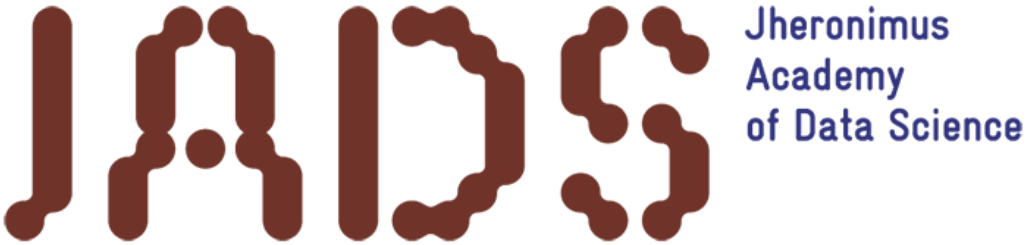


JM0140-M-6 - Data Engineering
Assignment 2
Group 12

Krzysztof Wiesniakowski 2124849
Kyriakos Koukiadakis 2074817
Xavier Paulus 2047901
Vishal Seghal 2109374
Chigozie Ifepe 2109365

# 1.Overview of the Data Pipelines

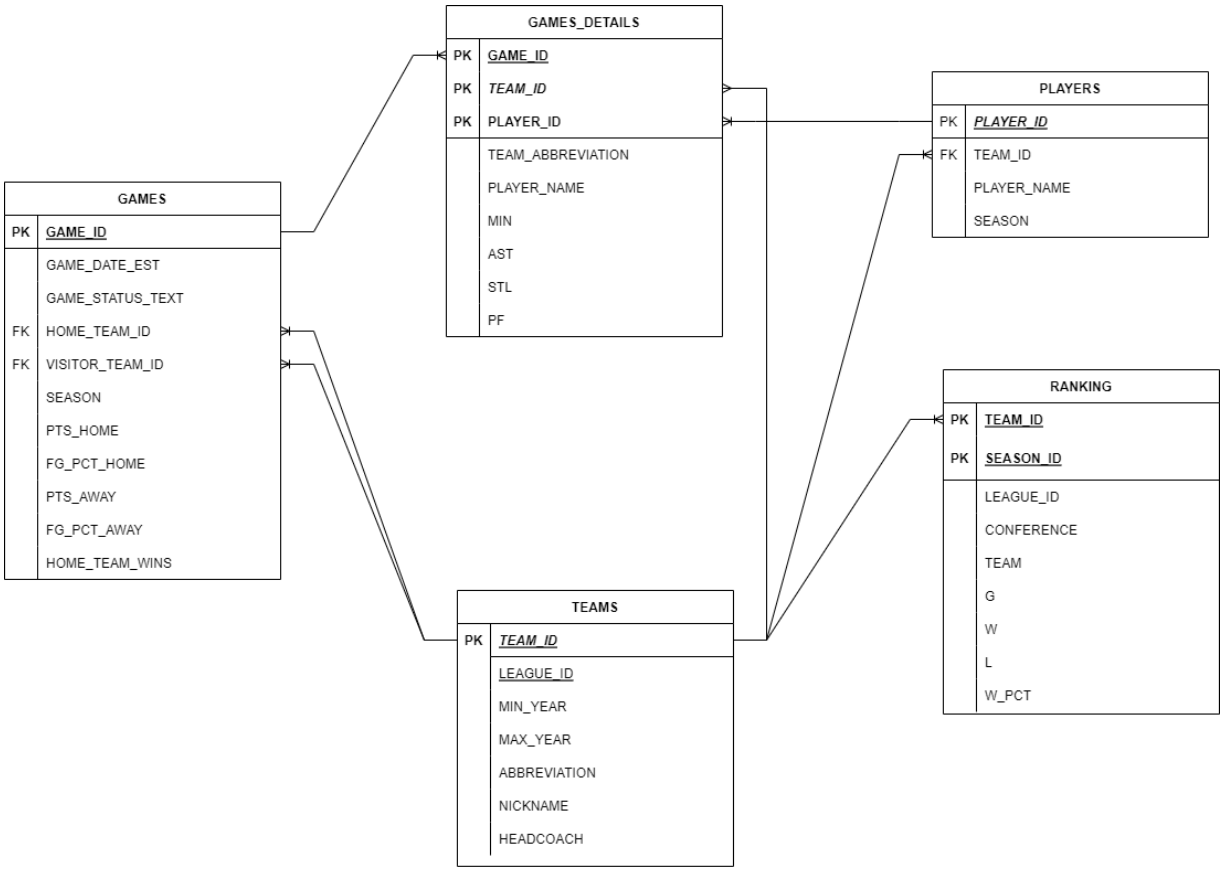


Figure 1: Entity Relationship Diagram

The objective of this task is to construct two data pipelines that offer compelling statistical insights intended for stakeholders in this case NBA enthusiasts. These pipelines aim to deliver comprehensive and current information, utilizing sophisticated visualizations and analytical approaches to present captivating statistics. The focus lies in providing enhanced insights and up-to-date data that offer compelling and engaging information for stakeholders invested in the NBA domain.

