approaches to solving a problem and or designing a system. Typically, represented as a diagram with boxes and lines, boxes representing elements of the system, and lines their interaction[32]. Software engineering still retains an artistic component, once the requirements are defined, the process of planning the architecture to satisfy them is not as systematic as what is expected of other engineering disciplines.

Considering that MATE has non trivial quality requirements and is the first project of the type both for MedicineOne and the intern, design and creating an adequate architecture was given a lot of attention. The end goal was making sure that no assumptions were wrong and that no mistakes were made at the very beginning that would latter threaten to make the project fail for lack of technical viability. Once defined, the requirements were distilled to yield a set of architectural drives later used to iteratively validate and refine architectural drafts.

This section exposes the drivers as well as the resulting architectural views.

7.1 Drivers

The functionality expected from the system is not exceptional or innovative, it can be generally described as a soft real time analytical processing platform, however, the quality requirements, like the need for near real time ingestion and availability, create specific needs that end up becoming the drivers for the system.

The following list contains the quality requirements that, for their reaching implications, were the deciding factors behind the proposed design.

[1]Software Architecture in Practice 2nd Edition Reading, MA: Addison-Wesley, 2003.

TABLE 7.1: Architecturally significant requirements.

ID	User Story
SR-PERF-SYS.02	The system shall present latencies inferior to 10 seconds between ingestion of an event and it being reflected on the metrics and/or data model.
SR-QOS-SYS.02	The system shall be horizontally scalable.
SR-QOS-SYS.03	The system shall be extensible by means of module addition. New modules shall be able to subscribe to the input even feed and query the existing data.
SR-REL-SYS.03	The system shall implement the necessary redundancy to ensure data persistence in the event of hardware failure.
SR-MAINT-SYS.01	The system shall employ general standardized interfaces so that components may be replaced with homologous ones when they exist, for example a standard SQL database.
SR-SAFE-SYS.02	To be safe against human error, the system shall provide an event replay mechanism, leveraging permanent storage.

From the list of drivers in table 7.1 resulted the following list of architectural drivers;

TABLE 7.2: Architectural drivers.

ID	Driver
AD.01	Low latency ingestion and processing, specifically, under 10 seconds.
AD.02	Horizontally scalability.
AD.03	Modular to allow for extension and composition.
AD.04	Highly available and redundant with tolerance to partial infrastructure and software failures.

7.2 Style

Generally, the MATE platform was designed to follow a service oriented architecture (SOA). According to Erl, Thomas[33], a SOA has the following objectives;

- Increased intrinsic interoperability;
- Increased federation;
- Increased vendor diversification options;
- Increased business and technology alignment;
- Increased return of investment (ROI);
- Increased organizational agility;

- Reduced information technology (IT) burden;

Since they overlap with the drivers for the system, this style, SOA, was selected. Namely, it allows the usage of existing open sources solutions and their composition and extension when necessary. As advised by Erl, Thomas[33] the architecture and service design was based around the following core principles;

- Standardized service contract;
- Service loose coupling;
- Service abstraction;
- Service reusability;
- Service autonomy;
- Service statelessness;
- Service discoverability;
- Service composability;

Following these guidelines, the applications developed on top of the MATE platform in the future, will themselves adhere to this style and be services. They will focus on implementing business logic and using the available support services to obtain, process and store data.

7.3 Views

The result of the architectural design was a set of diagrams and accompanying text describing how the system should be built, responsibilities distributed, functionality segmented and how they satisfy the drivers.

The diagrams and notation were based on Simon Brown's C4 model[34] for its elegance and objective approach, conveying only the important points. Even though the name C4 comes from the fact that the book describes four levels of detail, Context, Containers, Components and Classes, the author goes on to argument that in most cases a Class diagram is too close to implementation to be defined at this stage, since it would incur in considerable effort and yield very little return, making iterative validation too heavy to undertake.

In the particular case of this project, since most of the system will be made using open source components and frameworks, the intern decided that only the first two levels of detail, Context and Containers, would be useful.

7.3.1 Contextual View

The contextual view gives an overview of the system and surroundings as well as external interactions or integrations. Since MATE will consume events from an existing system, the context diagram was especially important when making sure the existing context was well understood and that the new system would be compatible with it.

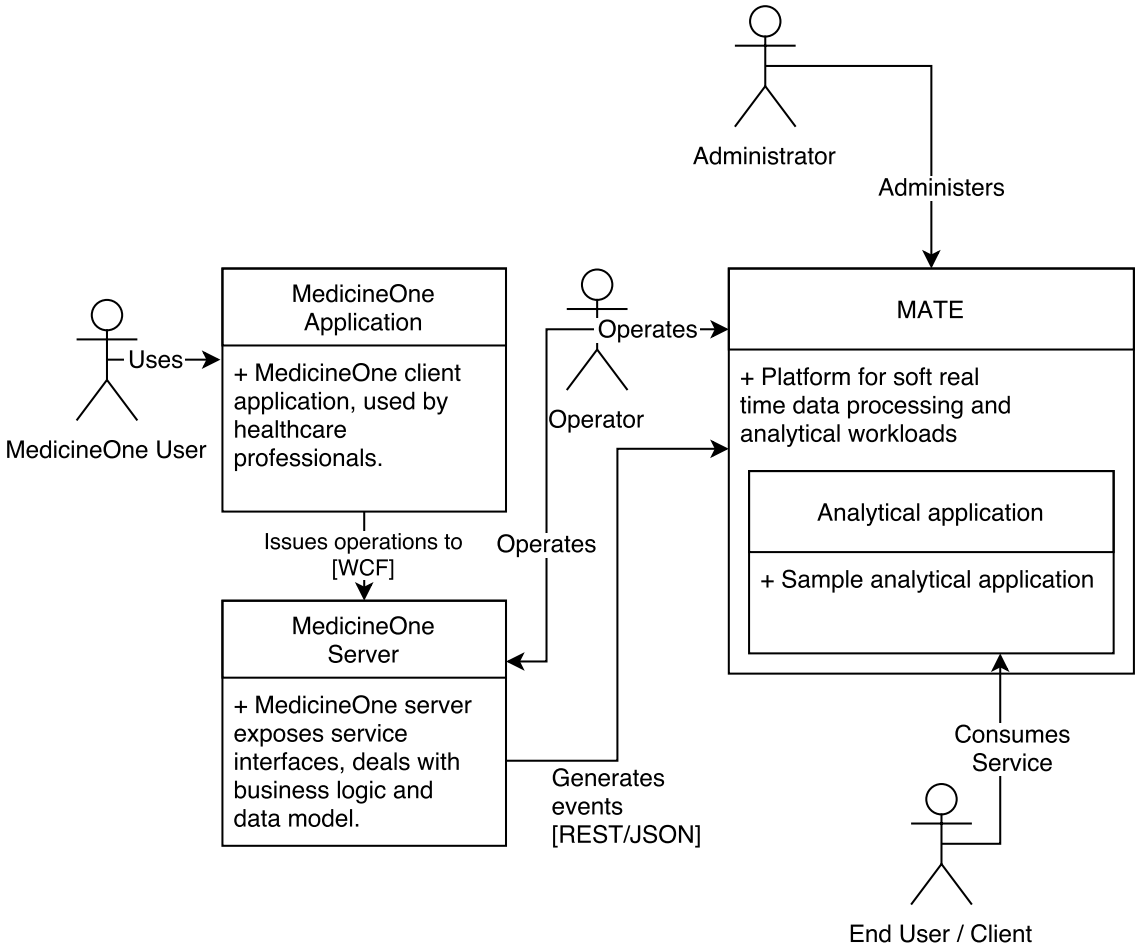


FIGURE 7.1: Contextual view of the system.

Actors:

- **MedicineOne User:** Healthcare professional using the CMS.
- **Operator:** The Software operator configures and manages the system.
- **Administrator:** The MATE administrator manages the platform, data and the applications running on it.
- **Client:** The user consumes the services.

Systems:

- **MedicineOne 8:** CMS composed by the Client and Server application.
- **MATE:** Soft real time event processing and storage platform.

Sample Flow

MedicineOne User uses the MedicineOne application, which acts as a client to the MedicineOne server, when relevant, the server generates and sends an event to MATE which persists, analysis, and indexes it and makes it available to the Client in soft real time.

7.3.2 Container View

MATE will be composed mostly of existing open source platforms, each ensuring a specific function in the soft real time processing stack. This view will be the most useful for the developer as it presents a clear road map of what needs to be deployed, connected and configured, plus it clarifies the responsibility distribution.

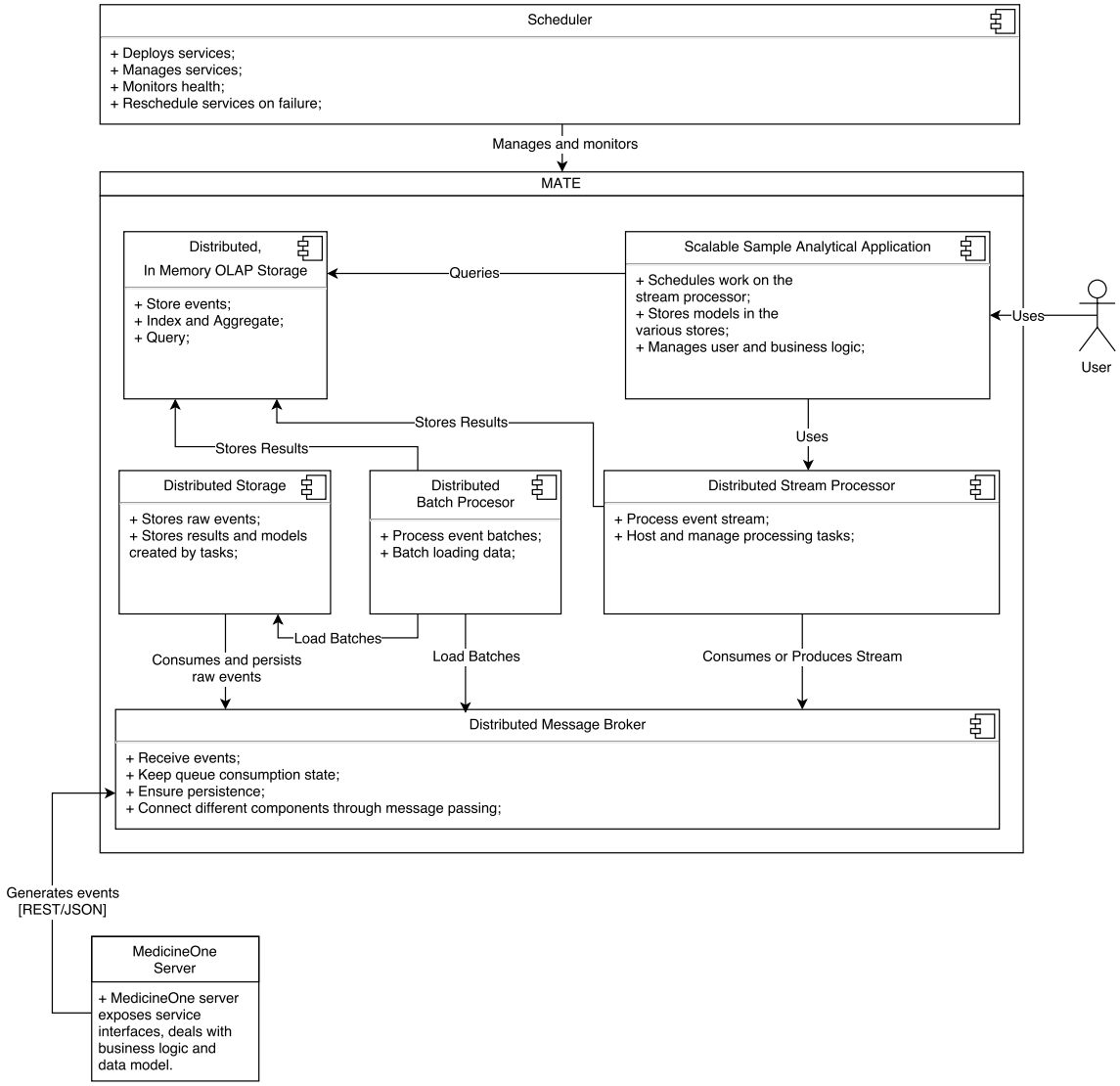


FIGURE 7.2: Container view, showing the main components.

