Bruno Manuel Paias Firmino

Bachelor of Science in Computer Science and Informatics Engineering

Smart Monetization - Telecom Revenue Management beyond the traditional invoice

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser:	José Rodrigues Carvalho, Revenue Management, Celfocus
Co-adviser:	Matthias Knorr, Prof. Auxiliar, Universidade Nova de Lisboa

Examination Committee

Chairperson:	Dr. Pedro Medeiros, Prof. Associado, Universidade Nova de Lisboa
Raporteur:	Dr. Paulo Lopes, Prof. Auxiliar, Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2019

Smart Monetization - Telecom Revenue Management beyond the traditional invoice

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Do. Or do not. There is no try.

Acknowledgements

I would like to start by thanking my advisors José Carvalho and Matthias Knorr for all the help and support provided during the writing of this thesis. I would also like to thank FCT-NOVA as a whole and more specifically the Department of Informatics, along with all the professors and colleagues who helped me grow academically, professionally and as a person. My thanks are extended to my colleagues at Celfocus for welcoming me and helping me get integrated in an environment that was new to me, as well as for all the entertaining coffee breaks, work meetings, general advice and valuable teachings. A special mention goes to Vasco Pereira for his guidance. On a more personal note I would like to thank my close friends and family for all the love and support over all these years, my academic family, and my mentor Tiago Santos in particular. A special thank you goes to my mom and dad for always loving me, supporting me and allowing me to pursue my studies, as well as my grandparents, some of whom are not with us anymore but I'm sure would be very proud.

Abstract

Nowadays, there is a fast and unpredictable technological evolution, with new systems constantly emerging on the market, with the capability of being monetized. However, these systems are not always fully and flexibly explored. Many hardware distributors are selling products without a clear view on sustainable business models for them, leaving these as an afterthought. Communications Service Providers are suddenly under pressure to modernize and expande their business models as to regain the ground claimed by Over-the-top service providers, who make use of existing infrastructures to provide their own services, which naturally may lead to substantial revenue loss from the actual infrastructure owners. The Smart Monetization project aims to explore this paradigm, with the design and implementation of a reusable asset, making use of Big Data and Analytics tools that can ingest and process usage and billing data from customers, detecting event patterns and correlations that can be monetized, leading to improved and new service experiences and ensuring, as well, greater transparency on the process of billing and charging of these services.

Keywords: Smart Monetization; Telecommunications; Revenue Management; Billing; Charging; Data processing; Big Data; Fast Data; Analytics

Resumo

No mundo atual existe uma acelerada e imprevisível evolução tecnológica, em que todos os dias são desenvolvidos novos sistemas com a capacidade de serem monetizados. No entanto, estes nem sempre são explorados ao máximo ou de uma forma flexível. Imensos distribuidores de hardware efetuam a comercialização de produtos sem dispor de uma visão concreta de modelos de negócio sustentáveis para os mesmos, deixando-os para segundo plano. Os *Communications Service Providers* estão sob uma súbita pressão para modernizar e expandir os seus modelos de negócio de modo a recuperar terreno reivindicado por provedores de serviços *Over-the-top*, que usufruem das infraestruturas já existentes para fornecer os seus próprios serviços, o que leva naturalmente a perdas substanciais de receita da parte dos atuais proprietários das infraestruturas. O projeto *Smart Monetization* visa explorar estes paradigmas, partindo do desenho e eventual implementação de um *asset* reutilizável, com recurso a ferramentas de *Big Data* e Analytics, que possa ingerir e processar dados de utilização e faturação dos consumidores, analisando padrões e correlação de eventos que possam ser monetizados, levando assim a melhores e novas experiências de utilização de serviços, garantindo também uma maior transparência no processo de faturação e cobrança dos mesmos.

Palavras-chave: *Smart Monetization*; Telecomunicações; *Revenue Management*; Faturação; Cobrança; Processamento de dados; *Big Data*; *Fast Data*; *Analytics*

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Overview

Celfocus[1] was created in 2000 as a joint venture between Novabase and Vodafone Portugal and provides solutions for the telecommunications market, working within domains such as Customer Relationship Management, Enterprise Application Integration, Business Intelligence and Intelligent Network Applications and Services.

The motivations for this project arise from the great potential that comes from Big Data and Analytics tools and mindsets that hasn't been fully explored yet. There is a large number of devices connected to the internet that produce vast quantities of all kinds of data, which ultimately may be gathered and stored but aren't availed, which is also not helped by the lack of adequate tools to harness and make use of this kind of data. This was revealed by an internal study Celfocus did on the subject, concluding a need to develop an in-house tool capable of answering these requirements.
Here is where **Smart Monetization** enters the picture. An asset capable of ingesting large quantities of data and analyse them in order to gather knowledge and have the potential to monetize it.
Over the course of this thesis, State of the Art tools were studied, architectural design decisions were made and a Proof of Concept was developed to show the viability and potential of this tool.
In order to provide a clearer understanding of the problem, it is useful to first succinctly introduce a few prevalent topics related to the subject, such as the concepts of Communications Service Provider, Over The Top services, Internet of Things and Big Data Analytics.

[1]https://www.celfocus.com/

1.1.1 Communications Service Provider

The use cases presented and experimented with, over the course of this work, were mostly focused with a Communications Service Provider (or CSP for short) perspective in mind, given its enormous potential given their reach and available infrastructure. A CSP is a service provider that transports information electronically - a telecommunications service provider, for instance.
CSP have traditionally offered a selection of bundled services such as triple-play which packages landline, television and broadband services altogether [Mar15]. After these types of bundles came quad-play, which aggregated mobile services to the aforementioned package, resulting in an ever increasing larger selection and bundling of different types of available services.
These bundles are more attractive to the consumer due to the added ease of subscription and cleaner billing process. More recently, due to the constant evolution and development of existing and brand new technologies, CSP find themselves encouraged to offer new bundles that include new services such as Internet of Things services and other innovative and disruptive services.

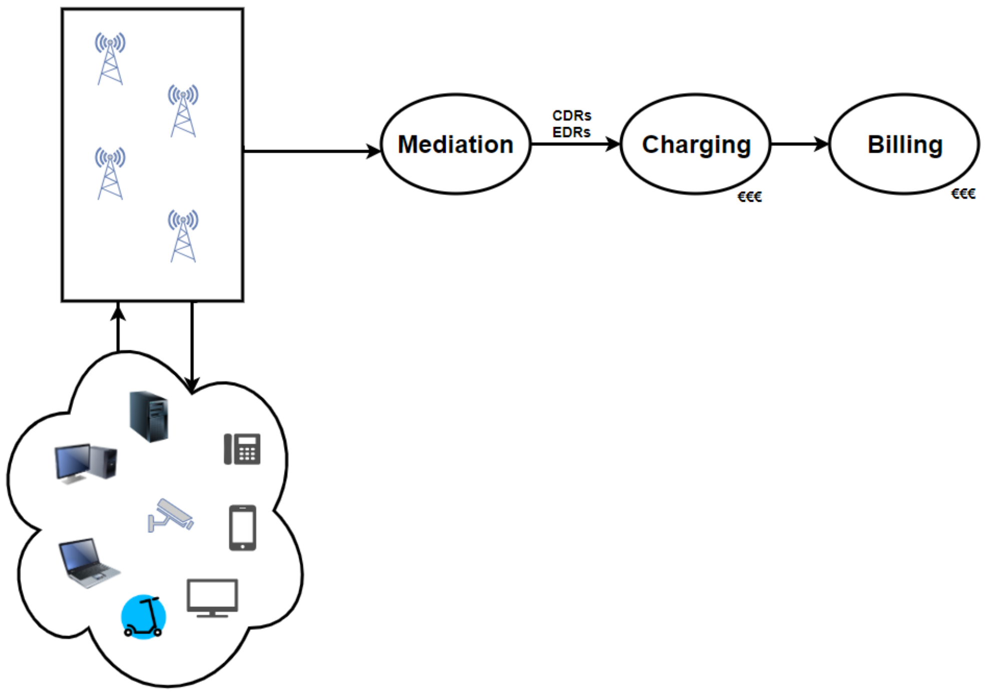


Figure 1.1: Traditional business model of a CSP

The traditional CSP business model is comprised of multiple fields but there are three layers most closely coupled and important when we talk about Big Data and Analytics and its monetization potential (Figure 1.1).
Firstly, there is the network layer and all its supporting structure such as antennas, network cables and data storage.

Secondly there is the mediation layer where customer data records are treated. These data records exist traditionally in the form of Call Data Records (CDR). These records are comprised of phone service use details and are then associated with predetermined monetary values for billing purposes. This is called the rating process. Eventually technology and the industry kept innovating and more general data records had to be created, leading to the creation of Extended Data Records or Event Data Records (EDR), which store more dynamic types of event information, not just phone records.
Lastly there is the billing (prepaid) and charging (postpaid) layer, destined to pass on this information back to the consumer and bill or charge them adequately.

1.1.2 Over The Top Services

Traditionally, the main revenue streams for telecommunications operators have been voice and messaging services, with data coming in at a far third.
While these operators had been quick to react to previous game-changing developments such as the internet explosion and the emergence of cellular mobile communications, they seem to have been lacking in the face of newer challenges such as Over The Top (OTT for short) service providers.
These service providers deliver audio, video and other media over the internet and bypass the traditional operator's network. Though they make use of the telecommunications operators' network and infrastructure, they do not contribute directly and are, in fact, beginning to pose a credible and measurable threat to their revenue. Enabled by technology advances such as smartphones, fast IP networks, open source platforms and a wide variety of innovative applications and services offered, OTT service providers are seeing an ever-increasing adoption rate. [Suj+15]

1.1.3 Internet of Things

The term Internet of Things (IoT for short) has been around for several years. When it was introduced it stood for the vision of a world where all physical objects are tagged with an RFID (Radio-frequency identification) transponder with a globally unique ID. RFID easily allows tracking the objects, and the EPC (Electronic Product Code) serves as a link to data which can be queried over the Internet about each individual object.
Since then its meaning has expanded. Using sensors or sensor networks, additional information about the objects or the environment that they are in can be recorded as well. Software embedded in the objects enables data processing directly on the item, and in combination with actuators, local control loops can be implemented. The Internet of Things is a key part of the internet of the future and many new opportunities can be foreseen for citizens, businesses and society as a whole. [Hal+08]

It is also pertinent to mention the existence of the concept of Internet of Everything

(IoE for short). While IoT is mostly focused on the physical objects and actuators communicating with each other, which allows these to work and perform tasks automatically with no human input, simply reacting to the environment surrounding them, IoE is what brings in network intelligence to bind all these concepts into a cohesive system. [Van16]

1.1.4 Big Data Analytics

Big Data is a loosely defined term used to describe large and complex data sets that are very difficult to work with using traditional statistical software [Sni+12].
Analytics, on the other hand, involves relatively conventional methods of exploring links and statistical relationships between data and respective output of results and information [Ass16].
Big Data and Analytics are tightly related concepts and are often coupled together. They both seek to glean knowledge from data and translate that into smart business decisions. However, there are a few important differences. Big Data implies the existence of three key characteristics, usually called the Three Vs of Big Data [MB12]:

- **Volume**: the vast quantity of data to process;
- **Velocity**: the fast rate at which new data is created, stored and processed;
- **Variety**: the broad heterogeneity of data and its sources.

More recently there have been a few added Vs to more accurately describe the concept of Big Data, two of them being [Rec18]:

- **Veracity**: the accuracy and integrity of data;
- **Value**: the significance and impact the data has on business goals.

These make up the so-called Five Vs of Big Data but there are other ones as well, such as Viscosity, Variability, Volatility, Viability and Validity [Kha+18].

In essence, Big Data Analytics focuses therefore on the collection of useful information from vast and diverse quantities of data from different data sources to make informed decisions, in order to obtain business advantage and discovery of new commercial opportunities to monetize services of interest to the consumer.
This area reveals a particular interest and potential in the previously mentioned IoT paradigm. In the words of Kevin Ashton, "If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best."[Ash09]. This potential, which will be further explored in later sections, comes from the fact that these devices and systems generate large amounts of

precious data.

It is also pertinent in the context of this work to mention the concept of Fast Data, which gives priority to the low latency necessary for Big Data to be useful in a business context, due to the need of real or near real-time data processing and information extraction.

1.2 Problem Description

Due to the increasing availability of Over The Top (OTT) services, as well as the explosive growth of the number of Internet of Things (IoT) devices connected to the global network being utilized for the most diverse use cases, traditional Communications Service Providers (CSP) find themselves struggling to regain lost ground, innovate and adapt their services.
Reports suggest that CSP have lost around $386 billion in revenue in the last six years due to the increasing consumer preference toward OTT message and voice (VoIP) services. It is predicted that spending on traditional telecommunication services will reduce by around 36% in the coming decade. [Ven18]
Currently, the processing of customer and service usage data is an increasingly important factor to business success. The analysis of this kind of data allowed the discovery of usage patterns and preferences that can be utilized to provide a better, more personalized service to every different user.
OTT service providers such as Netflix[2] or Uber[3] use analytics tools to better serve their consumers with the content they are most likely to want, which leads to customer satisfaction and retention. [Put18] [Dan18]

CSPs, more than any other service providers, have the ownership of the existing infrastructure with the capability of supporting this kind of consumer behaviour, whereby it is crucial to take advantage of such lead in order to provide the best possible service.
The emergence of Smart Devices, on the other hand, makes it possible to identify predetermined events without consumer intervention, such as an irrigation system that, with the use of sensors and actuators, detects the need to water plants automatically, acting accordingly without the need for user intervention.
However, these kinds of systems are not being fully availed, its settings requiring manual configuration by the user or, even if it is controlled externally, for instance by a CSP, there are limitations in regard to monetization of events, e.g., they are limited to a traditional static subscription model (e.g. charging 20€ monthly to water the grass, once a day, at a fixed hour), which does not allow for the level of flexibility and efficiency.

