Data Engineering-2 Project Report

Leveraging the Million Songs Dataset for Real-time Data Processing, Pipeline and Visualization.

Group Members:

1. Eshitha Dhanaraj - 11036071
2. Paramesh Anil - 11028138

Table of Contents:

1. Abstract
2. Introduction
3. Project Architecture
4. Dataset Description
5. Setup and Configuration
6. Data Flow and Processing
7. Results and Visualization
8. Challenges Faced
9. Conclusion
10. Future Work
11. References

1.Abstract:

The Million Songs project is an all-encompassing undertaking that puts forward a strong data pipeline for processing and analyzing the events generated from testing run on a simulated music streaming service. Designed to mimic user activity at a music website, the project incorporates advanced technologies for streaming data processing and rich analysis. Important objectives include observing user behavior, finding hits and assessing user's demographics.

2.Introduction:

The ever-changing world of data engineering is being compelled to renew itself by users increasing demands for high-tech analytical tools that can extract higher value results from more complex datasets. As for music streaming where user's preferences and actions determine the digital experience, using new tools of data engineering is vital. As part of Data Engineering 2, this project begins an adventure to discover and utilize the power of real-time data processing and analytics by stimulating the Million Songs Dataset.

2.1 Background of Million Songs Dataset:

The Million Songs Dataset stands as a testament to the transformative power of data in the music industry. Compiled by The Echo Nest, this comprehensive collection contains audio features, metadata, and contextual information for over a million songs. Covering a wide spectrum of genres, artists, and time periods, the dataset provides a rich foundation for exploring the intricacies of musical composition and user preferences.

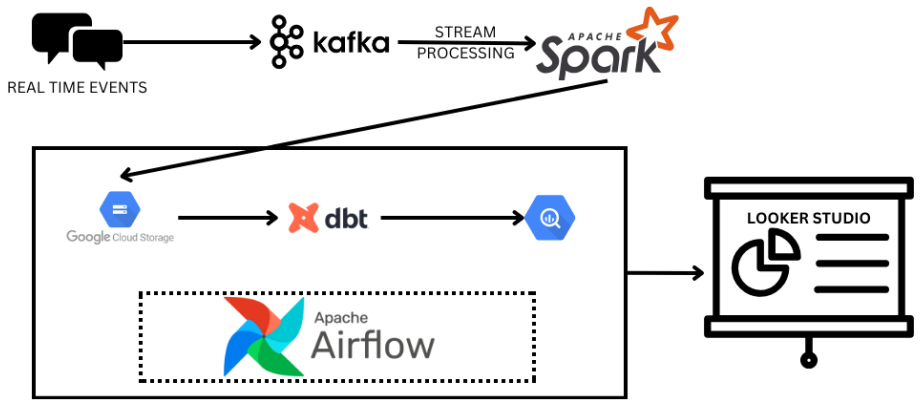
2.2 Significance of Real-Time Data Processing:

In the fast-changing world of music streaming, realtime data processing is a game changer. Traditional batch processing methods sometimes suffer from being unable to deliver the spot on insights that timely information about user behavior can provide, and cannot do so quickly enough to keep up with changing trends. Collecting and analysing the user interaction data with this project is one example of real-time data processing. This ability not only improves the accuracy of analytics, but also enables streaming platforms to react quickly to new trends. Thus, they can improve their content suggestions and user engagement.

2.3 Purpose and Scope of the Project:

The aim of this project is to demonstrate the powers that real-time data processing and analytics can wield in terms of the Million Songs Dataset. The purpose of this project is to collect, process and analyze events happening in real-time regarding user interactions within a music streaming service. By doing so we hope to be able to identifyuser behavior that drives song popularity or would otherwise provide guidance for interesting trends present in the dataset collected. This project, however, is not limited to data processing. It extends from the generation of synthetic events until completion of an overall analytics dashboard. Our projectLeveraging a broad range of tools and technologies, from Apache Kafka to Spark Streaming, Google Cloud Platform (GCP), etc., we hope that these can be brought together smoothly and usefully analysed as possible outcomes. We hope the exploration will not only just display technical skill but can create meaningful discourse that is applicable on a non-simulated scale; and allow us to dive deeper into what analytics talent could bring about in changing our perception of music.

