Weather and Space Alert System - Technical Documentation

Author: Shandro Mitra

Contact: shandro.mitra@gmail.com

GitHub: <https://github.com/ShandroMitra/weather_and_space_alert_system.git>

Language: Python 3.10.11

# Overview

The Weather and Space Alert System is a Python-based automated monitoring tool that gathers real-time weather and space data, processes and validates it, and triggers email alerts based on critical conditions. It supports data ingestion from public APIs (Open-Meteo and NASA), performs robust validation and transformation, and stores clean data in a PostgreSQL database with audit logging.

# Core Features

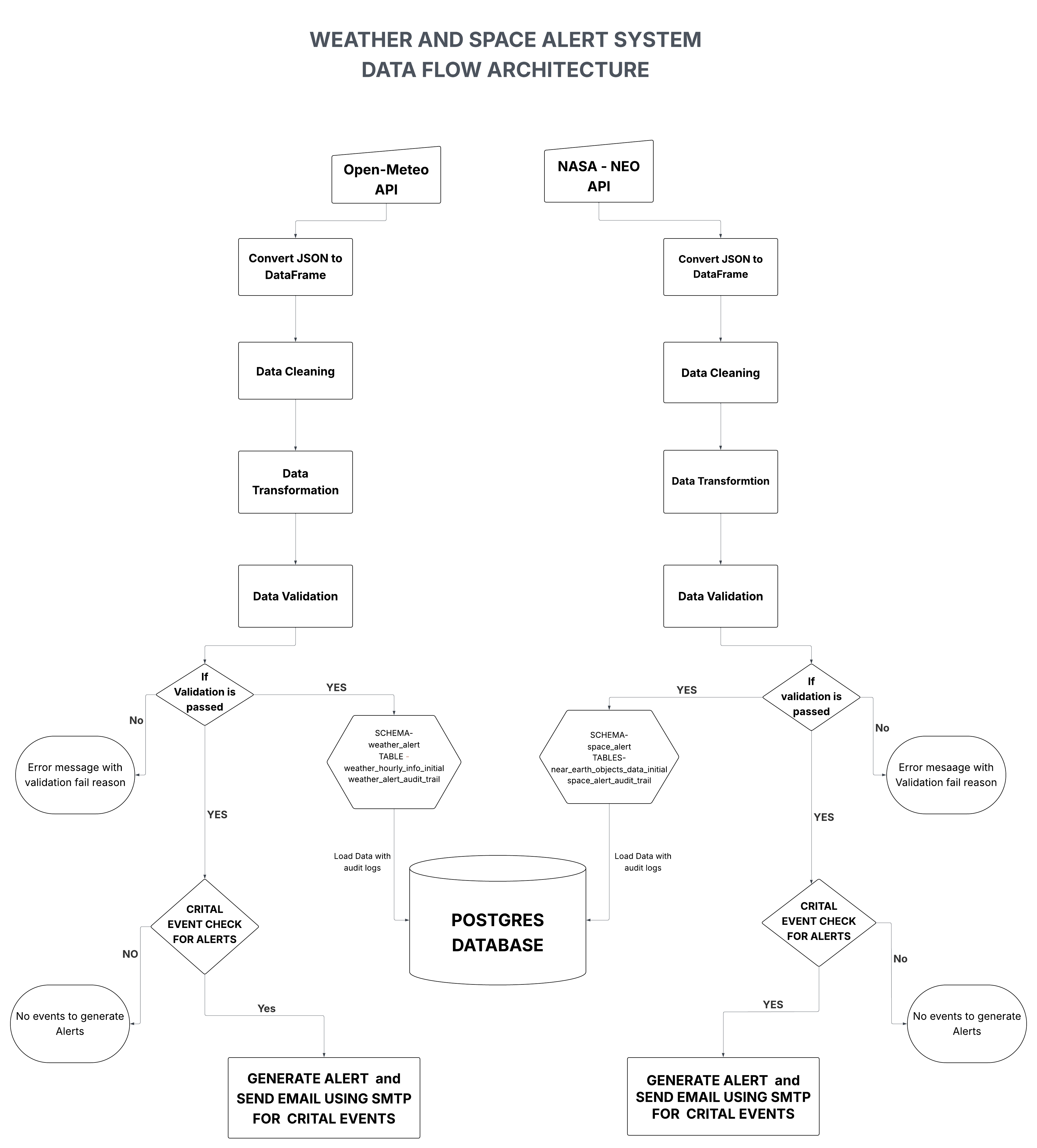
## 1. Weather Alert System

- Data Source: Open-Meteo API for hourly weather data across major Indian cities.  
- Processing: JSON to DataFrame, clean, mark as processed  
- Transformation: Feature engineering, unique ID, soft delete, batch ID  
- Validation: Type checks, range checks, uniqueness, date checks  
- Storage: Auto-creates schema/table, logs in audit table  
- Monitoring: Detects critical weather conditions, sends email alerts

## 2. Space Alert System

- Data Source: NASA Near Earth Object (NEO) API  
- Processing: JSON to DataFrame, clean, optimize types, time conversion  
- Transformation: Feature engineering, unique ID, soft delete, batch ID  
- Validation: Presence, uniqueness, range checks, date checks  
- Storage: Prevents duplicates, logs audit info  
- Monitoring: Detects NEO threats, sends email alerts

# Architecture

  
  
**ETL Pipeline — 7 Core Steps**

The following 7 steps define the ETL framework applied to both weather and space modules:

1. Extraction  
    → Pull JSON data from Open-Meteo and NASA APIs using robust retry logic (requests + retry-requests).
2. Parsing & Conversion  
    → Convert JSON responses into structured pandas DataFrames with proper datetime indexing.
3. Cleaning  
    → Round numeric columns, remove duplicates, detect and handle missing data, handle outliers, and convert UTC to IST (for space data).  
    → Mark each row with a processing\_status = 'cleaned'.
4. Transformation  
    → Feature engineering: custom logic to derive flags (e.g., is\_potentially\_hazardous, weather\_type).  
    → Add metadata: batch\_id (UUID), is\_deleted (soft delete flag), and unique IDs based on key identifiers.
5. Validation  
    → Implement checks:
   1. Data type integrity
   2. Null/mandatory fields
   3. Logical bounds (e.g., temperature, diameter)
   4. Uniqueness constraints
   5. Date constraints (no future timestamps)
6. Load  
    → Use SQLAlchemy to connect to PostgreSQL.  
    → If schema/table doesn't exist, create automatically.  
    → Load transformed data and append audit logs (row counts, batch ID, load timestamp).
7. Monitoring & Alerting  
    → Analyze cleaned, validated data to detect:
   1. Weather: heatwaves, cyclones, fog, etc.
   2. Space: hazardous NEOs, high velocity or close approaches  
       → If alerts are triggered, send an automated email using preconfigured SMTP server.

# Installation & Deployment

1. Clone the repository:  
 git clone <repository-url>  
  
2. Install dependencies:  
 pip install -r requirements.txt  
  
3. Create a `.env` file with the following keys:  
 API\_KEY\_NASA=  
 DB\_URL=  
 EMAIL\_USER=  
 EMAIL\_PASS=  
  
4. Run the main script:  
 python main.py  
  
5. Monitor logs for status and alerts

# Dependencies

- requests  
- python-dotenv  
- pandas  
- scikit-learn  
- numpy  
- SQLAlchemy  
- openmeteo-requests  
- requests-cache  
- retry-requests