Universities of Burgos, León and Valladolid

Master's degree

Business Intelligence and Big Data in Cyber-Secure Environments

Thesis of the Master's degree in Business Intelligence and Big Data in Cyber-Secure Environments

título del TFM

Presented by Adrián Riesco Valbuena
in University of Burgos — June 26, 2022
Supervisor: Álvar Arnaiz González

Universities of Burgos, León and Valladolid

Master's degree in Business Intelligence and Big Data in Cyber-Secure Environments

Mr. Álvar Arnaiz González, professor of the department named Computer Engineering, area named Computer Languages and Systems.

Exposes:

That the student Mr. Adrián Riesco Valbuena, with DNI 71462231N, has completed the Thesis of the Master in Business Intelligence and Big Data in Cyber-Secure Environments titled NOMBRE TFM.

And that thesis has been carried out by the student under the direction of the undersigned, by virtue of which its presentation and defense is authorized.

In Burgos, June 26, 2022

Approval of the Supervisor:

Mr. Álvar Arnaiz González

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android . . .

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Contents

List of Figures

List of Tables

Memory

Introduction

Social networks are currently a fundamental aspect of society. People usually use social networks to share experiences, opinions, and aspects of their lives and interact with other people. Using social networks as a data source we can access a huge amount of information and be able to build accurate analyses on practically any topic.

On the other hand, another aspect that has gradually permeated our society is the concept of subscriptions to services, be it music, movies, games, or almost any concept that we can think of. Not so many years ago, the concept of paying for subscriptions to services, where you do not actually own the content you pay for and instead get temporary access whose duration is defined by how long you continue to pay for the subscription, was relegated to very specific services and was not nearly as globalized as it is today.

The global acceptance of subscription as a service is reflected in social networks, where users can comment on the different music, movie, and game platforms, turning each new release into a social phenomenon. In order to take advantage of both worlds, this project uses the social network Twitter to obtain information about the latest music listened to by users (by searching for a particular hashtag) and then consult the data of the song and the artist involved that Spotify, a platform based on music as a service, has.

The development of the project has followed an agile methodology with two-week sprints, and has focused on learning and using recognized tools in the field of Big Data, such as Apache Airflow, Apache Spark, Apache Cassandra, Docker Compose and Flask, among others.

Project objectives

The initial objectives through which the use case was built consisted in the following points:

- Potential ability to obtain data in real time.
- Combination of at least two different data sources.
- Potential to scale in both technology and data volume.
- Implication of recognized technologies within the world of Big Data.
- Use of open source tools to avoid potential costs.

After a research, the author designed the use case and built the objectives of the projects:

- Build a pipeline to gather information about last songs listened from the Twitter API. The name of the endpoint queried is *recent search*.
- Find information about the songs (name, artist and audio features) through the Spotify API. The names of the endpoints queried are *search for item* and *get tracks' audio features*.
- Execute all the ETL[1] process in *Apache Spark*.
- Store all the data in a data warehouse under a known technology, *Apache Cassandra*.

[1]ETL is the acronym for Extract, Transform and Load, the three phases for data processing

- Visualize the information in a custom front-end and back-end created with *Bootstrap* and *Flask*.
- Orchestrate all the data flow with *Apache Airflow*.
- Develop the project with *Docker* and *Docker Compose* to ensure deployment through heterogeneous environments.

Through these global objectives, the low-level functional and technical requirements were specified. More information about these requirements is included in the appendix.

Theoretical concepts

In this section are covered the theoretical concepts in which the project has been based. All concepts are described in a detailed and simple way since this master's thesis can be aimed at technical and non-technical students.

3.1 Big Data

The term Big Data is used to refer to data sets whose size exceeds the capabilities of the software systems used to capture, preserve and process the data within an acceptable time window.

The definition of Big Data is usually linked to the three dimensions or Vs defined by Doug Laney in 2001 [12]:

- **Volume.** The amount of data to handle.
- **Velocity.** The rate at which new data is created and consumed.
- **Variety.** The diversity or form of the captured data.

After the definition of the three Vs, more dimensions have been added: Veracity, Value, Validity, Vagueness, Volatility... Up to a total of 42 different dimensions to define Big Data.

From the moment big data is discovered until it can be used and generate value, it goes through a life cycle made up of the following phases: discovery, raw data ingestion, raw data processing, storage, integration with other data, analysis, and presentation of results.

Due to the complexity of big data, it usually requires a complex and varied technological ecosystem to be managed.

3.2 ETL

An ETL process refers to the stages that every data processing exercise usually goes through:

1. **Extraction.** Raw data is collected from the source and fed into our Data Lake, which preserves it in its original form.

2. **Transformation.** The raw data is cleaned and transformed to be used in our environment.

3. **Load.** Once structured and filtered, the data is entered into our Data Warehouse.

3.3 API

An Application Programming Interface or API is an interface that defines the interactions that can be made with a software system. The APIs generally define the data that can be requested and sent to the system, the way to authenticate to it and the format of the returned data [10].

In relation to web development, most of the APIs work according to Hypertext Transfer Protocol (HTTP), a communication protocol that allows information transfers through files on the World Wide Web. Additionally and not exclusive, a large number of APIs are developed according to the REST architectural style, defined by Roy Fielding in the year 2000 and which is based on a series of principles that seek to facilitate development:

1. Uniform interface for all resources, forcing all queries made to the same resource (each with a specific Uniform Resource Identifier or URI) to have the same form regardless of the origin of the request.

2. Decoupling between the client and the server, making the only information that the client must know about the server is its identifier (URI) and that the only action to be carried out by the server is to return the data required in the request.

3. Stateless queries, meaning that each request must contain all the information necessary to be processed without requiring an additional request or storing any type of state.

4. Allow, whenever possible, both client-side and server-side caching to reduce the load of the former and increase the scalability of the latter.

5. Layer system, allowing multiple intermediaries between the client and the server and preventing them from knowing in any case if they are communicating with the other party or with an intermediary.

6. Although the resources exchanged are usually static, a REST architecture can optionally have responses that contain snippets of executable code.