Containers:

- **Distributed OLAP storage:** In memory, distributed OLAP storage with stream and batch ingestion capabilities for dimensional events.
- **Distributed storage:** Distributed, high performance storage.
- **Distributed stream processor:** Processes streams of events with low latency.
- **Distributed batch processor:** Batch processing of large amounts of data for applications that require a global vision of the data.
- **Distributed message broker:** Message broker for both external and internal messaging.
- **Scheduler:** Responsible for deploying all the services, monitoring their health and restart them if they fail.
- **Scalable scalable analytical application:** A sample analytical application implemented on top of the MATE platform.

Sample Flow

MedicineOne generates and sends an event, the message broker immediately persists it, locally and on the distributed storage, ensuring reliability and resilience. The stream processor then consumes events from the broker and passes them through the defined processing tasks, sometimes called topologies. It will for example calculate event metrics, like count by type and other dimensions, and when relevant notify applications through the message broker and push events and notifications to the users of a sample analytical application. The events themselves, and the time series resulting from the computed metrics will then be feed into the OLAP data storage, which will ensure optimal performance by managing the received data, separating recent data, from historic data. The batch processor can also be used to consume events from the storage or message broker and generate more complex metrics that require a global view of the data.

7.4 Validation and Traceability

For traceability purposes this section depicts the relationships between requirements, architectural drivers and the design decisions and mechanisms employed to fulfill them.

The properties given by the architectural style, in conjunction with the distributed nature of the components / services theoretically ensure that most of the objectives are achieved. However, some issues, mainly performance related, could arise from unforeseen emergent properties. Since the platform can be used uncountable types of applications, it is very hard to prove that all properties will apply to all of them. For the purpose of this project it was assumed that most analytical applications will mostly perform a linear usage of the services avoiding the risk of deadlocks between them and reducing the possibility of resource contention.

When it comes to the subject of scalability, the system is scalable as a result of being a composition of components that have been proven to be scalable by their respective developers and independent benchmarks. For this property to hold true for applications developed on MATE, the developers must ensure that they are themselves horizontally scalable, either through statelessness or workload distribution.

Table 7.3 is a traceability matrix that relates requirements with drivers and finally with the measures taken to ensure them.

TABLE 7.3: Traceability Matrix.

Driver	**Requirements**	**Measures**
AD.01	SR-PERF-SYS.02	Usage of distributed, scalable services and in memory processing and storage. Examples would be the stream processor and the in memory OLAP storage.
AD.02	SR-QOS-SYS.02	All services are provided by distributed, horizontally scalable components.
AD.03	SR-QOS-SYS.03, SR-MAINT-SYS.01	Service oriented architecture with well defined interfaces to allow interoperability and reduce coupling. Usage of a message broker and messaging passing communication pattern.
AD.04	SR-REL-SYS.03, SR-SAFE-SYS.02	Usage of distributed, redundant components. Multiple points of persistence. Usage of a scheduler to deploy, manage and monitor the services and ensure availability.

Chapter 8

Implementation

In this chapter the reader will find a description of the implementation process, and its different steps, from technology selection to deployment and operational concerns. As a complement, there are also narrative descriptions of the process and important events that came to pass.

8.1 Overview

As depicted in the chronogram in Figure 3.5 in chapter 3 the implementation started with an attempt to deploy and configure the main components, namely, Apache Kafka[27], Apache Storm[22], Apache HDFS[35] and Druid[20]. It became evident that the automation and management of the system has been greatly underestimated and that it would be a central concern. Just using Docker[36] containers to simulate a clustered environment would not be enough since it lacked important features such as networking across different machines, coordination as well as service discovery.

After some research and testing, Apache Mesos[37], an open source cluster manager, was selected for the task. The deployment was automated using Ansible[38], a software for declarative configuration, management and deployment. Some platforms like HDFS and Storm had already been ported to run on a Mesos cluster and could be deployed directly. By adding Mesosphere Marathon[39], a container scheduler, on top of Mesos, it became possible to run the remaining platforms by containerizing them. To provide a service discovery system, Mesos-DNS, a Mesosphere open source project was used, making it possible for services to find each other through name (domain name system (DNS)).

After noticing performance issues and some compatibility failures with Mesos, Apache HDFS was replaced with Apache Cassandra, a distributed not only SQL (NoSQL) data store with homologous capabilities.

Once Kafka, the message broker, had been deployed attention was shifted to the event production component for the CMS, MedicineOne 8, in other words, integration.

MedicineOne 8 was developed organically, to fit the emerging needs of the clients, using a feature oriented development process. Architecturally the system is divided in modules, web services, developed by different teams, as a consequence there is a lack of centralized technical documentation and data model.

This coupled with the lack of an application level logging layer, since a lot of it is done at database level using triggers, made full integration too expensive to be undertaken by the intern alone. As implementing such a mechanism required inside knowledge of each different module, cooperation from the different teams would be necessary to achieve full integration. Since this was not possible due to scheduling and prioritization matters, it was decided that the best course of action would be to implement a module in MedicineOne 8 to provide an event logging API, to be integrated in the business process at a later date. This module is responsible for receiving event data, complete it when necessary and send it to the MATE platform.

At the same time some testing tools were developed to feed the previously mentioned module with data from test databases.

After installing Storm, business logic was implemented as Apache Storm topologies, or processing tasks. Events are taken from the message broker, persisted to the distributed data storage, and then go thorough an ETL pipeline that prepares them for ingestion and stores them on the message broker, to be loaded by the OLAP data store.

Some difficulties were felt at this stage making sure event processing was transactional due to the fact that the Mesos ports of Storm and Kafka were not on the latest version. As to strengthen the decision to use Apache Mesos, a month after, Mesosphere open sourced their Data Center Operating System[3] (DC/OS), a higher level cluster management platform implemented over Apache Mesos, it was decided that updating would create the desired results. Due to the added workload the project deadline was delayed and the machine provisioning scripts updated to deploy DC/OS on the infrastructure.

This update greatly improved the deployment process as well as management with its useful graphical management interface and supporting services such as log management. Installing some tools, such as Kafka and Cassandra became as easy as clicking an icon or running a command. Storm and Druid has to be manually containerized and deployed with Marathon. Finally, it was necessary to update the topologies to run on Apache Storm 1.0.1.

8.2 Technology Evaluation and Selection

For some described functional modules in the architecture, there were multiple viable platforms. This sections lists them according to function and explains the process that lead to the final choices. In general the selection criteria were;

- Adequacy to the purpose;
- Compatibility;
- Performance;
- Fulfilling the low latency requirements necessary for soft real time;
- Ease of deployment;

8.2.1 Infrastructure

In the beginning there was uncertainty whether the platform should be deployed on a public cloud, using the available platforms, or on premise. In the end it was decided to develop for on premise installation, and with the decision to use Apache Mesos and DC/OS, explained next, the flexibility to deploy on cloud or on premise alike was attained to some extent.

The factors that weighed the most on this decision were financial and avoiding lock in to a particular vendor. At the time of the study, the smallest possible Storm cluster on Azure alone would cost an estimate of $892.80 / month. Also, using any of the proprietary services would lock the platform to a particular provider and possibly cause legal issues due to the nature of the data being stored.
Another factor that favored on premise deployment was the flexibility of being able to license the platform to clients for private, internal usage. The management costs of running the infrastructure were considered acceptable since MedicineOne has their own private cloud infrastructure and deploys all of its services on their own infrastructure or rented dedicated servers.

8.2.2 Distributed Message Broker

When looking for a distributed, redundant, persistent message brokering platform, the solutions considered were;

- Apache Kafka[27];
- RabbitMQ[40];

While RabbitMQ was a more mature solutions, it is not distributed by design as is the case with Kafka, which was built around that principle.

Moreover, Apache Kafka, presented higher performance, was easier to deploy and integrated both with Apache Mesos and Mesosphere DC/OS[3].

8.2.3 Distributed Stream Processor

For stream processing, at the time of research there were essentially 2 mature, industry tested options;

- Apache Storm[22];
- Apache Spark[30];

Storm was designed and used at twitter for stream processing, on the other hand, Spark is a general distributed computation platform with some available stream processing features. The biggest difference being Spark is more oriented to batch, or small batch processing. As this was a latency sensitive application, Storm, which processes events individually, yielding lower times, was selected. The simpler, easier to understand pipe and filter model used by Storm and the ability to simulate the cluster environment straight from IDE for fast testing weighed heavily on the decision.

8.2.4 Distributed Storage

The distributed storage was initially meant to be provided by Apache HDFS due to its compatibility with the Hadoop ecosystem. However due to compatibility issues with the cluster manager and the fact that it was not trivial to deploy on the small development cluster, it was replaced with Apache Cassandra. This was an easy choice since Cassandra is battle tested and widely used by the industry for its simplicity, performance and linear scalability[29].

8.2.5 Distributed, in Memory, OLAP Storage

Druid[20] was developed and open sourced by Metamarkets specifically for the purpose of high volume, soft real time OLAP workloads. Being a distributed system, capable of using Apache Cassandra for deep storage, and stream and batch ingestion, it perfectly fit the role like no other platform. The distributed architecture with in memory stream ingestion for recent data and post processing or batch ingestion of historic data follows the principles of a lambda architecture, making it adequate for soft real time applications. Druid also includes its own broker component, responsible for mapping queries over the two data layers and reducing the results back to the client.

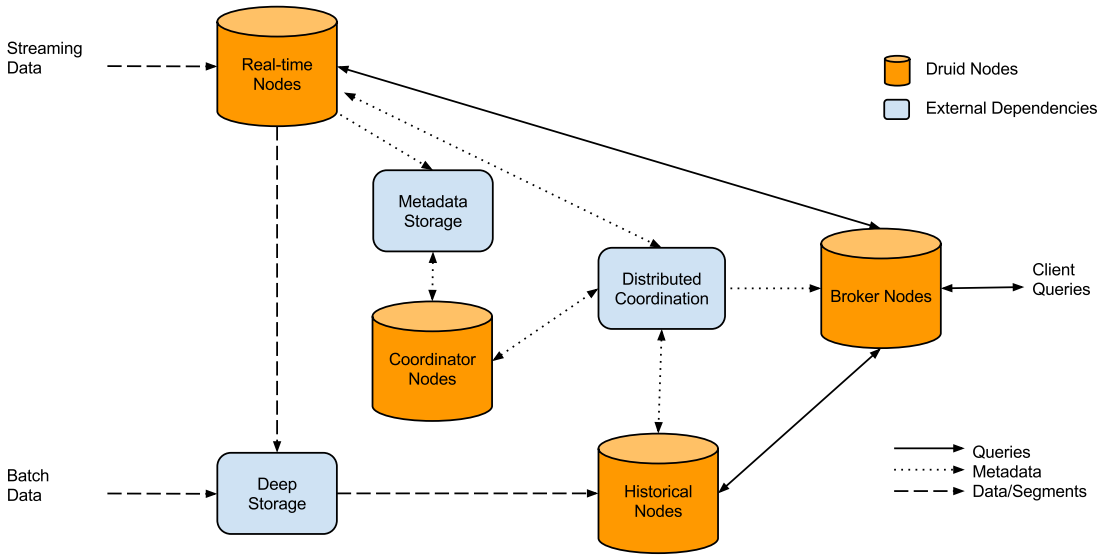


FIGURE 8.1: Druid Architecture (http://druid.io).

InfluxDB[19] was found during research and mentioned, but it is mean to be used on time series with a single dimension, for example, a temperate over time measured by a network of sensors.

8.2.6 Cluster Manager

For managing and monitoring the physical infrastructure and services, there were two viable options;

- Kubernetes[12];

- Apache Mesos[37];

In the end Apache Mesos was selected for its ability to run not only containers but also popular frameworks / platforms, some of which were relevant to the project, such as Spark, Storm, Cassandra and Kafka. This decision proved to be right when Mesosphere DC/OS[3], which runs on top of Mesos, was release a month later, bringing considerable benefits.

Apache Mesos and DC/OS where designed to abstract the infrastructure and make applications highly available, the architectural overview in figure 8.1 is presented for clarification.

