Project Documentation: Automated Environmental Air Quality Data Pipeline

Author: StanJohn04 **Date:** June 23, 2025 **Version:** 1.2

Table of Contents

1. Project Overview & Goals

1.1. Executive Summary

This project is a complete, automated data engineering pipeline designed to collect, store, and process daily environmental data for selected locations in Georgia, USA. The system leverages Windows Task Scheduler to run a daily Python script that fetches data from external weather and air quality APIs. The processed data is then stored in a local PostgreSQL database. This robust, automated data collection provides a clean and rich dataset that serves as the foundation for future time-series analysis, machine learning modeling, and environmental anomaly detection.

1.2. Core Objectives

- **Data Engineering:** Design and implement a robust, automated ETL (Extract, Transform, Load) pipeline.
- **Relational Databases (SQL):** Design a normalized database schema and manage a PostgreSQL database, including user roles and permissions.
- **Automation:** Create a reliable, scheduled task to ensure consistent daily data collection on a Windows PC.
- **Data Analysis & AI:** Prepare a clean, historical dataset suitable for future exploratory data analysis (EDA) and the development of predictive AI/ML models.

1.3. Key Technologies & Skills Demonstrated

- **Programming:** Python
- **Libraries:** Pandas, SQLAlchemy, Requests, python-dotenv
- **Databases:** PostgreSQL
- **APIs:** Google Air Quality API, Open-Meteo Weather API
- **Operating Systems:** Windows 11
- **Automation:** Windows Task Scheduler, Batch Scripting
- **Version Control:** Git, GitHub

- **Core Skills:** ETL Pipeline Development, Database Administration, API Integration, Task Automation, Data Modeling.

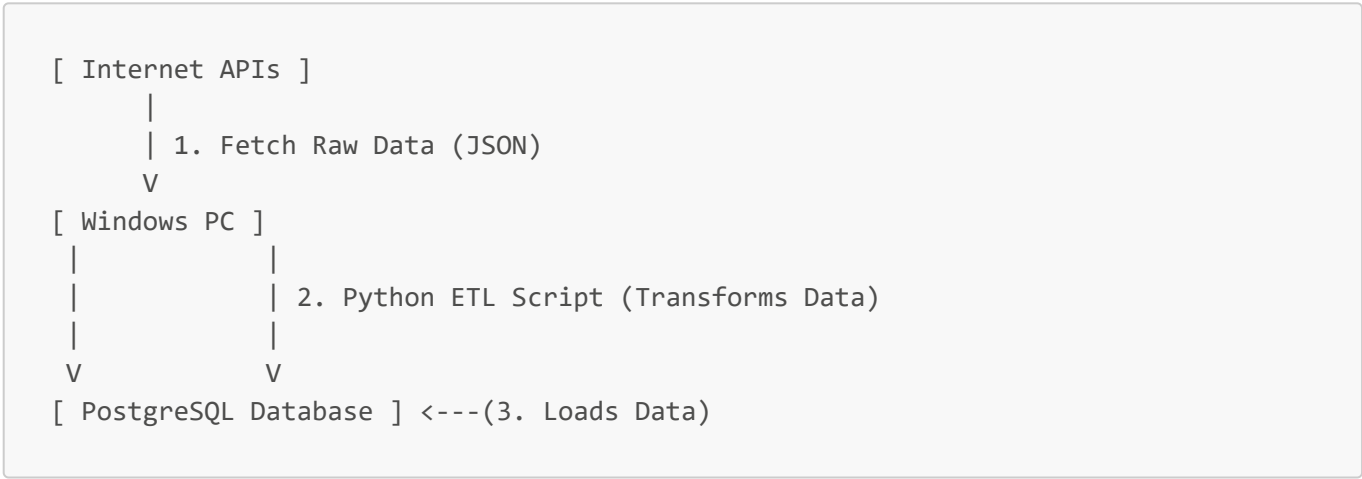
---

2. System Architecture

The system operates on a single-machine architecture, where all processes run locally on a Windows PC.

- **Windows PC (Host Machine):**
  - **Role:** Acts as both the ETL worker and the database server.
  - **Software:**
    - Runs a PostgreSQL server instance for data storage.
    - Executes the Python ETL script via an automated daily task.
  - **Data Flow:** The Python script, triggered by Task Scheduler, fetches data from internet APIs, transforms it, and loads it directly into the local PostgreSQL database.

Data Flow Diagram



---

3. Data Engineering & ETL Pipeline

3.1. Data Sources

- **Google Air Quality API:** Provides current and historical (30-day lookback) air quality data, including an overall Air Quality Index (AQI) and concentrations for key pollutants like PM2.5, PM10, O3, NO2, CO, and SO2.
- **Open-Meteo API:** Provides forecast and historical weather data, including mean temperature, precipitation sum, and max wind speed. The pipeline intelligently uses the `forecast` endpoint for recent dates and the `archive` endpoint for older dates.