According to IBM, nearly 90% of the data collected by IoT devices will remain unused

[2] https://www.netflix.com/
[3] https://www.uber.com/

[Was16]. This reveals the existence of a tremendous waste of a vast source of information that could be useful to measure consumer behaviour and ultimately lead to a refinement of offered services, with greater individual personalization, by also allowing the monetization of increasingly sought-after non-traditional innovative and disruptive services.

1.3 Challenges and Requirements

The Smart Monetization project emerges from a study conducted by Celfocus, more specifically the department of Revenue Management, which detected a lack of platforms with out-of-the-box capabilities that allow for the monetization of events created by current and future Internet of Things (IoT) services, in a distinct and innovative manner, giving an answer to the defined requirements. The existing platforms are relatively prepared to fulfill the challenges of Over The Top (OTT) services, but not those of a Communications Service Provider (CSP). The functionalities available are restricted to subscription billing and rudimentary predetermined event monetization.

Smart Monetization has the primary goal of taking advantage of Big/Fast Data and Analytics tools and techniques, allowing real or near real-time processing of data, in order to quickly provide information to the final user. This solution, which will be integrated with legacy Business Support Systems, specifically Revenue Management domains and processes such as mediation, rating, billing and invoicing, must not only be able to monetize events and/or event correlations, but also generate ways to transparently communicate, to the final consumer and adjacent accounting processes, exactly which components and subsequent values or costs are correlated to such events.

In terms of requirements this solution must take in consideration the following criteria:

- ***Performance*** - overall performance and throughput of the system;
- **Efficiency** - adequate and balanced results to system complexity ratio;
- **Scalability** - capacity to expand system resources to evenly process an increasing amount of work;
- **Security and compliance** - cyber attacks, GDPR, ISO, among others.

Some use cases this solution must support are, for instance:

- Dynamically rank and charge Internet of Things (IoT) events by the observation of patterns and event correlation;
- Distinctly charge users that move to a certain space such as concert venues;

- Define campaigns based on data usage during a period of time and a set place;
- Rationally charge the demands around Smart Cities (e.g. surveillance cameras video streaming, garbage collection sensors, automatic irrigation sensors, etc.);
- Direct costs to private entities when events are held in their venues.

1.4 Expected Contribution

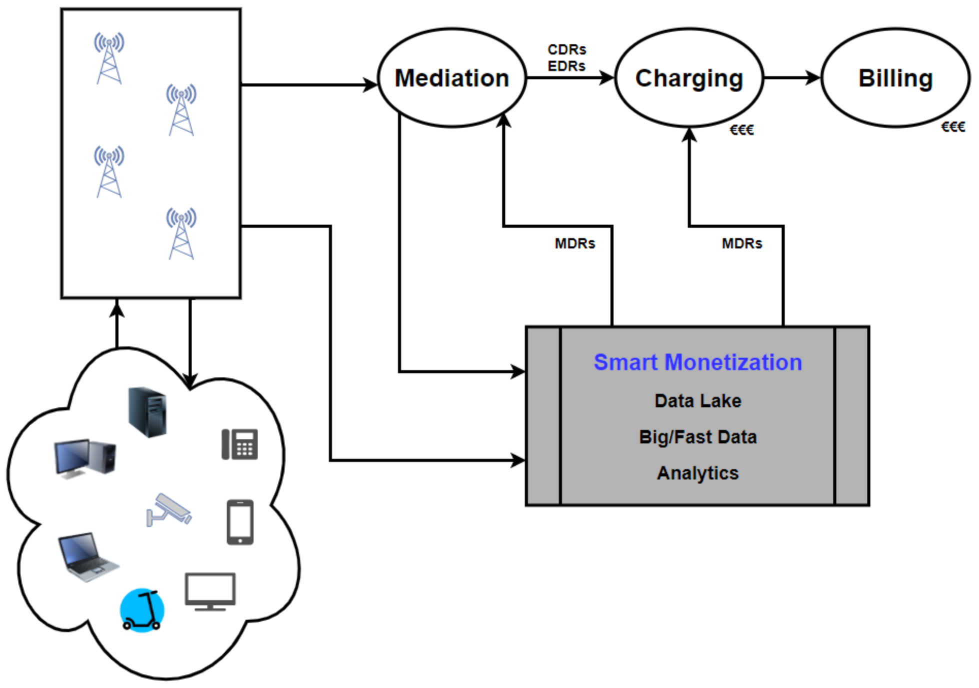


Figure 1.2: CSP business model with the introduction of Smart Monetization

The goal of this dissertation is the exploration of the previously described problem starting with the technical and architectural design of a reusable asset, using Big Data and Analytics tools and techniques, as well as the development of a Proof of Concept (PoC). The final solution aims to be homogeneously integrated in the existing system, as shown in Figure 1.2.
Adequate tools were studied and evaluated, followed by the implementation of a PoC in order to measure the feasibility of this solution, asserting as well the potential of Big Data Analytics from a business point of view. An investigation was also conducted to assess future work to add to this solution in order to expand its capabilities.
While the potential for this asset is large, the initial expected objectives for the PoC, in order to prove its viability, were to ingest data from a source, store such data in a database and then analyze it, producing adequate results that may be forwarded to other departments in order to be monetized.
In the course of this dissertation multiple contributions were made, both on an theoretic

level and a practical level. In short, a technical and architectural design was conducted, followed by the development of a Proof of Concept, using non productive usage data, representative of reality on a business context.
On a theoretical level, existing solutions were explored, and a study of the state of the art of the tools most adequate for each layer of this type of project was conducted. After the conclusion of the project, a study was also conducted to measure the effectiveness of this particular solution, which proves useful to substantiate the potential of the Big Data Analytics area on a business level.
On a practical level, a Proof of Concept resulted from this dissertation, whose purpose was the ingestion, analysis and storage of select data, which brings tangible benefits both to the service provider by allowing a larger scope of new monetization capabilities, and to the client who ultimately may receive a more attractive and innovative service usage experience and more new, innovative and disruptive services.

1.5 Document Structure

This document is structured in four major sections:

- Chapter 1 starts with an initial overview of this dissertation (Section 1.1) by introducing a brief approach to a few key concepts. Section 1.2 describes the problem that this work aimed to address followed by Section 1.3 that presents a few important challenges and requirements that were taken in consideration. This chapter ends with Section 1.4 which is dedicated to setting the expected contributions to be made over the course of this work;

- Chapter 2 describes the groundwork for the project. Related work is explored (Section 2.1), the high level architecture design is laid out (Section 2.2) and the tools most adequate to be used are compared and evaluated (Section 2.3);

- Chapter 3 is dedicated to the actual Proof of Concept development. There is a description of the development environment and installation of tools (Section 3.1 and Section 3.2, respectively), a description of Smart Meeting Room, the chosen use case to be used as an example for this Proof of Concept (Section 3.3), a section about what a typical pipeline looks like to get up and running (Section 3.4) and finally the testing and analysis of results from this implementation (Section 3.5);

- Lastly, overall conclusions about this project and dissertation are discussed, as well as an outline of potential future work (Chapter 4).

Chapter 2

Smart Monetization

2.1 Related Work

This project came to light from a 2018 study conducted by Celfocus where an investigation and evaluation of flexible state of the art charging and billing solutions was carried out, where it was then concluded that the existing platforms available on the market do not match the necessary proposed requirements.
The several existing tools that were analyzed in this study, among them Oracle Monetization Cloud - OMC, goTransverse TRACT and Zuora Billing, revealed either a lack of complexity, volume or agility. As such, it was concluded the necessity of the development of an in-house platform capable of answering these requirements with resort to low latency ingestion and analysis of vast quantities of heterogeneous data (Big/Fast Data).

2.2 High Level Architecture

This project's high level architecture is divided in three layers (Figure 2.1):

- **Data Ingestion**;
- **Data Storage**;
- **Data Analysis**.

Each of these layers uses one (or more) tool(s) that ultimately communicate with one another to form a Big Data Analytics pipeline.
The data ingestion layer is responsible for aggregating data from any source in order to reroute it to the appropriate storage and analytics layers. Eventually this layer can also be used to do light preprocessing work in order to filter or normalize some of the data, if

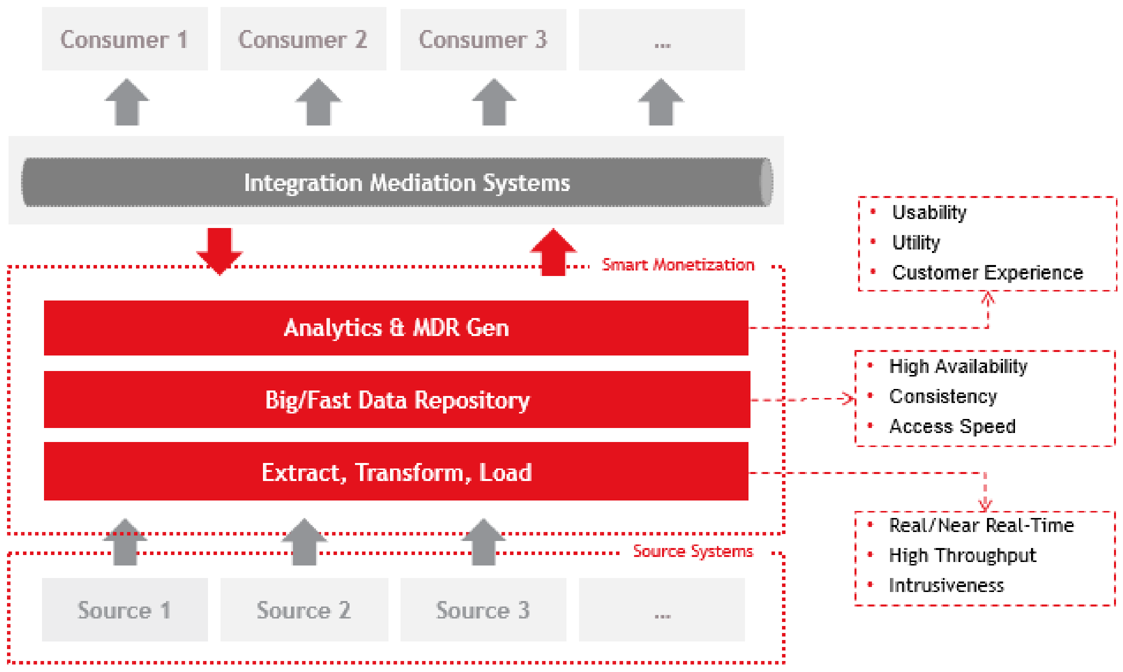


Figure 2.1: Smart Monetization - High level architecture

justified.
The data storage layer, effectively a Data Lake, stores all ingested and processed data in the system. A Data Lake, as opposed to a Data Warehouse, is a flexible data repository used to store vast quantities of raw structured, unstructured or semi-structured data [Mak18].
Lastly, the analytics layer processes the ingested data and, with resort to pattern observation and event correlation, will generate Monetizable Data Records that will eventually be able to be transparently visualized and perceived by the final consumer.

2.3 Tools Considered

The development of this project involves the use of multiple Big Data and Analytics tools. This area has several available tools, as can be seen in Figure 2.2 and as such, a state of the art analysis was necessary to evaluate the most adequate tools needed for each layer of this project in particular.
A few of these tools were first chosen in a preliminary evaluation to be further analyzed. Each evaluation had specific requirements and technical features kept in mind. There was also a certain preference, in the case of a tie, toward the more established tool, with the larger presence in the market and community, and with which Celfocus possesses more experience working with.

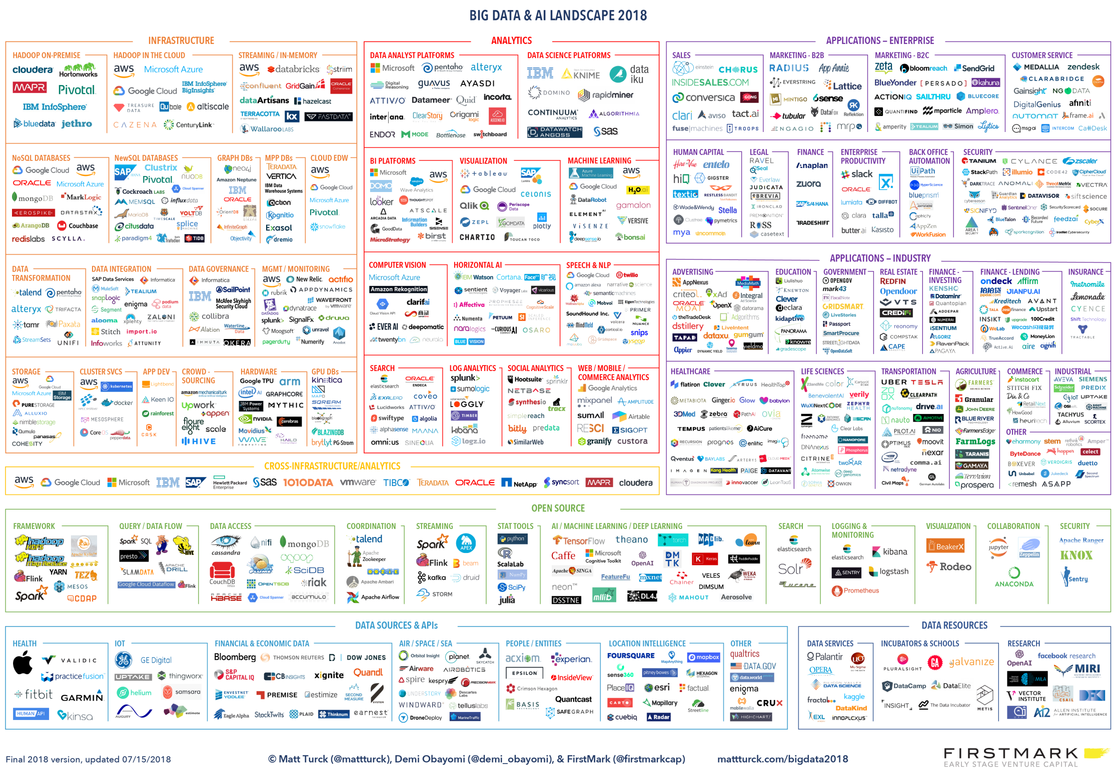

BIG DATA & AI LANDSCAPE 2018
INFRASTRUCTURE
ANALYTICS
APPLICATIONS – ENTERPRISE
APPLICATIONS – INDUSTRY
CROSS-INFRASTRUCTURE/ANALYTICS
OPEN SOURCE
DATA SOURCES & APIs
DATA RESOURCES
Final 2018 version, updated 07/15/2018
© Matt Turck (@mattturck), Demi Obayomi (@demi_obayomi), & FirstMark (@firstmarkcap)
mattturck.com/bigdata2018
FIRSTMARK EARLY STAGE VENTURE CAPITAL

Figure 2.2: Big Data tools used in 2018[1]

2.3.1 Data Ingestion

For the first layer, the data ingestion layer, responsible for capturing data from sources, there are several open source tools available with multiple implementation protocols.

