# ETL Pipeline for Music Streaming Data Analysis

---

## 1. Introduction

### 1.1 Purpose

This document outlines the technical solution for an ETL pipeline that processes music streaming data. The pipeline integrates data from Amazon S3 and a RDS, performs transformations, computes Key Performance Indicators (KPIs), and loads the processed data into Amazon Redshift for analytics.

### 1.2 Scope

- Extract streaming data from **Amazon S3**, user and songs data from Amazon **RDS**.
- Validate data integrity before processing.
- Transform data, compute KPIs (Genre-Level and Hourly KPIs).
- Load the processed data into **Amazon Redshift** using an optimized upsert strategy.
- Implement **error handling, logging, and cleanup mechanisms**.

### 1.3 Definitions, Acronyms, and Abbreviations

- **ETL** – Extract, Transform, Load
- **DAG** – Directed Acyclic Graph (Airflow workflow)
- **KPI** – Key Performance Indicator
- **S3** – Amazon Simple Storage Service
- **RDS** – Amazon Relational Database Service
- **Redis** – In-memory key-value store for caching

### 1.4 References

- [ETL with Airflow Documentation] (User's provided PDF)
- Apache Airflow Documentation (https://airflow.apache.org/)
- Amazon Redshift Documentation (https://docs.aws.amazon.com/redshift/)

### 1.5 Overview

The document provides an **architectural representation**, **use-case workflows**, **deployment strategy**, and **data handling processes** for the ETL pipeline.

---

## 2. Architectural Representation

### 2.1 Architecture Overview

The ETL pipeline is implemented using **Apache Airflow**, with tasks for **extraction, validation, transformation, KPI computation, and data loading**. Redis is used for **temporary storage**, and Amazon Redshift is used for **analytical queries**.

---

## 3. Architectural Goals and Constraints

### 3.1 System-wide Design Decisions

- Use **Airflow DAGs** for workflow orchestration.
- Store some temporary data in **Redis** instead of XCom due to size limitations.
- Use **Amazon Redshift** for analytical storage with optimized upserts.

### 3.2 Architectural Constraints

- **Data Volume**: Must handle growing streaming data efficiently.
- **Processing Time**: The pipeline should be completed within an acceptable time for daily analytics.
- **Reliability**: Implement error handling and retry.

---

## 4. Use-Case View

### 4.1 Use-Case Model

**Primary Users**

- **Data Engineer**: Manages pipeline and ensures data integrity.
- **Business Analyst**: Queries Redshift for insights.

**Use Cases**

1. **Extract users and songs RDS and streaming data from S3**

2. **Validate dataset integrity**
3. **Transform raw data into meaningful KPIs**
4. **Store and load data efficiently into Redshift**

---

**5. Logical View**

**5.1 Major Components**

- **Airflow DAG**: Manages task execution order.
- **Data Extraction Module**: Extracts user and songs and streaming data.
- **Data Validation Module**: Ensures required columns exist.
- **Data Transformation Module**: Processes raw data and computes KPIs.
- **Redshift Data Loading Module**: Efficiently loads data with upserts.
- **Logging & Cleanup Module**: Handles errors and temporary file cleanup.

---

**6. Process View**

**6.1 ETL Pipeline Execution**

1. **Extract data from S3 & RDS.**
2. **Store temporary data in Redis for faster access.**
3. **Validate data columns to ensure integrity.**
4. **Perform transformations and compute KPIs.**
5. **Load transformed data into Redshift using upsert strategy.**
6. **Perform cleanup of temporary Redis data and other temporary files.**

---

**7. Deployment View**

**7.1 Deployment Components**

- **Airflow DAG**: Deployed on an Airflow instance.
- **Amazon S3**: Stores raw streaming data.

- **Redis**: Caches temporary files and keys.
- **Amazon Redshift**: Stores processed analytical data.

---

**8. Implementation View**

**8.1 Development Environment**

- **Language**: Python (for Airflow DAG and ETL tasks).
- **Orchestration**: Apache Airflow.
- **Storage**: S3 (raw data), RDS (tables), Redis (temporary), Redshift (processed data).
- **Version Control**: Git.

**8.2 Configuration Management**

- **Environment Variables**: AWS credentials, database connection settings.
- **Logging**: Airflow logs pipeline execution details.
- **Error Handling**: Tasks use a failure cleanup strategy.

---

**9. Data View (Optional)**

**9.1 Data Storage Mechanism**

- **S3**: Stores raw streaming data files.
- **Redis**: Caches intermediate transformation results.
- **Redshift**: Stores processed and analytical data.

**9.2 Security & Integrity**

- **S3 Access Controls**: Restrict unauthorized data access.
- **Redshift Permissions**: Implement role-based access.

---

**10. Size and Performance**

- **Processing Time**: The DAG should complete within daily execution windows.

- **Scalability**: The pipeline should handle increasing streaming data volume efficiently.
- **Optimizations**:
    - Use **Redis** for caching instead of slow XComs.
    - Perform **batch inserts** into Redshift.
    - Implement **staging tables** in Redshift for upserts.

---

**11. Quality Attributes**

| Quality Attribute | Implementation Strategy |
|---|---|
| **Reliability** | Airflow DAGs with retries & failure handling. |
| **Scalability** | Distributed task execution in Airflow. |
| **Performance** | Redis caching, optimized Redshift inserts. |
| **Security** | AWS IAM roles & encrypted S3 storage. |
| **Maintainability** | Modular Python scripts for each task. |

---

**12. Appendices**

**12.1 Glossary**

| Term | Description |
|---|---|
| **DAG** | A directed acyclic graph in Airflow defining the ETL workflow. |
| **Redis** | An in-memory store used for temporary caching. |
| **Upsert** | A database operation that inserts new rows and updates existing ones. |

**12.2 Index**

- Git Repo: (https://github.com/Zukizuk/music-streaming-etl-airflow-workflow)

**12.3 Revision History**

| Date | Author | Description |
| --- | --- | --- |
| 2025-03-16 | Marzuk | Initial draft |