Data has been taken from Kaggle [1]. This is a dataset with all NBA games from the 2004 season to Dec 2022. Some data was found redundant for the scope of this assignment and for the sake of simplifying data structure some columns have been dropped. The tables as well as their relationship can be seen in Figure 1.

The first pipeline is a batch processing pipeline. This pipeline takes a batch of data uploaded to the GCS bucket and finds how many times each team finished in the top 3 in the NBA (regardless of the conference) based on points scored home. As it can be seen in Figure 2, each place on the podium is represented by a different color.

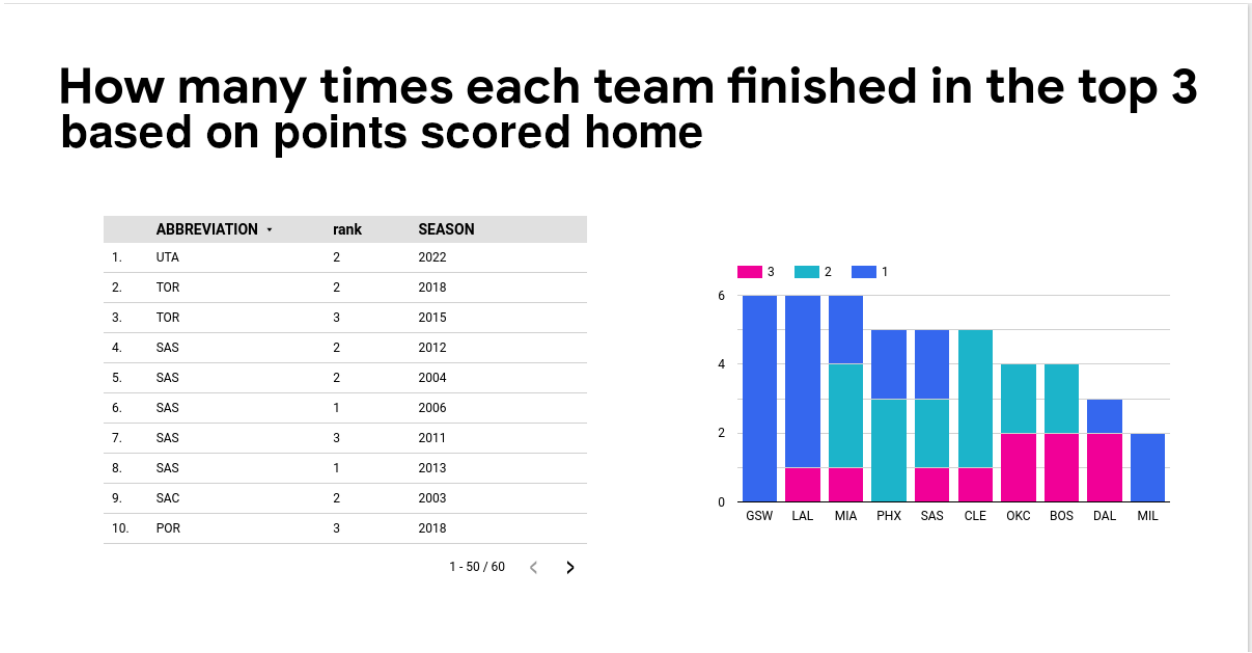


Figure 2: Visualization of the batch processing pipeline results

The second pipeline retrieves the NBA players in descending order based on their scored points. It is important to say that it is dummy data produced by ChatGPT which is personalized according to the information provided from the Entity Relationship diagram.

