

# P2: Trusted Zone, Exploitation Zone and Downstream Tasks

## Big Data Management

Bruna Barraquer & Haonan Jin & Hang Yao

June 8, 2025

### Abstract

This second phase of the Big Data Management project extends the initial **Data Lakehouse** framework by incorporating the **Trusted** and **Exploitation Zones**, thereby completing the end-to-end data pipeline. The architecture is designed to ensure modularity, scalability, and reliability through a unified technology stack consisting of **Apache Spark**, **Delta Lake**, and **Apache Airflow**. Data flows seamlessly from raw ingestion in the **Landing Zone** to curated outputs in the **Exploitation Zone**, where it supports a variety of consumption tasks, including real-time analytics, historical dashboards, and recommendation systems. To support data integrity and governance, the project tries to integrate a validation framework using **Great Expectations**. All components are containerized with **Docker** to ensure reproducibility and deployment portability. The final architecture demonstrates a robust, production-ready pipeline capable of supporting diverse business insights in the movie domain.

## 1 Architecture design

Our pipeline architecture is designed to provide a scalable, consistent, and maintainable framework for end-to-end data management. It builds upon the initial **Landing Zone** architecture by extending support for the **Trusted Zone** and **Exploitation Zone**, without introducing new database technologies. This decision is based on simplicity, tooling familiarity, and the scalability of the chosen stack: **Apache Spark**, **Delta Lake**, and local **Parquet**-based storage.

We adopted a layered folder-based structure to represent different zones within the **Data Lakehouse**—**Landing**, **Trusted**, and **Exploitation** are just layers in the same distributed **Data Lakehouse** with clear separation by folders and metadata. Hence, facilitates the **Delta Lake**'s transaction and **Spark**'s distributed computing capabilities throughout the pipeline. This strategy enables us to avoid unnecessary complexity from integrating heterogeneous databases (**DuckDB** or **MongoDB**), which would require extra orchestration and interoperability layers.

To orchestrate the batch data processing workflow across all zones, we use **Apache Airflow**. Its **Directed Acyclic Graphs** (DAGs) model allows clear dependency management between pipeline tasks, ensuring orderly execution. **Airflow** also adds scheduling, retry mechanisms, and monitoring capabilities, contributing to the robustness of our architecture.

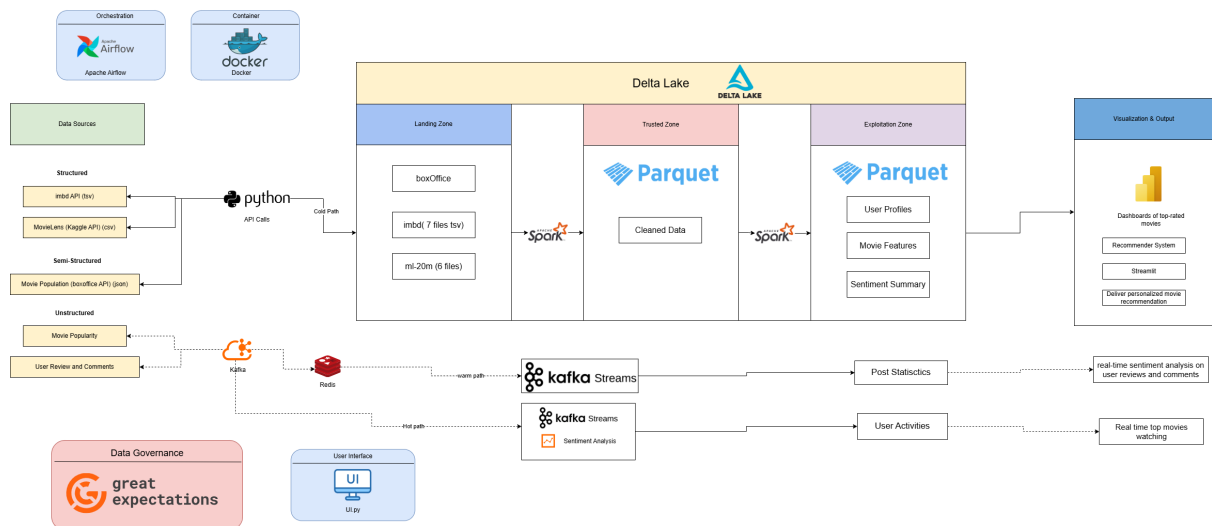


Figure 1: Proposed Diagram

All necessary components of the pipeline are containerized using **Docker**. For example, **Docker** containers encapsulate **Apache Kafka** (for real-time ingestion), **Apache Spark**, **Apache Airflow**, and auxiliary services, enabling seamless integration and simplified configuration. This guarantees reproducibility and environment consistency across development.

The architecture supports both batch and streaming paradigms, offering flexibility to accommodate cold, warm, and hot path consumption layers. Through modular pipeline stages and consistent data formats, each processing phase can evolve independently, without disrupting upstream or downstream processes.

Overall, this design provides a robust foundation for scalable big data processing, aligned with industry best practices and well-suited to the scope and constraints of our academic and technical objectives.

## 2 Trusted Zone

The **Trusted Zone** acts as the cornerstone of our data pipeline, serving as the first curated layer where raw ingested data is transformed into reliable, clean, and analysis-ready datasets. Its primary purpose is to enforce data consistency, quality, and integrity before it is passed on to downstream analytical or visualization tasks.

### 2.1 Data Storage Strategy for the Trusted Zone

We specifically selected **Delta Lake** for our **Trusted Zone** because it brings a suite of features that are crucial for keeping our data dependable, high-quality, and easy to manage. One more reason to choose **Delta Lake** for storage is because of Rock-Solid Transactions (ACID), **Delta Lake** gives us ACID (Atomicity, Consistency, Isolation, Durability) guarantees. In simple terms, this means data operations are all-or-nothing. They either complete perfectly, or they do not happen at all, which is a huge win for preventing data corruption. Anyone looking at the data always sees a consistent picture—absolutely vital for a zone we label as "trusted."

## What We Gain with Delta Lake (Pros):

- **Seriously Reliable Data:** Those ACID transactions and schema checks drastically cut down the risk of messed-up data.
- **Better Data Quality:** Schema validation and the option to add data quality rules directly lead to cleaner, more trustworthy data.
- **Easier Data Lake Management:** Features like time travel and merge take the headache out of many common data lake chores.

## Things We Considered (Cons):

- **A Bit More Under the Hood:** For super simple tasks or tiny datasets, **Delta Lake**'s features might add a tiny bit of storage or processing overhead compared to just using plain **Parquet** files. Usually, though, the benefits far outweigh this.

## What if we'd gone a different Store

- **Our Thoughts:** If we'd picked a classic data warehouse, our data transformations (similar to what is in our Python script) would likely have been done with SQL stored procedures or separate ETL/ELT tools loading data into the warehouse after transforming it. The whole setup would be very focused on structured, relational data.
- **The Advantages:** These are mature systems with years of development and great tools. SQL is dominant one, and everyone understands relational concepts. They plug into most BI tools easily, and cloud versions make them simpler to run.
- **The Disadvantages:** They can be a bit stiff when dealing with semi-structured data or schemas that change a lot. Scaling up can get pricey or complicated. ETL processes can sometimes feel more rigid than the freedom **Spark** gives us.