2.3.1.1 Ingestion Tools

Figure 2.3: Data ingestion tools considered

An initial shortlist (Figure 2.3) was picked after a superficial analysis, which is made

[1]Source: `http://mattturck.com/matt_turck_firstmark_big_data_landscape_2018_final/`

up of the following tools: Apache Kafka[2], Apache Flume[3], Apache Pulsar[4], Apache ActiveMQ[5] and Pivotal Software's RabbitMQ[6] (Table 2.1).

Several characteristics were taken into consideration during the evaluation and comparison of these tools, such as:

- **Scalability** - capacity to serve an increasing load of data;
- **Data consistency** - guarantee of transit and storage of data without anomalies or inconsistencies;
- **Fault tolerance and recovery** - capacity to prevent and recover from system faults (hardware or network related);
- **Security and compliance** - cyberattacks, GDPR, ISO, among others;
- **Synchronization method/protocol** - data transfer model type;
- **Data processing capability** - capacity to process data on top of ingesting it;
- **Throughput** - data transfer rate on a set time frame;
- **Maturity and available documentation** - amount of documentation, community and professional support availability and proven industry use;
- **Licensing** - licensing type for the use and commercialization of the respective tool.

Table 2.1: Overview of the ingestion tools considered.

	Kafka	**Flume**	**ActiveMQ**	**RabbitMQ**	**Pulsar**
Protocol	Pub/Sub	Passive/Active	Queue	Queue	Hybrid
Data processing	Yes	No	No	No	Yes
Initial release	2011	2012	2004	2007	2016
Licensing	Apache	Apache	Apache	Mozilla	Apache

Each of these tools offer different levels of scalability, data consistency, fault tolerance, security and compliance.

Kafka is a high-capacity, highly scalable distributed message broker originally developed by LinkedIn. It works on a Publisher/Subscriber (or Producer/Consumer) model in which producers send messages to Kafka brokers. These messages are associated to Kafka

[2]https://kafka.apache.org/
[3]https://flume.apache.org/
[4]https://pulsar.apache.org/
[5]http://activemq.apache.org/
[6]https://www.rabbitmq.com/

topics and will be received by consumers subscribed to the latter. A group of brokers is called a cluster and these are maintained by Apache Zookeeper (as of August 2019 this dependency is being addressed, where a discussion is being held looking to replace Zookeeper with a self-managed metadata quorom [McC19]).
One of Kafka's strengths is message persistence, where every message can be stored in disk for a configurable period of time [Nan15]. While performance may depend on several design choices and other system factors, this tool offers a very reasonable throughput of possibly more than 100000 messages per second [Hum17] with the capability of reaching much higher values, without the need for state of the art hardware [Kre14]. Furthermore, Kafka also offers a data processing library, Kafka Streams, which may be useful for an optional preprocessing or filtering of data before being transferred to the storage layer.

Flume is a distributed system that allows the collection, aggregation and transfer of data to a repository such as Hadoop. This tool features decent performance, with Kafka being slightly ahead on real-time data streaming [Cas+16]. While Kafka is a general purpose tool, Flume is specific purpose, great for log aggregation for instance. Unlike Kafka, it does not support message replication to other nodes, which may lead to easier data loss [Sic17].

Both ActiveMQ and RabbitMQ are traditional message brokers that work based on message queueing, supporting protocols such as Advanced Message Queuing Protocol (AMQP), Simple (or Streaming) Text Oriented Message Protocol (STOMP) and Message Queuing Telemetry Transport (MQTT). Being older tools, there is a greater feature and performance limitation when compared to more recent options [Tru14]. Being traditional message brokers, these tools do not offer data processing or analytics features like Kafka, for instance, with its Kafka Streams library, previously mentioned.

Lastly, Pulsar is the most recent out of the tools considered. This tool unifies high performance streaming (which Kafka pursues) and flexible traditional queueing (which RabbitMQ pursues) [Hal18b]. This tool shows better performance compared to Kafka [Str18] but, due to its relative recency, there is a greater lack of documentation compared to more mature tools.

After comparing these five tools, Kafka has been chosen as the most adequate tool for the project due to its flexibility and performance as well as the abundance of documentation and its maturity, being used in production in several industries [RK16]. Celfocus also already possesses experience with the use of this tool in other projects. Even though Pulsar offers performance gains, this trade-off was deemed not justifiable for what we want to achieve and demonstrate, but it could however be a potential tool to use in the future.

2.3.2 Data Storage

For the data storage layer, where we store all the data ingested for logging and analysis purposes, we effectively need a database which will serve as a Data Lake. Before going into the tool selection process, it is important to first approach a few relevant concepts and aspects that were taken in consideration for a project of this scope, handling vast quantities of possibly unstructured data.

2.3.2.1 SQL vs. NoSQL

This project revolves around Big/Fast Data and, as mentioned in the introductory section of this document, this is a factor that needs certain requirements to be taken into consideration due to the need to store and process, with low latency, vast quantities of heterogeneous data.
Relational databases, or traditional databases that use SQL (Structured Query Language), have been commonly rejected in projects of this kind in favor of alternative non-relational databases. These are also called NoSQL databases, which is debatably defined as either "No SQL"and/or "Not Only SQL"[Fow12].
The relational model is a favorable choice to work with well-defined, structured and sensitive data sets. It is also pertinent to highlight that databases of this kind are usually ACID compliant, required for transactions:

- **Atomicity** - all changes to data are performed as one, meaning they either are all successfully executed or none are;
- **Consistency** - data is in a consistent state both at the start and at the end of each transaction;
- **Isolation** - the intermediate state of a transaction is not visible to other transactions;
- **Durability** - after a successful transaction, data changes are persistent even in the case of system failure.

This type of integrity and anomaly-free guarantee is significantly important in financial systems, for instance, where data errors can not be tolerated.
However, this model is not particularly flexible, especially at a large scale, where the non-relational data model excels. This model was designed from the ground up to be scalable, flexible and have decent performance at the same time. NoSQL databases provide storage and access capacity to large quantities of possibly non-structured data sets, without the need to previously define the data types to be used [Ver18].
Commonly used relational databases (Oracle Exadata being a notable exception) were designed to run on a single machine, which hinders scalability, because instead of horizontally scaling, that is, adding more machines (i.e., nodes to a cluster), one must increase the processing capacity of a single node (vertical scaling), which may be more expensive (e.g.,

the node reaches its expansion limit and a larger one must be bought) and requires extra steps to provide fault tolerance, which is by default designed into modern, non-relational distributed databases (as a result from data being replicated across several machines) [SD12]. This kind of databases are called distributed databases, prevalent notion in non-relational databases.
With this in mind, for the scope of the Proof of Concept to be designed in this project, non-relational databases were considered to be the most adequate option.

2.3.2.2 CAP Theorem

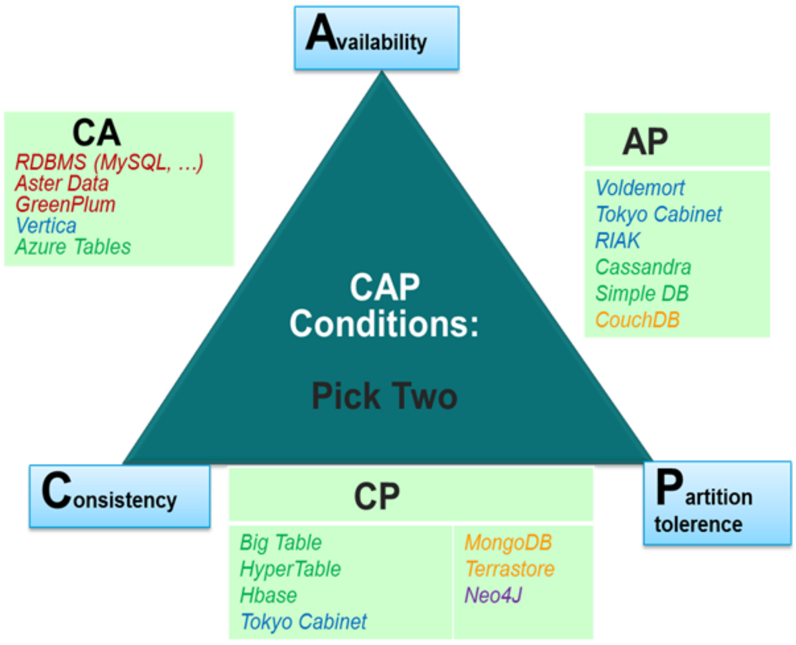


Figure 2.4: CAP theorem[7]

In the previous section (Section 2.3.2.1) an important relational database property was mentioned: ACID compliance. With non-relational databases the paradigm is somewhat different being that this was not originally a prioritized property due to the nature of its foundation.
This is due to a dilemma that goes by the name of CAP theorem or Brewer's theorem (Figure 2.4), that implies the following: of the three desirable distributed database properties, these being network partition tolerance, data consistency and system availability, we can only obtain two simultaneously [Sim12].
More precisely, this implies that in the presence of network partition tolerance, that is, the capacity of the system to keep processing data and serving requests even if any of its subsystems fails, something that is core to distributed data models, there must exist a trade-off between data consistency and system availability.
If a larger level of data consistency is required, the system may not always be available,

due to having to verify that every node in the system contains consistent data before performing a new operation. If a larger level of system availability is required, the nodes may not always contain the most recent data due to communication issues, which may lead to data inconsistency. This is not a binary decision to be made but a trade-off, only in the presence of network issues.
The relatively looser BASE property is therefore more predominant in non-relational databases in favor of ACID:

- **Basic Availability** - the database appears to work most of the time;
- **Soft-state** - stores do not have to be write-consistent nor do replicas have to be mutually consistent at all times;
- **Eventual consistency** - consistency is guaranteed at a later point, such as lazily at read time. [Sas18]

2.3.2.3 Storage Tools

Figure 2.5: Data storage tools considered

For the data storage layer, again, there are several tools to choose from. For this project, three of the most used non-relational databases in this kind of project were taken in consideration (Figure 2.5): Apache HBase[8], Apache Cassandra[9] and MongoDB[10] (Table 2.2).

For evaluating and comparing these tools, the following properties were taken in consideration:

- **High Availability** - system capability of continuously serving requests without returning errors;
- **Data Consistency** - viable data storage, without anomalies or read/write incoherences;
- **Fault tolerance** - system capability of preventing and recovering from system faults (hardware or network related);

[7]Source: http://ngvtech.in/droidhub/cassandra_datamodel/
[8]https://hbase.apache.org/
[9]http://cassandra.apache.org/
[10]https://www.mongodb.com/

- **Prioritized CAP theorem property** - priority and flexibility that the tool offers by default regarding data consistency and availability;
- **Data storage model** - database type (Key-value, Wide column, Document, Graph, among others);
- **Licensing** - licensing type for the use and commercialization of the respective tool.

Table 2.2: Overview of the databases considered.

	HBase	**Cassandra**	**MongoDB**
Architecture	Master/Slave	P2P homogeneous nodes	Master/Slave
Fault Tolerance	Backup master	Masterless	Master failover
CAP priority	Consistency	Availability	Consistency
Data model	Wide column	Wide column	Document
Licensing	Apache	Apache	SSPL

These three tools all implement the necessary requirements and in fact serve the same purpose but merely do so in different ways and with different priorities in consideration. HBase is a NoSQL database modeled after Google's BigTable. It follows a wide-column storage model which is a slight variation from the key-value format. Wide-column databases use rows and columns but, unlike in traditional relational databases, the names and formats of columns can be changed. Following a master/slave node architecture that runs on top of Hadoop Distributed File System (HDFS), Single Point of Failure (SPOF) is a possibility in case the master or primary node fails, even though a backup master can be set up. HBase prioritizes consistency when it comes to the CAP theorem, meaning it values and gives more importance to great data consistency in the system, due to data replication in partitions, an integral factor of HDFS. Beyond depending on Hadoop, this tool also depends on Apache Zookeeper for server management and a tool like Apache Hive for querying [Bek18c].

Cassandra, similarly to HBase, is a wide-column NoSQL database based on Amazon's DynamoDB. Unlike HBase, it follows a masterless node architecture, in which every system node is similar and serves the same function, which means there exists no SPOF [Bek18c]. Also unlike HBase, Cassandra prioritizes availability when it comes to the CAP theorem, meaning it values high availability in the system, even though this is adjustable according to the user's requirements [Tiw15]. Cassandra provides a query language named Cassandra Query Language (CQL), slightly similar to the traditional SQL, not depending on external tools like HBase does. While HBase shows great performance on database reads and is perfect to store bulky files, Cassandra is better for intensive writes and storage of multiple small files, which is more adequate to the project at hand [Bek18a] [Bek18c].

MongoDB is a document-oriented (using Binary JSON or just BSON) NoSQL database. Similarly to HBase it follows a master/slave node architecture, or primary/secondary node architecture. MongoDB processes that maintain the same data set are called replica sets. In case the master node fails, another one will be chosen to replace it, but this process may take several seconds. A decent data replication system is achievable but harder to set up compared to, say, Cassandra. Besides, using MongoDB implies the need to learn and use a query language different than the traditional SQL type language, one that is based on the JSON format.