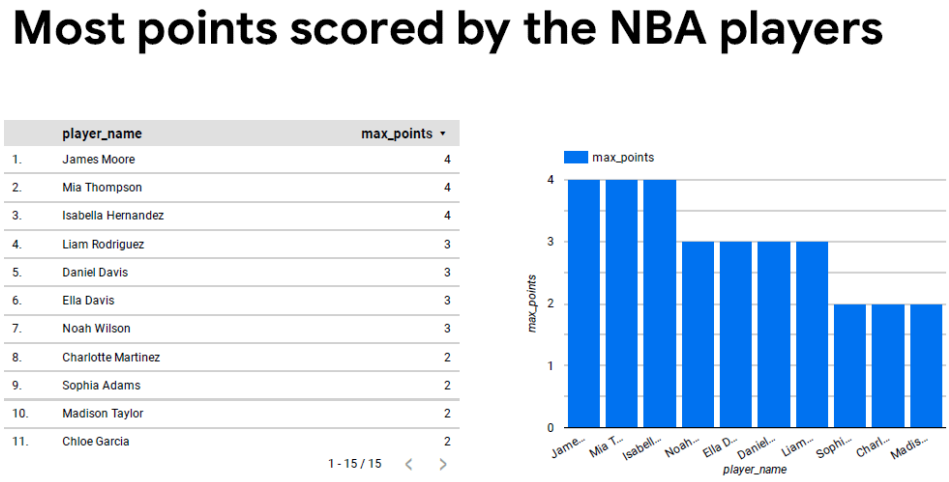


Figure 3: Visualization of the stream processing pipeline results

## 2.Design and Implementation of Data Architecture

The data architecture is similar across both pipelines. It operates within the Google Cloud environment, utilizing Google Cloud data storage, BigQuery, Looker, and the Compute Engine. The Spark application, along with its master and workers, resides on a virtual machine hosted in

Google Cloud. One worker was employed, dedicated to a specific pipeline. For the second pipeline, Kafka has been used to produce and continuously consume the data.