## 2.2 Organizing Our Trusted Data

You can see from our `main_cleaning_pipeline` function that we are careful about where each cleaned dataset goes. Each one becomes its own **Delta table**, so every source is created in directories within our **Trusted Zone**, it follows the same directory organization as the **Landing Zone**. With this structure we can manage, update, and even roll back each table on its own, thanks to **Delta Lake** and is easy for anyone who needs this trusted data can quickly find and understand what they're looking at.

## 2.3 Tasks

It is important to note that all transformation and cleaning tasks are centralized within the `data_cleaning_pipeline.py` script. This strategic consolidation ensures significant code reusability and prevents redundancy, as analogous processing steps are applied consistently across various datasources.

### 2.3.1 Data Cleansing & Standardization

We cleaned missing values by using `replace_nulls()` and `replace_column_values()` to standardize different null indicators like "N/A", "-", and "\N" into proper `None` values. This ensured consistent handling of missing data across all datasets. We also converted data into appropriate types—percentages into floats, currency strings into numeric values, runtimes into integers, date strings into `DateTime`, timestamps into `TimestampType`, and text booleans into actual `True/False`. This made the data ready for calculations and analysis. Additionally, we parsed and cleaned text using functions like `split_array_columns()` and `regexp_extract()` to turn delimited strings into arrays and extract useful information like release years and awards. We even considered text casing with `lower_case_columns()` to improve consistency.

### 2.3.2 Data Quality & Integrity

To remove duplicates, we applied `dropDuplicates()` using key columns like `[movieId, tagId]` and `[nconst]`, ensuring each record was unique. This helped maintain accurate aggregations. We also filtered out low-quality data—for example, dropping IMDb ratings where `averageRating` was zero or `numVotes` was missing—so that only meaningful and valid records remained in the dataset.

### 2.3.3 Data Formatting & Refinement

Finally, we used `pyspark_round()` to round numeric values, like relevance scores and financial figures, to consistent decimal places. This improved readability and made the data cleaner for reporting and visualization.

## 2.4 Moving Data from Landing to Trusted

The journey of our data from the `Landing/Persistent Zone` to the polished `Trusted Zone` is managed by the `main_cleaning_pipeline()` function in our Python script. **Apache Spark** handles the heavy processing, while **Delta Lake** serves as the underlying storage format throughout the pipeline. We use **Apache Spark** as our core distributed data processing engine. **Delta Lake** is used consistently both in the `Landing/Persistent Zone` (as seen in `load_delta_tables()`) and the `Trusted Zone`, reinforcing a uniform approach to data handling across the pipeline.

### 2.4.1 Pipeline from Landing To Trusted

- 1. Waking Up Spark:** The process starts with `get_spark_session()`, which initializes a **Spark** session with **Delta Lake** support and allocates the required resources.
- 2. Fetching Data from the Landing Zone:** Using `load_delta_tables()`, we read **Delta** tables from the `Landing/Persistent Zone`. These are assumed to be in **Delta** format already, offering schema enforcement and versioning benefits even in earlier pipeline stages.
- 3. Applying Transformations:** Each dataset (**MovieLens**, **BoxOffice**, **IMDb**) is cleaned using its dedicated function—`clean_ml_20m_tables`, `clean_boxoffice_table`, or `clean_imbd_tables`.

These perform all required cleaning, standardization, and validation to convert raw data into trusted, structured outputs.

**4. Saving to the Trusted Zone:** The final DataFrames are saved using `df.write.format('delta').mode('overwrite').save(...)`, replacing existing data with each run. The `overwrite` mode ensures the **Trusted Zone** always reflects the most up-to-date, fully processed version of the data.

### 3 Exploitation Zone

The **Exploitation Zone** serves as the final stage of our data pipeline, where processed and enriched data from the **Trusted Zone** is organized and transformed into a consumable format for analytics, reporting, and downstream applications. The strategy for data organization and processing in this zone reflects the rigorous standards established in the **Trusted Zone**, leveraging the benefits of **Delta Lake** for data reliability and manageability.

#### 3.1 Organizing Our Exploitation Data

Within the `exploitation_tables` function (located in `Python/Data_Management/Exploitation_Zone`), data from distinct sources undergoes separate processing and transformation. This approach is necessitated by the unique characteristics and requirements of each datasource. Each resulting table, whether it is a single table or a joined product of multiple tables, is stored in its dedicated directory within the **Exploitation Zone**. This structured organization, adhering to the **Delta Table** format, ensures consistency with other zones in our data lakehouse architecture. Such robust structure facilitates efficient data governance and enables reliable data rollbacks in the event of processing failures.

#### 3.2 Exploitation Tasks and Data Transformations

A fundamental limitation across our diverse datasources is the inconsistency in `movie ID` formats, excluding direct joins between them. For instance, while `BoxOffice` and `IMDb` sources share a similar ID format (`imdbID` and `tconst` respectively), we opted against joining them due to their distinct business domains. Similarly, the `MovieLens` datasource's IDs are entirely different, and its `link.csv` file's `imdbId` column does not align with the format provided by the `IMDb` source. Consequently, our strategy involves selecting the most relevant and significant tables or creating joined tables from each source to establish distinct business domains for the final exploitation tasks.

##### 3.2.1 *BoxOffice Source*

The `boxOffice_table` function is responsible for the necessary transformations to produce the final table for `BoxOffice` dataset. Given that this source comprises a single dataset, the primary transformation involves the selection of pertinent columns. Columns such as `imdbRating`, `imdbVotes`, and `Poster` are chosen for their intrinsic value in future analysis and modeling, ensuring the final table provides meaningful information for visualizing movie characteristics.

### 3.2.2 IMDb Source

For the IMDb source, our approach focuses on retaining and joining the most critical tables: `imbd_title_principals`, `imbd_name_basics`, `imbd_title_ratings`, `imbd_title_episode`, and `imbd_title_basics`. The remaining tables were deemed insignificant for exploitation due to a lack of relevant extractable information. From these five key tables, we construct two new tables, each enriched with newly derived **KPIs**:

**people\_movie\_kpi:** This table is derived from an inner join of `imbd_title_principals` and `imbd_title_ratings` on the `tconst` attribute, followed by an inner join with `imbd_name_basics` on `nconst`. The selected attributes include `id_person`, `person_name`, `category`, `average_rating`, and `id_title`. These attributes are crucial for retrieving information about individuals involved in movies and for the computation of **KPI**.

The **KPI** for this table represents the average rating of all movies in which an actor has participated. This metric is invaluable for assessing an actor's relevance or popularity, as a consistently high average movie rating often correlates with a celebrated individual in the sector.

**movie\_episode\_ratings\_kpi:** This table is constructed through an inner join of `imbd_title_episode` and `imbd_title_basics` on `tconst`, followed by a left join with `imbd_title_ratings` on the same attribute. The use of a left join is considered, as it ensures that information about episodes is retained even if a movie or series has not yet received a rating. Key attributes preserved in this table include `id`, `genres`, `seasonNumber`, `episodeNumber`, `averageRating`, and `numVotes`. These attributes provide critical insights for visualization and future analysis respecting movie and series relevance.