While these three databases that were considered are one way or another adequate for the project, Cassandra was the chosen tool. It offers the best performance compared to the other two [Cop15]; compared with MongoDB and HBase under mixed operational and analytical workload, Cassandra, even with its drawbacks, is by far the best performing out of the three [Bek18b]. It also offers flexible consistency configurations, it is highly scalable, it has excellent write performance and there is no SPOF. It also provides a query language, CQL, which is easier to assimilate for whoever is already familiar with the SQL language. due to their syntax similarities. One final point in favor of Cassandra is that Celfocus already has experience using this language.

2.3.3 Data Analytics

The last layer, the analytics layer, is responsible for the data processing and analytics.

2.3.3.1 Analytics Tools

Figure 2.6: Data analytics tools considered

For this last layer, yet again, several different tools were considered after an initial short list selection (Figure 2.6), with these tools being Kafka Streams[11], Spark Streaming[12], Storm[13], Flink[14] and Samza[15], all provided by Apache Software Foundation (Table 2.3).

[11]https://kafka.apache.org/documentation/streams/
[12]https://spark.apache.org/streaming/
[13]http://storm.apache.org/
[14]https://flink.apache.org/
[15]http://samza.apache.org/

For the evaluation and comparison of these tools, the following properties were taken in consideration:

- **In-memory processing capability** - capacity to process data in-memory rather than on-disk;
- **High Availability** - continuous data processing without interruption;
- **Architecture** - data processing model (Streaming, Batch);
- **Complex Event Processing** - CEP support;
- **Native Artificial Intelligence** - native capacity to use artificial intelligence or machine learning tools for data analytics;
- **Maturity and available documentation** - amount of documentation, community and professional support availability and proven industry use;
- **Licensing** - licensing type for the use and commercialization of the respective tool.

Table 2.3: Overview of the analytics tools considered.

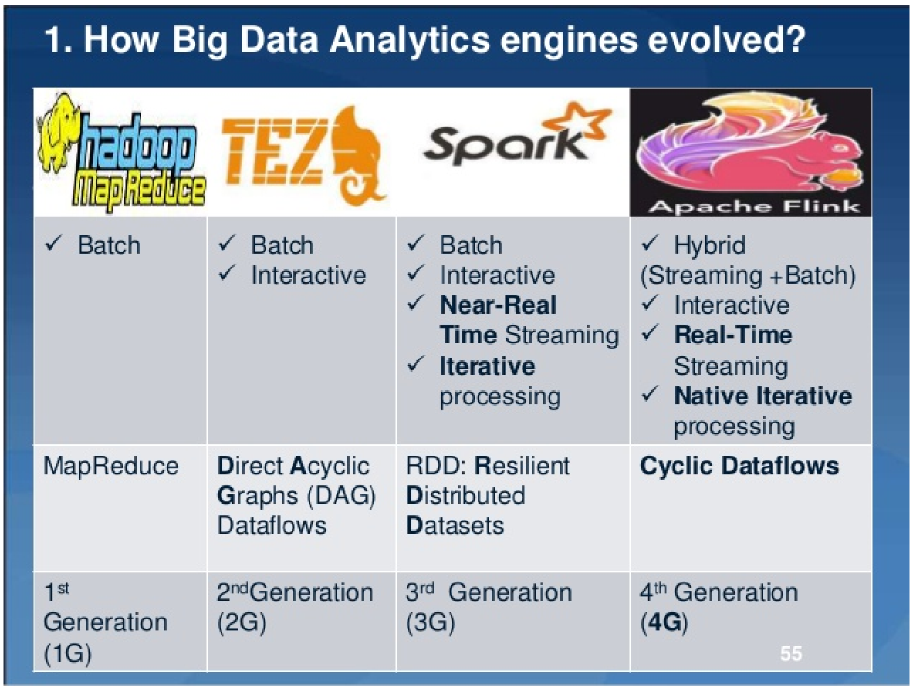
	Kafka	**Spark**	**Storm**	**Flink**	**Samza**
CEP	No	Yes	No	Yes	Yes
Native AI	No	Yes	No	Yes	No
Initial release date	2016	2012	2011	2014	2013
Licensing	Apache	Apache	Apache	Apache	Apache

Kafka Streams, unlike the other stream processing tools considered, is a lightweight library, so it does not need an additional cluster to operate, reducing the overall complexity of the system, and is specially fit for Kafka -> Kafka data flows. However, as expected, its performance and processing power is not comparable to other heavy lifting tools like Spark Streaming or Flink and has not been put to test at a large scale in the industry, due to its relative recency [Pra18]. All of this makes Kafka Streams not a very good fit for the analytics layer of Smart Monetization. It could, however, eventually be used with Kafka in the ingestion layer for preprocessing or data normalization purposes.

Spark Streaming came up as a Hadoop successor in terms of Batch Processing and was the first framework to support the Lambda architecture (where Batch and Streaming layers coexist, for data correction and speed, respectively). It is one of the most used tools in this area, being that, for instance, Netflix uses it for real-time recommendations for their consumers in its platform [SC18]. Traditionally, this tool implements near real-time processing, resorting to data micro-batching, which is the decomposition of a data stream in batches to be consumed periodically. However, with the launch of version 2.3, a new optional mode is available, Continuous Processing, allowing data processing with latency in the order of milliseconds [Tor+18], even though this option is recent and is still in experimental mode.

Storm is the Hadoop of the streaming world. It is the oldest, most mature and trustworthy open source stream processing tool and it is most suitable for use cases that require the processing of simple data events [Pra18]. While it is in fact considered a stable tool, there are better and more recent tools available out there, so it did not end up being considered much as a final choice.

Flink is Storm's successor like Spark is Hadoop's successor. It is one of the most recent and best performing tools, but its lateness to get into the market is a likely unfavourable point [Pra18]. Like Spark, which provides a machine learning library named MLlib, Flink provides FlinkML. Overall, like shown in Figure 2.7, if Spark is considered to be the 3G of Big Data, Flink is respectively the 4G [Bak16] of Big Data.



1. How Big Data Analytics engines evolved?

hadoop MapReduce	TEZ	Spark	Apache Flink
✓ Batch	✓ Batch ✓ Interactive	✓ Batch ✓ Interactive ✓ **Near-Real** Time Streaming ✓ **Iterative** processing	✓ Hybrid (Streaming +Batch) ✓ Interactive ✓ **Real-Time** Streaming ✓ **Native Iterative** processing
MapReduce	Direct Acyclic Graphs (DAG) Dataflows	RDD: Resilient Distributed Datasets	**Cyclic Dataflows**
1st Generation (1G)	2nd Generation (2G)	3rd Generation (3G)	4th Generation (**4G**)

55

Figure 2.7: Big Data Analytics tools evolution[16]

Samza is a framework described as a dimensioned version of Kafka Streams, both of them being fairly coupled with Kafka and being a good fit for Kafka -> Kafka data flows [Pra18]. Like Storm, it is considered to be a stable tool but over time it has been replaced by other more popular alternatives.

After comparing these five tools, Flink was determined to be the best fit for this layer.

Spark Streaming is more mature and proven in the industry, having vast documentation community and support available besides having good interoperability with the rest of the chosen tools [Hal18a] [Sar18] [Bag16], but Flink has been obfuscating that difference each passing day, with the increasing adoption rate from companies like Amazon, Uber or Alibaba [Fli19].
Flink offers also better performance which provides bigger potential and future proofness. It takes advantage of the Kappa architecture that, opposed to the Lambda architecture, allows a batch and stream data flow in a single pipeline, rather than separating them. The analytics layer is arguably the most important layer of the three, which supports the decision to go with a more recent tool with higher potential and future proofness such as Flink.

[16]Source: `https://www.slideshare.net/sbaltagi/why%2Dapache%2Dflink%2Dis%2Dthe%2D4g%2Dof%2Dbig%2Ddata%2Danalytics%2Dframeworks`

Chapter 3

Proof of Concept

A Proof of Concept is by definition the technical realization of a certain concept or idea in order to provide evidence of its feasibility and tangible, practical potential. The purpose of this particular Proof of Concept, for the scope of this project, is therefore to demonstrate how a minimal working pipeline of the architecture, that was designed over the initial planning period, looks like in practice. By analyzing its use and obtained results we may then be able to prove it to be a valuable and desirable asset from a business standpoint.
This section aims to report the details of the configuration and implementation of the work environment and the previously picked tools to be used, as well as describing the mock-up use case selected to be explored during this phase of the project, detailing for instance its data schema, expected kinds of results and general motivations for its choice of use.
This Proof of Concept's workflow consists of data ingestion from Kafka—data that was previously obtained from a random data generator and placed in a CSV (Comma Separated Values) file which is then sent over to Cassandra for data lake purposes and finally accessed and analyzed by Flink to output the desired results.

3.1 Environment

Initially the Operating System of choice to develop and run the project on was a minimal CentOS Virtual Machine which would ideally offer better performance and efficiency but, due to some difficulties encountered getting some tools and addons up and running, ultimately a Windows machine was used as it was more intuitive and less time consuming to troubleshoot, at least for the development process. This local machine (provided workspace laptop), for disclosure and further metrics purposes down the line, has the following specifications:

- **Brand and Model** - HP ProBook 640 G1;
- **Operating System** - Windows 10 Enterprise x64;
- **CPU** - Intel Core i7-4600M (2 cores, 4 logical processors) @ 2.9 GHz;
- **RAM** - 8GB;
- **GPU** - Intel HD Graphics 4600;
- **Storage** - 275GB Crucial MX300 SSD.

For development, testing and Proof of Concept purposes, these specifications are sufficient to handle low amounts of data processing (in the range of tens of thousands of records) at a satisfactory speed.

3.2 Tool Installation

This section will serve to describe the steps we took in order to install each of the tools that were used and their respective versions and dependencies.

Firstly we checked if there was a version of Java already present on the Operating System, which is needed to run and use the other tools. As it turned out Java was already installed, having both JRE 1.8.0_211 and JDK 1.8.0_77.

On the programming side, which is needed to customize the tool to develop, we opted by using Eclipse as the Integrated Development Environment (IDE) of choice as we are already familiar with it and have vast experience with it and, as such, proceeded to install version 4.12.0 (2019-06).

We then effectively started by downloading Kafka version 2.11-2.3.0 from Apache's website, necessary for the data ingestion layer of Smart Monetization.
We got it set up and then verified that it is working properly by running the provided default producer and consumer executables, kafka-console-producer.bat and kafka-console-consumer.bat. Before these can be run, the Kafka service itself needs to be running, which itself in turn first needs Zookeeper to be running for cluster management, coordination and configuration (as previously mentioned in the study of the state of the art, this dependency is being debated). Therefore, we proceeded to download Zookeeper version 3.5.5 from Apache's website.
If there is a need to configure Kafka's settings such as log file location and retention time, timeout values, etc., the go-to file is named server.properties, located on the Kafka installation folder, on the config subfolder.

Moving forward, seeing as Cassandra was the chosen tool for storing data, we downloaded version 3.11.4 from Apache's website. It needs Python as well, so we downloaded version 2.7.16. We then proceeded to practice using CQL (which is quite similar to traditional SQL) by creating, removing and looking up some tables and records. Cassandra's settings can be fine-tuned by editing the cassandra.yaml file on the conf subfolder of Cassandra's installation folder.

Lastly, we got Flink version 1.8.0 from Apache's website as well. We got a default Flink Job/Worker running in order to test it, then proceeded to start coding a custom java program to use it with the remainder of the tools.

Several dependencies (jar files) were required to be imported from within Eclipse to be able to code using these tools' libraries. These files, which for the most part are included with each tool's download files, are the following:

- cassandra-driver-core-3.5.0.jar;
- flink-dist_2.11-1.8.0.jar;
- guava-23.0.jar;
- kafka-clients-2.3.0.jar;
- log4j-1.2.17.jar;
- metrics-core-3.0.2.jar;
- netty-all-4.1.36-Final-sources.jar;
- netty-all-4.1.36.Final.jar;
- slf4j-api-1.7.26.jar;
- slf4j-log4j12-1.7.26.jar.

3.3 Smart Meeting Room

For the scope of this Proof of Concept, random use cases and data to be analyzed could promptly be generated, but it was thought that it would be more valuable to find and use something closer to reality, in order to derive final results that make more sense given the context.
After having a meeting and a dialogue with colleagues from the Internet of Things department, we concluded that it could be mutually interesting to use, as basis for the first use

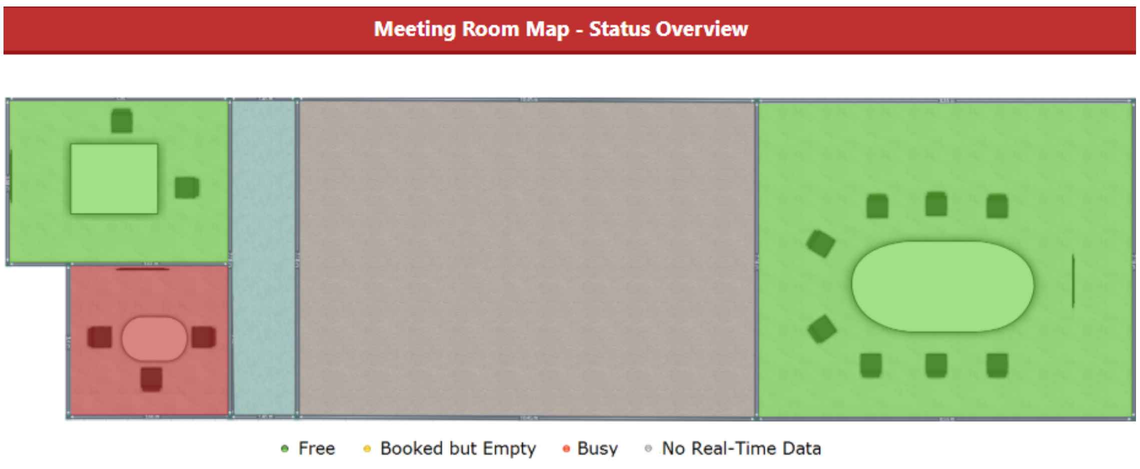


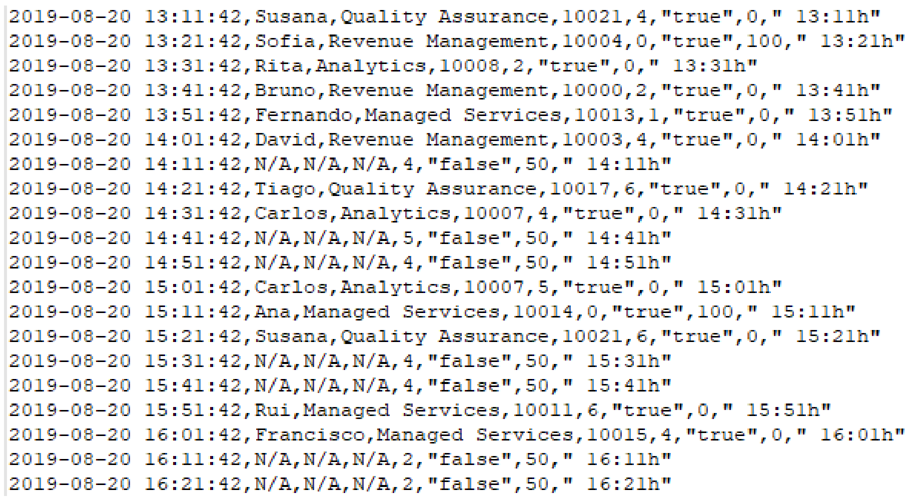
Figure 3.1: Snippet of interface for the chosen use case

case for this Proof of Concept, the data from one of their projects, Smart Meeting Room.