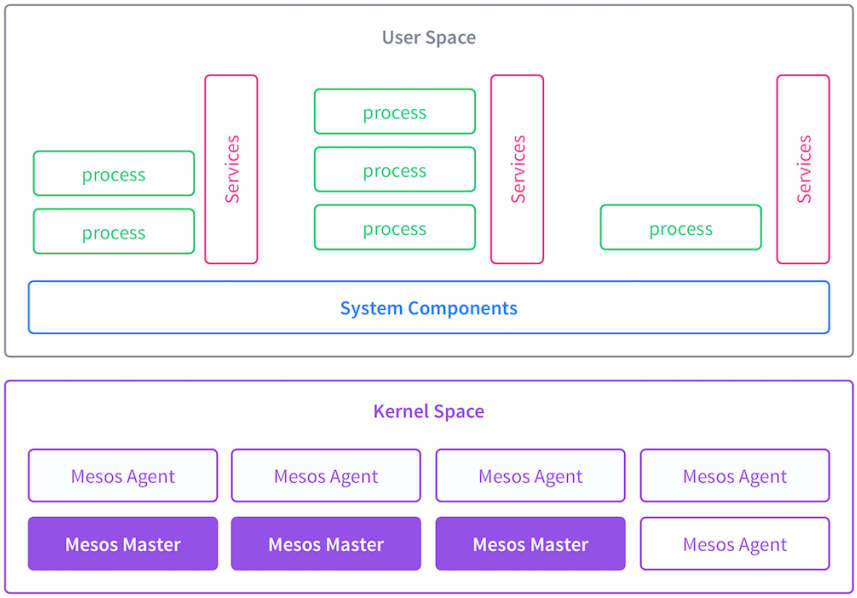


FIGURE 8.2: DC/OS architectural overview (https://dcos.io/).

Some interesting properties of Mesos and DC/OS[3] are;

- No single point of failure, redundant, active / passive master nodes;
- The ability to automatically move and reschedule tasks on node failure;
- Unified infrastructure management;
- Production proven, used by Twitter and Apple to name a few;
- Scalable and elastic, possible to add and remove nodes without interruptions;
- Getting a lot of attention and investment from large companies like Microsoft;
- Growing numbers of packages / frameworks;
- Enterprise version with paid support;

8.2.7 Container Scheduler

The following options were found during research and were compatible with Docker containers;

- Docker Swarm[36];
- Apache Aurora[41];

- Mesosphere Marathon[39];

Swarm proved early on that it was not adequate due to the lack of discovery and cluster wide networking mechanisms. Once it was decided to use Apache Mesos and Mesosphere DC/OS[3] Marathon was selected for already being bundled and deeply integrated. It is important to note that Marathon schedules containers over the infrastructure and takes care of resource reservation and assignment, allowing control over resource distribution and automatic port attribution and mapping.

8.2.8 Service Discovery

Since Mesosphere DC/OS[3] has both proxy and name(DNS) based service discovery, there was no need to choose, both can be used as required.

8.3 Operations

After the first development and deployment tasks it became evident that the real complexity of the system was not in the functionality itself but in the management of a complex system composed by a variety of distributed components. All of them needed to be deployed, managed, scaled, monitored and restarted in case of failure. After realizing that automating the provisioning and deployment with Ansible[38] the intern decided to slightly change the focus of the internship to operation concerns and use a cluster manager. The intent was achieving the following goals;

- Avoid static partitioning of the physical cluster, since dynamic scheduling is more efficient;
- Simplify deployment of distributed components;
- Provide basic low level services needed by multiple components, such as logging and state coordination;
- Service discovery;
- Reschedule services on partial infrastructure failure;
- Create a high availability environment;
- Streamline scale out process;
- Abstract physical infrastructure;
- Create a replicable deployment platform;

This addition is consistent with the architectural drivers of the project, making the operational layer friendly to SOA principles. Also, since it covers the functions of scheduling and monitoring it improves system maintainability and availability.
As described in the section Overview, the initial choice was Apache Mesos, however, Marathon DC/OS[3] 1.7 was released latter and adopted, adding higher level capabilities, a consistent command line interface and a graphical user interface.
DC/OS also made cluster provisioning a lot easier since it has its own configuration and installation tools.

From a deployment and operational point of view, the resulting system looks as depicted in figure 8.3.

FIGURE 8.3: Deployment diagram.

Apache Mesos has two types of nodes, masters, responsible for managing applications and slaves (also known as agents) where the actual applications are scheduled. To avoid having a single point of failure, the master nodes can be replicated with shared state being kept by a Zookeeper[42] ensemble. As described by its developers, "ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services"[42]. This way, even when a master fails, the remaining ones, as long as there enough to get the configured quorum, can elect a new active master and avoid failure or loss of availability. To clarify consult the overview of the architecture in figure 8.4.

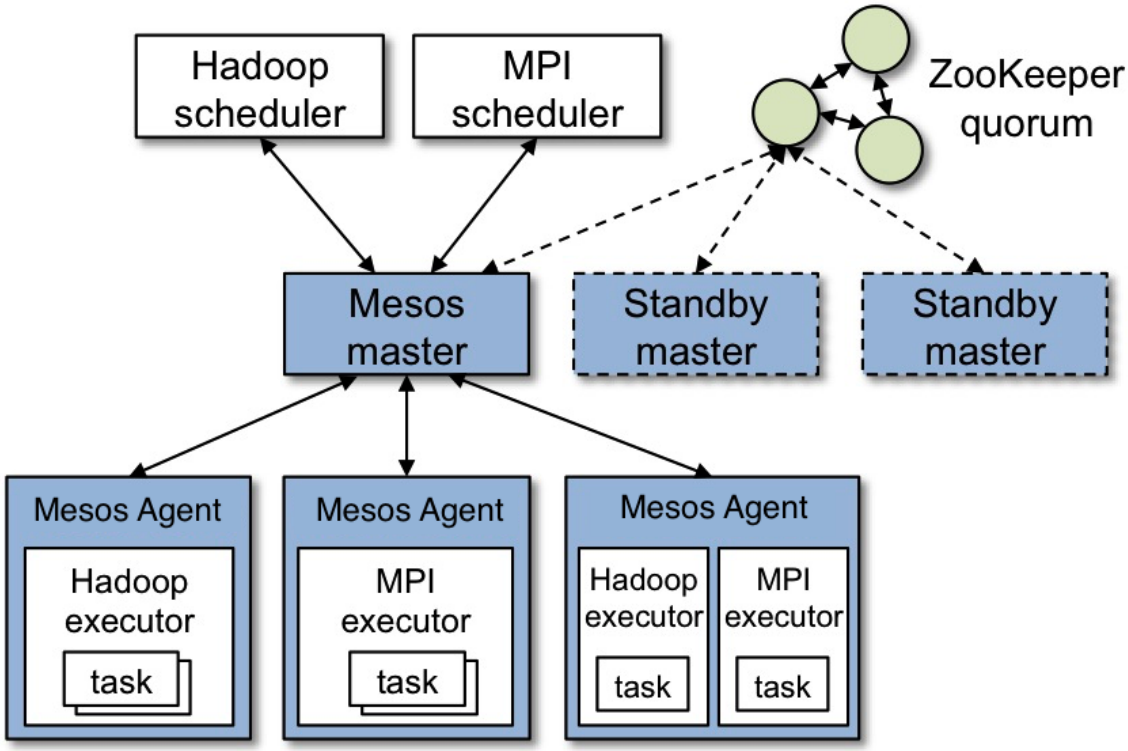


FIGURE 8.4: Apache Mesos architectural overview (Apache Mesos documentation).

8.3.1 Provisioning

Initially provisioning of Apache Mesos nodes was done using Ansible[38]. With this tool, a declarative description of the wanted for a set of machines is written, referred to as a play-book. When this play-book is executed, actions are executed over secure socket shell (SSH) on the target machines to get them to the desired state. This includes such actions as installing packages, creating configuration files and altering system settings. Below, is an example play-book that provisions an Apache httpd machine.

LISTING 8.1: Sample play-book from http://docs.ansible.com/

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
```

```
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

However, with the update to DC/OS provisioning can be done with the custom installer. For this purpose an extra node, called bootstrap node is necessary. This greatly simplified the provisioning process, Ansible is still used to get the nodes into the desired state when it comes to authentication, disk partitioning, etc. But the actual installation and configuration only requires creating a few configuration files and running a few commands on the bootstrap machine. The main configuration file used for the development deployment is listed below;

LISTING 8.2: Main DC/OS installation configuration file.

```
---
agent_list:
- 10.0.249.23
- 10.0.249.24
- 10.0.249.25
# Use this bootstrap_url value unless you
# have moved the DC/OS installer assets.
bootstrap_url: file:///opt/dcos_install_tmp
cluster_name: ClusterOne
master_discovery: static
exhibitor_storage_backend: static
master_list:
- 10.0.249.20
- 10.0.249.21
- 10.0.249.22
resolvers:
- 10.0.254.250
dns_search: medicineone.dom
ssh_port: 22
ssh_user: administrator
```

8.3.2 Monitoring and Failure Handling

The DC/OS has monitoring mechanisms inbuilt, both for packages / frameworks and containers. Marathon, the container scheduler, which is also responsible for scheduling the scheduler for other frameworks, monitors the health of each task, according to its definition. On the event any of them fails, or a subset of the nodes is lost due to hardware failure or network partitioning, they will be restarted or rescheduled on a different machine if necessary.

Since Apache Mesos also manages persistent volumes, applications that depend on this mechanism cannot be rescheduled on a different slave and will wait for the one with its data volume to be restored. Mesos has a tagging mechanism that allows distinguishing slaves according to arbitrary criteria, in this situation applications might also fail to reschedule if no suitable slaves are available.

A feature also implemented by Apache Mesos, and build upon by the DC/OS is log management. All the tasks are isolated inside containers under a supervisor controlled by Mesos, making the file system and logs available for consultation.

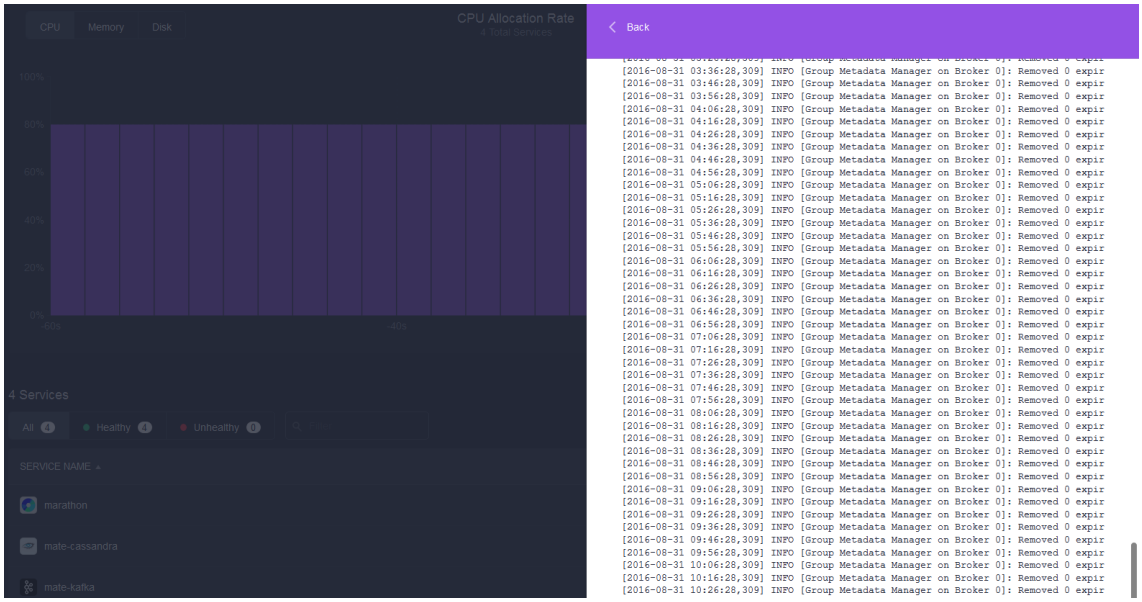


FIGURE 8.5: DC/OS log viewer.

8.3.3 Framework Installation

The Mesosphere DC/OS has a large selection of packaged frameworks and tools that can be easily installed. Well known examples are;

- Apache Spark;
- Apache Kafka;
- Apache Cassandra;
- Jenkins;
- Chronos;
- ArangoDB;

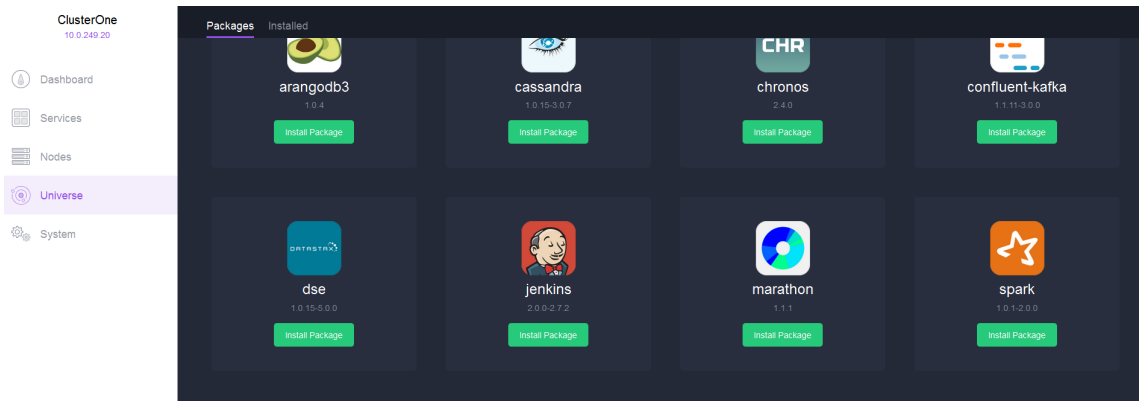


FIGURE 8.6: DC/OS package management dashboard.