The **KPIs** developed for this table are:

- **total\_episode:** Calculated as the product of `seasonNumber` and `episodeNumber`. This **KPI** serves as an indicator of a movie's or series' popularity; a greater number of episodes generally suggests higher engagement and popularity.
- **trend:** A binary indicator that signals whether a movie or series' overall average rating surpasses the overall average rating of all movies/series. This **KPI** offers insight into user preference and popularity, with a rating higher than the overall average signifying a well-received title.
- **score:** Derived from the product of `averageRating` and `numVotes` for each movie or series. The **score** KPI is a robust measure of relevance and popularity. A high score indicates a movie is both highly rated and widely viewed. Conversely, a movie with a low rating but a high number of votes might suggest that despite being popular, its content is not universally popular, potentially due to the involvement of famous actors whose performance did not meet expectations. Hence, by computing the **KPI**, we can gain an overview of the performance of each movie or series.

### 3.2.3 MovieLens Source

For the MovieLens source, we opted to preserve the `ml-20m_tag` and `ml-20m_rating` tables in their original form, as they are already highly informative and require no further transforma-

tions. The `ml-20m_link` table was excluded due to the incompatibility of its `imdbId` with the `IMDb` source. The remaining tables are consolidated into a single table, extending the information of `ml-20m_movie` by incorporating tag details.

This consolidation process involves an initial join between `ml-20m_genome_scores` and `ml-20m_genome_tags` on `tagId` to retrieve the tag names for each movie, compacting all tags into a single attribute using a list. Subsequently, an additional join with `ml-20m_movie` on `movieId` integrates these tags, enriching the movie information.

### 3.3 Moving Data from Trusted to Exploitation

The ordered transition of data from **Trusted Zone** to the highly structured **Exploitation Zone** is orchestrated by the `exploitation_tables` function within our Python script. This data movement strategy aligns with the principles established in preceding pipeline stages, involving the creation of dedicated folders within **Exploitation Zone** for each final **Delta table**. This separation by datasource ensures flexible data access and simplifies the management of individual **Delta** tables, as no direct joins are performed across sources.

#### 3.3.1 Pipeline from Trusted To Exploitation

**1. Waking Up Spark:** The process starts with the invocation of `get_spark_session()`, which establishes a **Spark** session. This session is configured with **Delta Lake** support and allocates the necessary computational resources, forming the foundation for subsequent data operations.

**2. Fetching Data from the Trusted Zone:** Data is retrieved from the **Trusted Zone** using the `load_delta_tables` function. These tables are inherently in **Delta** format, providing immediate benefits such as schema enforcement and versioning, even at earlier stages of pipeline.

**3. Applying Transformations:** Each dataset, originating from `MovieLens`, `BoxOffice`, and `IMDb`, undergoes specific transformations. These transformations are guided by our understanding of the most relevant movie-related information. This involves primarily selecting relevant columns and performing joins with other tables. Critically, new **KPIs** are computed for certain joined tables, particularly within the `IMDb` source. For instance, `score` KPI is derived from the product of `averageRating` and `numVotes`. Each datasource utilizes its dedicated function (`boxOffice_table`, `ml_movie_tables`, `imbd_tables`) to execute these transformations and **KPI** computations. It is noteworthy that only the `IMDb` source tables undergo **KPI** generation, as the attributes present in other sources were not deemed meaningful or useful for generating significant **KPIs**.

**4. Saving to the Exploitation Zone:** The final, transformed `DataFrames` are persisted in the **Exploitation Zone** using a **Spark** save command similar to those employed in previous zones. This ensures the **Exploitation Zone** consistently reflects the most current and fully processed version of the data, thereby maintaining data integrity and freshness.

## 4 Consumption Tasks

With the data management and processing stages established, the final step is to consume the data to generate insights. The focus of this project remains on the architectural and management aspects of the data pipeline rather than the analytical depth. The primary goal is to demonstrate a functional end-to-end system where data is ingested, processed, and visualized, showcasing the orchestration of the different components.

### 4.1 Top 10 movies - Real-Time Hot Path Analysis

This section details the "hot path", a real-time data consumption task designed to analyze streaming data as it is generated.

#### 4.1.1 Data Exploitation Mechanism

The chosen mechanism for the hot path is a **real-time dashboard** (examples in [6] and [5]). This approach allows for immediate visualization of incoming data, providing live insights into user reviews and movie ratings. The system is designed to process streaming data, perform a simple analysis, and update visualizations dynamically, which is a classic example of a hot path architecture where low-latency processing is critical.

#### 4.1.2 Tools Selection

The following tools were selected to build the real-time consumption layer:

- **Apache Kafka:** Serves as the central nervous system of the hot path. The `kafka_producer_hot_path.py` script continuously sends user review data to a **Kafka** topic, and the `hot_path_top_movies.py` script consumes it in real-time. **Kafka** was chosen for its high-throughput, low-latency, and fault-tolerant messaging capabilities, making it ideal for streaming data.
- **Streamlit:** A Python library used to create and share web apps for data science and machine learning projects. The entire user interface is built using **Streamlit** within the `hot_path_top_movies.py` and `app.py` scripts. It was selected for its simplicity and rapid development capabilities, which allow for the creation of interactive and auto-updating dashboards with minimal code.
- **Pandas:** Used for in-memory data manipulation. Although the analysis is simple, **Pandas** DataFrames are used within `hot_path_top_movies.py` to structure the recent reviews for easy display in the **Streamlit** app.
- **Plotly:** An interactive graphing library. It is used in `hot_path_top_movies.py` to create the dynamic bar chart that displays the top 10 movies based on their average ratings. **Plotly** was chosen for its ability to generate rich, interactive visualizations that can be seamlessly embedded into a **Streamlit** application.



### 4.1.3 Task: Real-Time Movie Rating Dashboard

The primary task is to display a live dashboard showing the top-rated movies based on a continuous stream of user reviews. This is accomplished through the following workflow:

1. **Data Generation:** The `generate_reviews.py` script provides the syntactic data by creating a diverse set of synthetic movie reviews. This is a placeholder for actual user activity, allowing the system to be tested and demonstrated without relying on live users. The reviews are generated using templates and enriched word banks to simulate positive, negative, and mixed sentiments.
2. **Producer (Data Ingestion):** The `kafka_producer_hot_path.py` script simulates a real-time data stream. It generates reviews using `generate_review_bank`, performs a quick sentiment analysis with VADER to assign a rating (1-5), and sends this structured data (including user ID, movie ID, review text, rating, and timestamp) to the `user-reviews-topic` in **Kafka**. This simulates a microservice or application component that generates events.
3. **Consumer and Application Logic (Data Consumption):** The `hot_path_top_movies.py` script is the core of the data consumption layer. It implements the `MovieAnalyzer` class, which handles the following:
  - **Kafka Consumption:** It runs a **Kafka** consumer in a background thread to listen for new messages from the `user-reviews-topic` without blocking the main application. This ensures the dashboard remains responsive.
  - **Data Processing:** As new reviews arrive, they are placed in a queue. The main application thread processes this queue, updating a dictionary (`movie_ratings`) that stores all ratings for each movie ID and a deque (`recent_reviews`) that holds the latest 50 reviews. This in-memory storage is sufficient for a real-time dashboard and avoids the latency of writing to and reading from a database for every new event.
  - **Live Visualization:** The script uses **Streamlit** to create a user interface with controls to start and stop the producer and consumer. It displays a **Plotly** bar chart of the top 10 movies, calculated on-the-fly from the `movie_ratings` data. It also shows a table of the most recent reviews. The dashboard can be set to auto-refresh (5 seconds), providing a near real-time view of the data.
4. **Application Entrypoint:** The `app.py` script acts as a simple navigator for a larger potential project, allowing a user to select from different "products". The "Top 10 Movies (Hot Path)" option loads and runs the `main` function from `hot_path_top_movies.py`, effectively launching the real-time dashboard.

