# Flight Delay Data Pipeline Setup with Airflow & SQL Server

# Flight Delay Data Pipeline with Apache Airflow and SQL Server

This document outlines the complete setup for a data engineering pipeline that ingests flight delay data from a CSV file into Microsoft SQL Server using Apache Airflow. It includes setup, configuration, and troubleshooting steps for Dockerized Airflow, ODBC integration, SQL Server authentication, and data loading.

---

## ✨ Project Overview

- \*\*Goal\*\*: Build an automated data pipeline using Airflow that loads flight delay data from a CSV file into SQL Server.

- \*\*Tools Used\*\*: Apache Airflow, Docker, SQL Server (SSMS), pyodbc, pandas

- \*\*Output\*\*: Structured SQL Server table ready for Tableau dashboarding.

---

## 🔍 Dataset Overview

- Source: Kaggle/real-world CSV dataset

- Key fields:

- `year`, `month`, `carrier`, `airport`, `arr\_flights`, `arr\_del15`, `carrier\_ct`, etc.

- File location inside Docker: `/opt/airflow/data/Airline\_Delay\_Cause.csv`

---

## 🚀 Phase 1: Local Script (Python + pyodbc)

1. Tested reading the CSV using pandas.

2. Connected to local SQL Server using pyodbc.

3. Inserted records into SQL Server table manually.

---

## 📆 Phase 2: Airflow DAG Development

### DAG Features:

- Reads CSV

- Connects to SQL Server

- Deletes existing data

- Inserts cleaned data into `flight\_delays` table

### DAG Operator Used:

- `PythonOperator`

### Path:

- DAG file: `dags/load\_flight\_data\_to\_sqlserver.py`

---

## 💪 Phase 3: Docker Configuration

### Custom Dockerfile

Used to install:

- `pyodbc`

- `ODBC Driver 17 for SQL Server`

- System dependencies

FROM apache/airflow:2.8.1-python3.10

USER root

RUN apt-get update && apt-get install -y \

gnupg curl apt-transport-https unixodbc-dev gcc g++ \

software-properties-common && \

curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add - && \

curl https://packages.microsoft.com/config/debian/10/prod.list > /etc/apt/sources.list.d/mssql-release.list && \

apt-get update && ACCEPT\_EULA=Y apt-get install -y msodbcsql17

USER airflow

RUN pip install pandas pyodbc

### docker-compose.yml Changes:

- Replaced `image:` with `build: .`

- Mounted `./data:/opt/airflow/data`

---

## 🛠️ Phase 4: SQL Server Configuration

### Enabled TCP/IP and static port 1433:

- Done via \*\*SQL Server Configuration Manager\*\*

### Enabled SQL Authentication:

- Switched to \*\*SQL Server and Windows Authentication mode\*\*

### Restarted SQL Server

---

## 👨‍💼 Phase 5: Login & Permissions

### Created login in SSMS:

CREATE LOGIN airflow\_user WITH PASSWORD = 'Test@12345!';

USE Airflow;

CREATE USER airflow\_user FOR LOGIN airflow\_user;

ALTER ROLE db\_datareader ADD MEMBER airflow\_user;

ALTER ROLE db\_datawriter ADD MEMBER airflow\_user;

---

## ⚠️ Troubleshooting Highlights

### Issue: `Trusted\_Connection` doesn't work in Docker

- ❌ Fixed by switching to SQL login with `UID` and `PWD`

### Issue: SSL certificate not trusted

- ❌ Fixed by adding `Encrypt=yes;TrustServerCertificate=yes;` to the connection string

### Issue: `TRUNCATE TABLE` failed

- ❌ Fixed by replacing with `DELETE FROM flight\_delays` (avoids ALTER permission requirement)

### Issue: Login failed (18456)

- ❌ Fixed by recreating login with correct password and mapping it to the DB

### Issue: Table not found

- ❌ Fixed by manually creating the table in SSMS

---

## 📊 Final DAG: Connection Example

conn\_str = (

f'DRIVER={{ODBC Driver 17 for SQL Server}};'

f'SERVER=host.docker.internal,1433;'

f'DATABASE=Airflow;'

f'UID=airflow\_user;'

f'PWD=Test@12345!;'

f'Encrypt=yes;'

f'TrustServerCertificate=yes;'

)

---

## 🏁 Outcome

- Airflow DAG successfully loads 170K+ records into SQL Server

- Data is accessible for Tableau/Power BI visualization

- Portable, documented pipeline for production extension

---

## 📖 Next Steps

- Add logging and metrics to DAG

- Create Tableau dashboard from `flight\_delays` table

- Add scheduling and notification logic

- Archive previous loads to a history table