The Weather Data Ingestion Pipeline is designed to automate the process of retrieving weather data from the OpenWeatherMap API, processing it, and storing it in a PostgreSQL database. This system leverages Docker for containerization and Apache Airflow for workflow orchestration.

Table of Contents

What is this technical documentation?

This technical documentation provides a detailed overview of the Weather Data Ingestion Pipeline project. It describes the architecture, components, data flow, deployment process, error handling, scalability, security considerations, and potential future enhancements.

System Architecture

Components

1. **PostgreSQL Database**
    - Purpose: To store ingested weather data.
    - Hosted within a Docker container.
    - Includes a schema designed to store city-specific weather information.
2. **Data Fetching Service**
    - A Python-based service.
    - Periodically fetches weather data from the OpenWeatherMap API.
    - Runs in a Docker container.
3. **Apache Airflow**
    - Manages and schedules the data ingestion pipeline.
    - Deployed via Docker Compose.
    - Ensures that the data fetching service runs at predefined intervals.

High-Level Architecture

The architecture consists of interconnected Docker containers, each responsible for a specific part of the pipeline:

1. A PostgreSQL container for the database.
2. A container for the data fetching service.
3. An Airflow container to orchestrate the pipeline.

These containers communicate over a Docker network, allowing seamless data transfer and management.

Component Descriptions

PostgreSQL Database

- **Image**: Official PostgreSQL Docker image.
- **Initialization**: The database and tables are initialized via SQL scripts that are executed when the container starts.
- **Schema**: Includes tables to store weather data with fields such as city, temperature, description, and timestamp.

Data Fetching Service

- **Technology Stack**: Python, Pandas, SQLAlchemy.
- **Functionality**:
    - Fetches data from the OpenWeatherMap API.
    - Transforms the data into a structured format suitable for database insertion.

    - Uses environment variables for API key management.
- **Execution**: Scheduled and triggered by Apache Airflow.

Apache Airflow

- **Image**: Official Apache Airflow Docker image.
- **Configuration**:
    - Airflow is configured to use the PostgreSQL database for metadata storage.
    - A Directed Acyclic Graph (DAG) defines the data ingestion workflow.
- **Tasks**:
    - The primary task is running the data fetching service script.
    - Scheduled to run at regular intervals (e.g., hourly).

Data Flow

1. **Triggering the Pipeline**:
    - Airflow schedules and triggers the data fetching service.
2. **Data Fetching**:
    - The data fetching service calls the OpenWeatherMap API.
    - Retrieves weather data for predefined cities.
3. **Data Processing**:
    - Transforms the raw API data into a structured format.
    - Includes data cleaning and structuring steps.
4. **Data Insertion**:
    - Inserts the processed data into the PostgreSQL database.
    - Ensures data integrity and handles potential duplicates or inconsistencies.

Deployment Process

1. **Clone the Repository**:
    - Clone the project repository from GitHub.
2. **Environment Configuration**:
    - Set up environment variables, including the OpenWeatherMap API key and database credentials.
3. **Build Docker Images**:
    - Build the Docker images for each service (database, data fetching, Airflow).
4. **Run Docker Compose**:
    - Use Docker Compose to start and orchestrate the services.
    - Ensure all containers are running and properly connected.
5. **Access and Monitor Airflow**:
    - Access the Airflow web interface to monitor and manage the pipeline.
    - Verify that the DAG runs successfully and data is ingested into the database.

Error Handling

1. **API Request Failures**:
    - Implement retries and exponential backoff for API requests.
    - Log errors and notify through Airflow alerts.
2. **Data Inconsistencies**:
    - Validate data before insertion.
    - Handle and log any data transformation errors.
3. **Database Connection Issues**:
    - Ensure robust database connection management.
    - Implement retries and connection pooling.

Scalability and Adaptability

1. **Adding New Cities**:
    - Update the configuration to include new cities.
    - Ensure the data fetching service dynamically adapts to the new list of cities.
2. **Handling Increased Data Volume**:
    - Optimize database schema and indexing.
    - Implement batch processing and efficient data insertion techniques.
3. **Integrating Additional Data Sources**:
    - Extend the data fetching service to handle new APIs.
    - Update the database schema to accommodate additional data fields.

Security Considerations

1. **API Key Management**:
    - Store API keys and sensitive information in environment variables.
    - Avoid hardcoding credentials in the codebase.
2. **Database Security**:
    - Use strong passwords and secure connections.
    - Regularly update and patch the PostgreSQL database.
3. **Network Security**:
    - Ensure that communication between containers is secure.
    - Use Docker network policies to restrict access.

Future Enhancements

1. **Advanced Data Processing**:
    - Implement more complex data transformation and aggregation.
    - Use machine learning to derive insights from the weather data.
2. **Enhanced Monitoring and Alerting**:
    - Integrate monitoring tools like Prometheus and Grafana.
    - Set up more comprehensive alerting for pipeline failures.

3. **User Interface**:
    - Develop a simple web interface to query and visualize the weather data.
    - Use frameworks like Flask or Django for the frontend.