3.Project Architecture:



3.1 Cloud Infrastructure (Google Cloud Platform): Google Cloud Platform (GCP) serves as the foundational cloud infrastructure for hosting and managing the entire system. GCP provides scalable computing resources, storage, and networking capabilities essential for deploying and operating the project components.

3.2 Infrastructure as Code (Terraform): Infrastructure as Code is implemented using Terraform, enabling the definition and provisioning of cloud resources in a consistent and repeatable manner. Terraform scripts automate the setup and configuration of the required infrastructure, ensuring efficiency and reliability.

3.3 Containerization (Docker, Docker Compose): Docker containers are employed for packaging applications and their dependencies, providing a consistent and portable environment across different stages of the project. Docker Compose is utilized for orchestrating the deployment of multiple containers and services.

3.4 Stream Processing (Apache Kafka, Spark Streaming):

- **Apache Kafka:** Acts as the backbone for real-time event streaming. It allows the ingestion and processing of synthetic events generated from the Million Songs Dataset, ensuring a reliable and fault-tolerant stream of data.
- **Spark Streaming:** Handles high-throughput stream processing, enabling the transformation and analysis of streaming data in real-time. Spark Streaming processes the incoming events to derive meaningful insights and prepares the data for storage.

3.5 Orchestration (Apache Airflow): Apache Airflow orchestrates the entire data pipeline by scheduling and coordinating tasks. It ensures the execution of real-time event processing tasks and triggers periodic data storage to the data lake, contributing to the overall efficiency and reliability of the system.

3.6 Data Storage (Google Cloud Storage): Google Cloud Storage serves as the data lake, storing processed data at regular intervals. It provides scalable and reliable storage, ensuring the availability of both raw and processed data for subsequent analysis.

3.7 Data Warehouse (BigQuery): BigQuery is utilized as the data warehouse, offering a powerful and scalable solution for executing analytics queries on structured data. It enables the exploration and extraction of valuable insights from the processed data.

3.8 Data Visualization (Data Studio): Data Studio is employed for creating intuitive and insightful visualizations. It transforms the analyzed data into a comprehensive dashboard, providing stakeholders with a user-friendly interface to interpret and derive insights from the metrics analyzed.

4.Dataset Description:

The Million Songs Dataset, a pioneering attempt to capture the essence of music through data. Built by the Echo Nest, data about each song is multifaceted; it covers a broad spectrum of genre, artists and time period. Essential features are track title, artist and release year; audio details such as tempo, key and loudness (louds), in each song with six melodic segments from the verses to choruses. In addition there is contextual information on familiarity of artists involved - how well one sings or plays guitar affects people's attitudes towards them-and whether a song sounds particularly This rich database provides researchers, data scientists and musical explorers with a unique opportunity to investigate the complex relationships between various aspects of music.

5. Setup and Configuration:

5.1 Google Cloud Platform (GCP) Initial Setup:

A. **Account and Project Creation:** We initiated our journey by creating a GCP account using our Google email ID. Subsequently, we set up our primary project named "Millionsongsde2," noting down the unique "Project ID" assigned to it. This Project ID will play a crucial role in the upcoming deployment of infrastructure using Terraform.

B. **Service Account Configuration:** To facilitate secure interactions with our GCP resources, we created a service account dedicated to our project. Initially, we granted this service account the "Viewer" role. We also obtained a key for this service account in JSON format, ensuring its confidentiality by renaming it to **google_credentials.json**

C. **SDK Download and Environment Setup:** To enable local development, we downloaded the Google Cloud SDK. We then set up an environment variable pointing to our downloaded GCP keys:**export GOOGLE_APPLICATION_CREDENTIALS= "google_credentials.json"**

This step ensures that our local environment is seamlessly configured for GCP interactions.

D. **Token Refresh and Authentication Check:**
- To verify and refresh our authentication, we ran the command:
  gcloud auth application-default login

This ensures that our local environment is authenticated and ready for further development.

Access Setup:

A.IAM Roles for Service Account: In the Identity and Access Management (IAM) section of the GCP Console, we fortified our service account by adding crucial roles such as Storage Admin, Storage Object Admin, and BigQuery Admin, in addition to the initial "Viewer" role.