## 2.1 Batch processing:

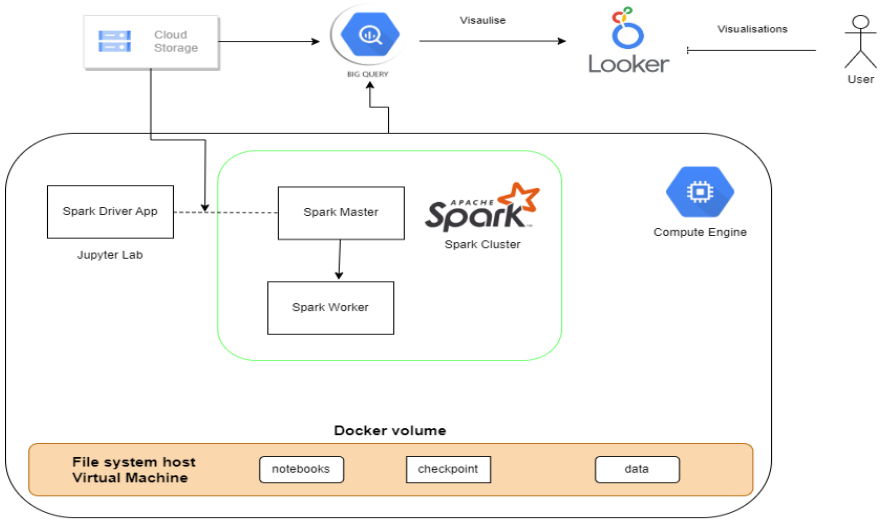


Figure 3: Batch processing data architecture

## 2.2 Stream processing

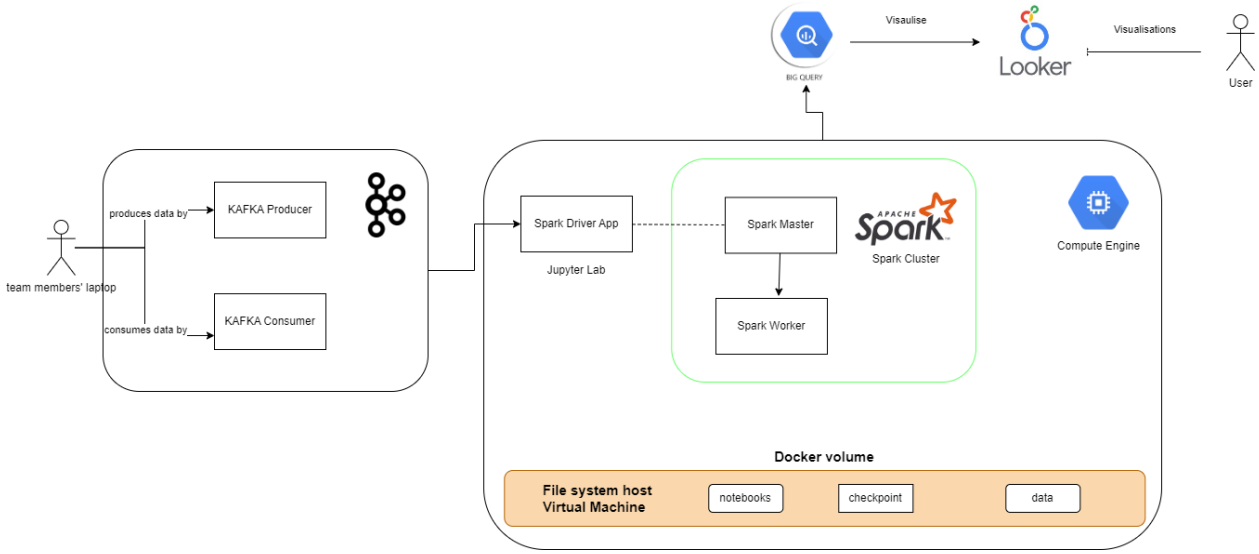


Figure 4: Stream processing data architecture

In the second pipeline data is produced by Kafka Producer and consumed by Kafka Consumer. Kafka has been chosen for that assignment because its architecture and capabilities make it a robust choice for handling large volumes of data, facilitating real-time data processing, and providing a reliable messaging system for modern data-driven applications. In this assignment both Kafka programs are run locally on team members' laptops and triggered manually.  Data

produced by Kafka Producer is in a JSON format with filled in fictional values as described in the previous section. Kafka Producer produces data only about games_details so it uses only part of the data from the tables.

The selection of BigQuery data for both pipelines output heavily relied on its integration with Looker Studio. Through BigQuery, data seamlessly feeds into Looker Studio, with any alterations to BigQuery tables instantly reflecting in Looker Studio. The ultimate visualization is crafted by utilizing the two tables derived from the preprocessing pipelines.

# 3.Design and Implementation of Data Pipelines

## 3.1 Batch processing implementation:

For the batch processing the design of the pipeline is depicted in the Figure below:

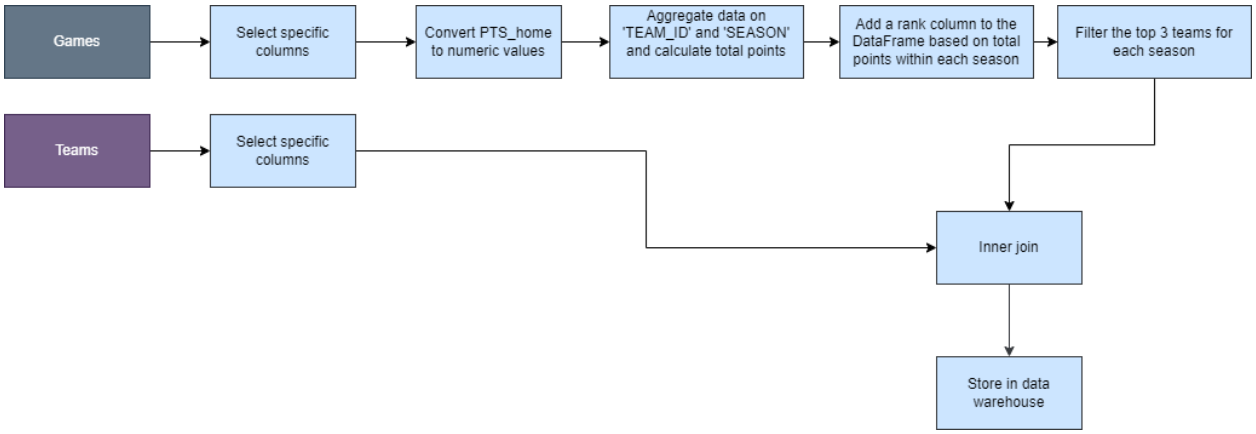


Figure 5: Visualization of batch processing pipeline

This pipeline begins by defining the data structure and creating a Spark Session for processing. It then retrieves data from a GCS bucket, filters it based on selected columns, converts pts_home to numeric values and aggregates it by team and season. Window specifications are established to partition the data. A rank column is added, determining team ranks based on total points. Finally, the pipeline filters the top three teams for each season, focusing on the highest-performing teams in every season's dataset.

## 3.2 Stream processing implementation:

For the stream processing the design of the pipeline is depicted in the Figure below:

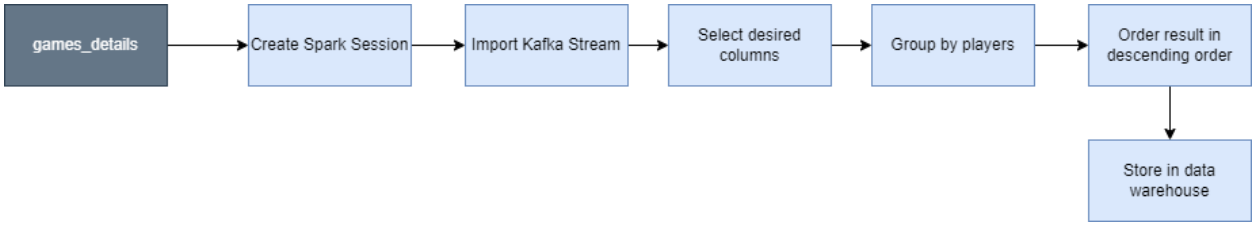


Figure 6: Visualization of batch processing pipeline

Because the stream processing pipeline is about real-time or near-real-time handling data and analysis of data as it flows through a system the focus of this pipeline is not to compute complex aggregate operations but to provide quickly relevant information in a nice visual way. After defining the schema and creating a spark session, the kafka stream is imported. Then relevant columns are chosen and aggregation is performed using the `groupBy()` function (in the current implementation, these are players who scored the most points). After aggregation, the result is ordered based on the maximum points scored by the player and it is queried to bigQuery.

# 4.Reflection on Design and Implementation of Data Architecture and Data Pipelines

Since the current batch processing pipeline has relatively few steps and can be considered simple there is not much room for changes and improvement with this specific outcome. Nevertheless, the pipeline could consist of more steps and could reveal more information from the data. For example the current pipeline could be enriched with the data from `Players` table to see which players scored the most points at home for the team.

When it comes to the stream processing pipeline there remains an opportunity for enhancement concerning the architecture and execution of the stream processing pipeline. While the current implementation serves as a foundational starting point, the current setup entails both the Kafka Producer and Kafka Consumer programs being executed locally on individual team member computers. This configuration limits their continuous operation, resulting in intermittent data production and collection.

