ETL Pipeline Using Apache Airflow and MySQL

1. Overview of the Document

This document describes how to build an end-to-end ETL (Extract, Transform, Load) pipeline using Apache Airflow, focusing on a sales dataset. It outlines the step-by-step process: from setting up the environment, building the pipeline tasks, transforming the data, and loading it into a MySQL database. The document also explains the rationale for saving DataFrames between steps, and ends with a summary of challenges encountered during implementation.

2. Airflow Overview

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. It allows you to break down complex workflows into tasks and manage them using Directed Acyclic Graphs (DAGs). Airflow ensures modularity, scalability, and reliability in data pipeline development.

3. Airflow Installation

1. Create a virtual environment:

```
python -m venv airflow_env
source airflow_env/bin/activate
```

2. Install Airflow with constraints:

```
pip install apache-airflow==2.9.0 --constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.9.0/constraints-3.8.txt"
```

3. Initialize the database:

```
airflow db init
```

4. Create a user and start the scheduler and web server.

ETL Pipeline Using Apache Airflow and MySQL

4. Connecting Airflow with MySQL

Use MySqlHook or SQLAlchemy.

Define connection in Airflow UI: Admin -> Connections -> Add Connection.

Use Docker host IP when running MySQL inside Docker.

Example:

```
conn = mysql.connector.connect(
    host='host.docker.internal',
    database='sales3',
    user='root',
    password='your_password'
)
```

5. Extract Code

Extracts data from a CSV file and saves it locally:

```
def extract():
    df = pd.read_csv('/path/to/raw_sales.csv')
    df.to_csv('/path/to/extracted.csv', index=False)
```

6. Transform Code

Transform Code 1:

- Rename columns
- Drop duplicates
- Separate categorical and numerical columns

Transform Code 2:

- Handle nulls (categorical -> mode, numerical -> mean)

Transform Code 3:

- Convert order_date to datetime

- Create order_year

Encode categorical and scale numerical columns.

Save as transformed.csv

7. Load Code

Loads transformed data into MySQL:

```
def load():
    engine = create_engine('mysql+mysqlconnector://root:password@host.docker.internal/sales3')
    df = pd.read_csv('transformed.csv')
    df.to_sql('sales_data', engine, index=False, if_exists='replace')
```

MySQL Table:

```
CREATE TABLE IF NOT EXISTS sales_data (...);
```

8. DAG Code

Airflow DAG automating the ETL:

```
with DAG('sales_etl_label_encode', ...) as dag:
    extract_task >> transform_task >> load_task
```

ETL Pipeline Using Apache Airflow and MySQL

9. Prove Clean Data Load into MySQL

Check loaded data:

SELECT * FROM sales_data LIMIT 10;

Verify:

- All values encoded
- No NULLs or duplicates
- Includes order_year

10. Challenges

- Environment setup
- Data validation
- Task isolation
- Datetime formatting
- SQL errors

Why Save the DataFrame at Each Step?

1. Separation of Concerns
2. Fault Tolerance
3. Debugging
4. Airflow Task Isolation