In general terms, an API based on a REST architecture serves to make it easier for developers to develop applications that interact with the resources published by it.

Twitter API

Twitter is an American social network founded in 2006 that allows users to share short posts (280 characters since 2017), known as tweets, and interact with those of other users through replies, likes, retweets or quotes [27]. Although it has recently incorporated additional payment functions, this social network is free to use and is accessible on multiple platforms. Currently, the social network has 217 million active users daily [23].

On the other hand, Twitter is also known for giving certain facilities to developers to make their products interact with the platform. The company has a Twitter Developer portal where a multitude of resources and useful documentation are posted [21]. This portal contains a description of the API that Twitter offers, how to authenticate, the different endpoints to which queries can be launched, and the associated usage limits.

The Twitter API, currently in its second version, allows the user to request and receive a wide variety of data. Depending on the query launched, the user can receive a series of different objects, each with its own fields and parameters:

- **Tweets**. It represents the basic block of communication between Twitter users.
- **Users**. It represents a user account and its metadata.
- **Spaces**. It represents a space (virtual places in Twitter where users can interact in live conversations) and its metadata.
- **Lists**. It represents a Twitter list (used to configure information visualized in the timeline) and its metadata.

- **Media**. It represents any image, video or GIF attached to a tweet and can be obtained by expanding the Tweet object.
- **Polls**. It represents a poll (choices, duration, end-time and results) and can be obtained by expanding the Tweet object.
- **Places**. It represents a place identified in a tweet and can be obtained by expanding the Tweet object.

Of all the endpoints available in the API (manage tweets, user lookup, search spaces, full-archive tweet search...), in this thesis only the Recent Search endpoint has been used, which returns a list of the most recent tweets based on the rules entered in the query. Both the number of tweets to receive and the id or date of the oldest tweet to be returned can be specified, and this endpoint allows receiving up to one hundred tweets per query and includes a pagination token to handle larger results [20].

Spotify API

Spotify is a company of Swedish origin founded in 2006 that provides audio streaming services, currently being one of the companies with the largest number of users among all those that have this type of service. Spotify has a catalog made up of music and podcasts, including more than 82 million songs, distributed through a free service (limited control and periodic announcements) with the option of a premium subscription. Its business model is based on advertising and paying users, and it pays royalties to artists based on the proportion of streaming of their songs compared to the total played [26].

Spotify has a developer portal that provides a wealth of documentation to help design and implement various use cases. Spotify has an API based on a REST architecture with different published endpoints that return metadata of artists, albums and songs from its own catalog, as well as information on users, lists and music saved by them, in JSON format [16].

The Spotify API has several endpoints to which queries can be sent to collect or modify information: Albums, Artists, Shows, Episodes, Search [18], Tracks [17], Users, Playlists, Categories, Genres, Player and Market. During this thesis the following have been used:

- **Search**. Search for Item allows to obtain information about the Spotify catalog of artists, songs, albums, playlists, shows or episodes.

You can specify the type or types of objects to return, in this case being the type “tracks”. Only one search can be performed per query.

- **Tracks**. Get Tracks’ Audio Features allows to obtain the characteristics of a set of songs specified by their id. The characteristics returned are as follows:
  - **Acousticness**. Confidence measure from 0.0 to 1.0 about whether the track is acoustic, with 1.0 representing high confidence that it is acoustic.
  - **Danceability**. It describes with a value between 0.0 and 1.0 how suitable a track is for dancing based on a combination of its musical elements, with 1.0 being the greatest danceability.
  - **Duration_ms**. It represents the duration of the track in milliseconds.
  - **Energy**. It represents with a value between 0.0 and 1.0 the conception of the energy level of the track, being 1.0 the maximum energy value.
  - **Instrumentalness**. It predicts with a value between 0.0 and 1.0 whether or not the track contains vocals. Values above 0.5 usually represent tracks without vocals, and the closer to 1.0 the more likely they are.
  - **Key**. It indicates the key (in a musical context, the dominant scale) the track is in, with each key having an assigned integer starting with 0. If no key is detected, the value is -1.
  - **Liveness**. It represents audience presence with a value between 0.0 and 1.0. Values greater than 0.8 indicate a high probability that the track was recorded live.
  - **Loudness**. Indicates the average volume of a track in decibels, with values generally contained between -60dB and 0dB.
  - **Mode**. Indicates the modality of the track, being the value 1 greater and 0 less.
  - **Speechiness**. Detects the presence of spoken words in a track. Values less than 0.33 typically indicate instrumental tracks without vocals, values between 0.33 and 0.66 songs with music and vocals, and values greater than 0.66 podcasts, audiobooks, and similar formats.
  - **Tempo**. Indicates the tempo or rhythm of a track in beats per minute.

– **Time_signature**. Represents the estimated time signature value, with values between 3 and 7 indicating 3/4 and 7/4 time signatures, respectively.
– **Valence**. Indicates with values between 0.0 and 1.0 the musical positivity transmitted by the track, where high values indicate greater positivity, while low values indicate greater negativity.

3.4 Orchestrator

The great variety of applications and services that exist in technological environments, where there are workflows with various actors with interdependencies between them, make their management and automation enormously complex. The more complex a system is, the more difficult it is to manage the intervening factors [7].

System automation usually improves efficiency, simplifies management, and reduces associated costs, both in terms of time spent and personnel required to control it. On the other hand, a distinction is made between automation and orchestration in that the former refers to a single task, while the latter comprises multi-step processes and workflows, being the scope of work of the orchestrators.

Two main orchestrators have been used in this project: Docker Compose, which acts as an orchestrator for the work environment containers, and Apache Airflow, which orchestrates the project's workflow.

3.5 NoSQL Databases

A database is a set of data belonging to the same context and stored for later use, and can be updated periodically. The best known type of databases are relational databases. In a relational database, the data attributes are stored in the form of columns, previously defined, and the values are stored in the rows of the table for all its columns or attributes. These databases have an associated query language called SQL (Structured Query Language).

Relational database properties are summarized in ACID properties:

- **A**tomicity. The process is done completely or it is not done.
- **C**onsistency. Only valid data is written.
- **I**solation. The operations are performed one at a time.