B. Enable Necessary APIs: APIs are the building blocks of our project, and we enabled essential ones through the GCP Console. This includes IAM API and IAM Credentials API, and other pertinent APIs as needed, like DataProc.

C.Environment Variable Confirmation: To double-check the correct setup, we ensured that the

GOOGLE_APPLICATION_CREDENTIALS environment variable is in place:
exportGOOGLE_APPLICATION_CREDENTIALS="Google_Credentials.json" This confirms that our local environment is well-aligned with GCP for seamless development and interaction.

5.2 Terraform Infrastructure Setup:

In the Millionsongsde2 project, we seamlessly set up our infrastructure using Terraform, simplifying the deployment and management process. As part of our project workflow:

A.Repository Cloning: We initiated the setup by cloning the project repository to our local machine:
git clone https://github.com/ && cd millionsongsde2/terraform

B.Initialization and Dependency Download: Inside the Terraform directory, we initialized the project and downloaded necessary dependencies:
terraform init

C.Terraform Plan and Input Values: To visualize the setup plan, we executed:
Terraform plan

During this step, we provided input values for the GCS bucket name and our GCP Project ID as prompted, maintaining consistency throughout the project.

A. Infrastructure Application: Applying the infrastructure was straightforward:
terraform apply

B. Project Conclusion and Teardown: After successfully concluding the Millionsongsde2 project, we efficiently tore down the infrastructure to halt billing:
terraform destroy

5.3 VM SSH Setup:

In setting up SSH access to our VM instances for the Millionsongsde2 project,

A. **SSH Key Generation:** To ensure a secure connection, we generated an SSH key on our local system within the .ssh folder. We referred to a guide to ensure proper key creation.
B. **Public Key Addition to VM Instance:** We added the public key (.pub) to the respective VM instances. This step ensures secure authentication during SSH connections, establishing a trusted communication channel.
C. **Configuration File Creation:** Within the .ssh folder, we created a configuration file:
touch ~/.ssh/config

The configuration file was customized with a snippet specifying the External IP addresses of the Kafka, Spark (Master Node), and Airflow VMs. We also included the relevant username and the path to the SSH private key.

Host millionsongsde2-kafka
HostName 34.171.111.162
User param
IdentityFile C:\Users\mailt\.ssh\gcp

Host millionsongsde2-spark
HostName 34.67.50.178
User param
IdentityFile C:\Users\mailt\.ssh\gcp

Host millionsongsde2-airflow
HostName 34.72.219.241
User param
IdentityFile C:\Users\mailt\.ssh\gcp

D. SSH Access: With this setup, accessing the servers became seamless. In separate terminals, we used the following commands to SSH into the servers. We made sure to update the IP addresses if VMs were restarted:
ssh millionsongsde2-kafka
ssh millionsongsde2-spark
ssh millionsongsde2-airflow

Additionally, we set up port forwarding from the VMs to our local machine, enabling us to easily monitor and manage Kafka and Airflow UIs.

5.4 Setting up Kafka VM:

In the Millionsongsde2 project, the establishment of the Kafka environment on a dedicated compute instance is a crucial step. Here's a detailed account of what we did:

A. **SSH Connection:** We initiated an SSH connection to the Kafka VM using the designated hostname:
Ssh millionsongsde2-kafka

B. **Repository Cloning and Navigation:** After accessing the VM, we cloned the project repository and navigated to the Kafka folder:
git clone https://github.com/ && cd millionsongsde2/kafka

C. **Dependency Installation:** We streamlined the setup process by executing a script that installs Anaconda, Docker, and Docker Compose:
bash ~/millionsongsde2/scripts/vm_setup.sh && exec newgrp docker

D. **Environment Variable Setup:** To facilitate communication, we set environment variables, specifically noting the External IP of the Kafka VM:
export KAFKA_ADDRESS=34.171.111.162

It's important to set these variables every time a new shell session is created or if the VM undergoes a stop/start cycle.