In an improved iteration of the stream processing pipeline, a sustained scrapping of data from the NBA official website at fixed intervals (e.g., hourly) in JSON or XML format would be integral. This data would be made available via an endpoint, while a consistently active Kafka consumer (potentially hosted on a separate server) would continuously read this data stream, subsequently feeding it to the Spark Driver Application. Such an approach ensures a constant inflow of updated data, thereby ensuring that visualizations consistently reflect the most recent information available.

The alternative design for the stream processing pipeline can be seen below.

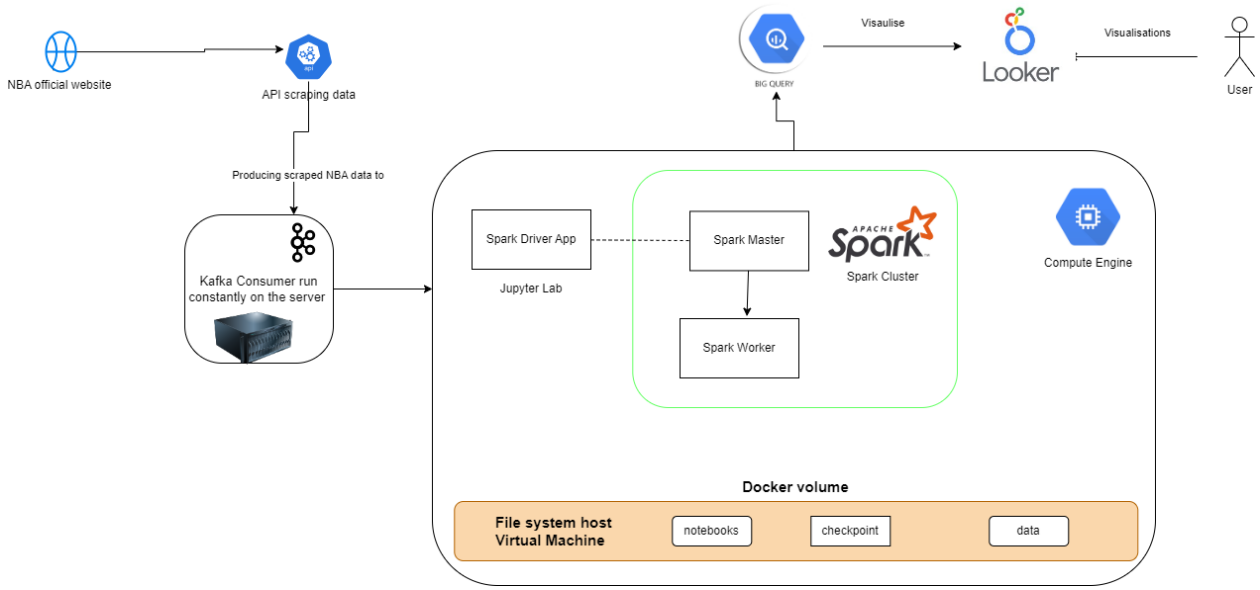


Figure 7: Alternative Design for Stream Processing pipelines

# 5. Individual Contributions of Students

Krzysztof Wiesniakowski:
set up a GitHub repository, created a ground to batch processing, did a stream processing pipeline, wrote the substantial part of the report and made sure everything is ready for the submission.

Koukiadakis Kyriakos:
Created the batch processing pipeline, and the visualizations of this pipeline with Looker studio.

Xavier Paulus:
Came up with the NBA data and what to do with it. I also help to write the report.

Vishal Seghal:
Worked on the processing pipeline, and made sure everything was organized. I set up some technical parts to run online.

Chigozie Ifepe:
I helped to come up with the idea for NBA data and what we could explore there. I helped with writing the report as well.

References:
[1] NBA Games data. (2022, 23 december). Kaggle.
https://www.kaggle.com/datasets/nathanlauga/nba-games/data

[2] Cloud computing icon - Unlimited download. Cleanpng.com. (n.d.). cleanpng.com. https://www.cleanpng.com/png-bigquery-google-cloud-platform-google-analytics-bi-1534320/download-png.html

[3] Wikipedia contributors. (2023, November 8). *Apache Spark*. Wikipedia. https://en.wikipedia.org/wiki/Apache_Spark