- **D**urability. When an operation is performed, it persists and is not undone even if the system crashes.

However, in the face of the massive volumes of data that are associated with the concept of Big Data, relational databases have a series of limitations:

- Reading the data is expensive, since the data is represented in tables, queries involve joining large data sets and filtering the results.
- The stored information usually has similar structures, a concept that does not agree well with Big Data, where the variety of data structure is greater.
- Scalability is not their strongest factor, since they were initially designed considering a single server or, at most, having replicas and load balancing.

Distributed databases are limited by the CAP theorem:

- **C**onsistency. The information remains coherent and consistent after any operation, with all copies having the same data at all times.
- **A**vailability. The system continues to function even if any of its nodes or parts of the software or hardware fail, and all reads and writes complete successfully.
- **P**artition tolerance. The system nodes will continue to function even if the connection between them fails or messages are lost, maintaining their properties.

According to CAP's theorem, any distributed database with shared data among its nodes can have at most two of the three properties at the same time. This theorem resulted in databases with relaxed ACID properties, that is, with BASE properties:

- **B**asically **A**vailable: The store works most of the time, even if failures occur.
- **S**oft-State: Stores or their replicas do not have to be consistent at all times.

- **E**ventually Consistent: consistency happens eventually, as it is something that is taken for granted at some point in the future. All copies will gradually become consistent if no further updates are run.

Non-relational databases or NoSQL (Not Only SQL), a term introduced by Carl Strozzi in 1998 that describes all those databases that do not follow the same design patterns as relational databases. Non-relational databases follow the BASE properties and have advantages such as:

- They do not require a fixed data schema.
- The data is replicated on multiple similar nodes and can be partitioned.
- They are horizontally scalable, that is, by adding new nodes.
- They are relatively inexpensive and simple to implement, with a host of open source alternatives.
- They provide fast read and write speeds, with fast key-value access.

However, the main disadvantages of non-relational databases is that they do not support certain features of relational databases (join, group by, order by...) except within their partitions, they do not have a query language standard such as SQL and its relaxed ACID or BASE properties give lesser guarantees.

Non-relational databases are mainly divided into four groups:

- **Key/value**. Their data model is very simple, since they only store keys and values. They are very similar to a hash table, they are fast, they have great ease of scaling, eventual consistency and fault tolerance, although they cannot support complex data structures.
- **Column oriented**. Data is stored in columns instead of rows. The data is semi-structured, easily distributable, provides high reading speed, calculations on attributes are faster (especially aggregations such as averages) and are perfect when you want to do many operations on large data sets, but they are not the most efficient for writing or when you want to retrieve all records. The data model has columns, super columns, column families, and super column families.

- **Document oriented**. These are key-value stores in which the value is stored as a document with a defined format, so the final data model is collections of documents with a key-value structure (JSON, PDF, XML...). They are schema-less, highly scalable, programmer-friendly, and support rapid development.

- **Graph oriented**. They represent information as the nodes of a graph and their relationships as edges, using graph theory to traverse it. Their strength is the analysis of the relationships between their objects and they represent hierarchical information very well, but they are not particularly good for scaling and tend to have a higher learning curve.

3.6 Containers

Containerization is the packaging of code together with its dependencies, configurations and libraries to form a lightweight executable that can be executed in any infrastructure regardless of its system [9]. In this way, developers can focus on developing applications safely and quickly without worrying about subsequent execution, since the code they develop will be compiled into a package with all its dependencies, abstracting it from the operating system, isolating it and making it portable.

The main advantages of containerization are summarized in:

- **Portability**. The container's abstraction from the host operating system allows it to run consistently on any platform.

- **Agility**. The emergence of open source container engines like Docker has made it easier to integrate with DevOps elements and to run on different operating systems.

- **Speed**. Containers are light and fast to run due to their lack of an operating system and their limited content.

- **Fault isolation**. Each container is isolated and runs independently of the rest, so failures do not propagate between them.

- **Efficiency**. The container software shares the kernel of the operating system of the machine and the application layer can be shared between containers, making better use of system resources.

- **Ease of management**. Orchestrators make it extremely easy to install, manage, scale, and maintain containers, and the simplicity of containers also works in its favor.
- **Security**. Isolating containers acts as a security barrier against the spread of malware throughout the container environment.

A container is considerably lighter than a virtual machine, since it contains only an application and the elements necessary for its execution, while virtualization includes the entire operating system. The container has an engine to be executed and an orchestrator is usually used to manage several containers and their interconnections.

3.7 Continuous Integration / Continuous Delivery

Continuous integration / continuous development or CI/CD is a software development and delivery method based on the introduction of automation in the stages of the development process, allowing work on iterables of the project subjected to testing phases. Continuous integration refers to the automation of development processes and building iterations, while continuous development refers to the continuous delivery of software and its deployment to the production environment [8].

A well-constructed CI/CD cycle helps developers merge new changes with the original project, as well as validate changes to ensure no new deficiencies or bugs are introduced into the product. Each functionality added to the main repository is tested both unitarily and functionally in an automated way, including these tests and allowing a quick analysis of possible conflicts before launching the new iteration of the product.

3.8 Template engines

Section explaining Template engines -> Jinja.

Techniques and tools

In this section are presented the methodological techniques and development tools used to carry out the project.

4.1 GitHub

GitHub [2] is a collaborative development platform created in 2008 and based on the Git version control system. Github has a freemium model and provides the ability to host both public and private projects, focusing primarily on code development [25].

GitHub is the repository in which the project has been managed and hosted, and Git is the version control through which the commits have been made from the local environment. For the agile methodology, the Milestones have been used as sprints and the Issues as the tasks to be carried out in each of them.

4.2 Postman

Postman [3] is a tool that allows the user to build and use APIs in a simple way. Some of its characteristics are the API repository (easier storage, cataloging and collaboration), the availability of tools to help in the API design, testing and documentation, the workspaces to organize the work, and built-in integrations with tools such as GitHub, Azure DevOps, Jenkins, Splunk, Slack and Microsoft Teams. In addition, Postman is based on open source technologies, which provides the ability to be easily extended [13].