Installation is easy, all that is required is creating a file with the desired configurations and installing the packages with the command line tool, dcos-cli. The command for installing Apache Spark with the default settings would be;

LISTING 8.3: Apache Spark package installation using the command line.

```
$ dcos package install spark
```

Another possibility would be using the graphical user interface, by connecting to one of the master nodes and clicking the desired package. Process depicted in figure 8.7.

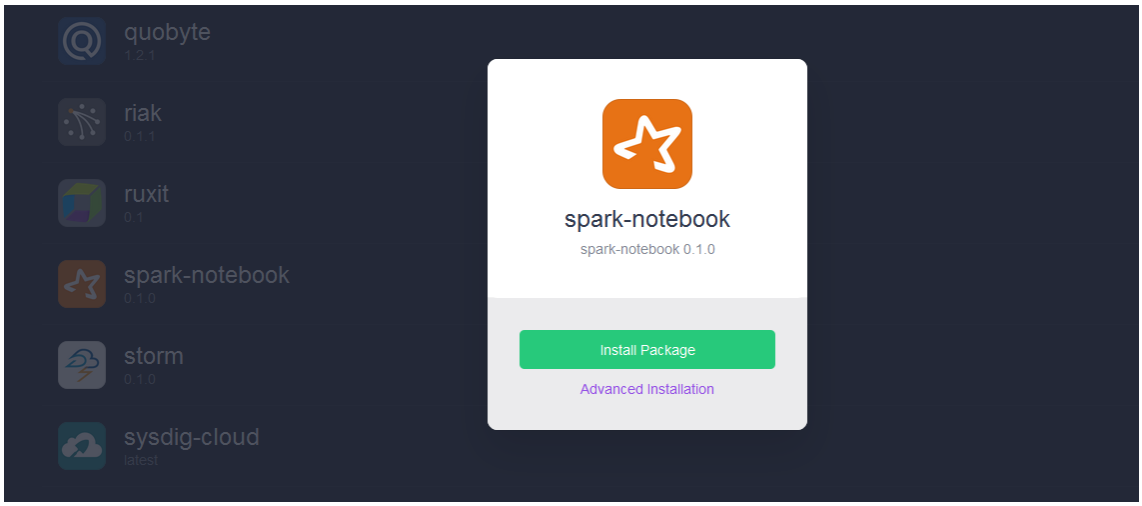


FIGURE 8.7: DC/OS package installation example.

8.3.4 Container Scheduling

In DC/OS containers are scheduled and monitored by Marathon[39] and referred to as tasks. Tasks are defined in javaScript object notation (JSON) in a declarative fashion. To achieve task isolation, Mesos can employ different technologies, including Docker containers. Since Docker has an online repository mechanism for containers, this makes it easy to distribute executables across the cluster as containers. Marathon also supports directly running binaries or commands, however the users has to provision a file server, or distributed file system to make them available on all nodes.
To schedule as task one creates a declarative file defining how many instances should be scheduled and what resources should be attributed to each one, such as cpu, memory, disk and network ports. A workload, that can be a binary, command or docker image must also be declared. Below is an example of a task that launches one instance of a python HTTP server.

LISTING 8.4: Sample Marathon task.

```
{
  "id": "pdtest",
  "cmd": "python3 -m http.server 8080",
  "cpus": 0.05,
```

```
  "mem": 128.0,
  "disk": 128.0,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "python:3",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8080, "hostPort": 0 }
      ]
    }
  },
  "healthChecks": [{
        "protocol": "TCP",
        "portIndex": 0
    }]
}
```

8.3.5 Hardware

The development system was deployed on 6 virtual machines running on MedicineOne's private cloud infrastructure. The cluster was setup with 3 master nodes for redundancy as well as 6 slave nodes for the actual workload. While not ideal, that is what was available, and it was good enough to test redundancy / replication and partitioning / distribution. For proper benchmarking and for the system to truly shine, more resources will be necessary.

Master nodes specifications:

- **CPU:** 4 x vCPU;
- **RAM:** 3GB;
- **Storage:** 256GB;
- **OS:** CentOS 7;

Slave nodes specifications:

- **CPU:** 4 x vCPU;
- **RAM:** 6GB;
- **Storage:** 256GB;
- **OS:** CentOS 7;

8.4 Development

In addition to the supporting platform, there was the need to define data models, as well as develop integration and configure components, those steps are explored in this section.

8.4.1 Integration

MedicineOne 8, the CMS generating the information MATE will process follows a SOA. Furthermore, it grew organically, developed by distinct teams and individuals, as such it is pretty heterogeneous and requires in depth knowledge to extend, especially when it comes to crosscutting concerns. The system was meant to send events such as pathology diagnosis, and drug and diagnostic test prescriptions to MATE, however there is no single point in MediconeOne 8 where these can be caught. Moreover, there is no logging functionality at the application level, most of it being currently done at the database level using triggers. Given the high effort necessary to implement this feature, it was decided to instead implement a service / module exposing a event generation and publishing API. This module was named MateEventLogger and is responsible for receiving the necessary information, generate an event payload in JSON and publish it to MATE. The rational was that it could be later integrated into the business logic by the teams responsible for each area.

The MateEventLogger service was developed in dotNet, more specifically C#, published with windows communication foundation (WCF), which are the basal technologies for MedicineOne 8. For testing and validation purposes, extra tooling was developed to load data from existing databases and generate events with past data.

The service exposes the following contract;

LISTING 8.5: MateEventLogger service contract;

```
namespace MateEventLoggerLibrary.Contract {
    /// <summary>
    /// This API exposes the endpoints of the Mate event logger
    /// </summary>
    [ServiceContract]
    public interface IMateEventLogger {
        /// Logs an event.
        [OperationContract]
        void LogEvent(Event eventInstance);
        /// Generates an event based on the PK's of the involved
            entities.
        /// organizationLocal: PK of the ORG_LOCAIS_ORGANIZACAO
        /// physician: PK of the physician, from
           ORG_UTILIZADORES
        /// patient: PK of the patient from, CLI_UTENTES</param>
        [OperationContract]
        Event CreateBaseEvent(Guid organizationLocal, Guid
           physician, Guid patient);
    }
}
```

The *Event* returned by *CreateBaseEvent* is a general event that can then be converted into a specific event by adding a data object with the specific details, for example, if it was a dug prescription event, an object would be added with the classification system and code of the prescribed drug.

8.4.2 Data Model

Data models were defined both for event representation and for the raw event storage table;

Events

In the MateEventLogger service, events are represented by the data objects depicted in the class diagram in figure 8.8;

FIGURE 8.8: MateEventLogger event data objects class diagram.

Before being sent to MATE events are serialized to JSON, as illustrated by the example below;

LISTING 8.6: MateEventLogger serialized event;

```
{
  "Drug": {
    "Classification": "PRT",
    "Code": "3220092"
  },
  "Id": "7efd8810-f117-4aff-83d3-d6bade719f8f",
  "Timestamp": "2016-08-31T15:17:43.1098015Z",
  "Type": "DrugPrescriptionEvent",
  "ServerId": "2ffff04a-6f8e-11e6-8b77-86f30ca893d3",
  "IsTest": true,
  "Organization": {
    "Uuid": "9531ddc5-2fd1-450e-9f9d-89c8e63564e0",
```

```
    "Name": "USF Cruz de Celas",
    "Type": "SNS"
  },
  "Location": {
    "Country": "PRT",
    "Distrito": "60000",
    "Concelho": "60300",
    "Freguesia": null,
    "PostalCode": "3000063",
    "CoordinatesSource": null,
    "Latitude": null,
    "Longitude": null
  },
  "Physician": {
    "Specialties": [
      "MEDICINA GERAL E FAMILIAR"
    ],
    "SpecialtiesPk": [
      "ad4f10b6-cc03-4c2b-9043-9f6c01c6331b"
    ],
    "Age": null,
    "Sex": "Female"
  },
  "Patient": {
    "Age": 37,
    "Sex": "Male",
    "Height": 174.0,
    "Weight": 69.0
  }
}
```


Raw Storage

For persistence and failure tolerance purposes, all events are persisted to the distributed data storage, provided by Apache Cassandra. This task is carried out by a Storm topology that streams the events from Kafka, the message broker, and inserts them Cassandra. No processing is done, and for performance reasons, events are partitioned / segmented according to MedicineOne 8 server id and the date (day). This ensures even partitioning and scalability while keeping related events together (locality) for batch processing workloads. Furthermore, the keyspace where the data is stored has a replication factor of 2, ensuring one of the replicas can be lost without loss of data or availability. Note that this value was used for development purposes and might need to be increased in production.

The Cassandra table where the events are stored as well as its keyspace are defined as follows, using cassandra query language (CQL);

LISTING 8.7: Mate keyspace definition.

```
CREATE KEYSPACE IF NOT EXISTS mate WITH REPLICATION = { 'class'
   : 'SimpleStrategy', 'replication_factor' : 2 };
```

LISTING 8.8: Cassandra table definition for raw event storage.

```
CREATE TABLE IF NOT EXISTS mate.raw_events (
    server_id uuid,
    date text,
    time_stamp timestamp,
    type varchar,
    event_json varchar,
    PRIMARY KEY((server_id, date), time_stamp)
)
WITH CLUSTERING ORDER BY (time_stamp ASC);
```

8.4.3 Stream Processing Topologies

Two stream processing topologies were developed for the MATE system during the internship;

- Raw event persistence topology;
- Event ETL topology (prepares events for Druid);

These topologies run on top of Apache Storm, which distributes them across the available hardware; Storm topologies are pipe and filter processing tasks based around the concepts of Spouts and Bolts. Spouts represents the data origins, for example a spout can read data from Kafka and emit the messages to a graph of Bolts, each representing a processing step. Parallelism of both spouts and bolts can be controlled by the developer. Topologies can be made transactional using an acknowledgment mechanism at each processing step.

Raw Event Persistence

This topology reads events from Kafka, deserializes them, extracts the date of the event from the times stamp and inserts it into Cassandra.

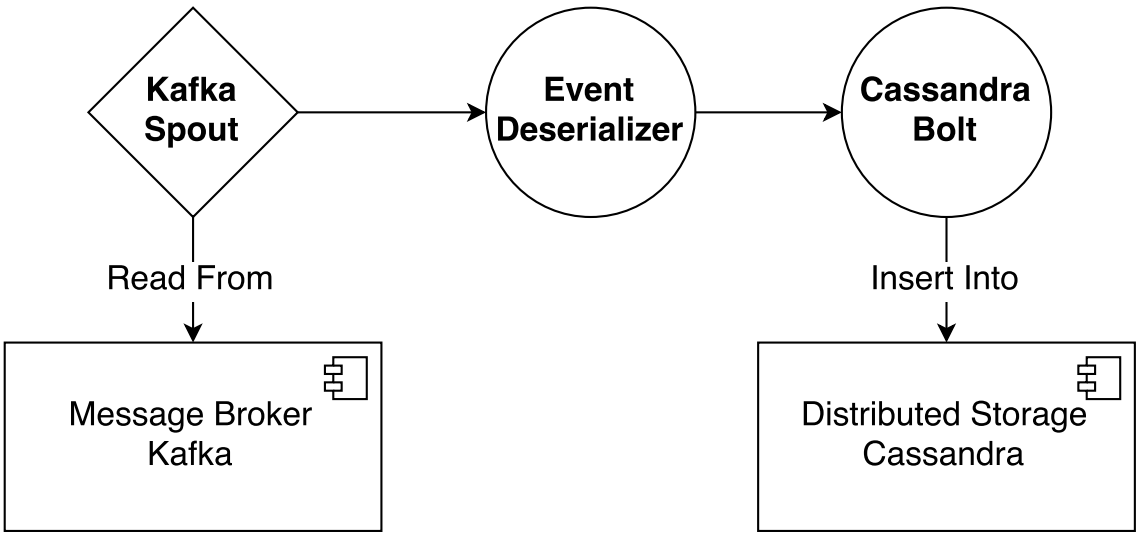


FIGURE 8.9: Raw event persistence Storm topology.