This project revolves, in short, around having Internet of Things (IoT) devices with cameras connected to the network, placed in selected meeting rooms in the workplace (Figure 3.1). These devices continuously capture data such as the number of occupants detected in the respective room and the system ultimately compares it with the data from Outlook, tool used to book these rooms for a period of time, providing interesting near real-time information and statistics.
As such, we can detect for instance if a room was booked but is currently empty, or if it is occupied but was previously unbooked and thus withdraw conclusions from such data events that may be used for workplace optimization and monetization opportunities.
Using this in conjunction with Smart Monetization we can pick up this data, analyze correlated events and, for example, take due actions like penalizing people who frequently book rooms but do not use them after that, which generates unintended and avoidable inefficiency in the workplace.
This data is captured from device sensors every 3 minutes, is stored in CSV files and has the following schema:

- **Timestamp** - a timestamp for the time of the captured event, e.g. 2019-07-04 11:58:00;
- **Name** - name of the person who booked the room ('N/A' if it has not been booked);
- **Department** - department name of the person who booked the room ('N/A' if it has not been booked);
- **ID** - Employee ID number of the person who booked the room ('N/A' if it has not been booked);

- **Number of persons** - the number of persons detected in the room at the given time, e.g. 3;
- **Booking status** - whether the room is currently booked according to Outlook data, i.e. true or false;
- **Verdict** - one of three numbers: 0, 50 or 100, being that 50 means the room is unbooked but currently occupied, 100 means the room is booked but currently empty and 0 meaning a normal state, which would be unbooked and empty or booked and occupied;
- **String of the timestamp** - a shorter string representation of the current time, e.g. 11:58h.



```
2019-08-20 13:11:42,Susana,Quality Assurance,10021,4,"true",0," 13:11h"
2019-08-20 13:21:42,Sofia,Revenue Management,10004,0,"true",100," 13:21h"
2019-08-20 13:31:42,Rita,Analytics,10008,2,"true",0," 13:31h"
2019-08-20 13:41:42,Bruno,Revenue Management,10000,2,"true",0," 13:41h"
2019-08-20 13:51:42,Fernando,Managed Services,10013,1,"true",0," 13:51h"
2019-08-20 14:01:42,David,Revenue Management,10003,4,"true",0," 14:01h"
2019-08-20 14:11:42,N/A,N/A,N/A,4,"false",50," 14:11h"
2019-08-20 14:21:42,Tiago,Quality Assurance,10017,6,"true",0," 14:21h"
2019-08-20 14:31:42,Carlos,Analytics,10007,4,"true",0," 14:31h"
2019-08-20 14:41:42,N/A,N/A,N/A,5,"false",50," 14:41h"
2019-08-20 14:51:42,N/A,N/A,N/A,4,"false",50," 14:51h"
2019-08-20 15:01:42,Carlos,Analytics,10007,5,"true",0," 15:01h"
2019-08-20 15:11:42,Ana,Managed Services,10014,0,"true",100," 15:11h"
2019-08-20 15:21:42,Susana,Quality Assurance,10021,6,"true",0," 15:21h"
2019-08-20 15:31:42,N/A,N/A,N/A,4,"false",50," 15:31h"
2019-08-20 15:41:42,N/A,N/A,N/A,4,"false",50," 15:41h"
2019-08-20 15:51:42,Rui,Managed Services,10011,6,"true",0," 15:51h"
2019-08-20 16:01:42,Francisco,Managed Services,10015,4,"true",0," 16:01h"
2019-08-20 16:11:42,N/A,N/A,N/A,2,"false",50," 16:11h"
2019-08-20 16:21:42,N/A,N/A,N/A,2,"false",50," 16:21h"
```

Figure 3.2: Snippet of randomly generated data to be analyzed

The original schema for this data does not include names of persons, department names and employee IDs. These were however added for this example use case seeing that it may bring final output results that are closer to the schema of a traditional CDR/EDR, allowing for more and more interesting conclusions to derive, allowing to further and more easily demonstrate the capabilities of Smart Monetization.

The data stored by this system is currently only in the tens of thousands of records. As such, I took the liberty of coding a small program which generates random but valid records based on the system previously described, allowing however many records we want to analyze, be it hundreds of thousands or even millions, if needed (useful for benchmarking purposes).
The **pseudocode** for this program is listed below in Listing 3.1. A few randomly generated rows from an example CSV file are shown in Figure 3.2.

Listing 3.1: Pseudocode for Random Data Generator

```
for(NUMBER OF DESIRED RECORDS)
{

  do
  {
    event time = event time + DELAY;
  } while(event time not in office hours);

  numberOfPersons = random int;
  roomIsBooked = Math.random() < 0.5;

  if((roomIsBooked && numberOfPersons>0)
  || (!roomIsBooked && numberOfPersons==0))
  {
    verdict = 0;
  }
  else if(!roomIsBooked && numberOfPersons>0)
  {
    verdict = 50;
  }
  else
  {
    verdict = 100;
  }

  if(roomIsBooked)
  {
    create random employee from list of names, departments and IDs;
  }
  else
  {
    name and department and ID = "N/A";
  }

  write line to CSV file;

}
```

We can generate however many records we wish to analyze with the **for** loop. The **do while** loop serves the purpose of only allowing valid timestamps (9 to 6 shift). We then use a few random number generators to get a booked or non-booked room and number of people in it. The verdict is calculated with this information accordingly. Each line while be output to the CSV file which will be used later.

3.3.1 Cassandra Setup

In order to store the data from this project in Cassandra to be further analyzed by Flink, Cassandra needs to be set up appropriately.

First we created a new keyspace, which is the highest abstraction in a distributed data store and is basically a schema work space that will contain column families. This is accomplished with the command

```
CREATE KEYSPACE smr WITH REPLICATION = ’class’ : ’SimpleStrategy’, ’replica-
tion_factor’ : 1 ;
```

on the CQL shell (sqlsh).

From here on we will use this keyspace in Cassandra for anything related to this Proof of Concept, and we can access it by typing

```
use smr;
```

on cqlsh.

Next we defined the table for the data with the command

```
CREATE TABLE smr.datatest1
(
Timestamp timestamp,
PersonName text,
PersonDepartment text,
EmployeeID text,
NumPersons int,
IsBooked Boolean,
Verdict int,
TimestampString text,
Primary key (Timestamp)
);
```

on cqlsh.

After this we can interact and access the data we want with queries similar to traditional SQL such as select count(*) from smr.datatest1;.

3.3.2 Data Analytics Use Case

The core substance of this project is to take data and analyze it in order to extract useful information that can be ultimately monetized. As such, and embracing the Smart Meeting Room project idea, a specific example use case was generated.
Kafka will obtain randomly generated data based on the Smart Meeting Room project from a CSV file source and transfer it to Cassandra. For this, a few lines of custom code were implemented for getting a Kafka producer client to get the data from the CSV file into a topic and then a Kafka consumer client to redirect it over to Cassandra.

Following this, the data will be accessed and analyzed from Cassandra by Flink and Monetizable Data Record outputs will be produced. More specifically in this case, Flink will analyze who booked rooms and how often, associating each booked room with an arbitrary cost of 1€. Flink will detect events in which a room was booked but ended up remaining unused and, on each occasion that this happens three times in a row, the person and/or the department who booked the room will be fined an arbitrary value of 5€.

After analyzing a certain time frame, Flink will thus generate Monetizable Data Records that may then be redirected to adjacent billing and invoicing systems to charge the respective parties adequately.

3.4 Tool Workflow

In order to display how the system works we start by creating a random dataset by running DataGenerator.java which will generate an adjustable number of records and save them as a CSV file.

We can then initialize Cassandra by going into Cassandra's folder and typing cassandra in the command line. We can interact with Cassandra from its own shell, which can be accessed by typing cqlsh on the command line.

After this we may start Zookeeper (zkserver on the command line) and proceed to run Kafka with

```
<kafka folder> .\bin\windows\kafka-server-start.bat .\config\server.properties
```

on the command line.

We then need to create a topic for which messages and their respective producers and consumers will be associated, which can be done by typing

kafka-topics.bat –create –bootstrap-server <ip:port> –replication-factor X –partitions Y –topic <name>

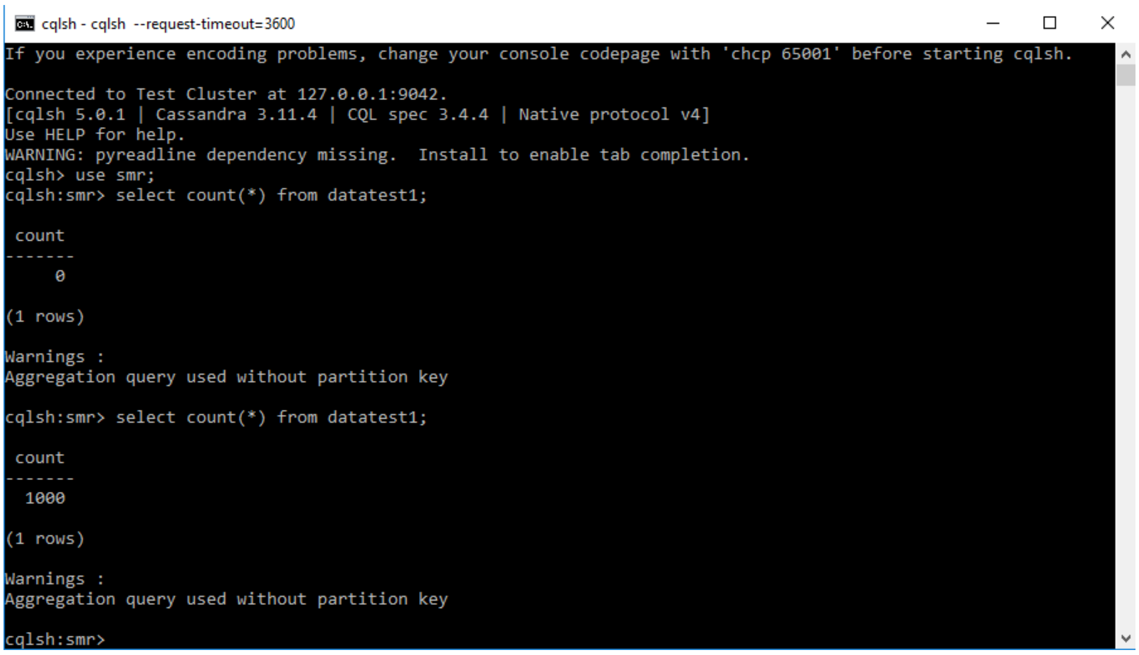
on the command line.

In this specific case, for development purposes, localhost:9092 was used and we have a replication factor of one, a single partition and a topic named datatest1.