This implementation fulfills the project's requirements by creating a fully orchestrated, albeit simplified, data consumption pipeline. It successfully ships data from a streaming source (**Kafka**) to a consumption system (**Streamlit**) and implements a task that mimics a real-world analytics process, prioritizing the demonstration of a well-architected data flow over the complexity of the analysis itself.

### 4.1.4 Example

Figure 2 illustrates a concise example of **Real-Time Movie Analysis**. This real-time data processing necessitates the prior initialization of both producer and consumer components. The

figure subsequently displays the processed data through a dynamic plot, complemented by a table presenting recent user reviews. This table includes additional informative attributes, such as the leverage rating and the most common rating for enhanced real-time insights.

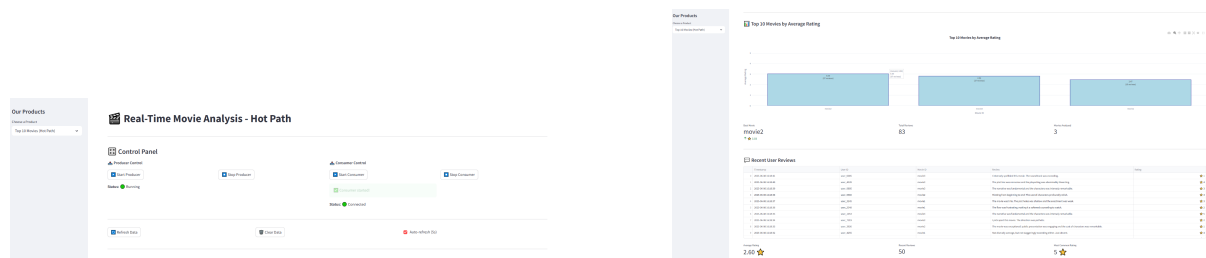


Figure 2: Hot Path Top Movies Example

## 4.2 Top 10 Movies by Clicks - Near-Real-Time Warm Path Analysis

Following the hot path's real-time analysis, the warm path is introduced to handle near-real-time data processing. This path focuses on aggregating data over a slightly longer time window, providing insights that are still timely but less instantaneous than the hot path. The goal is to demonstrate a functional end-to-end system where streaming data is ingested, aggregated, and visualized to showcase the orchestration of different architectural components for near-real-time analytics.

This section details the "warm path," a near-real-time data consumption task designed to analyze streaming data on a hour basis but you can check the data on a refresh of 2 seconds.

### 4.2.1 Data Exploitation Mechanism

The chosen mechanism for the warm path is a near-real-time dashboard that visualizes aggregated movie click data. This approach allows for the timely visualization of user interactions, providing insights into which movies are currently trending. The system is designed to process streaming data, perform an aggregation over a defined time window (one hour), and dynamically update visualizations. This is a classic example of a warm path architecture where processing latency is slightly higher than in a hot path, but it still delivers up-to-date information.

### 4.2.2 Tools Selection

The tools were the same as the hot path to build the near-real-time consumption layer, as demonstrated in the `warm_path_movie_click_relevance.py` script.

### 4.2.3 Task: Near-Real-Time Movie Click Dashboard

The primary task is to display a live dashboard showing the top 10 most-clicked movies based on a continuous stream of simulated user clicks. This is accomplished through the following workflow:

1. **Data Generation (Producer):** The `near_real_time_processing` function within the script acts as a producer, simulating a real-time stream of user clicks. It randomly generates click data for a predefined set of movies and sends this information (movie ID and number of clicks) to the 'movie-click-rate' topic in **Kafka**. This simulates a microservice tracking user interactions.
2. **Consumer and Application Logic (Data Consumption):** The `MovieClickAnalyzer` class within `warm_path_movie_click_relevance.py` is the core of the data consumption layer. It performs the following functions:
  - **Kafka Consumption:** A **Kafka** consumer runs in a background thread to listen for new messages from the 'movie-click-rate' topic. This non-blocking approach ensures the dashboard remains responsive while continuously fetching new data.
  - **Data Processing and Aggregation:** As new click data arrives, it is placed into a queue. The main application thread processes this queue, aggregating the total clicks for each movie ID in a dictionary. This aggregation occurs over a one-hour window, which resets automatically. This in-memory aggregation is efficient for near-real-time dashboarding, avoiding database latency for every new event.
  - **Live Visualization:** The script uses **Streamlit** to create a user interface with controls to start and stop both the data producer and consumer. It displays a **Plotly** bar chart of the top 10 movies based on the aggregated click counts within the current one-hour window, which means, the data is reset every hour. A countdown timer shows when the next data reset will occur. The dashboard is configured to auto-refresh (2 seconds), providing a near-real-time view of movie popularity.

This implementation successfully creates a fully orchestrated warm path data pipeline. It effectively channels data from a streaming source (Kafka) to a consumption and visualization system (Streamlit), demonstrating a practical analytics process that prioritizes the delivery of timely and aggregated insights over the instantaneous.

#### 4.2.4 Example

Figure 3 shows the initialization sequence, which is analogous to that described in the preceding hot path section. Subsequently, it presents a **Near-Time Analysis** demonstrating movie relevance, measured by click interactions. This analysis is updated and reset hourly, reflecting the continuous aggregation of click data within that specific time window. The corresponding table also displays the cumulative number of clicks for each movie.

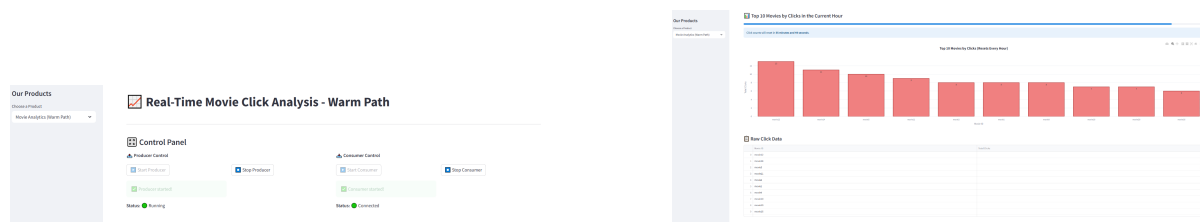


Figure 3: Warm Path Movie Clicks Example

### 4.3 Recommender System - Cold Path Predictor

Following the analysis and processing conducted in the hot and warm paths, this section details the development of a recommender system based on cold path data. Specifically, we leverage the `ml-20m_movie` and `ml-20m_rating` datasets sourced from the `MovieLens` collection. This cold path approach is designed to provide recommendations to the users.