[2] https://github.com/
[3] https://www.postman.com/

4.3 Apache Airflow

Apache Airflow [4] is a service orchestrator that allows you to plan, manage, and monitor workflows [1]. It was created in 2014 by Airbnb with the aim of handling the company's huge data flows, and published in 2015 under an open source license. In March 2016 the project joined the Apache Software Foundations incubator and was published as a top level project in 2019.

Airflow is used to automate jobs by breaking them down into smaller tasks. For example, this project uses Airflow to automate the ETL process that consumes data from Twitter and Spotify, processes it, and serves it to the user. Among the main features of Airflow are scalability and ease of integration with other tools.

The main element used by Airflow are the Directed Acyclic Graphs or DAGs, which are groups of tasks connected to each other through dependencies like the nodes of a graph. The word *direct* indicates that the existing relationships in the graph must only have one direction (bidirectional relationships between nodes or tasks are not allowed), while the word *acyclic* means that cycles cannot exist in the graph (nodes or tasks cannot be executed more than once). Tasks are defined by means of an operator and there is a very extensive library with operators that allow defining a wide variety of services such as BashOperator, to execute Bash commands, or SparkSubmitOperator, to submit a task to Spark. Regarding the programming language, Python is the one in which DAGs are developed.

Airflow allows you to have control of the tasks executed through a record of their executions, the time, the current or final status and the generated logs. In addition, it allows certain parameters to be associated with each task, such as, for example, the maximum execution time allowed.

4.4 Apache Spark

Apache Spark [5] is a multi-language engine that emerged in 2009 at the University of California used for data processing and machine learning designed for both simple single-node and distributed architectures [11]. Apache Spark supports the Python, Scala, SQL, Java and R programming languages and allows to design pipelines for large volumes of data for batch and streaming processing. Currently, Spark is one of the most widely used large-scale data processing frameworks.

[4] https://airflow.apache.org/
[5] https://spark.apache.org/

Key features of Apache Spark include its ability to scale, its speed, and the multitude of free resources made available by the large Apache community. In addition, Apache Spark has been adapted to the main cloud solutions, such as Databricks, Amazon Web Services, Google Cloud Platform and Microsoft Azure.

The Apache Spark architecture is mainly composed of:

- **Driver.** Turn code into tasks to distribute to worker nodes. This code goes on to form DAGs, which indicate the order of the tasks and the node in which they are going to be executed.
- **Executors.** They run on worker nodes and execute assigned tasks.

Finally, one of the most characteristic elements of Spark is the Resilient Distributed Dataset or RDD. RDDs are abstractions that represent a collection of immutable objects that can be divided among the different nodes of a cluster and executed in a distributed way.

4.5 Cassandra

Apache Cassandra [6] is a widely used open source non-relational database that emerged in 2008 as a Google code open source project and in March 2010 as an Apache top level project. It is based on a system that acts as a mix between key-value and column-oriented and introduces Cassandra Query Language or CQL, an alternative query language to SQL with a similar syntax [2, 24].

Among its main features are:

- **Scalability**. Its performance increases linearly by adding new nodes, which can be added in real time without affecting system performance.
- **Fault tolerance**. Due to its distributed system architecture, data is automatically replicated across multiple nodes and enables replication and redundancy to be provided across multiple datacenters. Cassandra works with partitions, where each node owns a set of tokens whose ranges determine what data to send to each node within the cluster.
- **Decentralization**. All nodes in the cluster are assigned the same role, with no master node that can act as a single point of failure.

[6] https://cassandra.apache.org/_/index.html

- **Consistency**. Considering the CAP theorem (Consistency-Availability-Partition tolerance), Cassandra is built to be AP, that is, to sacrifice consistency to ensure continuous operation without failures. Cassandra allows you to adjust this level of consistency by selecting the minimum number of nodes that must validate a read or write operation before it is considered successful.

Cassandra was used in the project as the main database to store Twitter and Spotify data.

4.6 Flask

Flask [7] is...

4.7 Bootstrap

Bootstrap [8] is...

4.8 Docker

Docker [9] is an open-source tool that emerged in 2013 that allows you to add an abstraction layer to the deployment of applications by automating them in a container from software [5]. Through Docker, software development can be simplified and applications can be executed in different work environments more easily.

The containers raised by Docker are associated with an image. To build an image in Docker, a file called Dockerfile with a specific syntax is used. Once built, the image will contain the Dockerfile along with the libraries and code specified, and any desired containers can be launched from it.

Docker allows integration with tools such as Ansible and Jenkins, and with cloud providers such as Microsoft Azure, Google Cloud Platform and Amazon Web Services, among others.

[7] https://flask.palletsprojects.com/en/2.1.x/
[8] https://getbootstrap.com/
[9] https://www.docker.com/

4.9 Docker Compose

Docker Compose [10] is a solution that allows you to manage multiple Docker containers on a single host machine [6].

Docker Compose uses a yaml file to indicate the services you want to build along with the images each one should be based on. In this file you can indicate environment variables, dependencies, ports and even commands that must be launched at startup.

While Docker Compose is the Docker container management solution on a single machine, Docker Swarm is the orchestration tool for managing containers on multi-machine distributed architectures.

[10] https://docs.docker.com/compose/

Relevant aspects of the project

The first step of the project was to assess the feasibility and viability analysis of the concept devised. The author was looking to use two data sources with:

- Real and updated data, preferably related to the social interest.
- The possibility of getting a stream data flow.
- The potential to combine both to get an added value.

Considering the previous points, the author found an interesting option on Twitter and Spotify providers. Both of them provides solid APIs for a fluid development and have the characteristics needed to combine the data collected. Consequently, the author designed the following use case:

1. The Twitter API is consulted to gather the *tweets* with the hashtag `#NowPlaying`.
2. The tweet is cleaned, removing the stopwords and the other hashtags and getting the song name and artist as isolated as possible.
3. The Spotify API is consulted to gather the information of the song identified.
4. The data is joined and moved to the database, ready to be stored and visualized.

