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 — March 19, 2022
Tutor: 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, March 19, 2022

Approval of the Tutor:

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

Contents iii

List of Figures vi

List of Tables vii

Memory 1

1. Introduction 3

2. Project objectives 5

3. Theoretical concepts 7
- 3.1 API . . . 7
- 3.2 Orchestrator . . . 10
- 3.3 NoSQL Databases . . . 10
- 3.4 Containers . . . 10
- 3.5 Continuous Integration / Continuous Delivery . . . 10
- 3.6 Template engines . . . 11
- 3.7 Web Server Gateway Interface . . . 11
- 3.8 Tables . . . 11

4. Techniques and tools 13
- 4.1 GitHub . . . 13
- 4.2 APIs . . . 13
- 4.3 Postman . . . 13
- 4.4 Apache Airflow . . . 13

4.5 Apache Spark . . . 13
4.6 Cassandra . . . 14
4.7 Flask . . . 14
4.8 Bootstrap . . . 14
4.9 Docker . . . 14
4.10 Docker Compose . . . 14

5. Relevant aspects of the project **15**

6. Related works **17**

7. Conclusions and future work lines **19**

Appendix **20**

Appendix A Project Plan **23**
A.1 Introduction . . . 23
A.2 Temporary planning . . . 23
A.3 Feasibility study . . . 26

Appendix B Requirements **27**
B.1 Introduction . . . 27
B.2 General objectives . . . 27
B.3 Catalog of requirements . . . 27
B.4 Requirements specification . . . 28

Appendix C Design specification **29**
C.1 Introduction . . . 29
C.2 Data design . . . 29
C.3 Procedural design . . . 29
C.4 Architectural design . . . 29

Appendix D Programming technical documentation **31**
D.1 Introduction . . . 31
D.2 Directory structure . . . 31
D.3 Programmer's guide . . . 31
D.4 Compilation, installation and execution of the project . . . 33
D.5 System tests . . . 33

Appendix E User documentation **35**
E.1 Introduction . . . 35

E.2 User requirements 35
E.3 Installation 35
E.4 User's manual 35

Bibliography **37**

List of Figures

List of Tables

3.1 Tools and technologies used 11

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 analysis 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, each new development becoming a social phenomenon. With the aim of taking advantage of both worlds, in this project the social network Twitter will be used to obtain information on the latest music listened to by users (by searching for a certain hashtag) and subsequently 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.

Project objectives

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

- 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.
- Use of open source tools.

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, 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 process in *Apache Spark*.
- Store all the data in a data warehouse under a known technology, *Cassandra*.
- 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* 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 API

An Aplication 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 [1].

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 [4]. 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 [3].

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[2]. 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 [**?**].

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 [**?**].

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[**?**].

The Spotify API has several endpoints to which queries can be sent to collect or modify information: Albums, Artists, Shows, Episodes, Search [**?**], Tracks [**?**], 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".
- **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 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.

3.2 Orchestrator

Section explaining Flow Orchestrator -> Airflow.

3.3 NoSQL Databases

Section explaining NoSQL Databases.

3.4 Containers

Section explaining Containers.

3.5 Continuous Integration / Continuous Delivery

Section explaining CI/CD.

3.6 Template engines

Section explaining Template engines -> Jinja.

Tools	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
TEXMaker				X
Astah				X

Table 3.1: Tools and technologies used

3.7 Web Server Gateway Interface

Section explaining Web Server Gateway Interface (WSGI).

3.8 Tables

TablaSmall.

Techniques and tools

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

4.1 GitHub

GitHub is the repository where the project was uploaded and its evolution was tracked.

4.2 APIs

During this project there were used APIs from two different providers to gather the information: Twitter API and Spotify API.

4.3 Postman

Postman is a tool that allows the user to send HTTPS request in a simple way.

4.4 Apache Airflow

Apache Airflow is a flow orchestrator that allows the user to...

4.5 Apache Spark

Apache Spark is...

4.6 Cassandra

Cassandra is a NoSQL database that...

4.7 Flask

Flask is...

4.8 Bootstrap

Bootstrap is...

4.9 Docker

Docker is...

4.10 Docker Compose

Docker Compose is...

Relevant aspects of the project

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

- Real and updated data, preferable 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 vector values of the cleaned Twitter data and the name of the song returned by Spotify are compared to ensure they are the same.
5. The data is moved to the database, ready to be stored and visualized.

During the design phase, the author analyzed the output of both APIs using Postman.

The project development was undertaken following an Agile methodology.

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.

Appendix

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 31st, 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 31st and February 7th. 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 16th. 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 14th and February 21st. 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 28$^{\text{th}}$ and March 7$^{\text{th}}$. 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 16$^{\text{th}}$.

Sprint 4. Weeks of March 14$^{\text{th}}$ and March 21$^{\text{st}}$. This Sprint had the following tasks assigned:

- Task1.

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

Sprint 5. Weeks of March 28$^{\text{th}}$ and April 4$^{\text{th}}$. This Sprint had the following tasks assigned:

- Task1.

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

Sprint 6. Weeks of April 11$^{\text{th}}$ and April 18$^{\text{th}}$. This Sprint had the following tasks assigned:

- Task1.

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

Sprint 7. Weeks of April 25$^{\text{th}}$ and May 2$^{\text{nd}}$. This Sprint had the following tasks assigned:

- Task1.

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

Sprint 8. Weeks of May 9th and May 16th. This Sprint had the following tasks assigned:

- Task1.

The end-of-sprint meeting was held on M— May –th.

Sprint 9. Weeks of May 23rd and May 30th. This Sprint had the following tasks assigned:

- Task1.

The end-of-sprint meeting was held on M— June –th.

Sprint 10. Weeks of June 6th and May 13th. This Sprint had the following tasks assigned:

- Task1.

The end-of-sprint meeting was held on M— June –th.

A.3 Feasibility study

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

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.

Appendix B

Requirements

B.1 Introduction

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 15 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.

F4 -

The technical requirements that the project had to meet were:

T1 Ability to be deployed in different environments with minimum effort.

T2 Automated data flow, with hole process orchestrated by a unique tool.

T3 Data warehouse with ability to escalate in terms of a Big Data problem.

T4 -

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:

Spotify data structure

Thhe data received from Spotify queries has the following structure:

Cleaned data

After the cleaning process, the resulting data structure...

C.3 Procedural design

Flow diagram.

C.4 Architectural design

Component diagram.

Appendix D

Programming technical documentation

D.1 Introduction

D.2 Directory structure

Estructura GitHub.

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 dockerfile.
2. Build the image: `docker build -f spark.Dockerfile -t ar/spark-ubuntu .`
3. Create a Docker network: `docker network create -driver bridge spark-network`
4. Run the image: `sudo docker run -d -t -name master -network spark-network ar/spark-ubuntu`

D.4 Compilation, installation and execution of the project

D.5 System tests

Appendix E

User documentation

E.1 Introduction

E.2 User requirements

E.3 Installation

E.4 User's manual

Bibliography

[1] International Business Machines Corporation (IBM). Ibm - rest apis, 2022. [Internet. February 12th 2022].

[2] Inc. Twitter. Twitter developer, 2022. [Internet. February 5th 2022].

[3] Variety. Twitter daily users 2021, 2021. [Internet. February 26th 2022].

[4] Wikipedia. Twitter, 2022. [Internet. February 19th 2022].