3.2. Database Schema

The PostgreSQL database (`aqi_db`) contains two core tables:

1. `locations`: Stores static information about each data collection site.

```
CREATE TABLE locations (
    id SERIAL PRIMARY KEY,
    city VARCHAR(100) NOT NULL,
    country VARCHAR(100) NOT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    UNIQUE(city, country)
);
```

2. `daily_readings`: Stores the time-series data, linking a location to a specific day's environmental readings. The `ON CONFLICT` clause in the load script ensures data integrity by updating a day's record if the script is run more than once, making the pipeline idempotent.

```
CREATE TABLE daily_readings (
    id SERIAL PRIMARY KEY,
    location_id INTEGER NOT NULL REFERENCES locations(id),
    reading_date DATE NOT NULL,
    aqi REAL,
    pm10 REAL,
    pm25 REAL,
    o3 REAL,
    no2 REAL,
    co REAL,
    so2 REAL,
    temperature_celsius REAL,
    precipitation_mm REAL,
    wind_speed_kmh REAL,
    UNIQUE(location_id, reading_date)
);
```

3.3. The ETL Scripts

- `scripts/etl.py`**:** The primary operational script. It is configured to fetch the current day's data for all specified locations, transform the API responses into a clean, structured format, and load the results into the local PostgreSQL database.
- `scripts/historical_backfill.py`**:** A one-time script used to populate the database with the initial ~30 days of historical data. This script contains specialized logic to interact with the historical endpoints of the APIs and average hourly data into daily summaries.

3.4. Automation

The daily execution of the pipeline is handled by **Windows Task Scheduler**. A task is configured to run a batch script (`run_etl.bat`) at a set time each day. The batch script is essential for reliably activating the correct Anaconda environment before executing the Python script.

`run_etl.bat` **contents:**

```
@echo OFF
ECHO Starting daily AQI ETL process...

:: This command initializes Conda in the batch script's environment
call C:\Users\stant\anaconda3\Scripts\activate.bat

:: This command activates your specific project environment
call conda activate AQI_Predict

:: This command runs your python script using its full path
python C:\Users\stant\Documents\Projects\AQI_Predict\scripts\etl.py

ECHO ETL process finished.
```

---

4. System & Environment Configuration

4.1. Python Environment Setup

- **Anaconda (PC):** A Conda environment (`AQI_Predict`) was used for all development and execution. This ensures all project dependencies (e.g., Pandas, SQLAlchemy) are isolated from the system's base Python installation. All required packages are documented in the `requirements.txt` file.

4.2. Environment Variables (`.env`)

The scripts securely manage credentials via a `.env` file, which is excluded from version control. The following variables are required:

- `GOOGLE_API_KEY`: The API key used for authenticating with the Google Air Quality API.
- `DB_HOST`: The hostname of the database server. For this project, it is set to `localhost`.
- `DB_PORT`: The port for the database server, typically `5432` for PostgreSQL.
- `DB_NAME`: The name of the database to connect to (e.g., `aqi_db`).
- `DB_USER`: The username for connecting to the database (e.g., `stant`).
- `DB_PASSWORD`: The password for the specified database user.

---

5. Next Steps & Future Enhancements

With a robust and automated data pipeline in place, the project is now ready for the analysis and deployment phases.

5.1. Immediate Next Steps: Analysis & Modeling

This work will be conducted in the `notebooks/01_data_analysis_and_modeling.ipynb` Jupyter Notebook.

1. **Exploratory Data Analysis (EDA):**
    - Analyze trends, seasonality, and patterns in the collected data over time.
    - Create visualizations (time-series plots, correlation heatmaps) to understand the relationships between weather patterns and pollutant levels.

2. **Time-Series Forecasting:**
    - Develop a machine learning model to predict the next day's Air Quality Index (AQI) based on weather forecasts and previous days' readings.
3. **Anomaly Detection:**
    - Implement an unsupervised learning model to identify days with unusual or unexpected pollution events.

5.2. Future Enhancement: Raspberry Pi Deployment

To create a more robust, 24/7 data collection system that does not depend on the PC being on, a planned future enhancement is to migrate the ETL process to a Raspberry Pi.

- **Proposed Architecture:** The Raspberry Pi would become the dedicated ETL worker, running the `etl.py` script on a `cron` schedule. The PC would be reconfigured to act as a dedicated database server, accepting remote connections from the Pi over the local network.
- **Benefits:** This distributed architecture is more energy-efficient and reliable for long-term, uninterrupted data collection, separating the data processing task from the data storage and analysis environment.

---

6. Appendix: Documentation & Resource Links

- **Version Control:** Project GitHub Repository
- **Data Sources:**
    - Google Air Quality API Documentation
    - Open-Meteo API Documentation
- **Technologies:**
    - PostgreSQL Documentation
    - Anaconda Documentation
    - Windows Task Scheduler Documentation