After this, we need to individually initialize both the Kafka producer and the Kafka consumer, by typing

kafka-console-producer.bat –broker-list localhost:9092 –topic datatest1
and
kafka-console-consumer.bat –bootstrap-server localhost:9092 –topic datatest1

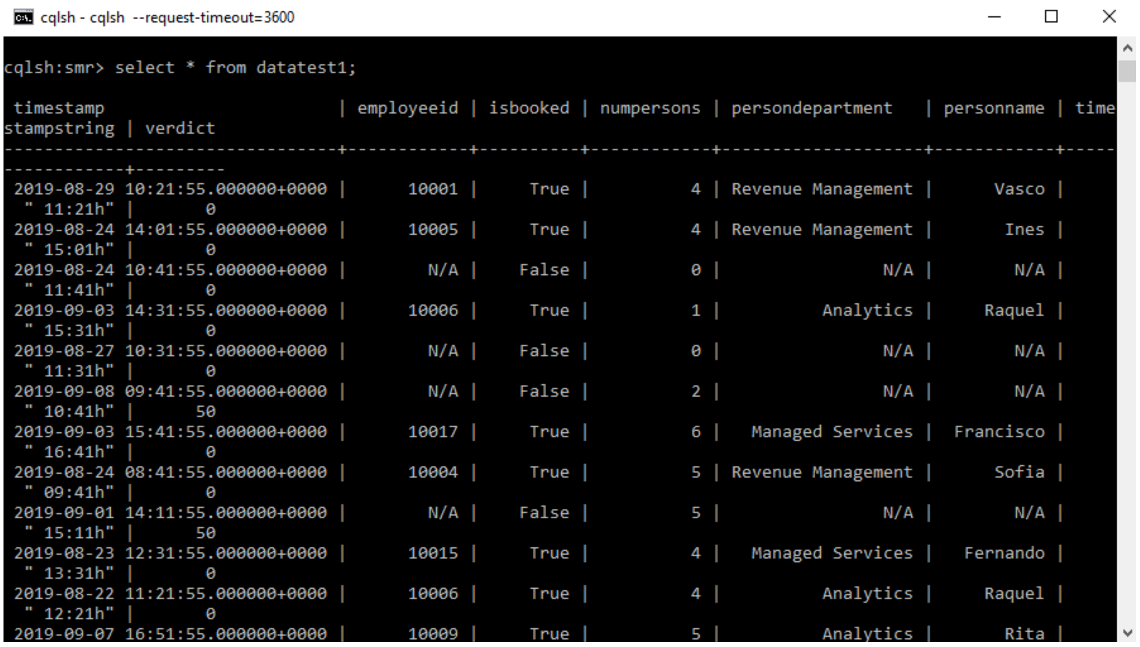
on the console, respectively.



```
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> use smr;
cqlsh:smr> select count(*) from datatest1;

 count
-------
     0

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:smr> select count(*) from datatest1;

 count
-------
  1000

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:smr>
```

Figure 3.3: Cassandra query over the SQL shell

With these services running we can finally push data by running the custom Java code for the data ingestion layer and for the data analytics layer. First we initialize the consumer by running TopicToCassandra.java, listed in Listing 3.2. What this does is



```
cqlsh:smr> select * from datatest1;

 timestamp                       | employeeid | isbooked | numpersons | persondepartment   | personname | time
stampstring | verdict
---------------------------------+------------+----------+------------+--------------------+------------+-----
------------+---------
 2019-08-29 10:21:55.000000+0000 |      10001 |     True |          4 | Revenue Management |      Vasco |
  " 11:21h" |       0
 2019-08-24 14:01:55.000000+0000 |      10005 |     True |          4 | Revenue Management |       Ines |
  " 15:01h" |       0
 2019-08-24 10:41:55.000000+0000 |        N/A |    False |          0 |                N/A |        N/A |
  " 11:41h" |       0
 2019-09-03 14:31:55.000000+0000 |      10006 |     True |          1 |          Analytics |     Raquel |
  " 15:31h" |       0
 2019-08-27 10:31:55.000000+0000 |        N/A |    False |          0 |                N/A |        N/A |
  " 11:31h" |       0
 2019-09-08 09:41:55.000000+0000 |        N/A |    False |          2 |                N/A |        N/A |
  " 10:41h" |      50
 2019-09-03 15:41:55.000000+0000 |      10017 |     True |          6 |   Managed Services |  Francisco |
  " 16:41h" |       0
 2019-08-24 08:41:55.000000+0000 |      10004 |     True |          5 | Revenue Management |      Sofia |
  " 09:41h" |       0
 2019-09-01 14:11:55.000000+0000 |        N/A |    False |          5 |                N/A |        N/A |
  " 15:11h" |      50
 2019-08-23 12:31:55.000000+0000 |      10015 |     True |          4 |   Managed Services |   Fernando |
  " 13:31h" |       0
 2019-08-22 11:21:55.000000+0000 |      10006 |     True |          4 |          Analytics |     Raquel |
  " 12:21h" |       0
 2019-09-07 16:51:55.000000+0000 |      10009 |     True |          5 |          Analytics |       Rita |
```

Figure 3.4: Cassandra query over the SQL shell

continuously listen to the associated Kafka topic for incoming messages and then send them over to Cassandra. Then, by running CsvToTopic.java, listed in Listing 3.3, we get the actual data from the specified CSV file and send it to that same topic the consumer is subscribed to. The data is then sent to and is persistently stored by Cassandra for eventual further actions. We can confirm the data was fully sent to Cassandra by querying it over the CQL shell, as shown in Figures 3.3 and 3.4.

Listing 3.2: TopicToCassandra.java code

```java
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.Session;

public class TopicToCassandra {

  KafkaConsumer<String, String> consumer = null;
  private final String topic = "datatest1";
  String serverIp = "localhost";
  String keyspace = "smr";
  Cluster cluster = Cluster.builder().addContactPoint(serverIp).build();
  Session session = cluster.connect(keyspace);

  public void initialize() {

    Properties props = new Properties();
    props.put("log.retention.ms", "60000");
    props.put("bootstrap.servers", "localhost:9092");
    props.put("group.id", "console-consumer-7456");
    props.put("zookeeper.sync.time.ms", "300");
    props.put("auto.commit.interval.ms", "1000");
    props.put("key.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
    consumer = new KafkaConsumer<>(props);

  }

  public void consume() {

    System.out.println("Now␣listening...");

    Duration duration = Duration.ofMillis(100);
    consumer.subscribe(Arrays.asList(topic));

    while (true) {

      ConsumerRecords<String, String> records = consumer.poll(duration);

      for (ConsumerRecord<String, String> record : records) {

        String lineToSplit = record.value();
        String timestamp = new String(lineToSplit.split(",")[0]);
        String personName = new String(lineToSplit.split(",")[1]);
        String personDepartment = new String(lineToSplit.split(",")[2]);
        String employeeID = new String(lineToSplit.split(",")[3]);
        int numPersons = Integer.parseInt(new
        String(lineToSplit.split(",")[4]));
        Boolean isBooked = Boolean.parseBoolean(new
        String(lineToSplit.split("\"")[1]));
        int verdict = Integer.parseInt(new String(lineToSplit.split(",")[6]));
        String timestampString = new String(lineToSplit.split(",")[7]);
        String data = String.format("insert␣into␣smr.datatest1(Timestamp,
␣␣␣␣␣␣␣␣PersonName,␣PersonDepartment,␣EmployeeID,␣NumPersons,
␣␣␣␣␣␣␣␣IsBooked,␣Verdict,␣TimestampString)␣values
␣␣␣␣␣␣␣␣('%s','%s','%s','%s',%d,%b,%d,'%s')", timestamp,
        personName, personDepartment, employeeID, numPersons,
        isBooked, verdict, timestampString);
        session.execute(data);

      }

    }

  }

  public static void main(String[] args) throws InterruptedException {

    TopicToCassandra kafkaConsumer = new TopicToCassandra();
    kafkaConsumer.initialize();
    kafkaConsumer.consume();

  }
}
```

What this TopicToCassandra.java program does is keep on waiting for new messages to be sent to the respective Kafka topic and then format the data to be sent to Cassandra, according to its schema.

Listing 3.3: CsVToTopic.java code

```
import java.io.BufferedReader;
import java.io.File;
import java.io.InputStreamReader;
import java.util.Properties;
import java.io.FileInputStream;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class CsvToTopic {

  private static Producer<String, String> producer;
  private static final String topic= "datatest1";
  private String inputFile =
  "D:/NB25167/Documents/Documentos␣do␣Estágio/Shared␣VM␣Folder/
␣␣NewGeneratedBigData1.csv";

  public void initialize() {

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092");
    props.put("serializer.class", "kafka.serializer.StringEncoder");
    props.put("request.required.acks", "1");
    props.put("key.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer",
    "org.apache.kafka.common.serialization.StringSerializer");
    producer = new KafkaProducer<>(props);

  }

  public void publishMessage(Object[] args) throws Exception {
```

```java

    File file = new File(inputFile);
    FileInputStream fstream = new FileInputStream(file);
    BufferedReader br = new BufferedReader(new InputStreamReader(fstream));

    String msg = null;

    /*
     * Read input file line by line
     */
    while ((msg = br.readLine()) != null) {

      ProducerRecord<String, String> record =
      new ProducerRecord<String, String>(topic, msg);
      producer.send(record); //This publishes the line to the given topic

    }

    br.close();

  }

  public static void main(String[] args) throws Exception {

    long startTime = System.nanoTime();
    CsvToTopic kafkaProducer = new CsvToTopic();
    kafkaProducer.initialize();
    kafkaProducer.publishMessage(args);
    producer.close();
    long endTime   = System.nanoTime();
    long totalTime = endTime - startTime;
    double totalTimeInSeconds =
    (double) totalTime / 1000000000; //1 second = 1.000.000.000 nanoseconds
    System.out.println("It␣took␣" + totalTimeInSeconds +
    "␣seconds␣to␣process␣this␣file.");

  }

}
```

On the other hand, what this CsVToTopic.java program does is pick up a set CSV file where the data dump is located and forward it to the respective Kafka topic, which will then be treated by the previous program.

All that is left to do is run the custom Flink code to process the data stored in Cassandra. This is accomplished by running FlinkAnalytics.java, listed in Listing 3.5. By doing this, the data will be processed and suitable results will be produced, including Monetizable Data Records, which will be output to a file named MDR.csv and a more visual description of total costs and fines applied, which will be output to a file named FlinkResults.txt.

Listing 3.4: CassandraToFlink.java code

```java
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Session;

public class CassandraToFlink {

  String serverIp, keyspace;
  Cluster cluster;
  Session session;;

  public CassandraToFlink() {

    this.serverIp = "localhost";
    this.keyspace = "smr";
    this.cluster = Cluster.builder().addContactPoint(serverIp).build();
    this.session = cluster.connect(keyspace);

  }

  public ResultSet getData(){

    return session.execute("SELECT * FROM datatest1");

  }

}
```

This small auxiliary class, to be used by the FlinkAnalytics.java program, retrieves all the data from Cassandra to be analyzed.

Listing 3.5: FlinkAnalytics.java code

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.tuple.Tuple8;

import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;

public class FlinkAnalytics {

  public static void main(String[] args) throws Exception {

    final int roomBookCost = 1; //1 euro
    final int roomUnusedPenalty = 5; //5 euro
    final int nrUntilPenalty = 3;

    final String outputFile = "D:/NB25167/Documents/Documentos␣do␣Estágio/
␣␣␣␣Shared␣VM␣Folder/FlinkOutput.txt";
    final String resultsFile = "D:/NB25167/Documents/Documentos␣do␣Estágio/
␣␣␣␣Shared␣VM␣Folder/FlinkResults.txt";
    final String mdrFile = "D:/NB25167/Documents/Documentos␣do␣Estágio/
␣␣␣␣Shared␣VM␣Folder/MDR.csv";

    long startTime = System.nanoTime();

    ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
    CassandraToFlink getDataFromCassandra = new CassandraToFlink();
    ResultSet rs = getDataFromCassandra.getData();

    List<Row> rows = rs.all();

    List<Tuple8<String, String, String, String, Integer, Boolean, Integer,
    String>> tupleList = new ArrayList<Tuple8<String, String,
    String, String, Integer, Boolean,
    Integer, String>>();

    for(int x=0;x<rows.size();x++) {
      String lineToSplit = rows.get(x).toString(); // Row[<TIMESTAMP>, <ID>,
      <ISBOOKED>, <NRPERSONS>, <DEPARTMENT>, <PERSON>, <TIMESTAMPSTRING>,
      <VERDICT>]
      String timestamp = new
      String(lineToSplit.split(",")[0].split("\\[")[1].trim());
      String personName = new String(lineToSplit.split(",")[5].trim());
      String personDepartment = new String(lineToSplit.split(",")[4].trim());
      String employeeID = new String(lineToSplit.split(",")[1].trim());
      int numPersons = Integer.parseInt(new
      String(lineToSplit.split(",")[3].trim()));
      Boolean isBooked = Boolean.parseBoolean(new
      String(lineToSplit.split(",")[2].trim()));
      int verdict = Integer.parseInt(new
      String(lineToSplit.split(",")[7].split("\\]")[0].trim()));
      String timestampString = new String(lineToSplit.split(",")[6].trim());

      tupleList.add(x, new Tuple8<String, String, String, String, Integer,
      Boolean, Integer, String>(timestamp, personName, personDepartment,
      employeeID, numPersons, isBooked, verdict, timestampString));
    }

    DataSet<Tuple8<String, String, String, String, Integer, Boolean,
    Integer, String>> ds = env.fromCollection(tupleList);

    DataSet<SMR> smrs = ds.map(new MapFunction<Tuple8<String, String, String,
    String, Integer, Boolean, Integer, String>, SMR>() {

      private static final long serialVersionUID = 6202380453385032823L;

      @Override
      public SMR map(Tuple8<String, String, String, String, Integer, Boolean,
      Integer, String> csvLine) throws Exception {
        String timestamp = csvLine.f0;
        String personName = csvLine.f1;
        String personDepartment = csvLine.f2;
        String employeeID = csvLine.f3;
        int numPersons = csvLine.f4;
        boolean isBooked = csvLine.f5;
        int verdict = csvLine.f6;
        String timestampString = csvLine.f7;
        return new SMR(timestamp, personName, personDepartment, employeeID,
        numPersons, isBooked, verdict, timestampString);
      }
    });

    /*
     * return dataset with every "booked␣but␣empty" room and print
     every 3rd time this happens for a fine to be applied
     */
    DataSet<SMR> filteredSMR = smrs.filter(new FilterFunction<SMR>() {

      HashMap<String, int[]> peopleCosts = new HashMap<String, int[]>();
      // person name | nr of booked rooms, nr of verdict 100 received
      HashMap<String, int[]> departmentCosts = new HashMap<String, int[]>();
      // department name | nr of booked rooms, nr of verdict 100 received

```

```java
    String mdrOutput = ""; //text for MDR output file

    private static final long serialVersionUID = -8790897659555390061L;

    @Override
    public boolean filter(SMR smr) throws Exception {

      if(smr.getIsBooked()) {

        String persName = smr.getPersonName();
        String depName = smr.getPersonDepartment();

        if(!peopleCosts.containsKey(persName)) {
          peopleCosts.put(persName, new int[]{0,1});
        }
        if(!departmentCosts.containsKey(depName)) {
          departmentCosts.put(depName, new int[]{0,1});
        }

        if(smr.getVerdict()==100) { // if booked but unused

          if(peopleCosts.get(persName)[1]%nrUntilPenalty==0) {
            mdrOutput += smr.getEmployeeID() + "," + smr.getPersonName() +
            "," + smr.getPersonDepartment() + "," + smr.getTimestamp() +
            "," + (roomBookCost + roomUnusedPenalty) + "\n";
            System.out.println(persName + "␣has␣been␣fined␣" +
            roomUnusedPenalty + "␣euros!\n");
          }
          if(departmentCosts.get(depName)[1]%3==0) {
            System.out.println(depName + "␣have␣been␣fined␣" +
            roomUnusedPenalty + "euros!\n");
          }
          peopleCosts.put(persName, new int[]{peopleCosts.get(persName)
          [0]+1, peopleCosts.get(persName)[1]+1});
          departmentCosts.put(depName, new int[]{departmentCosts
          .get(depName)[0]+1, departmentCosts.get(depName)[1]+1});
        }
        else { // if booked but used
          mdrOutput += smr.getEmployeeID() + "," + smr.getPersonName() +
          "," + smr.getPersonDepartment() + "," + smr.getTimestamp() + ","
          + roomBookCost + "\n";
          peopleCosts.put(persName, new int[]{peopleCosts.get(persName)
          [0]+1, peopleCosts.get(persName)[1]});
          departmentCosts.put(depName, new int[]{departmentCosts
          .get(depName)[0]+1, departmentCosts.get(depName)[1]});
        }