#### 4.3.1 Data Exploitation Mechanism

Our recommender system incorporates two distinct collaborative filtering methodologies to generate recommendations for a given user: a **user-based** collaborative filtering approach [2] and an **item-based** collaborative filtering approach [3]. Both strategies are widely recognized for their effectiveness in providing significant recommendations. These systems are designed to retrieve the top  $N$  movies for a specified user, with a subsequent validation step that computes the genre similarity between movies a user has been seen and the recommended movies.

#### 4.3.2 Tools Selection

Our recommender system's consumption layer builds upon the tools used in previous tasks and leverages a modular set of Python scripts for its core functionality. The `user_based_recommender.py` and `item_based_recommender.py` modules are responsible for training their respective collaborative filtering models and generating predictions. Additionally, the `utils.py` module provides crucial utility functions, particularly for evaluating genre similarity between movie sets. All these specialized scripts are orchestrated and executed within the main `recommender_system.py` script, which drives the visualization layer.

#### 4.3.3 Task: Recommendation for Users

The primary objective of this module is to present a list of top  $N$  movies for a given user, determined by the selected collaborative filtering method. Furthermore, it assesses the similarity of these predicted movies to the user's watched movies. This comprehensive workflow is executed through the following sequential tasks:

1. **Data Extraction:** The initial step involves extracting the necessary datasets from the **Delta** tables within the **Exploitation Zone**. Those data form the foundation for model building and subsequent prediction tasks.
2. **Model Training:** The extracted rating dataset is partitioned into training and validation sets. The training set is then used to fit the chosen collaborative filtering model, which computes **user-item interaction matrices**. These matrices encapsulate implicit or explicit user preferences and item characteristics, forming the basis for future predictions.
3. **Model Prediction:** Given a specific user, the trained model predicts the top  $N$  movies they are most likely to enjoy. This involves computing a score for each unseen movie based on the overall user or item preferences encoded in the matrices. The movies with the highest scores are then selected as recommendations.

4. **Validation:** Upon generating the top  $N$  movie recommendations, a crucial validation step is performed. This involves computing the genre similarity between these unseen recommended movies and the movies the user has watched. To accurately capture the relevance of genres within the movies, **TF-IDF** is employed. Unlike simple genre counts, **TF-IDF** effectively weights genres by their significance. Subsequently, **Cosines Similarity** is utilized to quantify the genre similarity. A high similarity score indicates that the model has made good predictions aligned with the user's observed genre preferences. For instance, if a user's viewing history is predominantly composed of "action" movies, the recommend movies are expected to exhibit a high relevance in the "action" genre.
5. **Display:** Finally, the top  $N$  recommended movies are presented to the user. This display includes comprehensive information such as genres, tags, titles, and the computed similarity score. This detailed presentation allows for an intuitive observation of the model's performance and the rationale behind the recommendations.

This implementation successfully establishes a fully orchestrated cold path data pipeline. It effectively retrieves the requisite tables from the **Exploitation Zone** and performs the necessary data transformations and model inferences to achieve our business domain objectives and provide a satisfactory user experience through personalized movie recommendations.

#### 4.3.4 Example

Figure 4 shows the initial configuration interface for the recommender system. Here, the user can specify the recommendation method, which in this example is **user-based**. They can also input a User ID (e.g., 1) and define the number of top movies to recommend (e.g., 10). After setting these parameters, clicking "Get Recommendations" starts the computation. The system then displays the results in a table, showing the top  $N$  recommended movies along with their calculated genre similarity (e.g., 0.8919).

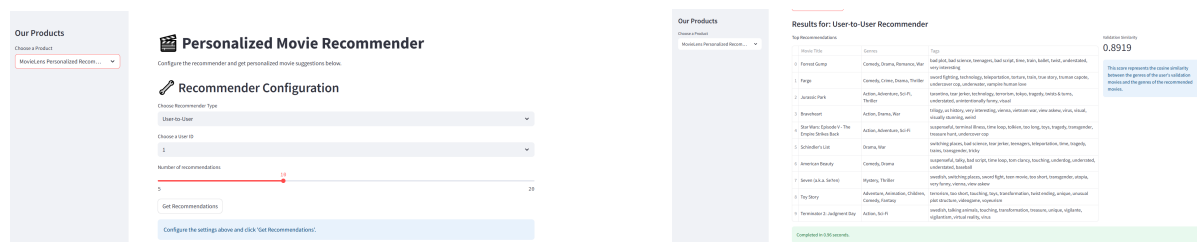


Figure 4: MovieLens Recommender System Example

## 4.4 IMDb KPI Dashboard - Cold Path Analysis

This final task focuses on the IMDb datasource, specifically leveraging the pre-computed **KPIs** within the `movie_episode_ratings_kpi` and `people_movies_kpi` tables located in the **Exploitation Zone**. This section describes the development of a historical **KPI** dashboard for cold path analysis, designed to provide comprehensive insights into IMDb data.

#### 4.4.1 Data Exploitation Mechanism

The core data exploitation mechanism for IMDb cold path involves the creation of a dynamic dashboard. This dashboard facilitates the visualization of title-specific historical data, allowing interactive filtering by `startYear` (release year) and `genre`. This interactive filtering provides deep insight into various title-related **KPIs**. Furthermore, the dashboard includes dedicated sections for actors and directors, presenting key metrics such as the number of titles participated in and the average rating across those titles, thereby offering meaningful individual-level **KPIs**.

#### 4.4.2 Tools Selection

For the development of this historical consumption layer, the toolset employed is consistent with those used in the preceding tasks. The implementation details and specific functionalities are encapsulated within the `imdb.kpi_dashboard.py` script.

#### 4.4.3 Task: KPI Dashboard

The primary objective of this task is to construct and visualize an interactive dashboard designed for the comprehensive analysis of movies/series titles and associated individuals (actors/directors). The workflow is executed through the following sequential steps:

1. **Data Extraction:** The initial step involves extracting the required datasets from the **Delta** tables within the **Exploitation Zone**. For efficient in-memory processing and subsequent conversion to **Pandas** DataFrames, these **Delta** tables are first converted to **Parquet** format. This intermediate step is crucial due to Parquet's optimized columnar storage, which enhances data loading speed and memory efficiency for local file operations, thus forming the foundational data for analysis.
2. **Filter:** Initially, the dashboard allows for interactive filtering of title-related information based on `startYear` and `genre`. By default, it displays data for the five most recent years and the top five ordered genres. Subsequently, the relevant **KPIs** corresponding to the applied filters are presented.
3. **Filtered Visualizations:** The filtered data is presented in a clear tabular format, displaying key columns such as `originalTitle`, `startYear`, `genres`, `total_episodes`, `trend`, and `score`. This provides a comprehensive overview of the title-specific information, with an option to export the filtered table as a **CSV** file.