E. **Kafka Start-up:** Moving on to Kafka start-up, we navigated to the Kafka folder, built the Docker images, and initiated the Kafka and eventsim containers: **cd ~/millionsongsde2/kafka && \ docker-compose build && \ docker- compose up**

F. **Kafka Control Center Check:** We ensured the availability of the Kafka Control Center on port 9021. This served as a vital checkpoint to verify the smooth operation of the Kafka setup.

G. **Eventsim Initialization:** In a separate terminal session on the Kafka VM, we initiated the generation of events using eventsim:
bash ~/millionsongsde2/scripts/eventsim_startup.sh

This script creates events for 50000 users over the next 2 hours. The container runs in detached mode, and we monitored progress through the logs:
docker logs --follow million_events

5.5 Setting up Spark Cluster:

In our project, configuring the Spark Streaming process in the DataProc cluster is a pivotal step for seamless communication with the Kafka VM instance. Here's breakdown of the process we followed:

A. SSH Connection to Master Node: We initiated an SSH connection to the Spark cluster's master node for configuration:

ssh millionsongsde2-spark

B. Repository Cloning and Navigation: Upon accessing the Spark cluster, we cloned the project repository and navigated to the Spark Streaming folder: **git clone https://github.com/ && cd millionsongsde2/spark_streaming**

C. Environment Variable Setup: To establish the necessary communication links, we set up environment variables, including the External IP of the Kafka VM and the name of the GCS bucket provided during Terraform setup:
export KAFKA_ADDRESS=34.171.111.162
export GCP_GCS_BUCKET=songsde2

It's essential to configure these variables in each new shell session or after restarting the cluster.

D. Reading Kafka Messages with Spark: We initiated the Spark Streaming process to read messages from Kafka using the following Spark-submit command: **spark-submit \ --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 \ stream_all_events.py**

Successful execution results in the creation of new Parquet files in the designated GCS bucket. Spark writes a file every two minutes for each topic, providing a structured storage of the streaming data.

E. Monitored Topics: The Spark Streaming process is configured to read from specific Kafka topics, namely:
   a. **listen_events**
   b. **page_view_events**
   c. **auth_events**

5.6 Setting up Airflow VM:

In the project, the establishment of the Airflow environment on a dedicated compute instance was a critical step, facilitating the orchestration of tasks and workflows. Here's a step-by-step guide to the process we followed:

A. **SSH Connection:** We began by establishing an SSH connection to the Airflow VM:
   ssh millionsongsde2-airflow

B. **Repository Cloning and Navigation:** Once connected, we cloned the project repository and navigated to the project root directory:
   git clone https://github.com/ && cd millionsongsde2

C. **Dependency Installation:** To streamline the setup, we executed a script that installs Anaconda, Docker, and Docker Compose:
   bash ~/millionsongsde2/scripts/vm_setup.sh && exec newgr docker

D. **Service Account File Transfer:** We transferred the service account JSON file from the local machine to the VM's **~/.google/credentials/** directory. It's crucial to ensure the filename is **google_credentials.json** to avoid DAG failures.

E. **Environment Variable Setup:** We set up environment variables, aligning them with the values used during Terraform setup:
   export GCP_PROJECT_ID=songsde2
   export GCP_GCS_BUCKET=millionsongsde2

F. **Airflow Startup:** The Airflow startup process commenced with the execution of the following script.
   bash ~/millionsongsde2/scripts/airflow_startup.sh &&
   cd ~/millionsongsde2/airflow

G. **Airflow Access and Logging:** Once the setup was complete, Airflow became available on port 8080. We accessed the

Airflow interface using the default username and password, which is set to 'airflow.' The system runs in detached mode.
To monitor logs from Docker, we used the following command:
docker-compose logs --follow

H. **Stopping Airflow:** When needed, Airflow can be stopped using the following command:
docker-compose down

6. Data Flow and Processing:

The Millionsongsde2 project employs a sophisticated data flow and processing pipeline to handle events generated from a simulated music streaming service. The end-to-end process involves several key stages, technologies, and components working seamlessly together:

Event Generation:
Source: Eventsim generates synthetic event data, simulating user interactions on a music streaming website.
Objective: Mimic real-world user behavior, creating diverse events like song listens, page views, and authentications.