      }

      String results = ""; //text for results output file
```

```

        for (HashMap.Entry<String, int[]> entry : peopleCosts.entrySet()) {
          int totalBookings = entry.getValue()[0];
          int totalUnusedRooms = entry.getValue()[1];
          results += entry.getKey() + "␣has␣booked␣" + totalBookings +
          "␣rooms,␣spending␣" + totalBookings*roomBookCost +
          "euros,␣and␣was␣fined␣" + (totalUnusedRooms-1)/nrUntilPenalty +
          "␣times␣with␣a␣cost␣of␣" + ((totalUnusedRooms-1)/nrUntilPenalty)*
          roomUnusedPenalty + "euros!\n";
        }

        for (HashMap.Entry<String, int[]> entry : departmentCosts
        .entrySet()) {
          int totalBookings = entry.getValue()[0];
          int totalUnusedRooms = entry.getValue()[1];
          results += entry.getKey() + "␣have␣booked␣" + totalBookings +
          "␣rooms,␣spending␣" + totalBookings*roomBookCost +
          "euros,␣and␣were␣fined␣" + (totalUnusedRooms-1)/nrUntilPenalty +
          "␣times␣with␣a␣cost␣of␣" + ((totalUnusedRooms-1)/nrUntilPenalty)
          *roomUnusedPenalty + "euros!\n";
        }

        BufferedWriter writer = new BufferedWriter(new
        FileWriter(resultsFile));
        writer.write(results);
        writer.close();
        BufferedWriter writer2 = new BufferedWriter(new FileWriter(mdrFile));
        writer2.write(mdrOutput);
        writer2.close();

        return smr.getVerdict() == 100;
      }
    });

    filteredSMR.writeAsText(outputFile);

    long endTime   = System.nanoTime();
    long totalTime = endTime - startTime;
    double totalTimeInSeconds = (double) totalTime / 1000000000;
    //1 second = 1.000.000.000 nanoseconds

    System.out.printf("Data␣successfuly␣processed␣in␣%f␣seconds.\n␣\n",
    totalTimeInSeconds);

    env.execute("Executing...");

  }

}
```

This FlinkAnalytics.java program analyzes each record and will output both a summary of its findings and a file with Monetizable Data Records.

Each time we repeatedly run the system there may be issues due to repeated data and leftover metadata stored in the system. Due to this it is recommended that, on top of clearing the Cassandra table the data is stored on (TRUNCATE command), tool logs are better off cleared too.
I made a short and simple batch file in order to do this, listed in Listing 3.6.

Listing 3.6: Batch code to delete logs

```
echo Batch to delete logs
D:
del "D:\NB25167\Downloads\kafka_2.11-2.3.0\.kafka-logs\*.*" /s /f /q
del "D:\NB25167\Downloads\kafka_2.11-2.3.0\logs\*.*" /s /f /q
rmdir /s /q "D:\NB25167\Downloads\logs\kafka"
mkdir D:\NB25167\Downloads\logs\kafka
C:
del "C:\Users\NB25167\.data\*.*" /s /f /q
echo Done!
```

3.5 Testing and Results

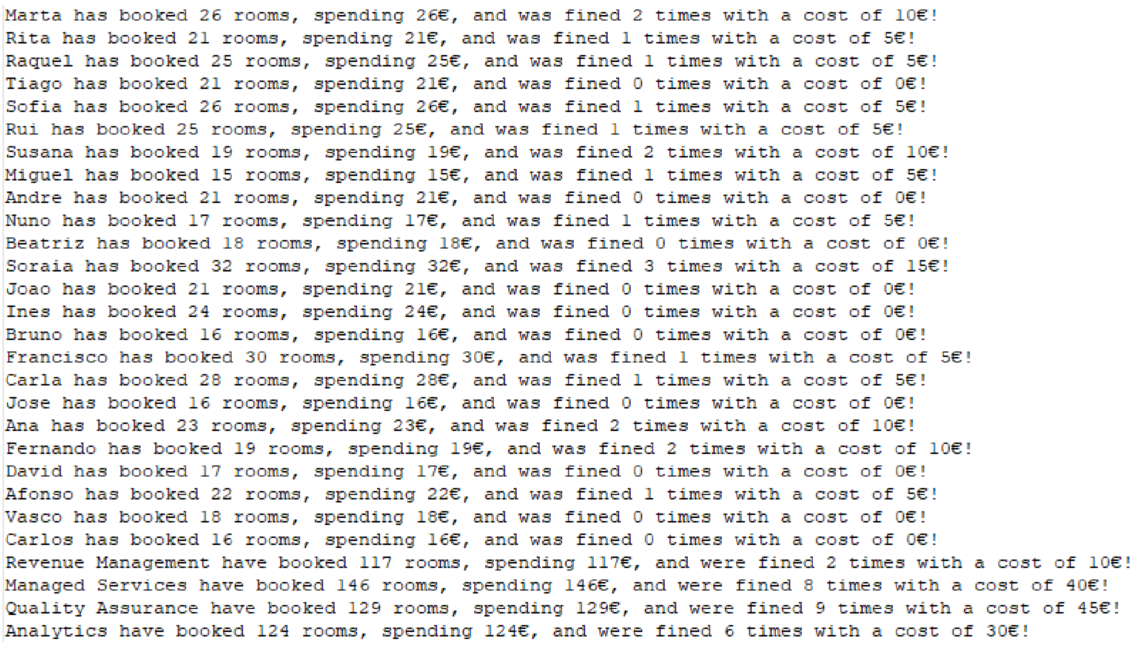
For purposes of testing the use case selected for this Proof of Concept, and for purposes of random data generation, details for twenty-four fictitious persons from four different fictitious departments were created in order to be used when we want to generate random data to be analyzed, keeping in mind we want to be able to generate Monetizable Data Records from it so we need certain information such as Employee ID's. The random data generator takes these fictitious employees and creates random bookings and uses of rooms within a defined time frame, depending on how many records (lines) we want to generate and the delay between each event, in minutes, keeping in mind the standard daily and weekly work hours and shifts.
For this particular test, one thousand records were generated with a ten minute delay between each.
The list of fictitious employees used for this use case is shown in Table 3.1.

After generating random data using this information, stored in a CSV file, we are able to run Smart Monetization, which in essence ingests said data from the CSV file and sends it over to Cassandra who stores it in the appropriate defined schema. Then, Flink analyzes the data and produces the type of desired results we are expecting. These results are separated in two files: the first file allows a rather human-readable summary of the data analyzed and results obtained, detailing how many rooms each employee booked and how many times they were fined, along with the total costs sustained. At the end, a

ID	Name	Department
10000	Bruno	Revenue Management
10001	Vasco	Revenue Management
10002	Jose	Revenue Management
10003	David	Revenue Management
10004	Sofia	Revenue Management
10005	Ines	Revenue Management
10006	Raquel	Analytics
10007	Miguel	Analytics
10008	Carlos	Analytics
10009	Rita	Analytics
10010	Andre	Analytics
10011	Marta	Analytics
10012	Joao	Managed Services
10013	Rui	Managed Services
10014	Carla	Managed Services
10015	Fernando	Managed Services
10016	Ana	Managed Services
10017	Francisco	Managed Services
10018	Soraia	Quality Assurance
10019	Tiago	Quality Assurance
10020	Nuno	Quality Assurance
10021	Afonso	Quality Assurance
10022	Beatriz	Quality Assurance
10023	Susana	Quality Assurance

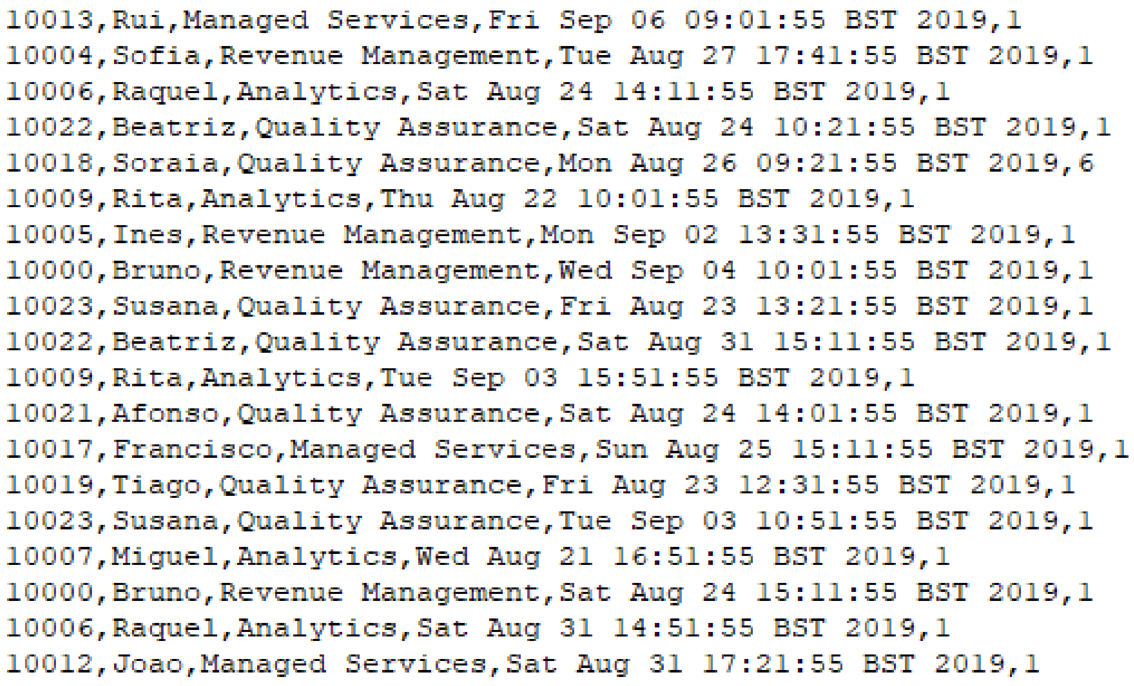
Table 3.1: List of fictitious employees

summary for each department is also shown in order to provide a different point of view. The content of this file is shown in Figure 3.5; The other file contains the data that constitutes the Monetizable Data Records in an adequate schema, made up of the employee ID, employee name, name of the department, timestamp for the event and value to charge, depending on the type of situation pertained to the associated event. A glimpse of the content of this file is shown in Figure 3.6.



```
Marta has booked 26 rooms, spending 26€, and was fined 2 times with a cost of 10€!
Rita has booked 21 rooms, spending 21€, and was fined 1 times with a cost of 5€!
Raquel has booked 25 rooms, spending 25€, and was fined 1 times with a cost of 5€!
Tiago has booked 21 rooms, spending 21€, and was fined 0 times with a cost of 0€!
Sofia has booked 26 rooms, spending 26€, and was fined 1 times with a cost of 5€!
Rui has booked 25 rooms, spending 25€, and was fined 1 times with a cost of 5€!
Susana has booked 19 rooms, spending 19€, and was fined 2 times with a cost of 10€!
Miguel has booked 15 rooms, spending 15€, and was fined 1 times with a cost of 5€!
Andre has booked 21 rooms, spending 21€, and was fined 0 times with a cost of 0€!
Nuno has booked 17 rooms, spending 17€, and was fined 1 times with a cost of 5€!
Beatriz has booked 18 rooms, spending 18€, and was fined 0 times with a cost of 0€!
Soraia has booked 32 rooms, spending 32€, and was fined 3 times with a cost of 15€!
Joao has booked 21 rooms, spending 21€, and was fined 0 times with a cost of 0€!
Ines has booked 24 rooms, spending 24€, and was fined 0 times with a cost of 0€!
Bruno has booked 16 rooms, spending 16€, and was fined 0 times with a cost of 0€!
Francisco has booked 30 rooms, spending 30€, and was fined 1 times with a cost of 5€!
Carla has booked 28 rooms, spending 28€, and was fined 1 times with a cost of 5€!
Jose has booked 16 rooms, spending 16€, and was fined 0 times with a cost of 0€!
Ana has booked 23 rooms, spending 23€, and was fined 2 times with a cost of 10€!
Fernando has booked 19 rooms, spending 19€, and was fined 2 times with a cost of 10€!
David has booked 17 rooms, spending 17€, and was fined 0 times with a cost of 0€!
Afonso has booked 22 rooms, spending 22€, and was fined 1 times with a cost of 5€!
Vasco has booked 18 rooms, spending 18€, and was fined 0 times with a cost of 0€!
Carlos has booked 16 rooms, spending 16€, and was fined 0 times with a cost of 0€!
Revenue Management have booked 117 rooms, spending 117€, and were fined 2 times with a cost of 10€!
Managed Services have booked 146 rooms, spending 146€, and were fined 8 times with a cost of 40€!
Quality Assurance have booked 129 rooms, spending 129€, and were fined 9 times with a cost of 45€!
Analytics have booked 124 rooms, spending 124€, and were fined 6 times with a cost of 30€!
```

Figure 3.5: Flink results summary