During the design phase, the author performed the following tasks:

- Parse the output of both APIs using Postman to create the script to extract the data.
- Identify the most appropriate software tools to meet the project requirements. At this point, Apache Airflow was determined for flow orchestration, Apache Spark for data processing, Apache Cassandra for data storage, Flask and Bootstrap for data visualization, and Docker and Docker Compose for container management of the services.
- ...

During the development phase, the following tasks were performed:

- A Python script (PySpark, Python API for Apache Spark) was created to collect, join, transform and store the data from the Twitter and Spotify APIs.
- Containers for each service were defined in Docker Compose and custom images were created in Docker (if needed):
  - Apache Airflow containers configured for flow orchestration: webserver, scheduler, worker, init, triggerer, redis, postgres, client, and flower.
  - Apache Spark containers were configured for data processing: master and three workers. The Apache Cassandra container was configured for data storage.
  - Linux container was configured for data visualization with Flask and Bootstrap.
- The DAG in Apache Airflow was configured to automate data extraction, transformation, and loading.
- Apache Spark was configured to be able to receive the script sent by Apache Airflow and communicate with Apache Cassandra.
- Apache Cassandra was configured with the database and structure required.

The project development was undertaken following an Agile methodology, with Sprints of 2 weeks of duration being represented as *Milestones* in Github and the tasks as *Issues*.

Related works

This section would be similar to a state of the art of a thesis or dissertation. In a final master's thesis, its presence does not seem so obligatory, although it can be left to the tutor's judgment to include a small commented summary of the works and projects already carried out in the field of the current project.

Conclusions and future work lines

Every project must include the conclusions derived from its development. These can be of a different nature, depending on the type of project, but normally there will be a set of conclusions related to the results of the project and a set of technical conclusions. In addition, it is very useful to make a critical report indicating how the project can be improved, or how work can continue along the lines of the completed project.

Appendixes

Appendix A

Project Plan

A.1 Introduction

The project planning was decided in an initial meeting between the author and his tutor. It was based in an Agile methodology, with two-weeks *sprints* and meetings between the author and his tutor conditioned to their availability.

The project repository was stored in GitHub under the url https://github.com/AdrianRiesco/Data-Engineer-project. Each *sprint* was created as a *milestone*, with the *issues* contained there being the tasks assigned. The *issues* were created to reflect tasks at most eight hours, allowing the author segregate his work and manage each *sprint* better. The author closed an *issue* when the task was finished and a *milestone* when the *sprint* was over, regardless of its state. If a task remained in an open state when a *sprint* reached its planned end date, the *issue* was transfered to the next *milestone*.

A meeting was held by the author and his tutor at the end of each sprint. During these meetings, both of them reviewed the state and development of the tasks of the corresponding sprint and planned the tasks of the next sprint. All the *milestones* and *issues* can be consulted in the project repository.

A.2 Temporary planning

The sprints carried out for the development of the project are described below with their corresponding dates:

Initial meeting. Held on Monday January 31^{st}, it was the start point for the first sprint. During this meeting, the objective of the project, the data source and the tools to be used were validated by both the author and his tutor. The author previously made a research and came with an idea and the tutor exposed his point of view to create the final goal.

Sprint 1. Weeks of January 31^{st} and February 7^{th}. This Sprint had the following tasks assigned:

- Configure the work environment.
- Configure the project memory template.
- Write a draft of the objectives and main goals.
- Write a brief description of the tools selected.
- Write a brief explanation of the selected tools and the work methodology.
- Inspect Twitter API.
- Inspect Spotify API.

The end-of-sprint meeting was held on Wednesday February 16^{th}. Analysis: Most of the activities were realized by the author, excepting the Inspection of the Spotify API. Regarding the Twitter API, the author inspected the output and he concluded that it had the characteristics needed to be used to launch queries to the Spotify API (the tweet could be cleaned to get the song name and artist).

Sprint 2. Weeks of February 14^{th} and February 21^{st}. This Sprint had the following tasks assigned:

- Write the code to gather information from Spotify.
- Write the code to gather information from Twitter.
- Write a description of Spotify and Twitter APIs.
- Write the API inspection process in the “Programmer guide” section.
- Write the project introduction.
- Write the Twitter and Spotify data description.

The end-of-sprint meeting was held on Wednesday February 2nd. Analysis: All the activities were accomplished on time. The author identified some potential problems when extracting information from the Spotify API, such as the name of the song must be quite accurate to be able to get the search results.

Sprint 3. Weeks of February 28th and March 7th. This Sprint had the following tasks assigned:

- Write a description of Twitter API.
- Write a description of Spotify API.
- Write the description of the Twitter data.
- Write the description of the Spotify data.
- Improve and unify the code written to collect data from both APIs.
- Set up the Spark environment using Docker.

The end-of-sprint meeting was held on Wednesday March 16th. Analysis: During this sprint, the author had difficulties configuring the Docker environment, which derived in a delay of the other tasks. These tasks were transfered to the next sprint.

Sprint 4. Weeks of March 14th and March 21st. This Sprint had the following tasks assigned:

- Write a description of NoSQL Databases.
- Set up the Spark environment using Docker.
- Set up the Airflow environment using Docker.

The end-of-sprint meeting was held on Wednesday May 4th. Due to personal reasons, the author could not continue with the project during the month of April, so there was a temporary pause in the planning.

Sprint 5. Weeks of May 2nd and May 9th. This Sprint had the following tasks assigned:

- Configure the DAG in Airflow.
- Learn the fundamentals of Flask.
- Redesign the project plan excluding the month of April

The end-of-sprint meeting was held on Wednesday May 18th.

Sprint 6. Weeks of May 16th and May 23rd. This Sprint had the following tasks assigned:

- Deploy Cassandra environment.
- Write the description of the Twitter data.
- Write the description of the Spotify data.
- Write a description of an orchestrator.
- Write a description of CI/CD.

The end-of-sprint meeting was held on Monday May 30$^{\text{th}}$.

Sprint 7. Weeks of May 30$^{\text{th}}$ and June 6$^{\text{nd}}$. This Sprint had the following tasks assigned:

- Integrate Cassandra with Airflow and Spark.
- Create the ETL workflow to load the data to Cassandra.
- Write a description of Apache Spark.
- Write a description of Docker and Docker Compose.
- Write a description of Flask, Jinja and Bootstrap.