Event ETL for Druid

For interoperability and generality, events are sent by the MedicineOne 8 CMS as nested objects serialized in JSON. While druid can ingest data in JSON it requires them to be flat, as there is no support for nested dimensions. This topology loads the events from the message broker, deserializes them, flattens the JSON and depending on the type of event saves them to a different queue on the message broker. This way druid can then effortlessly load them from the message broker.

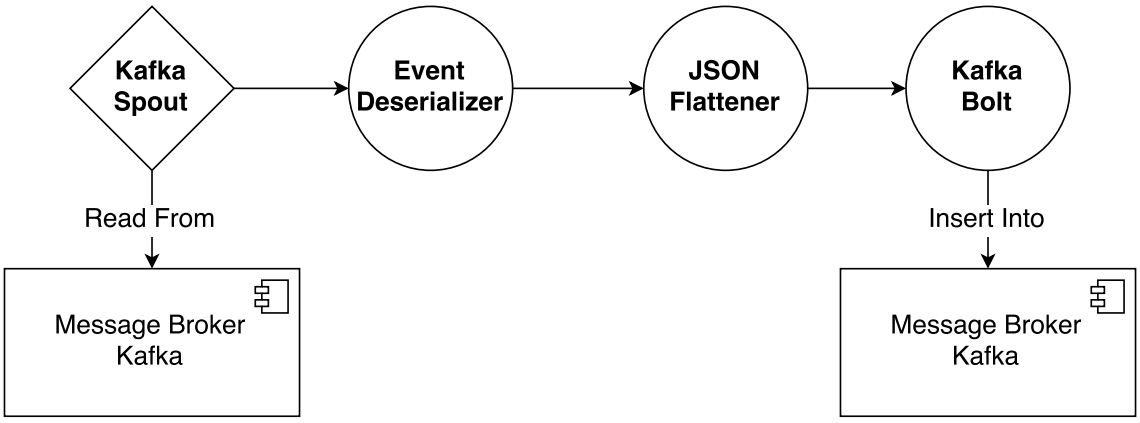


FIGURE 8.10: Event ETL Storm topology.

8.4.4 System deployment

The DC/OS has a consistent command line interface client that allows scripting cluster management easily. Since it allows framework / package installation, task management on Marathon, etc, it was possible to automate the deployment of the MATE system. A folder was created with the configuration files for each component, and Marathon task definitions. The administrator only needs to make sure the configurations are correct and run a script. This will take care of installing the necessary frameworks and tasks, namely;

- Apache Kafka;
- Kafka Manager (Web based graphical management interface for Kafka);
- Apache Cassandra;
- Apache Storm;
- Druid;
- Pivot (Druid OLAP exploratory visualization framework);
- Raw event persistence topology;
- Event ETL topology (prepares events for Druid);

Being able to effortlessly deploy and destroy the system, or create multiple instances for testing greatly accelerates development and debugging.

8.4.5 Application Deployment

Firstly, any application developed for MATE can use any of its components, from the message broker to the OLAP or distributed NoSQL storage.

Future applications developed on top of MATE can be deployed using various strategies;

For processing tasks, that load events, process them and store the results, the most natural way would be to leverage Apache Storm. This entails writing the processing logic using the Storm API and submitting it to Storm. From then on, Storm will make sure the application is distributed across the cluster and kept running even on the event of partial infrastructure failures.

If the workload requires some framework supported by Mesos and DC/OS, it can be installed and the application submitted to the new service, Apache Spark would be the prime example.

As a last example, for user facing applications, a very convenient alternative would be developing the application in whichever is considered the best technology or language. This application would then be encapsulated as a Docker container, a Marathon task defined and then scheduled on Marathon. This application would be monitored, kept running and scalable with a command or click on the web graphical user interface.

The listed approaches and the resulting compositions create a very stable, modular deployment platform, that makes extension safe and effortless.

Chapter 9

Results

The real time analytics field is changing very fast as more and faster hardware and tools are available. The lack of stability felt during the project is a direct result of that, new tools and approaches are released every week. The main example is Mesosphere's DC/OS that came out close to the end of the internship and while making some previous work obsolete, was well received by the community and users.

The result of the internship is a general, low latency, or soft real time dimensional event processing system for analytic applications. It has the necessary primitives / tools to do stream processing, batch processing and even machine learning and prediction applications. The necessity to generalize and not commit to any particular use case was a consequence of MedicineOne 8 not being yet ready for data publishing for lack of a standardized representation convention and consistent data model. Allied to this, comes the fact that the product, from a market perspective has not yet been developed and the MATE will have to adapt to the clients necessities.

While this was unexpected, it is in no way a negative result. The performance and quality requirements exceed the expectations and the current state of the art allows for more than initially expected. According to the threshold of success defined in the requirements and the intern's opinion the project was a success, the next sections present evidence in favor of this statement.

9.1 Management

Management of the platform is to a large degree taken care of by Mesos', Marathon's and DC/OS' graphical user interfaces (GUIs). The command line and REST APIs make it possible to automate all management processes or make the system elastic to a degree. However, day to day management can easily be done through the GUI. For illustrative purposes a few screenshots are presented in figure 9.1.

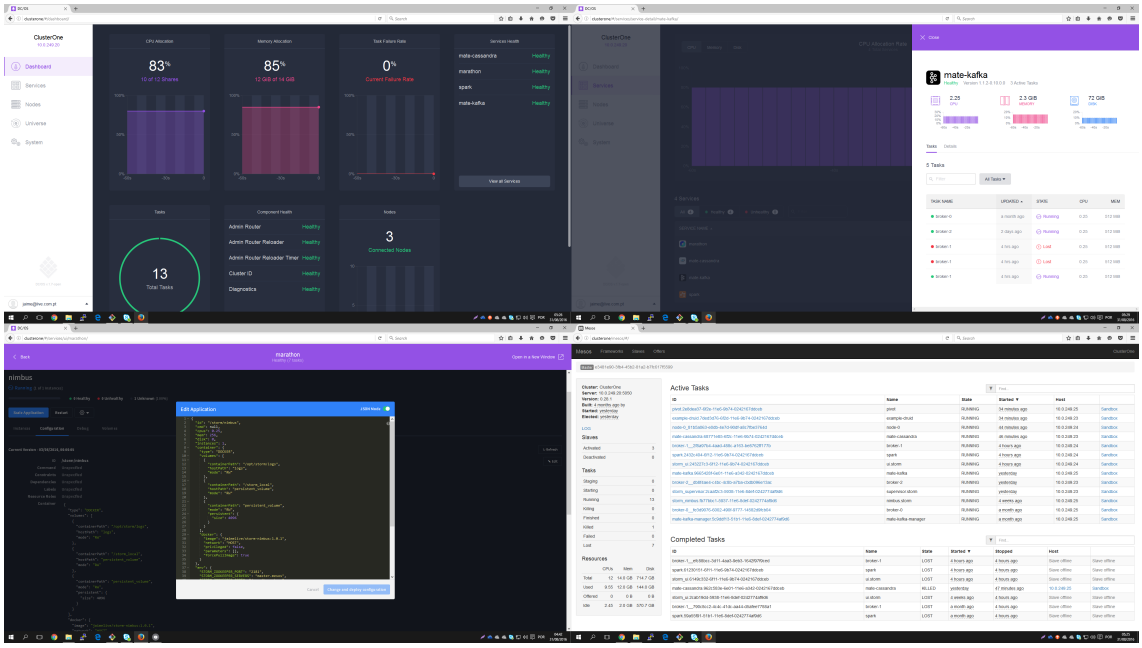


FIGURE 9.1: Management interfaces.

The GUIs let the operators easily scale, schedule and terminate tasks as well as consulting their logs or redefine their access to computational resources. It is also possible to install and configure packages / frameworks entirely through the GUI and with very little effort. For example, to manage kafka and inspects its performance, an application called Kafka-Manager[43] was installed and configured through a simple graphical interaction.

9.2 Performance

Since the main function of the system is performing analytical operations over streams of events, the metric chosen for performance testing is events / second. Even though benchmarking is presented, the reader should be aware that since the tools being benchmarked were designed for much larger deployments, these results, while more than good enough to cover the requirements, are not at all representative of the full potential of the system. This was mostly a consequence of infrastructure availability and to compensate for it, third party benchmarks are referenced.

When it comes to system wide performance, it far exceeds the requirements by at least one order of magnitude. As shown on the more detailed subsections, the bottleneck will be the stream processor, at 821 events / second processing rates. However, as long as event volumes stay within that limit, latency from ingestion to availability at the data storages will be way under the stipulated limit of 10 seconds.

9.2.1 Message Broker

With 2 message brokers, a queue with 2 partitions and a replication of 2, the message broker service was able to consistently ingest an average of 9358 events / second, each approximately 1Kilobyte in size with a sample size of 16 millions events. These were less than ideal conditions, machines properly tunned for Kafka, will yield much better results with linear scaling as verified by Linkedin's benchmark[44]. However, as stated, this more than covers the performance requirements of the system and leaves enough performance for functionality expansion.

9.2.2 Stream Processor

Processing data from a queue loaded with 16 million events, both topologies performed as expected. The raw event persistence operated at the distributed data stores capability. The event ETL topology managed to process and write on average 821 events / second. The bottleneck was JSON deserialization and flattening, on a larger cluster this could be fixed by increasing the number of messages allowed to be in flight inside a topology before acknowledge and increasing the parallelism of the flattening bolt. While a definite bottleneck, it still performs adequately for the project requirements and will accommodate the growth in the foreseeable future.

9.2.3 Distributed Storage

Apache Cassandra's scheduler on DC/OS had a known bug at the time of writing that made it impossible to run each node with less than 2Gigabytes of memory. This severely limited what could be done on the small development cluster. The write benchmark was performed with a single node, in this condition, it averaged 889 writers / second with 4 clients in parallel. Once again, since this result is not representative due to the limitations, a good source of performance information is the benchmark done by Netflix[29], which shows an average of 11429 writes / second per node with linear scalability from 50 to 350 nodes.

9.2.4 Distributed, In memory OLAP Storage

Druid is an in memory data storage, and while historic data may be paged to disk for a performance / cost trade-off, recent data must be in memory. The fact there wasn't enough memory available to run Druid fully distributed made the benchmark hold very little value. Information about performance is available at the official website[45]. The data supports that as long as there is enough memory to hold all the events expected between segment archiving events, Druid will not be the bottleneck in the system. This information is corroborated by Linkedin's benchmark[46] where druid can ingest 20000 rows per real time process while keeping sub-second query latency.

9.3 Validation

As predicted during the risk analysis there was no time to work on an actual implementation of a user facing analytical soft real time dashboards application, and work was centered around back-end and operations.

Specifically the following requirements were not fulfilled;

- SR-F-FE-01
- SR-F-FE-02
- SR-F-FE-03
- SR-F-FE-04
- SR-F-FE-05
- SR-F-FE-06
- SR-F-FE-07
- SR-F-FE-08
- SR-PERF-SYS.03

All of them were low priority and the risk of not having time to do them had been accepted.

Excluding the previously mentioned, the High and Medium priority are validated in the next two subsections.

9.3.1 Functional Requirements

All the functional requirements are directly fulfilled by the services that compose the MATE system. For details consult table 9.2.

TABLE 9.1: Functional Requirements Validation.

ID	Priority	Validation
SR-F-M1S.01	High	Generation and publishing assured by the MateEventLogger service. Ingestion and availability for processing is ensured by the message broker.
SR-F-BE.01	High	The message broker is replicated and persistent for a period of 1 week, and the Raw Event Storage topology persists them permanently on the distributed data storage.
SR-F-BE.02	High	Functionality provided by the message broker and the OLAP data storage.
SR-F-BE.03	High	Functionality provided by the distributed data storage and the OLAP data storage.
SR-F-BE.04	High	Functionality provided by the distributed data storage, for maximum granularity, and the OLAP data storage for aggregates.
SR-F-BE.05	High	Functionality provided by the OLAP data storage.

9.3.2 Quality Requirements

Quality requirements are generally satisfied by the components and the architecture. For details consult table 9.2. However, since the mean time between failure

of components is unknown, the intern was not able to accurately calculate the necessary replication to achieve the desired availability goals.

TABLE 9.2: Quality Requirements Validation.