```
10013,Rui,Managed Services,Fri Sep 06 09:01:55 BST 2019,1
10004,Sofia,Revenue Management,Tue Aug 27 17:41:55 BST 2019,1
10006,Raquel,Analytics,Sat Aug 24 14:11:55 BST 2019,1
10022,Beatriz,Quality Assurance,Sat Aug 24 10:21:55 BST 2019,1
10018,Soraia,Quality Assurance,Mon Aug 26 09:21:55 BST 2019,6
10009,Rita,Analytics,Thu Aug 22 10:01:55 BST 2019,1
10005,Ines,Revenue Management,Mon Sep 02 13:31:55 BST 2019,1
10000,Bruno,Revenue Management,Wed Sep 04 10:01:55 BST 2019,1
10023,Susana,Quality Assurance,Fri Aug 23 13:21:55 BST 2019,1
10022,Beatriz,Quality Assurance,Sat Aug 31 15:11:55 BST 2019,1
10009,Rita,Analytics,Tue Sep 03 15:51:55 BST 2019,1
10021,Afonso,Quality Assurance,Sat Aug 24 14:01:55 BST 2019,1
10017,Francisco,Managed Services,Sun Aug 25 15:11:55 BST 2019,1
10019,Tiago,Quality Assurance,Fri Aug 23 12:31:55 BST 2019,1
10023,Susana,Quality Assurance,Tue Sep 03 10:51:55 BST 2019,1
10007,Miguel,Analytics,Wed Aug 21 16:51:55 BST 2019,1
10000,Bruno,Revenue Management,Sat Aug 24 15:11:55 BST 2019,1
10006,Raquel,Analytics,Sat Aug 31 14:51:55 BST 2019,1
10012,Joao,Managed Services,Sat Aug 31 17:21:55 BST 2019,1
```

Figure 3.6: Flink Monetizable Data Records

Chapter 4

Conclusions

4.1 Final Thoughts

The aim of this thesis was to select an interesting and promising topic within the vast computer science field to analyze, study and explore in order to apply and test all the knowledge acquired through these five years of studies consolidated in one larger final academic work. The opportunity to be able to achieve this goal in a professional context was an additional added value taken in consideration that allowed for a broader learning experience opportunity.

This internship and subsequent thesis did correspond to the expectations perceived during its proposal and the predefined goals were met upon its completion.
This work consisted of exploring new methods of monetization of dynamic events by the use of Big Data tools. A state of the art study first took part in the earlier stages of this project, followed by a technical design and following implementation of a Proof of Concept in order to demonstrate its viability.
During the course of this project several interesting decisions had to be taken into consideration which allowed for a deeper understanding of the work ethics in a professional telecommunications company as well as the limitations and strong suits of both software tools and hardware solutions each more adequately regarded for specific solutions.

The scope of this project was slightly restructured during the planning period given the valuable feedback obtained from both colleagues and professors upon the thesis preparation presentation in April. Following this presentation, beyond general document and writing-related feedback, advice and opinions on the project itself were also given, such as the ambitiousness of its scope, which was adequately reduced from a broad idea to a

more plausible implementation given the time given and scope of this thesis.
The opportunity of interning at Celfocus to explore this project and develop this thesis was also very enriching and contributed to the learning of great deals of different subjects and domains not only in the world of telecommunications and consulting, but also allowing a deeper understanding of different technologies and concepts and how they may be used to solve problems and create value.

4.2 Contributions

Fullfiling the objectives that have been defined in the thesis preparation period, this project is materialized in the form of both a report that analyses a set of tools and their features, and in the design, implementation and demonstration of a prototype application.
An initial design of the high level architecture of Smart Monetization took place, followed by a study of the state of the art of several big data and analytics tools and solutions, comparing and picking the most adequate ones for each layer of the project. This planning period was followed by an implementation period, where we would bring together these tools and develop a Proof of Concept able of demonstrating the capabilities of Smart Monetization.

In the end, after testing and verifying the results obtained, we were able to in fact ascertain the usefulness and potential of this tool. The tool is able to gather data, store it and then analyze it as expected, producing desirable outputs such as Monetizable Data Records that may be forwarded to adjacent departments for billing and charging purposes. Due to hardware and time limitations we were not yet able to evaluate its performance on a large scale environment (multiple and diverse sources, extraordinarily large amounts of data), but given the choice of tools, which were chosen precisely with scalability in mind, this should not pose a problem.

4.3 Future Work

Given the time and scope limitations of this project, it has not yet been fully explored to its full capacity and, as such, it can be extended and improved in multiple ways in the foreseeable future.

In the future it would be interesting to evaluate the system's throughput using several, more powerful, computing nodes - if necessary, cloud-based to allow us to thoroughly test the prototype scalability. That could produce some very interesting results and truly allow the tool to shine, by analizing large amounts of data with very high throughput.

In terms of features there is a lot of potential to explore given the groundwork that has been laid in these past few months. Different kinds of data sources and data sinks may be tested, the flexibility of the system may be improved, more use cases may be produced and analyzed, further integration with other systems may be achieved and even though it is not the core purpose of the tool, better visualization methods and tools may be used in conjunction with Smart Monetization in order to better portray the results and information obtained from it.

Bibliography

[Ash09] K. Ashton. "That 'Internet of Things' Thing." In: *RFID Journal* (2009).

[Ass16] G. Association. *IoT Big Data Framework Architecture*. Tech. rep. GSM Association, 2016.

[Bag16] A. Baghel. *Traffic Data Monitoring Using IoT, Kafka and Spark Streaming*. 2016. URL: https://www.infoq.com/articles/traffic-data-monitoring-iot-kafka-and-spark-streaming (visited on 02/27/2019).

[Bak16] S. Bakliwal. *4G of Big Data - Apache Flink. A Comprehensive Guide covering all the aspects*. 2016. URL: https://www.linkedin.com/pulse/4g-big-data-apache-flink-comprehensive-guide-all-aspects-bakliwal (visited on 03/01/2019).

[Bek18a] A. Bekker. *Apache Cassandra vs. Hadoop Distributed File System: When Each is Better*. 2018. URL: https://www.scnsoft.com/blog/cassandra-vs-hadoop (visited on 02/28/2019).

[Bek18b] A. Bekker. *Cassandra Performance: The Most Comprehensive Overview You'll Ever See*. 2018. URL: www.scnsoft.com/blog/cassandra-performance (visited on 07/29/2019).

[Bek18c] A. Bekker. *Cassandra vs. HBase: twins or just strangers with similar looks?* 2018. URL: https://www.scnsoft.com/blog/cassandra-vs-hbase (visited on 02/28/2019).

[Cas+16] M. O. de Castro, C. Bertolini, and E. Preuss. "Avaliação experimental dos Brokers Kafka e Apache Flume no Contexto de Big Data." Master's thesis. Universidade Federal de Santa Maria - UFSM, 2016.

[Cop15] E. P. Coporation. *Benchmarking Top NoSQL Databases*. Tech. rep. End Point Corporation, 2015.

[Dan18] E. Dans. *How Analytics Has Given Netflix The Edge Over Hollywood*. 2018. URL: https://www.forbes.com/sites/enriquedans/2018/05/27/how-analytics-has-given-netflix-the-edge-over-hollywood/ (visited on 03/11/2019).

[Fli19] Flink. *Powered by Flink*. 2019. URL: https://flink.apache.org/poweredby.html (visited on 03/01/2019).

[Fow12] M. Fowler. *NosqlDefinition*. 2012. URL: https://martinfowler.com/bliki/NosqlDefinition.html (visited on 06/24/2019).

[Hal18a] A. Hall. *Processing streams of data with Apache Kafka and Spark: ingestion, processing, reaction, examples*. 2018. URL: lenadroid.github.io/posts/distributed-data-streaming-action.html (visited on 02/27/2019).

[Hal18b] S. Hall. *Kafka Alternative Pulsar Unifies Streaming and Queuing*. 2018. URL: https://www.thenewstack.io/kafka-alternative-pulsar-unifies-streaming-and-queuing/ (visited on 02/27/2019).

[Hal+08] S. Haller, S. Karnouskos, and C. Schroth. "The Internet of Things in an Enterprise Context." In: *Future Internet Symposium* (2008).

[Hum17] P. Humphrey. *Understanding When to use RabbitMQ or Apache Kafka*. 2017. URL: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka (visited on 02/27/2019).

[Kha+18] N. Khan, M. Alsaqer, H. Shah, G. Badsha, A. Abbasi, and S. Salehian. "The 10 Vs, Issues and Challenges of Big Data." In: 2018.

[Kre14] J. Kreps. *Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines)*. 2014. URL: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines (visited on 02/27/2019).

[Mak18] I. Makaranka. *Alternative approaches to implementing your data lake*. 2018. URL: https://www.scnsoft.com/blog/data-lake-implementation-approaches (visited on 02/27/2019).

[Mar15] S. Martins. *O futuro dos serviços convergentes - será triple-play ou quad-play que sai por cima?* 2015. URL: https://pt.linkedin.com/pulse/o-futuro-dos-servi%C3%A7os-convergentes-ser%C3%A1-triple-play-sidcley-martins (visited on 03/01/2019).

[MB12] A. McAfee and E. Brynjolfsson. "Big Data: The Management Revolution." In: *Harvard Business Review* (2012).

[McC19] C. McCabe. *KIP-500: Replace ZooKeeper with a Self-Managed Metadata Quorum*. 2019. URL: https://cwiki.apache.org/confluence/display/KAFKA/KIP-500%3A+Replace+ZooKeeper+with+a+Self-Managed+Metadata+Quorum (visited on 08/06/2019).

[Nan15] N. Nannoni. "Message-oriented Middleware for Scalable Data Analytics Architectures." Master's thesis. KTH Royal Institute of Technology, 2015.

[Pra18] C. Prakash. *Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza : Choose Your Stream Processing Framework*. 2018. URL: https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b (visited on 02/27/2019).

[Put18] S. Putri. *How Uber Depends on Big Data Analysis to Improve Their Service*. 2018. URL: https://blog.sonarplatform.com/index.php/2018/01/04/how-uber-depends-on-big-data-analysis-to-improve-their-service/ (visited on 03/11/2019).

[RK16] J. Rao and J. Kreps. *Powered by*. 2016. URL: https://cwiki.apache.org/confluence/display/KAFKA/Powered+By (visited on 02/27/2019).

[Rec18] M. Reca. *The 3+ Vs of Big Data: Volume, Velocity, Variety, and a whole lot more*. 2018. URL: https://www.flydata.com/blog/3-vs-of-big-data/ (visited on 02/19/2019).

[Sar18] R. Saraf. *How we build a robust analytics platform using Spark, Kafka and Cassandra*. 2018. URL: https://medium.com/walmartlabs/how-we-build-a-robust-analytics-platform-using-spark-kafka-and-cassandra-lambda-architecture-70c2d1bc8981 (visited on 02/27/2019).

[Sas18] B. M. Sasaki. *Graph Databases for Beginners: ACID vs. BASE Explained*. 2018. URL: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/ (visited on 06/24/2019).

[SC18] N. Sharma and E. Chow. *Near Real-Time Netflix Recommendations Using Apache Spark Streaming*. 2018. URL: https://databricks.com/session/near-real-time-netflix-recommendations-using-apache-spark-streaming (visited on 02/27/2019).

[SD12] V. Sharma and M. Dave. "SQL and NoSQL Databases." In: *International Journal of Advanced Research in Computer Science and Software Engineering* (2012).

[Sic17] T. Siciliani. *Big Data Ingestion: Flume, Kafka, and NiFi*. 2017. URL: https://dzone.com/articles/big-data-ingestion-flume-kafka-and-nifi (visited on 02/27/2019).

[Sim12] S. Simon. *Brewer's CAP Theorem*. Tech. rep. University of Basel, 2012.

[Sni+12] C. Snijders, U. Matzat, and U.-D. Reips. ""Big Data": Big Gaps of Knowledge in the Field of Internet Science." In: *International Journal of Internet Science* (2012).

[Str18] Streamlio. *Apache Pulsar Outperforms Apache Kafka by 2.5x on OpenMessaging Benchmark*. 2018. URL: https://streaml.io/about/newsreleases/apache-pulsar-outperforms-apache-kafka-on-openmessaging-benchmark (visited on 02/27/2019).

[Suj+15] J. Sujata, S. Sohag, D. Tanu, D. Chintan, P. Shubham, and G. Sumit1. "Impact of Over the Top (OTT) Services on Telecom Service Providers." In: *Indian Journal of Science and Technology* (2015).

[Tiw15] A. Tiwari. *Why Cassandra is an Excellent Choice for Real Time Analytics Workload*. 2015. URL: http://blogs.shephertz.com/2015/04/22/why-cassandra-excellent-choice-for-realtime-analytics-workload/ (visited on 02/28/2019).

[Tor+18] J. Torres, M. Armbrust, T. Das, and S. Zhu. *Introducing Low-latency Continuous Processing Mode in Structured Streaming in Apache Spark 2.3*. 2018. URL: https://databricks.com/blog/2018/03/20/low-latency-continuous-processing-mode-in-structured-streaming-in-apache-spark-2-3-0.html (visited on 02/27/2019).

[Tru14] Y. Trudeau. *Exploring Message Brokers: RabbitMQ, Kafka, ActiveMQ, and Kestrel*. 2014. URL: https://dzone.com/articles/exploring-message-brokers (visited on 02/27/2019).

[Van16] Vandana. *Internet of Everything Explained*
. 2016. URL: https://internetofthingswiki.com/internet%2Dof%2Deverything%2Dexplained/690/ (visited on 06/17/2019).

[Ven18] S. Venkat. *Telcos need an Over-The-Top (OTT) strategy to regain lost ground*. 2018. URL: https://www.cerillion.com/Blog/September-2018/Telcos-need-an-Over-The-Top-(OTT)-strategy (visited on 02/19/2019).

[Ver18] A. Verma. *NoSQL vs. SQL – How NoSQL is Better for Big Data Applications?* 2018. URL: https://www.whizlabs.com/blog/nosql-vs-sql/ (visited on 02/22/2019).

[Was16] S. Wasserman. *IBM Watson IoT Platform to Help Engineers with Product Development*. 2016. URL: https://www.engineering.com/IOT/ArticleID/11759/IBM-Watson-IoT-Platform-to-Help-Engineers-with-Product-Development.aspx (visited on 02/18/2019).