The end-of-sprint meeting was held on Tuesday June 14$^{\text{th}}$.

Sprint 8. Weeks of June 13$^{\text{th}}$ and June 20$^{\text{th}}$. This Sprint had the following tasks assigned:

- Create the front-end with Flask and Bootstrap.
- Integrate the ETL workflow with the front-end.
- Write section 5 "Relevant aspects of the project".
- Write section 7 "Conclusions of the project".

The end-of-sprint meeting was held on M— June –$^{\text{th}}$.

Sprint 9. Weeks of June 27$^{\text{rd}}$ and July 4$^{\text{th}}$. This Sprint had the following tasks assigned:

- Task1.

The end-of-sprint meeting was held on M— July –$^{\text{th}}$.

A.3 Feasibility study

The architecture of the project and the use case were designed to ensure its feasibility.

Tools	Twitter	Spotify	Apache	BSD-3-Clause	MIT	GPL
Twitter API	X					
Spotify API		X				
Apache Airflow			X			
Apache Spark			X			
Apache Cassandra			X			
Flask				X		
Bootstrap					X	
TeXMaker						X

Table A.1: Tools and technologies' licenses

Economic feasibility

The project is based on open-source platforms to ensure its economic and legal feasibility. The APIs where the information was gathered are free to use if the developer keeps his queries under specific limit rates.

Legal feasibility

The project is based on open-source platforms to ensure its economic and legal feasibility. The licenses in which the technology and tools used in the project are based are:

- **Twitter License.** Free with usage limitations and requires developer account [22].
- **Spotify License.** Free with usage limitations and requires developer account [15].
- **Apache License.** Free with limitations [3].
- **Flask License.** Free with limitations, uses BSD-3-Clause license [14].
- **Bootstrap License.** Free with limitations, uses Massachusetts Institute of Technology (MIT) license [19].
- **Texmaker License.** Free with limitations, uses GNU General Public License (GPL) license [4].

Appendix B

Requirements

B.1 Introduction

This section lists the general objectives and requirements identified during the initial planning of the project and on whose fulfillment the development of the project has focused.

B.2 General objectives

The requirements through which the use case was built were the following:

- Ability to obtain data in real time.
- Combine at least two different data sources.
- Potential to scale in both technology and data volume.
- Involve various technologies from the world of Big Data.
- Mostly open source tools.

B.3 Catalog of requirements

The functional requirements that the project had to meet were:

- **F1**. The data must be obtained from the Twitter hashtag `#NowPlaying` every 30 minutes, taking care of API rate limits.

- **F2**. The visualizations must show last songs name, artist and audio features.
- **F3**. The visualization must have a link to the source tweet.

The technical requirements that the project had to meet were:

- **F1**. Ability to be deployed in different environments with minimum effort.
- **F2**. Automated data flow, with hole process orchestrated by a unique tool.
- **F3**. Data warehouse with ability to escalate in terms of a Big Data problem.

B.4 Requirements specification

Appendix C

Design specification

C.1 Introduction

C.2 Data design

Twitter data structure

The data received from Twitter queries has the following structure (the data has been obtained from a real query and its output has been reduced by trimming certain elements due to their length):


```
{"data":[
      {
         "id":"1533311938209382403",
         "entities":{
            "annotations":[
               {
                  "start":43,
                  "end":62,
                  "probability":0.6061,
                  "type":"Other",
                  "normalized_text":"Breakfast In America"
               }
            ],
            "urls":[
               {
                  "start":84,
```

```
                    "end":107,
                    "url":"https://t.co/YiXxSepm8x",
                    "expanded_url":"https://rideshare.airtime.pro",
                    "display_url":"rideshare.airtime.pro",
                    "images":[
                        {
                            "url":"https://pbs.twimg.com/news_img/1532561096858640397/3mUiSDDN?format=jpg&name=orig",
                            "width":1920,
                            "height":1200
                        },
                        {
                            "url":"https://pbs.twimg.com/news_img/1532561096858640397/3mUiSDDN?format=jpg&name=150x150",
                            "width":150,
                            "height":150
                        }
                    ],
                    "status":200,
                    "title":"Rideshare Radio",
                    "description":"Hits from the 70's 80s 90s 00s 10s 20s, No Talking just back to back Music totally commercials free 24/7",
                    "unwound_url":"https://rideshare.airtime.pro"
                }
            ],
            "hashtags":[
                {
                    "start":0,
                    "end":11,
                    "tag":"Nowplaying"
                },
```

```
                {
                    "start":129,
                    "end":139,
                    "tag":"Rideshare"
                }
            ]
        },
        "created_at":"2022-06-05T04:57:08.000Z",
        "text":"#Nowplaying 2010 Remastered - Supertramp - Breakfast In America  - Stream here-&gt; https://t.co/YiXxSepm8x - Non Stop Hits 24/7 #Rideshare #Radio #Hits #Uber #RideshareRadio #Petrol #Parcoursup #PlatinumJubilee #ENGvNZ #PrideMonth"
    },
    {
        Second tweet.
    }
],
"meta":{
    "newest_id":"1533311938209382403",
    "oldest_id":"1533311921256124416",
    "result_count":10,
    "next_token":"b26v89c19zqg8o3fpyzltxkhapj3hf1q96mc01w4yl3el"
}
}
```


The fields required for the project are:

- **id.** Tweet id, useful uniquely identify the tweet.
- **text.** Tweet text, useful to identify the song played.
- **entities.** useful to clean the text and remove hashtasg, cashtags, mentions and urls.
- **created_at.** Tweet creation date.