ID	Priority	Validation
SR-PERF-SYS.01	High	Satisfied, consult Performance section.
SR-PERF-SYS.02	High	Satisfied under the assumption there are enough resources, consult Performance section.
SR-QOS-SYS.02	Medium	The system is a composition of horizontally scalable services.
SR-QOS-SYS.03	High	Satisfied through SOA and the presence of scheduling services such as Marathon.
SR-REL-SYS.01	Medium	Satisfied by the usage of redundant components under the assumption there enough replicas.
SR-REL-SYS.02	Medium	Satisfied by the usage of redundant components and a monitoring service, under the assumption there enough replicas.
SR-REL-SYS.03	High	Satisfied by the redundant message broker and redundant distributed data store.
SR-MAINT-SYS.01	High	Satisfied by the architecture.
SR-MAINT-SYS.02	High	Satisfied by the usage of redundant components under the assumption there enough replicas.
SR-SAFE-SYS.02	High	The message broker is replicated and persistent for a period of 1 week, and the Raw Event Storage topology persists them permanently on the distributed data storage.

9.3.3 Stream Processing Topologies

Since the Apache Storm library allows local testing of the topologies and each bolt has a well defined function, topologies were validated by unit testing each part. As they were developed in Java jUnit and maven was used for test management. Topologies were considered successful when all tests were passed. Tests consisted of a few examples where the results where known and considered representative of the domain. Furthermore, the system was left running for 16 million events and no exceptions were logged at any point.

9.4 Applications

Possible applications for the MATE system are;

- **OLAP Dashboard:** Druid can be for aggregation and other analytical queries while using Cassandra for state storage and making the actual front end application stateless. By wrapping it in a docker container and scheduling it with a Marathon task, it would be extremely easy to manage and scale as necessary.
- **Machine Learning:** Machine learning from the events streams and historic data could be easily implemented by using DC/OS package manager to install Apache Spark. With the help of MLlib[47] it would be easy to train a given model from aggregates (Druid), historic data (Cassandra, Druid) and even streams (Kafka). The state of said model could be saved on Cassandra or other easily deployable high performance storage like Redis. Predictive analysis could then be applied to incoming events using Spark or Storm.

Chapter 10

Conclusion

Looking back, the internship started full on uncertainty, exploring a completely new field, however it is definitely a success technically and academically. Thanks to the freedom given at MedicineOne it was possible to explore the viability limits of soft real time analytics platforms as well as explore the area of operations management. The result of the project is a soft real time processing platform adequate to MedicineOne's data processing needs for the foreseeable future. And while less focus was placed on the actual application of the platform, initially a dashboard application, a lot more was put into provisioning, deployment, and quality attributes. As it stands, MATE is more than just a frail prototype, it is a reliable, easy to deploy anywhere, cloud or premise, replicable environment that can support analytics and even some more performance demanding machine learning applications. At a personal level it was incredibly rewarding to explore an emerging field, surrounded by active communities and constant change, even if it came at the cost of having to deal with unexpected change and compatibility issues.

I thank all the involved parties for the opportunity to explore, learn and grow.

10.1 Future Work

For the future there are some topics worth exploring, namely;

- Security, which according to the road map will be a big part in the next DC/OS update;
- Data schema abstraction, developing an abstraction layer that made it possible to define, in a declarative format, new events to be ingested would make the system much more general and flexible;
- Better stream processing, new alternatives have surfaced, namely Concord and Heron, both are worth exploring;
- Study and characterize failure, most platforms used are relatively new and their failure patterns are now well known;
- Application level logging on MedicineOne 8, adding logging at application level would make the system more friendly to analytics and intelligence extraction;

Appendix A

System Requirements Specification

Authors and Contributors		
Name	Initials	Contact
Jaime Correia	JC	jaime.correia@medicineone.net
Marco Tinoco	MT	marco@medicineone.net

Revision History			
Version	Date	Author	Description
1.0	14/11/2015	JC	Requirement document based on previous meetings and state of the art study.
2.0	11/01/2015	JC, MT	Corrections suggested by Marco Tinoco and added requirement SR-F-FE-08.

Access List	
Internal Access	External Access
Public	Restricted

Table of Contents

1. Introduction

1.1. Objective

The purpose of this document is giving the reader a formal definition of the requirements of the real time healthcare aggregation and visualization platform to be developed.
As well as a guide for the developer and a management tool for the Product Owner, it will also serve as communication channel between the two parties and help materialize and scope the system.

Since the project was born from an idea and hadn't had previous formalization or development, this document will provide a much needed common ground to help convey the idea and validate its understanding and expected functionality.

1.2. Scope

This document pertains to the RTHAV project, and seeks to expose its various types of requirements and constraints as well as the process that lead to their gathering.

It will serve as the first formal definition of the project and condition and drive the next steps of the development process. Finally it will be a tool to validate the final product and thus categorize it in terms of completeness and readiness to be used in production and/or marketed.

1.3. Audience

This document is mainly meant for internal use by M1, specifically by the developers and other involved technical personnel as well as the stakeholders, specifically the Project Owner.

The technical sections are aimed at the developers and will serve as a starting point and general guide both for development and validation of the end product.

For the remaining stakeholders this document will ensure that their input and ideas were properly understood and materialized and as a mean to rectify and approve them.

1.4. Definitions and acronyms

Table 1 presents the list of definitions used throughout this document.

Name	Description
Applicable Document	A document is considered applicable if it contains provisions that through reference in this document incorporate additional provisions to the present document.
Reference Document	A document is considered a reference document if it is referred but not applicable to the present document.
Real Time Healthcare Aggregation and Visualization (RTHAV)	Code name for the real time healthcare data aggregation and visualization platform this documents pertains to.
Project Owner	The product owner represents the stakeholders and the customers. In the current project this is the person who idealized and created the project.

Table 1: Definitions

Table 2 presents the list of acronyms used throughout this document.

Name	Description
AD	Applicable Document
M1	MedicineOne
RD	Reference Document
TBC	To be confirmed
TBD	To be defined
RT	Real Time
RTHAV	Real Time Healthcare Aggregation and Visualization
UML	Unified Modelling Language
SRS	Software Requirements Specification

Table 2: Acronyms

1.5. Reference Documents

Reference	Source	Version/Date
Kanban	https://github.com/agilelion/Open-Kanban	27/10/2015
Scrum	http://epf.eclipse.org/wikis/scrum/	27/10/2015
A User Case Template: Draft for discussion, Derek Coleman	http://www.bredemeyer.com/pdf_files/use_case.pdf	27/10/2015
External Interface Modelling	http://www.agilemodeling.com/shared/ExternalInterfaceSpecTemplate.doc	27/10/2015

Table 3: Reference documents

2. Standards, conventions, procedures and tools

To achieve a homogenous documentation and development process, this section will detail the standards and lifecycle the project will follow.

2.1. Design Standards

For design needs, UML will be used when possible and expressive enough.

Even though the project will follow an agile life cycle without focus on formal documentation, there will be at least an SRS with the initial requirements, extracted from use cases.

2.2. Project Life Cycle

2.2.1. Considerations and Rational

Given that the project will be developed under the guise of a curricular internship, and divided into two stages, (first and second semester) with distinct levels of dedication (part time during first semester), it is especially important to use an adequate life cycle.

During the first semester, the project will be developed in part time, with an average weekly effort of 16 hours, on the second semester full time attention will be available, totalizing 40 hours of weekly dedication.
This distribution conditions the life cycle greatly, and trying to fight against it would not be productive, therefore, and since the project will require considerable architectural and documental effort, the first semester will be dedicated to precisely that regardless of chosen life cycle.

The second semester will be used for implementation, testing, benchmarking and finalizing documentation.

Looking at the project itself, a scalable event aggregation system with support for real time visualization, and the development environment, an incremental, iterative life cycle confers agility to the process and makes it easy to gradually steer the project in the desired direction while making decisions based on the reception of previous builds.

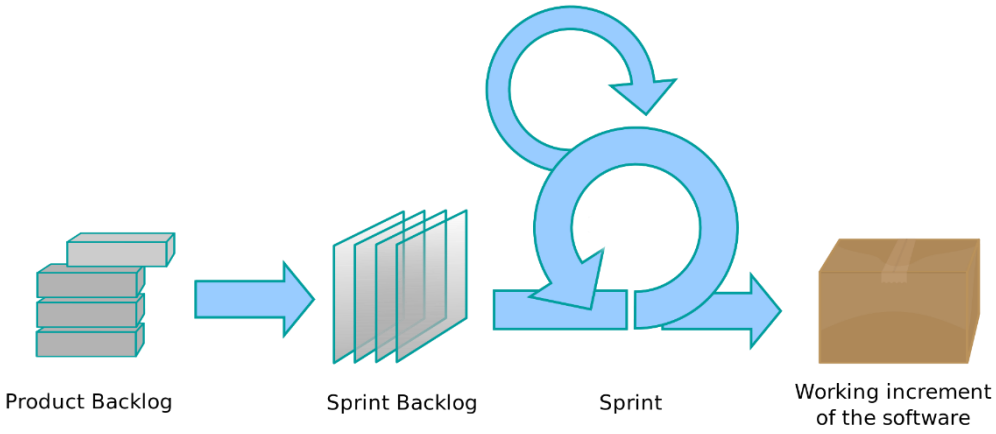
As a final note, there is no development team, the project is the sole responsibility of one developer, therefore only some portions of the chosen Life Cycle, or specifically process, will be used, namely task management, development / deployment / testing pipeline and requirements and change management.

2.2.2. Life Cycle

Certain considerations made the choice of life cycle evident. Since analytics and business intelligence is not an area where the developer is well versed or especially knowledgeable about, and there is no readily available expert in the organization, the only sane option is to define end goals and interactively try to achieve one at a time.

Starting from the integration with the existing system, and moving in a direction of adding functionality trying to achieve a functional front end application.

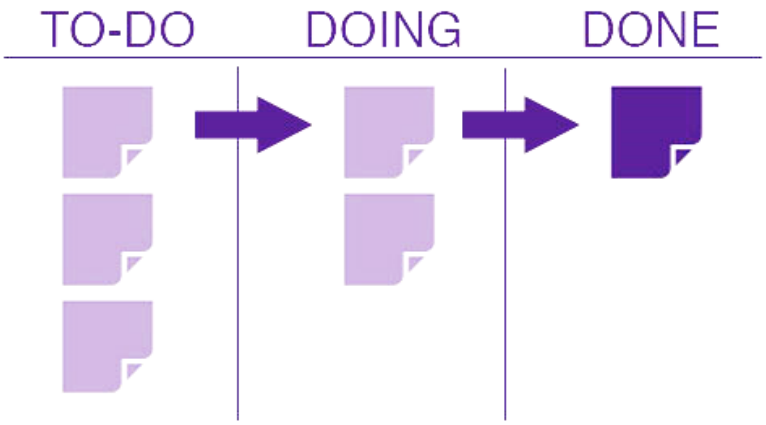
The project will follow an **incremental** life cycle, based on **SCRUM**, maintaining the notion of backlogs and sprints.



During the development stage, features will be segmented and incrementally added to the backlog, this way, at the end of each sprint the project owner will have the opportunity to experience the resulting artifact and possibly introduce changes.

For backlog and sprint management a **Kanban** board will be used. This choice makes it easy to estimate progress, time spent on each feature and to easily get an overview of the remaining work and in the end, evaluate the readiness of the product.

The planned sprint duration is **1 week.**



2.2.3. Change Management

It is expectable that during the requirement analysis, architectural design or after every sprint, the product owner might want to change certain aspects of the system to better fit company objectives.

After evaluation by the developer, changes will be dealt with according to cost and complexity.

Minor changes that do not add cost to the project will be added to the backlog and dealt with through the normal process.

Since there are very limited resources, and the project is part of a curricular internship, changes that require extending the scope of the project should be solved by allocating more human resources to the project and isolation of that proposed functionality in a new module.

2.3. Software Tools

This section lists the software tools necessary for the production of documentation and management of the development process as well as tools that are foreseeable to be needed later on. Even though the last part is not a part of the Requirements in the strict sense, the information is still useful to have for planning purposes.

Name	Version	Notes
MS Word	<versions>	Word Processor
Git	<version>	Repository and versioning tool
MS Visio	<version>	Diagram authoring
Sublime Text	2	Text Editor
Visual Studio	2013	IDE to manipulate and integrate the existing system.
SQL Server Management Studio	2013	Manipulating the existing data sources.
IntelliJ Idea	<latest>	Software development.
Java SDK	8	Software development.
Build Platform	?	Platform to automate solution building.
Deployment Platform	?	Platform to automate deployment.
Logging Platform	?	Platform to collect and analyse logs.