Complementing the tabular view, a dynamic plot visualizes the distribution of title scores group by either `genre` or `startYear`. This visualization highlights the comparative relevance and impact of different genres or release years based on their aggregated title scores.

4. **People Analysis:** Separated tables are presented to showcase the top 10 directors and actors, ranked by their average rating across all participated titles. This analysis effectively highlights the popularity and observed quality associated with each individual's contributions to the film sector.
5. **Summary:** A concise summary section provides an overview of the loaded and filtered data, including key rating statistics. This offers a quick yet insightful study into characteristics of the analyzed dataset.

This implementation successfully establishes a fully orchestrated cold path data pipeline. It effectively retrieves the necessary tables from the **Exploitation Zone**, performs the necessary data format conversions, and visualizes in a comprehensive dashboard that provides insightful overviews of both titles and individuals within the **IMDb** dataset.

#### 4.4.4 Example

Figure 5 displays the **KPIs dashboard** for IMDb titles and individuals. Initially, it applies default filters for genres and years, as illustrated. The dashboard then presents a filtered table containing relevant attributes for each title, alongside a plot visualizing title scores categorized by genre or year. Subsequently, tables detailing individual relevance are displayed, followed by a summary providing an overview of the analyzed data.

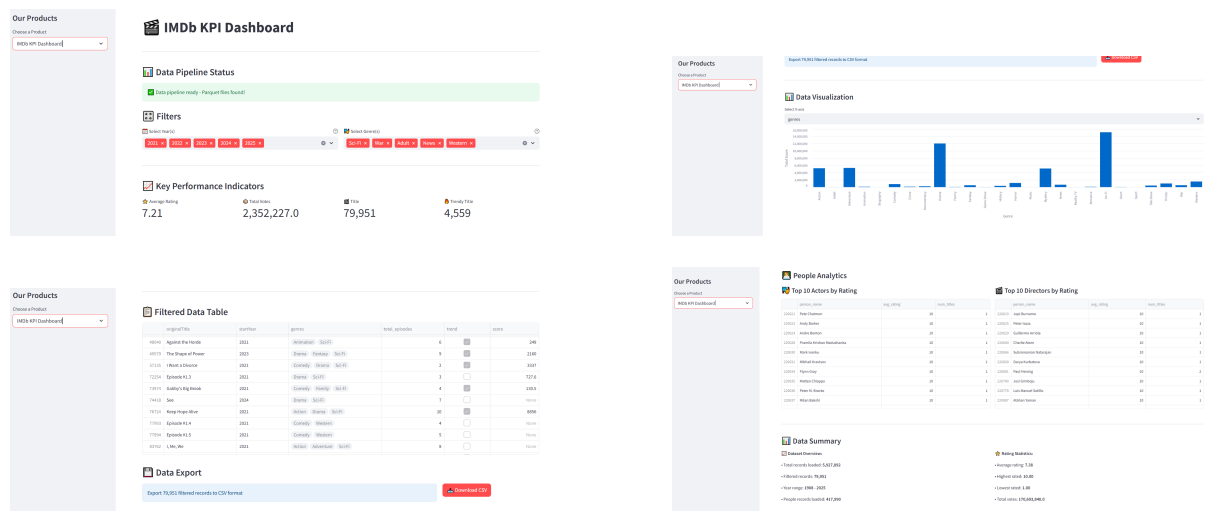


Figure 5: IMDb KPI Dashboard Example

## 5 Governance Tasks

This section outlines the data governance strategy for our big data architecture. We will shift our focus from purely technical aspects to a more organizational and strategic perspective, ensuring that our governance proposals are well-aligned with our business context.

### 5.1 Organizational and Strategic Point of View

Our business context revolves around the movie industry, utilizing datasets from **MovieLens**, **BoxOffice**, and **IMDb**. The primary goal is to derive meaningful insights about movie performance, audience preferences, and actor relevance.

#### 5.1.1 Relevant Data Domains

Based on our business context, we have identified the following key data domains:

- **Movie & Series Information:** This domain includes core metadata about movies and series, such as titles, genres, release dates, and production details.
- **Audience Ratings & Reviews:** This domain covers user-generated data, including ratings, reviews, and tags, which are crucial for understanding audience sentiment.
- **BoxOffice Performance:** This domain consists of financial data related to movie revenues, providing insights into the commercial success of films.
- **Cast & Crew Information:** This domain contains data about individuals involved in movie production, such as actors, directors, and writers.

### 5.1.2 Proposed Data Products

Based on these domains, we propose the following data products for our **Exploitation Zone**:

- **Movie Performance Analysis:** A comprehensive data product that integrates **IMDb** ratings and **BoxOffice** data to provide a holistic view of a movie's critical and commercial success.
- **Audience Sentiment Trends:** A product derived from **MovieLens** ratings and tags, designed to track and analyze audience preferences and trends over time.
- **Actor Career Analytics:** A data product that focuses on actors' filmographies, combining their movie roles with ratings and **BoxOffice** data to assess their career trajectories and bankability.

### 5.1.3 Comparison and Insights

The data products we initially created in our **Exploitation Zone**, such as `people_movie_kpi` and `movie_episode_ratings_kpi`, are excellent starting points. They serve as foundational analytical tables. However, by reframing them as data products, we can enhance their value. The key insight is that a data product is more than just a table; it is a curated, documented, and reliable asset designed for specific business use cases. Our proposed data products are an evolution of our initial tables, designed to be more intuitive and directly consumable by business users, with a clear focus on answering specific business questions.

## 5.2 Detailed Data Product Example: Movie Performance Analysis

Here, we provide a detailed specification for the **Movie Performance Analysis** data product, using a structure inspired by the **Data Product Canvas** [1].

- **Description:** This data product provides a unified view of movie performance by combining critical ratings from **IMDb** with financial data from **BoxOffice**. It is designed to help stakeholders assess the overall success of a movie.
- **Owner:** The "Movie Analytics" team.



- **Output Port:** The data product is delivered as a **Delta** table in the **Exploitation Zone**, accessible via **Spark SQL**.
- **Data Model:**
  - `movie_id` (string, primary key)
  - `title` (string)
  - `release_year` (integer)
  - `genres` (array of strings)
  - `average_rating` (float)
  - `num_votes` (integer)
  - `box_office_revenue` (long)
  - `score` (float)
- **Quality Metrics:**
  - **Completeness:** `box_office_revenue` should be non-null for at least 95% of movies released after 1990.
  - **Freshness:** The data is updated daily to include the latest ratings and revenue figures.
  - **Validity:** `average_rating` must be between 0 and 10.
- **Policies:**
  - **Access Control:** Access is restricted to members of the "Analytics" and "Marketing" user groups.
  - **Data Lineage:** Lineage is tracked from the source `IMDb` and `BoxOffice` tables, through the `Trusted Zone` cleaning and transformation processes, to the final `Exploitation Zone` table.

### 5.3 Governance Mechanism: Data Quality Validation with Great Expectations

To ensure the reliability of our data products, we will implement data quality validation using **Great Expectations** [4]. This validation framework will be applied after data processing but prior to loading it into the **Trusted Zone**. Data will only be loaded once it successfully meets all defined *expectations* specified in the **Great Expectations suite**.