Spotify data structure

The data received from Spotify queries to Search endpoint ("Search for Item") has the following structure (the data has been obtained from a real query and its output has been reduced by trimming certain elements due to their length):


```
{"tracks":{
      "href":"https://api.spotify.com/v1/search?query=Savoy+Brown+Wang+Dang+Doodle+&type=track&offset=0&limit=1",
      "items":[
         {
            "album":{
               "album_type":"album",
               "artists":[
                  {
                     "external_urls":{
                        "spotify":"https://open.spotify.com/artist/17obwOahRWI121iMUZznh2"
                     },
                     "href":"https://api.spotify.com/v1/artists/17obwOahRWI121iMUZznh2",
                     "id":"17obwOahRWI121iMUZznh2",
                     "name":"Savoy Brown",
                     "type":"artist",
                     "uri":"spotify:artist:17obwOahRWI121iMUZznh2"
                  }
               ],
               "available_markets":[
                  "MX",
                  "US"
               ],
               "external_urls":{
                  "spotify":"https://open.spotify.com/album/5oq20r8iNOO9fpw8R2h3vE"
               },
```

```
            "href":"https://api.spotify.com/v1/albums/5oq20r8iNOO9fpw8R2h3vE",
            "id":"5oq20r8iNOO9fpw8R2h3vE",
            "images":[
                {
                    "height":640,
                    "url":"https://i.scdn.co/image/ab67616d0000b2735dd44bf0a252e30d4bb2e7c8",
                    "width":640
                },
                {
                    "height":300,
                    "url":"https://i.scdn.co/image/ab67616d00001e025dd44bf0a252e30d4bb2e7c8",
                    "width":300
                }
            ],
            "name":"Street Corner Talking",
            "release_date":"1971-01-01",
            "release_date_precision":"day",
            "total_tracks":8,
            "type":"album",
            "uri":"spotify:album:5oq20r8iNOO9fpw8R2h3vE"
        },
        "artists":[
            {
                "external_urls":{
                    "spotify":"https://open.spotify.com/artist/17obw0ahRWI121iMUZznh2"
                },
                "href":"https://api.spotify.com/v1/artists/17obw0ahRWI121iMUZznh2",
                "id":"17obw0ahRWI121iMUZznh2",
                "name":"Savoy Brown",
                "type":"artist",
```

```
                    "uri":"spotify:artist:17obw0ahRWI121iMUZznh2"
                }
            ],
            "available_markets":[
                "MX",
                "US"
            ],
            "disc_number":1,
            "duration_ms":440733,
            "explicit":false,
            "external_ids":{
                "isrc":"GBF077120720"
            },
            "external_urls":{
                "spotify":"https://open.spotify.com/track/7p99XDR7dKaIMTYV3zia0V"
            },
            "href":"https://api.spotify.com/v1/tracks/7p99XDR7dKaIMTYV3zia0V",
            "id":"7p99XDR7dKaIMTYV3zia0V",
            "is_local":false,
            "name":"Wang Dang Doodle",
            "popularity":19,
            "preview_url":"None",
            "track_number":7,
            "type":"track",
            "uri":"spotify:track:7p99XDR7dKaIMTYV3zia0V"
        }
    ],
    "limit":1,
    "next":"https://api.spotify.com/v1/search?query=Savoy+Brown+Wang+Dang+Doodle+&type=track&offset=1&limit=1",
    "offset":0,
    "previous":"None",
    "total":11
  }
}
```

The fields required for the project are:

- **id.** Track id, useful uniquely identify the track.
- **name.** Track name, useful to identify the track.
- **popularity.** Popularity of the track.
- **artists' id.** ID of the artists.
- **artists' name.** Name of the artists.

The data received from Spotify queries to the Tracks endpoint ("Get Tracks' Audio Features") has the following structure (the data has been obtained from a real query and its output has been reduced by trimming certain elements due to their length):


```
{"audio_features":[
      {
         "danceability":0.516,
         "energy":0.36,
         "key":7,
         "loudness":-11.264,
         "mode":1,
         "speechiness":0.03,
         "acousticness":0.83,
         "instrumentalness":0.885,
         "liveness":0.116,
         "valence":0.144,
         "tempo":127.176,
         "type":"audio_features",
         "id":"7dg3XqARw7qOrkt9pZZNRF",
         "uri":"spotify:track:7dg3XqARw7qOrkt9pZZNRF",
         "track_href":"https://api.spotify.com/v1/tracks/7dg3XqARw7qOrkt9pZZNRF",
         "analysis_url":"https://api.spotify.com/v1/audio-analysis/7dg3XqARw7qOrkt9pZZNRF",
         "duration_ms":233812,
         "time_signature":4
      },
```

```
        {
            Group of features of the second track.
        }...]
}
```

The fields required for the project are:

- id.
- danceability.
- energy.
- key.
- loudness.
- mode.
- speechiness.
- acousticness.
- instrumentalness.
- liveness.
- valence.
- tempo.
- duration_ms.
- time_signature.

Cleaned data

After the cleaning process, the resulting data structure is:

- id_tweet.
- text.
- created_at.
- url_tweet.

- id_track.
- name.
- popularity.
- artists_id.
- artists_name.
- danceability
- energy
- key
- loudness
- mode
- speechiness
- acousticness
- instrumentalness
- liveness
- valence
- tempo
- duration_ms
- time_signature

C.3 Procedural design

Flow diagram.

C.4 Architectural design

Component diagram.

Appendix D

Programming technical documentation

D.1 Introduction

D.2 Directory structure

The project repository, hosted on GitHub, has the following directory structure:

- `airflow`.
- `doc`. It contains the project report.
- `docker`.
- `spark`.
- `README.md`.

D.3 Programmer's guide

Analysis

During the analysis phase, the author inspected the output of Twitter and Spotify APIs using Postman. In the first place, relying on the Twitter documentation, the author inspected the Twitter API by following the next steps:

1. Get access to the Twitter Developer Portal.
2. Get the credentials needed to consult the different endpoints of the API.
3. Import the *Twitter API v2* collection on Postman.
4. Create a fork of the automatically created environment (*Twitter API v2*) and collection *Twitter API v2* to be able to edit the values.
5. Modify the environment to include the following developer keys and tokens:
   - Consumer key (`consumer_key`).
   - Consumer secret (`consumer_secret`).
   - Access token (`access_token`).
   - Token secret (`token_secret`).
   - Bearer token (`bearer_token`).
6. In the collection tab, select the endpoint *Search Tweets* $\longrightarrow$ *Recent search* for the initial exploration. Configure the following parameters:
   - `query` = `#NowPlaying`
   - `tweet.fields` = `created_at,entities`
   - `max_results` = `10`
7. Now, we can send our query `https://api.twitter.com/2/tweets/search/recent?query=%23NowPlaying&max_results=10&tweet.fields=created_at,entities` to get the 10 most recent tweets with the hashtag ***#NowPlaying*** and receive their basic information (id, text) as well as the entities (hashtags, urls, annotations...) and the creation time stamp.