Table 4: List of software tools to be used

3. Load Estimations

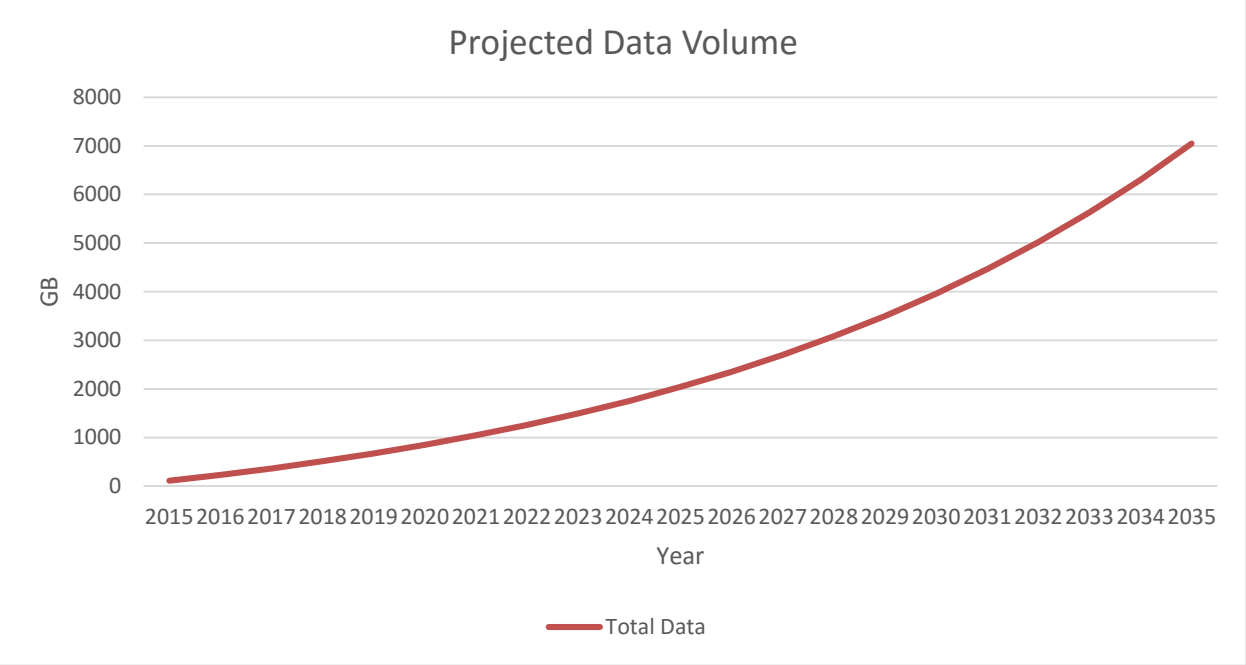
From operating information provided by MedicineOne, taken from two of their largest running systems, and assuming that the real volume of data is 4 times as much and will grow 10% each year, as suggested by the Project Owner, the following projections have been made for the next 20 years.

It is also assumed that each event will be around 2Kbytes in size.

Even if the values end up not representing real life condition, they should at least be in the same order of magnitude and serve as a good benchmark.

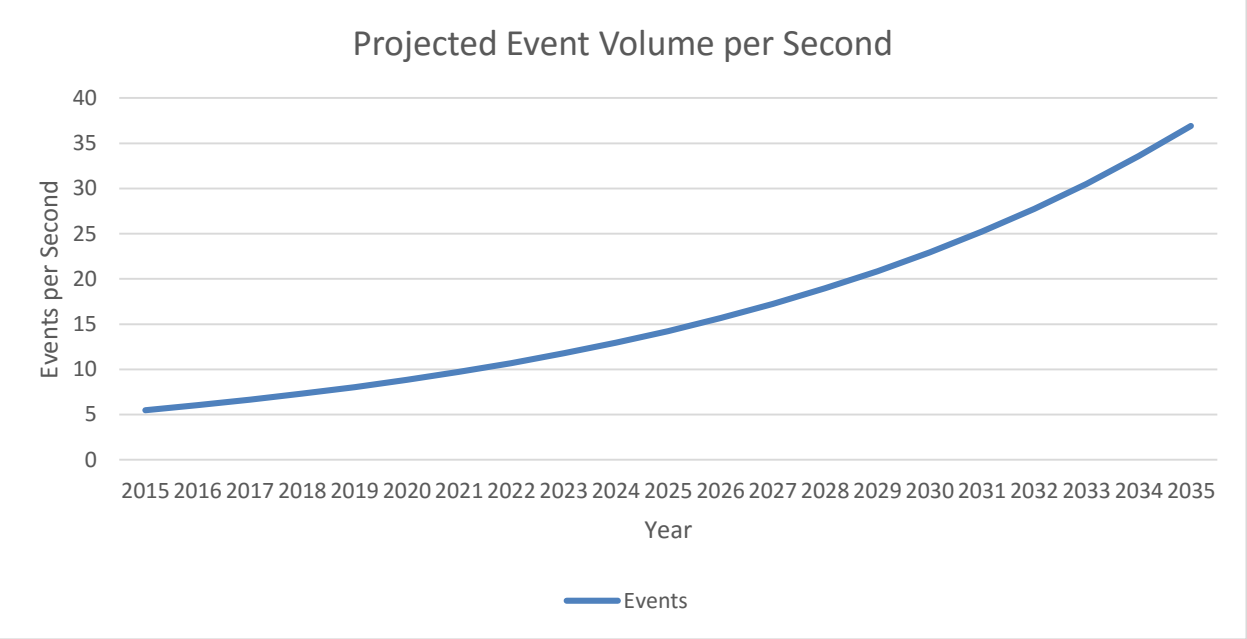
3.1. Data Volume

Data volume estimation will heavily impact technology selection, especially data persistence components such as databases.



3.2. Event Volume

This estimation will serve to establish some performance requirements, therefore being extremely important to set metrics for system validation.



4. Software Requirements Catalogue

4.1. Functional Requirements

4.1.1. MedicineOne Server - Existing System

SR-F-M1S.01	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: Medium
Details	Created on 05-11-2015.

The **MedicineOne Server** shall generate events for the listed actions, and reliably send them to the **Central Repository** serialized as JSON.

Each event should contain the following information:

- Timestamp
- Location (Zip-Code or Coordinates)
- Organization
- Physician Information
    a. Specialty
    b. Age
- Patient Information
    a. Sex
    b. Age

The relevant actions are listed below, with their particular attributes to include in the event, nested.

- **Drug prescriptions;**
    - Country Identifier *(M1SYSTEM.InstallationSettingValue, SettingId = InstallationCountry)*
    - National Identifier *(dbo.cli_prescricoes_utentes.Cod_Nacional)*
- **Pathology diagnoses;**
    - Classification System *(dbo.std_classificacoes_patologias.?(PK? | Nome?))*
    - Pathology Code *(dbo.cli_patologias.Codigo)*
- **Diagnostic test prescription;**
    - Specialty Code *(dbo.cli_actos_clinicos.codigo_area_convencao)*
    - Sub Type Code *(dbo.cli_actos_clinicos.Codigo)*

4.1.2. Backend

SR-F-BE.01	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: Medium
Details	Created on 05-11-2015.

The **Central Repository** shall persist received events.

SR-F-BE.02	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The **Central Repository** shall provide metrics related to the events in real time for subscription. The events are described in the requirement SR-F-M1S.01. The metrics to be made available in real time are: • Total events of each type; • Top N of each type of event, by specific attributes of each type and respective count; • The previous metrics, segmented by geographical region;	

SR-F-BE.03	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The **Central Repository** shall provide time series related to the events in real time for subscription. The events are described in the requirements SR-F-M1S.01 and SR-F-BE.02. The time series to be made available in real time on subscription are: • Count for each type; • Count for the specific attributes of each type; • The previous time series, segmented by geographical region; • Evolution of the previously mentioned metrics.	

SR-F-BE.04	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The **Central Repository** shall provide access to historic portions of the data artefacts described in SR-F-BE.02	

SR-F-BE.05	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The **Central Repository** shall let other systems query the event data and multidimensional aggregates of that same data, both real time, and historic.	

4.1.3. Frontend

SR-F-FE-01	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall authenticate users with a username and password, and credit them with the permissions associated with the account. The system will provide the following user profiles: • Operator: Full permissions, access to all data, user and respective permissions management. • Administrator: User and respective permissions management. • User: Access to data defined on personal permissions.	

SR-F-FE-02	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall let the Operator and Administrator set the permissions for each user, allowing control over which time series, time intervals, metrics and aggregates he can access.	

SR-F-FE-03	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall provide visualisations for the data elements described of the previous SR-F-FE requirements. There should be the following elements, according to type of data and information available: • Line charts (time series) • Gauges (metrics, aggregates) • Pie charts (metrics, aggregates) • Bar charts (metrics, aggregates) • Geographical charts (location information available)	

SR-F-FE-04	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall let the user overlay real time data series with sections of historic data of the same time series for comparison purposes.	

SR-F-FE-05	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall update visualisations pertaining to real time data, in real time as well. Real time in this context means, as soon as it gets the notification from the backend.	

SR-F-FE-06	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall let the users select between the three types of events available.	

SR-F-FE-07	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall let the user, select the different metrics, time series and aggregates in each section.	

SR-F-FE-08	
Type	Functional
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: Medium
Details	Created on 05-11-2015.
The **Frontend** shall, when necessary, provide a translation layer to show and input identifiers of drugs, pathologies and diagnostic tests in a user friendly format.	

4.2. Performance Requirements

SR-PERF-SYS.01	
Type	Performance
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The system shall be able to scale up to ingest at least 40 events per second. This metric comes from the estimations presented on a previous sections.	

SR-PERF-SYS.02	
Type	Performance
Status	Proposed. Version 1.0. Phase 1.0 Priority: Medium Difficulty: High
Details	Created on 05-11-2015.
The system shall present latencies inferior to 10 seconds between ingestion of an event and it being reflected on the metrics and/or data model.	

SR-PERF-SYS.03	
Type	Performance
Status	Proposed. Version 1.0. Phase 1.0 Priority: Low Difficulty: High
Details	Created on 05-11-2015.
The Frontend shall be replicable to accommodate a growing number of users, limited only by the performance of the data backend. This can be achieved by making the Frontend stateless.	

SR-PERF-SYS.04	
Type	Performance
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The data storage and processing capability shall be scalable horizontally to accommodate a growing number of events and queries to the aggregates and metrics.	

4.3. Interface Requirements

SR-I-SYS.01	
Type	Interface
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: Medium
Details	Created on 05-11-2015.
When not constrained by proprietary component implementations, interfaces, shall use well defined and standardised protocols and languages for interoperability. At the time of writing, JSON over HTTPS seems like a good option.	

SR-I-SYS.02	
Type	Interface
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: Medium
Details	Created on 05-11-2015.
System components should be as decoupled as allowed by performance requirements. This is not a hard requirement, more of a design guideline or recommendation.	

4.4. Operational Requirements

The operation of the system as a whole shall be done using the web based front end, after the operator has authenticated with the necessary credentials.

Operation of the used components and infrastructure will depend on their individual characteristics since at this point decisions on that end still haven't been made.

4.5. Resource Requirements

There are no explicit limits. Ideally, as specified in the performance requirements, the system should be able to scale to be able to accommodate a growing number of requests. When possible, components that scale horizontally should be used, and when it is not, bottleneck mitigation plans should be established.

4.6. Verification Requirements

The system shall produce, store and make available enough logs and metrics, such as performance, ingestion speed and hardware load, to make it possible to verify requirements.

When it comes to functional requirements, a **checklist** shall be made, containing tests, either manual or automated to ascertain the validity of each one.

4.7. Documentation Requirements

In the interest of saving time, after the initial planning documentation, only the absolute necessary to understand and operate the system will be produced, such as deployment and building documentation as well interface documentation.

4.8. Security Requirements

The system transfers health-care data, therefore it should always use **encrypted channels when crossing public networks and systems**, however, and since the data is anonymized there should be no pressing legal issues on this end.

Operation and usage of the system should require authentication and provide **auditability** and **accountability** through adequate logging.

4.9. Portability Requirements

In the interest of portability the following characteristics are required:

1. The frontend shall be web based and run on at least the 3 most popular browsers;
2. Interface between modules shall use well documented standards and protocols;
3. Deployment requirements and procedures shall be documented and or automated;

4.10. Quality Requirements

SR-QOS-SYS.01	
Type	Quality
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The system shall present latencies inferior to 10 seconds in real time metrics, from ingestion to availability for presentation.	

SR-QOS-SYS.02	
Type	Quality
Status	Proposed. Version 1.0. Phase 1.0 Priority: medium Difficulty: High
Details	Created on 05-11-2015.
The system shall be horizontally scalable.	

SR-QOS-SYS.03	
Type	Quality
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: Medium
Details	Created on 05-11-2015.
The system shall be extensible by means of module addition. New modules shall be able to subscribe to the input even feed and query the existing data.	