Real-Time Streaming with Kafka:
Source: Simulated events are streamed in real-time to a Kafka broker.
Objective: Utilize Kafka as a scalable and fault-tolerant event streaming platform.
Communication: Dedicated VM instance with an open port (9092) facilitates communication.

Spark Streaming for Real-Time Processing:
Spark DataProc Cluster: Deployed to consume events from Kafka topics.

Processing: Real-time processing of events using Spark Streaming.
Destination: Processed data is written to Google Cloud Storage (GCS) in Parquet format.
Objective: Enable near real-time data transformation and storage.

Data Warehouse and Transformation with dbt:
BigQuery Load: Processed data is loaded into BigQuery, serving as the project's data warehouse.
dbt Transformation: Utilize dbt for transformations and aggregations on the raw data.
Objective: Prepare data for analytics with meaningful tables and structures.

Workflow Orchestration with Airflow:
Airflow DAGs: Define workflows with Directed Acyclic Graphs.
Task Scheduling: Coordinate tasks such as Spark Streaming jobs, data loading to BigQuery, and dbt transformations.
Objective: Efficiently manage and schedule the entire data pipeline.

Data Visualization with Data Studio:
BigQuery as Source: Utilize analytics-ready data in BigQuery.
Data Studio Dashboards: Create interactive and customizable dashboards.
Objective: Visualize metrics like popular songs, active users, and user demographics.

Infrastructure Management with Terraform:
Infrastructure as Code (IaC): Manage GCP resources, Kafka, Spark, and Airflow components using Terraform.
Objective: Ensure reproducibility, scalability, and ease of infrastructure maintenance.

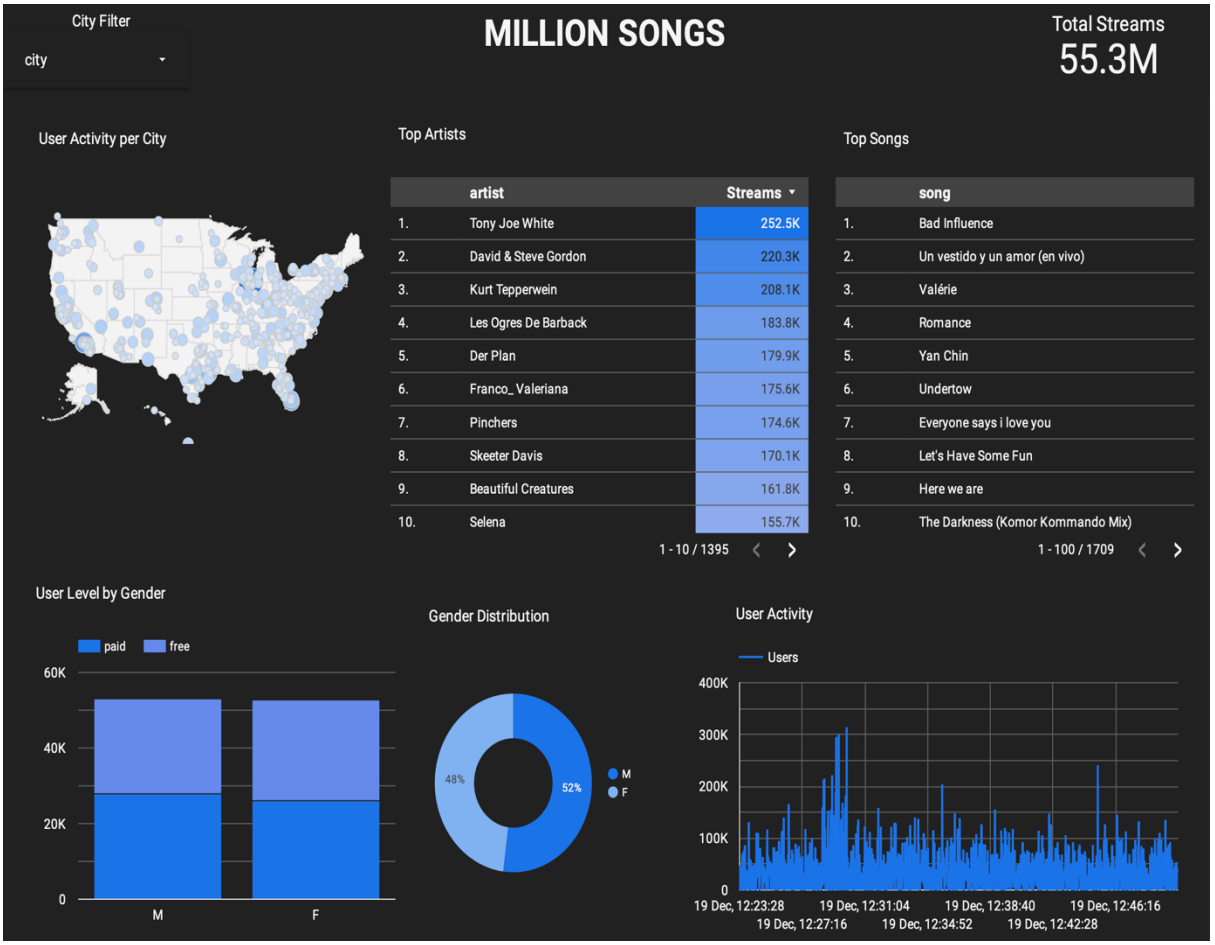
Continuous Monitoring and Logging:

Logging Mechanisms: Docker logs, Airflow logs, and Spark logs provide continuous monitoring.
Objective: Identify and resolve issues promptly, ensuring the health and performance of each component.

Teardown Process:
Resource Management: Terraform scripts for efficient teardown of infrastructure.
Objective: Avoid unnecessary costs by deprovisioning resources when not in use.

7. Results and Visualization:

GCP's built-in compatibility across our entire data pipeline was a big factor in why we chose it for Google Cloud Platform (GCP) and Looker. GCP's tight integration, especially with BigQuery as our data warehouse, means easy movement of the data and maximum performance. That's why Looker, which was picked for its innovative analytics platform, complements GCP by building in easy-to-use tools for data exploration and visualization. The central data modeling capabilities of Looker also help build a common point-of-view across our analyses. A user-friendly way of understanding music streaming metrics is the platform's interactive dashboards, which allow users to look at data dynamically.



Business Questions:

A) What patterns emerge when comparing user activity in different cities, and how can this information be leveraged to enhance user engagement and satisfaction in specific regions?

B) Based on the popularity of top artists and songs, how can the music streaming service optimize its content strategy to cater to the preferences of its diverse user base?

8. Challenges Faced:

a) GCP Credits
b) Setting up the instances/Google Console manually.
c) Not sending messages to our Kafka broker with Eventsim.
d) Creating DAG for the dataset
e) Credits end up soon when we simulate more users and vice versa if the users are less there are no insights.

9. Conclusion:

In wrapping up the Millionsongsde2 project, we've successfully built a dynamic and efficient data processing pipeline for real-time analytics in the music streaming realm. Leveraging Google Cloud Platform (GCP) as our reliable infrastructure and Looker for insightful visualizations, we've created a seamless and scalable ecosystem. From generating synthetic events to uncovering valuable insights about user behavior and popular songs, our project exemplifies the power of cloud computing and data visualization. The combination of GCP and Looker not only met our project goals but also laid the groundwork for future enhancements. Millionsongsde2 is more than just a data project; it's a successful endeavor in understanding and enhancing the virtual music experience.

10.Future Work:

- Build dimensions and facts incrementally instead of full refresh.
- Create dimensional models for additional business processes.
- Add more visualizations.
- Cloud Composer for Airflow

11.References:

Dataset link: http://millionsongdataset.com/

Eventsim: viirya/eventsim: Event data simulator. Generates a stream of pseudo-random events from a set of users, designed to simulate web traffic. (github.com)

Terraform: hashicorp/terraform: Terraform enables you to safely and predictably create, change, and improve infrastructure. It is a source-available tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned. (github.com)

Looker Studio Visualization Link: https://lookerstudio.google.com/reporting/8f3be2eb-157b-45c5-b86b-fe1a762008d3

Contribution of work on this Project: Since both of us are new to these technologies, we worked together on each and every step from setting up the GCP, Kafka, Spark Cluster, Airflow as well as Visualization. Both of us contributed equal amount of Time and Energy for successfully completing the project.s