After analyze the data gathered from the Twitter API, the author inspected the Spotify API (more specifically the endpoint "Search for Item"), following the next steps:

1. Get access to the Spotify Developer Portal.
2. Enter in the developers console and select the "Search for Item" endpoint.

3. Specify the parameters of the search. We can specify the type "track" and add a limit of one to only receive the first song found.

4. After click on get the bearer token, we can use that token by clicking on Try Me or just copy the resulting query in our Linux console to check the output.

5. The result of this query is the first result of the search containing information of the artist, the song and the album. With the artist and song ids we can consult other endpoints to get an audio analysis, the audio features and the artist information, between others.

Development

During the development phase, the following items were installed in the system:

- Docker version 20.10.12, build e91ed57.
- docker-compose version 1.29.2, build 5becea4c.
- docker-compose version 1.29.2, build 5becea4c.

Steps followed:

1. Create the Docker Compose file.
2. Launch the environment with `sudo docker-compose up -remove-orphans`.
3. ***Build the extended image: `sudo docker build . -f Dockerfile -pull -tag extended/airflow:2.3.0`.
4. ***Build the extended image: `sudo docker-compose build`.
5. ***Execute airflow-init: `sudo docker-compose up airflow-init`.
6. ***`sudo docker-compose up`.
7. ***In Airflow UI, create Spark connection.

D.4 Compilation, installation and execution of the project

The instructions to execute the project are the following ones:

1. Create the Docker Compose file.
2. Launch the environment with `sudo docker-compose up -remove-orphans`.

D.5 System tests

To test that the project is working in a proper way, there are a few tests that can be performed:

1. Test 1.
2. Test 2.
3. Test 3.

Appendix E

User documentation

E.1 Introduction

This section summarizes the elements that the user must have and the steps that must be followed to correctly execute the project.

E.2 User requirements

E.3 Installation

E.4 User's manual

Bibliography

[1] Apache. Apache airflow, 2022. https://airflow.apache.org (March 19th 2022).

[2] Apache. Apache cassandra, 2022. https://cassandra.apache.org/_/index.html (May 20th 2022).

[3] Apache. Apache license version 2.0, 2022. https://www.apache.org/licenses/LICENSE-2.0 (March 5th 2022).

[4] Pascal Brachet. Texmaker, 2022. https://www.xm1math.net/texmaker/ (February 5th 2022).

[5] Inc. Docker. Docker, 2022. https://www.docker.com (February 19th 2022).

[6] Inc. Docker. Docker compose, 2022. https://docs.docker.com/compose/ (February 19th 2022).

[7] Red Hat. Orchestrator, 2022. https://www.redhat.com/es/topics/automation/what-is-orchestration (March 19th 2022).

[8] Red Hat. Orchestrator, 2022. https://www.redhat.com/en/topics/devops/what-is-ci-cd (May 17th 2022).

[9] International Business Machines Corporation (IBM). Containerization, 2022. https://www.ibm.com/cloud/learn/containerization (March 19th 2022).

[10] International Business Machines Corporation (IBM). Est apis, 2022. https://www.ibm.com/cloud/learn/rest-apis#:

~:text=An%20API%2C%20or%20application%20programming,representational%20state%20transfer%20architectural%20style. (February 12th 2022).

[11] InfoWorld. Apache spark, 2022. https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html (March 19th 2022).

[12] KDnuggets. The 42 v's of big data, 2022. https://www.kdnuggets.com/2017/04/42-vs-big-data-data-science.html (February 5th 2022).

[13] Inc. Postman. Postman, 2022. https://www.postman.com (February 6th 2022).

[14] Pallets Projects. Flask license 2.1, 2022. https://flask.palletsprojects.com/en/2.1.x/license/ (March 26th 2022).

[15] Spotify Technology S.A. Spotify developer terms, 2022. https://developer.spotify.com/terms/ (February 5th 2022).

[16] Spotify Technology S.A. Spotify for developers, 2022. https://developer.spotify.com (February 5th 2022).

[17] Spotify Technology S.A. Spotify for developers - get tracks' audio features, 2022. https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features (February 12th 2022).

[18] Spotify Technology S.A. Spotify for developers - search for item, 2022. https://developer.spotify.com/documentation/web-api/reference/#/operations/search (February 12th 2022).

[19] Bootstrap Team. Bootstrap license, 2022. https://github.com/twbs/bootstrap/blob/main/LICENSE (March 26th 2022).

[20] Inc. Twitter. Recent search, 2022. https://developer.twitter.com/en/docs/twitter-api/tweets/search/introduction (February 19th 2022).

[21] Inc. Twitter. Twitter developer, 2022. https://developer.twitter.com (February 5th 2022).

[22] Inc. Twitter. Twitter developer agreement and policy, 2022. https://developer.twitter.com/es/developer-terms/agreement-and-policy (February 5th 2022).

[23] Variety. Twitter daily users 2021, 2021. https://variety.com/2022/digital/news/twitter-q4-2021-earnings-users-growth-1235176882 (February 26th 2022).

[24] Wikipedia. Apache cassandra, 2022. https://es.wikipedia.org/w/index.php?title=Apache_Cassandra&oldid=143581116 (May 20th 2022).

[25] Wikipedia. Github, 2022. https://es.wikipedia.org/w/index.php?title=GitHub&oldid=143733693 (May 27th 2022).

[26] Wikipedia. Spotify, 2022. https://en.wikipedia.org/w/index.php?title=Spotify&oldid=1077879878 (February 19th 2022).

[27] Wikipedia. Twitter, 2022. https://en.wikipedia.org/w/index.php?title=Twitter&oldid=1075330090 (February 19th 2022).