Real-Time Data Pipeline & Visualization

📊 A real-time data pipeline with visualization using Prefect, LakeFS, and Streamlit – all containerized with Docker Compose

This project was developed as part of the DSI321: Big Data Infrastructure course, focusing on building automated data pipelines for real-time analytics. It uses Prefect.io to manage task orchestration, and the entire system runs in Docker containers. Data insights are visualized using Python-based tools, making it easy to monitor and analyze the incoming data stream.

🔍 Overview

This project is a hands-on implementation of a real-time data pipeline. It fetches data from an Air4Thai API on an hourly basis, stores it in a versioned data lake , and visualizes insights through an interactive web dashboard.

The workflow uses **Python** as the main programming language, leverages **Prefect** for scheduling, **LakeFS** for data versioning, and **Streamlit** for front-end visualization. Everything runs seamlessly via **Docker Compose**.

🔁 Workflow

1. **Scheduled API Fetching with Prefect**
   Data is automatically fetched from an external API every hour using Prefect, which handles task scheduling and flow orchestration.

2. **Data Storage with LakeFS**
   The fetched data is saved in Parquet format and versioned using LakeFS, enabling reproducibility and easy tracking of changes over time.

3. **Mounting Data Locally**
   LakeFS is mounted locally, allowing the system to access versioned data directly through the file system for downstream processing.

4. **Visualization with Streamlit**
   A Python-based Streamlit app reads the Parquet data and displays interactive charts and insights with Basic metrics such as mean, count, and distribution are calculated using descriptive statisticsand A Large Language Model (LLM) is then used to generate natural language summaries of key insights.

5. **Containerized with Docker Compose**
   All services—including Prefect, LakeFS, and Streamlit—are containerized using Docker Compose for easy deployment and environment consistency.

▶️ How to Use

Follow these steps to set up and run the full data pipeline and dashboard:

```
git clone https://github.com/chokoon123/dsi321_2025.git
cd Real-time-Data-Processing-Pipeline
```

Visit: http://localhost:8888 Before executing deploy.py, you must first create a repository named air-quality-data in LakeFS.

Deploy Prefect Flow: Deploy the pipeline with a start of the hour (minute 40).:

```
python deploy.py
```

This sets up a deployment named data-pipeline in the default-agent-pool work pool, scheduled to run at the 40th minute of every hour (cron="40 * * * *").

Data Schema

The data schema is defined in SCHEMA.md. For this air quality data example:

```
{
  "columns": [
    "timestamp", "stationID", "nameTH", "nameEN", "areaTH",
    "areaEN", "stationType", "lat", "long", "PM25.color_id",
    "PM25.aqi", "year", "month", "day", "hour"
  ],
  "types": [
    "datetime64[ns]", "string", "string", "string", "string",
    "string", "string", "float64", "float64", "int64",
    "float64", "int64", "int64", "int32", "int32"
    ],
  "key_columns": [
    "timestamp", "stationID", "nameTH", "nameEN", "areaTH",
    "areaEN", "stationType", "lat", "long", "PM25.color_id",
    "year", "month", "day", "hour"
  ]
}
```

- timestamp: ISO format timestamp of data collection.
- stationID: Station ID code.
- nameTH: Station name in Thai.
- nameEN: Station name in English.
- areaTH: Area name in Thai.
- areaEN: Area name in English.
- stationType: Type of the station (e.g., roadside, general area).
- lat: Latitude of the station.
- long: Longitude of the station.
- PM25.color_id: Color ID for visualization based on PM2.5 level.
- PM25.aqi: PM2.5 Air Quality Index (AQI).
- year: Year of data record.
- month: Month of data record.

- day: Day of data record.
- hour: Hour of data record.

Key columns are used for data quality checks (no missing values and Reading with no object data type). Adapt the schema for your data source as needed.