4.11. Reliability Requirements

SR-REL-SYS.01	
Type	Reliability
Status	Proposed. Version 1.0. Phase 1.0 Priority: Medium Difficulty: High
Details	Created on 05-11-2015.
The system shall be available to ingest events at least 99% of the time.	

SR-REL-SYS.02	
Type	Reliability
Status	Proposed. Version 1.0. Phase 1.0 Priority: Medium Difficulty: High
Details	Created on 05-11-2015.
The system shall be available to reply to user queries at least 99% of the time.	

SR-REL-SYS.03	
Type	Reliability
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The system shall implement the necessary redundancy to ensure data persistence in the event of hardware failure.	

4.12. Maintainability Requirements

SR-MAINT-SYS.01	
Type	Maintainability
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
The system shall employ general standardized interfaces so that components may be replaced with homologous ones when they exist, for example a standard SQL database.	

SR-MAINT-SYS.02	
Type	Maintainability
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
Failure or planned maintenance of a clustered component shall not cause system downtime.	

4.13. Safety Requirements

SR-SAFE-SYS.01	
Type	Maintainability
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
Failure or planned maintenance of a clustered component shall not cause system downtime.	

SR-SAFE-SYS.02	
Type	Maintainability
Status	Proposed. Version 1.0. Phase 1.0 Priority: High Difficulty: High
Details	Created on 05-11-2015.
To be safe against human error, the system shall provide an event replay mechanism, leveraging permanent storage.	

5. Use Cases Catalogue

This section will list the use cases elicited from the project owner, which after analysis, originated the previous section, a listing of requirements.

5.1. Actors List

The described system will be modeled according to the following list of actors:

- **MedicineOne Server:** Represents the actual MedicineOne Server software, which in this case is the real time event producer.
- **Central Repository:** This is the main component of the RTHAV responsible for receiving and processing events and exposing a real time and historic data access interface.
- **Visualization Platform:** User facing part of the system responsible for producing visualizations and managing users.
- **Operator:** Operator of the RTHAV system.
- **User:** The end user that will access the Visualization Platform.

5.2. Use Cases

UC.1	
Use Case	Generate and send event
Description	An event is generated for each drug prescription, diagnostic test and diagnostic that arrives at the MedicineOne Server.
Actors	MedicineOne Server (primary) Central Repository
Assumptions	The event generation is started by the responsible component of the server.
Steps	1. Server receives a request to process one of the tasks that should create events; 2. The responsible module gathers the necessary information to generate the event; 3. Information is sent to the event generation module of the server; 4. The event is serialized to JSON; 5. REPEAT a. Persist even in queue; b. Establish secure connection to the Central repository; c. Send serialized event; 6. UNTIL operation is successful; 7. Remote event from persistent queue;
Non-Functional	**Reliability:** The system shall not lose events; **Performance:** Each event, when connection is successfully established should not take more than 5 seconds between generation and arriving at the Central Repository.
Issues	If the system takes too long to send the event, real-time performance will not be achieved and late events might cause problems.

UC.2	
Use Case	Compare prescriptions of a drug in the current week with the same period last year.
Description	The user wants to compare the prescriptions of a determined drug with the number of prescriptions on a homologous period during the previous year.
Actors	User (primary) Visualisation Platform
Assumptions	The system is running and the user is authenticated and has the necessary permissions.
Steps	1. User selects the prescription section 2. User selects the drugs of interest 3. User selects a time series of prescription volume. 4. User initiates the overlay addition. 5. User selects historic period of interest. 6. The visualisation system presents both graphs on the same plot.
Non-Functional	
Issues	

UC.3	
Use Case	Monitor prescriptions nearby
Description	A user is interested in monitoring the sales of a specific drug in the vicinity.
Actors	User (primary) Visualisation Platform
Assumptions	The system is running and the user is authenticated and has the necessary permissions.
Steps	1. User selects the prescription section 2. User selects the drugs of interest 3. User applies a geographical filter 4. The visualisation system will plot prescription count in real time
Non-Functional	**Latency:** The visualisation needs to be as low latency as possible. Events should be pushed to the client.
Issues	

UC.4	
Use Case	Monitor pathology dissemination.
Description	A user wants to visualise in real time the diagnoses of a determined pathology.
Actors	User (primary) Visualisation Platform
Assumptions	The system is running and the user is authenticated and has the necessary permissions.
Steps	1. User selects the pathology section 2. User selects pathology of interest 3. User selects geographical visualization 4. The system presents the number of occurrences on each geographical area.
Non-Functional	**Latency:** The visualisation needs to be as low latency as possible. Events should be pushed to the client.
Issues	

UC.5	
Use Case	Visualize market share of drugs in a category.
Description	A user wants to know what the most successful drugs are in a determined category.
Actors	User (primary) Visualisation Platform
Assumptions	The system is running and the user is authenticated and has the necessary permissions.
Steps	1. User selects the prescription 2. User selects category of interest 3. User selects market share visualization 4. The system presents top N drugs and their respective market share.
Non-Functional	
Issues	

6. External Interfaces Specification

EXT-INTERF.01	
Description	Event ingestion interface.
Frequency	30 times a second
Sizing	Must be able to ingest 30 events a second; Each even will weight around 2Kbytes.
Format	JSON
Security	TLS

EXT-INTERF.02	
Description	Time Series / Metrics Query
Frequency	Up to once a second for each end user.
Timing	---
Sizing	Upper estimate of 100kbs per user.
Format	JSON
Security	HTTPS

EXT-INTERF.03	
Description	Historic Data / Aggregate Query
Frequency	Up to once every 10 seconds for each user.
Timing	Should load in under 10 seconds.
Sizing	Upper estimate of 100kbps per user.
Format	JSON
Security	HTTPS

EXT-INTERF.04	
Description	Web Front End
Frequency	Up to once every 10 seconds for each user.
Timing	Should load the basic front end (excluding data from other interfaces) under 1 second.
Sizing	Upper estimate of 100kbps per user.
Format	HTTP
Security	HTTPS for authentication.

7. Pending issues

1. **Technological selection** would require a better understanding of the domain than is possible to have until prototyping and testing is done. To deal with the issue, the life cycle was adjusted to be forgiving and allow change to both chosen technologies and architectures.

2. **The project might be too large** for the time frame and resources allocated to it. This issue cannot be easily dealt with, therefore, if it proves to be true, the project will only go as far as possible, priority being assigned to requirements as described in the respective sections.

Instituto Pedro Nunes
Rua Pedro Nunes
Quinta da Nora
3030-199 Coimbra

Tel (+351) 239 103 500
Fax (+351) 239 103 501
E-mail geral@medicineone.net

www.medicineone.net

Bibliography

[1] "Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (Dec. 2010), pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835.

[2] K. G. Shin and P. Ramanathan. "Real-time computing: a new discipline of computer science and engineering". In: *Proceedings of the IEEE* 82.1 (Jan. 1994), pp. 6–24. ISSN: 0018-9219. DOI: 10.1109/5.259423.

[3] *Mesosphere DC/OS*. URL: https://mesosphere.com/ (visited on 06/15/2016).

[4] Eric Tschetter. *Real Real-Time. For Realz.* 2013. URL: http://druid.io/blog/2013/05/10/real-time-for-real.html (visited on 11/14/2015).

[5] *What is Scrum?* URL: https://www.scrum.org/resources/what-is-scrum (visited on 12/14/2015).

[6] *Scrum Process*. 2009. URL: https://upload.wikimedia.org/wikipedia/commons/5/58/Scrum_process.svg (visited on 11/12/2015).

[7] *What is Scrum? An Agile Framework for Completing Complex Projects.* URL: https://www.scrumalliance.org/why-scrum (visited on 12/14/2015).

[8] Dan Radigan. *A brief introduction to kanban.* URL: https://www.atlassian.com/agile/kanban/ (visited on 11/14/2015).

[9] David Peterson. *What is Kanban?* URL: http://kanbanblog.com/explained/ (visited on 11/14/2015).

[10] *How To Do 3-Point Estimating*. URL: http://4pm.com/3-point-estimating-2/ (visited on 04/12/2016).

[11] *Docker Swarm*. URL: https://docs.docker.com/swarm/ (visited on 12/11/2015).

[12] *Kubernetes*. URL: http://kubernetes.io/ (visited on 03/28/2016).

[13] *Practice Fusion*. URL: http://www.practicefusion.com (visited on 10/29/2015).

[14] *Insight Practice Fusion*. URL: https://insight.practicefusion.com/ (visited on 10/29/2015).

[15] *Insight Practice Fusion*. URL: http://www.practicefusion.com/pharma/ (visited on 12/18/2015).

[16] Surajit Chaudhuri and Umeshwar Dayal. "An Overview of Data Warehousing and OLAP Technology". In: vol. 26. 1. New York, NY, USA: ACM, Mar. 1997, pp. 65–74. DOI: 10.1145/248603.248616. URL: http://doi.acm.org/10.1145/248603.248616.

[17] Nickerson Ferreira and Pedro Furtado. "Real-time Data Warehouse: A Solution and Evaluation". In: *Int. J. Bus. Intell. Data Min.* 8.3 (Feb. 2013), pp. 244–263. ISSN: 1743-8195. DOI: 10.1504/IJBIDM.2013.059046. URL: http://dx.doi.org/10.1504/IJBIDM.2013.059046.

[18] *Apache Hadoop*. URL: https://hadoop.apache.org/ (visited on 11/19/2015).

[19] *InfluxDB*. URL: https://influxdata.com (visited on 11/18/2015).

[20] *Druid*. URL: http://druid.io/ (visited on 11/18/2015).

[21] *Apache HBase*. URL: https://hbase.apache.org/ (visited on 11/19/2015).

[22] *Apache Storm*. URL: http://storm.apache.org/ (visited on 11/18/2015).

[23] Raúl Rojas and Ulf Hashagen, eds. *The First Computers: History and Architectures*. Cambridge, MA, USA: MIT Press, 2000. ISBN: 0-262-18197-5.

[24] *Redis*. URL: http://redis.io/ (visited on 11/20/2015).

[25] Nathan Bijnens Michael Hausenblas. *Lambda Architecture*. 2014. URL: http://lambda-architecture.net/ (visited on 11/18/2015).

[26] *Netflix Suro*. URL: http://techblog.netflix.com/2013/12/announcing-suro-backbone-of-netflixs.html (visited on 01/18/2016).

[27] *Apache Kafka*. URL: http://kafka.apache.org/ (visited on 11/19/2015).

[28] *Apache Cassandra*. URL: http://cassandra.apache.org/ (visited on 03/08/2016).

[29] *Apache Cassandra Scalability Benchmark, Netflix*. URL: http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html (visited on 03/22/2016).

[30] *Apache Spark*. URL: http://spark.apache.org/ (visited on 11/18/2015).

[31] Matei Zaharia et al. "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=2228298.2228301.

[32] Linda Northrop. *The Importance of Software Architecture*. 2003. URL: http://csse.usc.edu/GSAW/gsaw2003/s13/northrop.pdf (visited on 11/14/2015).

[33] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007. ISBN: 0132344823.

[34] Simon Brown. *The Art of Visualising Software Architecture. Communicating software architecture with sketches, diagrams and the C4 model.* Leanpub, Jan. 2016. URL: https://leanpub.com/visualising-software-architecture.

[35] *HDFS*. URL: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html (visited on 11/19/2015).

[36] *Docker*. URL: https://www.docker.com/ (visited on 12/11/2015).

[37] *Apache Mesos*. URL: http://mesos.apache.org/ (visited on 03/28/2016).

[38] *Ansible*. URL: https://www.ansible.com/ (visited on 04/11/2016).

[39] *Mesosphere Marathon*. URL: https://mesosphere.github.io/marathon/ (visited on 03/29/2016).

[40] *RabbitMQ*. URL: https://www.rabbitmq.com/ (visited on 11/19/2015).

[41] *Apache Aurora*. URL: http://aurora.apache.org/ (visited on 03/29/2016).

[42] *Apache Zookeeper*. URL: https://zookeeper.apache.org/ (visited on 11/19/2015).

[43] *Kafka-Manager*. URL: https://github.com/yahoo/kafka-manager (visited on 08/03/2016).

[44] *Kafka benchmark at Linkedin*. URL: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines (visited on 03/08/2016).

[45] *Druid Benchmark*. URL: http://druid.io/blog/2014/03/17/benchmarking-druid.html (visited on 03/08/2016).

[46] *Druid Benchmark at Linkedin*. URL: https://www.linkedin.com/pulse/20140909054748-5574162-true-performance-measure-of-realtime-big-data-analytics-platforms (visited on 03/08/2016).

[47] *MLlib*. URL: http://spark.apache.org/mllib/ (visited on 06/15/2016).