#### 5.3.1 Implementation

The proposed implementation will follow these steps:

1. **Expectation Suite Creation:** For each dataset in the **Trusted Zone**, we will create a suite of expectations. For instance, for the movies dataset, we can define expectations such as:

```
# Example of a Great Expectations suite
{
  "expectation_suite_name": "movies_trusted_suite",
  "expectations": [
```

```

    {
        "expectation_type": "expect_column_values_to_not_be_null",
        "kwargs": {"column": "imdb_id"}
    },
    {
        "expectation_type": "expect_column_values_to_be_between",
        "kwargs": {
            "column": "average_rating",
            "min_value": 0,
            "max_value": 10
        }
    }
]
}

```

Listing 1: Great Expectations Code

2. **Validation in the Pipeline:** The validation step should be integrated into our **Airflow DAGs** and the **UI.py** batch ingestion. After the cleaning and transformation tasks for the **Trusted Zone** data, we will validate those data and if they are success we will save it to the **Trusted Zone**.
3. **Handling Failures:** If the validation fails, the pipeline will be configured to take action. For instance, the data will not be loaded into the **Exploitation Zone**, and an alert will be sent to the data governance team for review.

### 5.3.2 Benefits and Trade-offs

- **Benefits:**

- **Improved Data Trust:** By validating data quality automatically, we can ensure that the data products in the **Exploitation Zone** are reliable and trustworthy.
- **Proactive Issue Detection:** Data quality issues are caught early in the pipeline, preventing them from propagating to downstream systems and affecting business decisions.

- **Trade-offs and Costs:**

- **Development and Maintenance Overhead:** Creating and maintaining the expectation suites requires an initial investment of time and effort. As the data sources and business requirements evolve, these suites will need to be updated.
- **Computational Cost:** Running the data quality validations will add to the overall pipeline execution time and consume additional computational resources.
- **Complexity:** Integrating a new tool into the data stack adds a layer of complexity to the architecture and requires the team to learn and manage it.

## 6 Complementary Aspects

This section elaborates on the supplementary facets integral to the second phase of this project. It addresses key improvements and modifications concerning the initial business domains, the expansion of orchestration and containerization strategies, and the evolution of the user interface to cover data preparation. These crucial aspects are detailed as follows:

**Business Domain Evolution:** The ultimate use cases realized in this phase of the project differ from the initial proposals. This deviation is primarily due to constraints imposed by the characteristics and limitations of our chosen datasets, which were not conducive to generating the originally expected products. Consequently, we have instead adopted the business domains as detailed in Section 4, which are more aligned with the available data capabilities.

**Enhanced Orchestration:** The project's orchestration framework has been significantly extended to incorporate the processing tasks for the **Trusted** and **Exploitation Zones**, thereby completing the pipeline for preparing the final data tables. Specifically, within **Apache Airflow**, new tasks have been integrated. These tasks run daily, alongside the initial data ingestion and **Landing Zone** storage operations, ensuring a continuous and automated data preparation workflow.

**Containerization:** Our reliance on **Docker** containers persists in this second phase. The containerization strategy has been further developed by extending the necessary **Docker** images. This includes the provision of an additional **Kafka** image specifically for the Warm Path, distinct from the primary **Kafka** image used by the Hot Path. These updated images now encapsulate all dependencies and environments required to perform the newly integrated data processing tasks, maintaining consistency and portability across different environments.

Furthermore, the `requirements.txt` file has been updated to facilitate the construction of the virtual **Python** environment within **Docker**. This update includes the addition of essential module such as `streamlit` for building interactive web applications, `altair` for statistical visualizations, and `scikit-learn` for machine learning functionalities, among others.

**UI.py Development:** The `UI.py` has been enhanced to include functionalities corresponding to the tasks performed in the **Trusted** and **Exploitation Zones**. This expanded **UI** now allows users to initial processes for cleaning and preparing data, making it ready for subsequent analysis or modeling. It is imperative that these UI-driven data preparation tasks are executed prior to any consumption tasks.

**Code Reusability:** To optimize efficiency and maintainability, the tasks within the **Trusted** and **Exploitation Zones** necessitated frequent **Spark** session creation and **Delta** table loading operations. To address this, a dedicated utility module, `utils.py`, has been developed. This module encapsulates common functions, notably those for initializing **Spark** sessions and loading **Delta** tables. By centralizing these functionalities, other scripts responsible for **Trusted** and **Exploitation Zone** tasks can invoke them as needed, significantly enhancing code reusability and reducing redundancy.

These complementary aspects collectively contribute to the project's overall consistency, enabling continuous data management through robust orchestration and containerization. This approach ensures that all necessary project images are consistently available, mitigating potential distortions from local environment configurations.

## 7 Conclusion

This second phase of our **Big Data Management** project successfully extends the data pipeline architecture to cover the **Trusted** and **Exploitation** Zones, building a robust, modular, and scalable solution for data preparation and consumption.

In the **Trusted Zone**, we have established a reproducible and reliable data foundation by leveraging **Spark** for distributed processing and **Delta Lake** for ACID-compliant storage. Our data cleaning logic is centralized in reusable Python scripts, ensuring consistency and maintainability across datasets. This structured approach enables the **Trusted Zone** to serve as a dependable source for high-quality data.

The **Exploitation Zone** builds upon this foundation by transforming trusted data into well-structured, business-oriented tables. These tables are organized around core entities such as movies, individuals, and user interactions. We designed the schema to support a wide range of downstream use cases—including recommendation systems, KPI dashboards, and real-time analytics—while avoiding redundancy and maintaining schema consistency.

Our pipeline architecture has changed from the initial proposal. While we originally intended to integrate **Power BI** [7] and **Tableau** [8] dashboards via **Streamlit**, we ultimately chose a lighter and more flexible approach by creating dashboards using **Pandas** to display the project’s key performance indicators (**KPIs**). This approach, adopted across all consumption tasks, enabled us to rapidly build interactive front-end interfaces for our data products and visualizations, aligning well with the project’s scope and constraints.

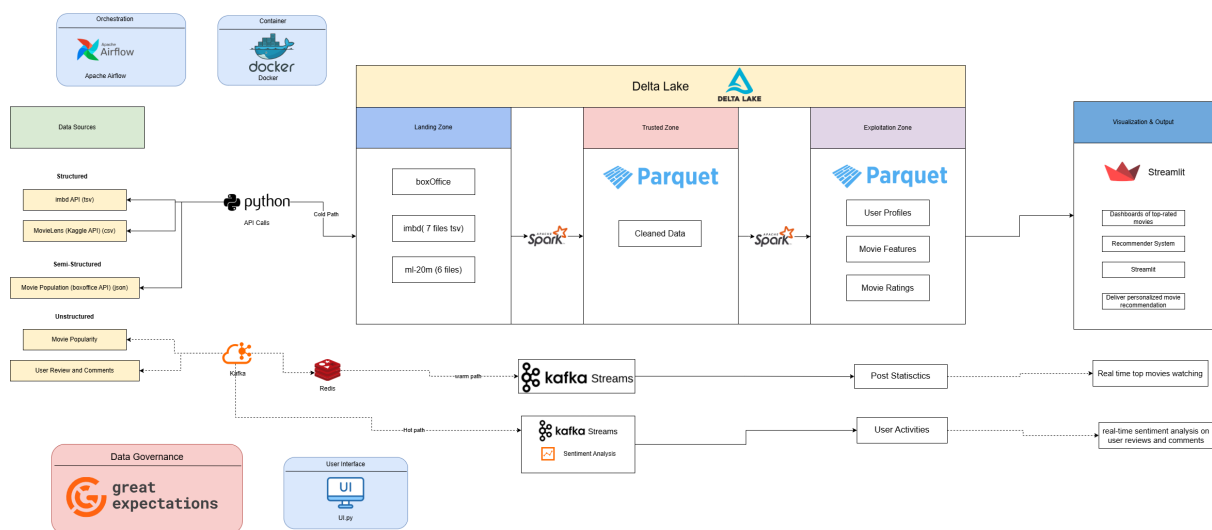


Figure 6: Actual Diagram Pipeline

On the governance side, we attempted to integrate **Great Expectations** [4] for data quality validation prior to loading data into the **Trusted Zone**. Although technical challenges with library versions hindered full implementation, the framework for incorporating such validations has not been established.

Overall, our pipeline demonstrates a fully orchestrated, end-to-end data architecture—spanning batch and streaming ingestion, transformation, and consumption. The system is containerized for portability, automated via **Airflow**, and designed with modularity and future extensibility in

mind. This foundation sets the stage for more advanced analytical capabilities and production-ready deployments.

## 8 Future Work

Some of the potential future works can be outlined as follows:

- **API Development:** Not all the available data have been utilized in the current products. Some datasets may be valuable for other applications. Therefore, a promising direction for future work is to develop **APIs** that allow external access to all data points, enabling broader utility and integration.
- **Extending the Business Domain:** Certain columns in the "box office" dataset, such as the movie "poster" field, have not been fully leveraged. A potential product could be the creation of a blog or web page that highlights top trending movies based on ratings, enriched with visual content. Additionally, the `ml-20m_tag` table, which contains user-generated tags, could be also integrated into the exploitation products. This would enhance the richness of user interaction data and help expand the business domain with user-centric insights.
- **Unit Tests:** To validate the functionalities across the **Trusted** and **Exploitation Zones**, we can perform unit tests similar to those conducted for the **Landing Zone** and **Data Ingestion**.

## References

- [1] Data Mesh Architecture. Data product canvas. <https://www.datamesh-architecture.com/data-product-canvas>. Accessed: 2025-06-08.
- [2] GeeksforGeeks. User-based collaborative filtering. <https://www.geeksforgeeks.org/user-based-collaborative-filtering/>, January 2023. Accessed: 2025-05-30.
- [3] GeeksforGeeks. Item-to-item based collaborative filtering. <https://www.geeksforgeeks.org/item-to-item-based-collaborative-filtering/>, September 2024. Accessed: 2025-05-30.
- [4] Great Expectations. Getting started tutorial overview (v0.14.13). [https://legacy.017.docs.greatexpectations.io/docs/0.14.13/tutorials/getting\\_started/tutorial\\_overview/](https://legacy.017.docs.greatexpectations.io/docs/0.14.13/tutorials/getting_started/tutorial_overview/), 2021. Legacy Documentation.
- [5] Mark Needham. Display a race on a live map. <https://blog.streamlit.io/display-a-race-on-a-live-map/>, June 2023. Accessed: 8 June 2025.
- [6] Author Name or Organization. Title of the item. <https://example.com/path/to/resource>, June 2025. Accessed: 8 June 2025.
- [7] Salma.Brahem, asehami, and Arvindra. Powerbi dashboards and streamlit. Online discussion at Streamlit forum. Thread started July 22, 2021.
- [8] Streamlit. Connect streamlit to tableau. <https://docs.streamlit.io/develop/tutorials/databases/tableau>, 2023. Streamlit Documentation.

## A GitHub

Here we attach you the link for the **GitHub** repository: <https://github.com/brunabarraquer/BDM>, where you can find the **README** file with instructions on how to run all code required to deploy the proposed design—Python scripts.

## B Instructions to Execute Apache Airflow

To run **Apache Airflow**, **Docker** must be active. Then, open your browser and go to <http://localhost:8080/home>. You should see the following interface (Figure 7):

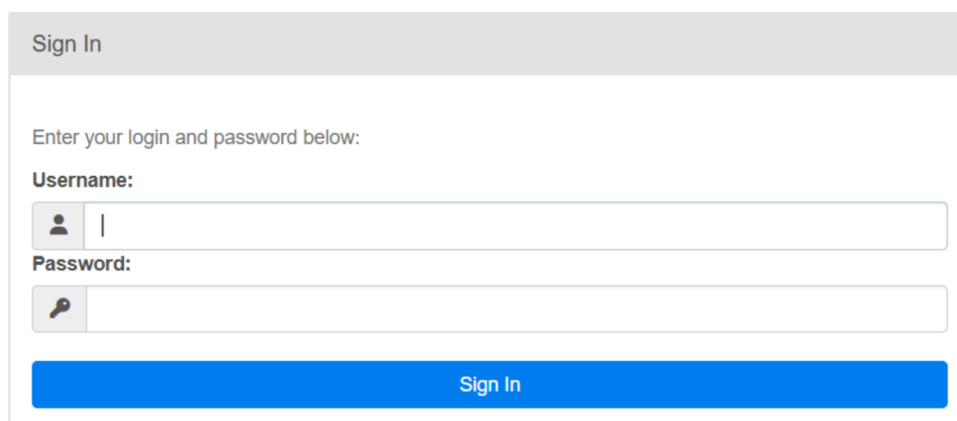
The image shows the Apache Airflow web interface's sign-in page. It has a light gray header with the text "Sign In". Below the header, there is a white box containing the login form. The form starts with the instruction "Enter your login and password below:". It then has two input fields: "Username:" with a user icon and "Password:" with a key icon. At the bottom of the form is a large blue button labeled "Sign In".

Figure 7: Apache Airflow Web Interface

The username and password is “admin” and “admin”. When you get in you will see the next screen (Figure 8) :

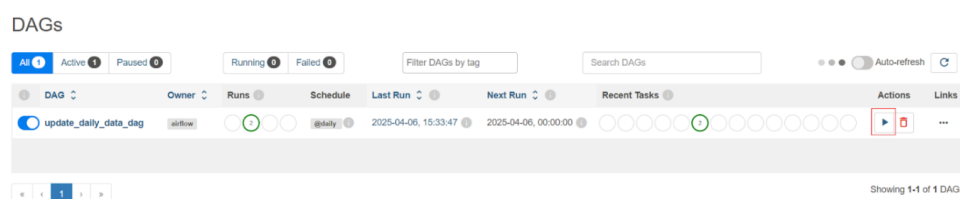


Figure 8: DAGs

Then you need to click on the play button to run the execution for our batch ingestion. With this, the execution for batch ingestion, data enrichment and preparation will start, and will download the files of the different datasources and moved to the folders that we have created (probably this will stay minutes).