Weather to Warehouse — Data Engineering Home Assignment

Overview

You are asked to design and implement an end-to-end **Data Engineering pipeline** that ingests raw weather data from a public API, processes it through **Bronze / Silver / Gold** layers, and loads analytics-ready data into a **data warehouse**.

This assignment is designed to evaluate your ability to think and work like a Data Engineer: data modeling, pipeline structure, reliability, clarity, and trade-off reasoning — not just writing code.

Business Context

A fictional company wants to analyze **weather trends by city over time** to support downstream analytics and reporting. The data must be reliable, reproducible, and easy to query.

The system should: - Collect weather data regularly - Preserve raw data for traceability - Produce clean, analytics-ready tables - Be easy to extend in the future

---

Data Source

Use the **OpenWeather API** (current weather or forecast endpoints).

Minimum requirements: - City name - Timestamp (UTC) - Temperature - Feels-like temperature - Humidity - Weather condition (e.g., clear, rain, clouds)

You may enrich the dataset with additional fields if you wish.

Architecture Requirements

Your pipeline must include **three explicit layers**:

1. Bronze Layer (Raw)

Purpose: preserve source data exactly as received.

Requirements: - Store raw API responses with **no transformations** - Include ingestion metadata (ingestion timestamp, source) - Data must be append-only - Schema evolution should not break ingestion

Accepted formats: - JSON or Parquet

---

2. Silver Layer (Cleaned & Structured)

Purpose: clean, normalize, and structure the data.

Requirements: - Parse and flatten raw JSON - Enforce data types - Handle missing or malformed values - Standardize timestamps (UTC) - Deduplicate records if applicable - Clearly define the schema

This layer should be **queryable and consistent**.

---

3. Gold Layer (Analytics-Ready)

Purpose: provide data optimized for analytics and reporting.

Requirements: - Aggregated or modeled tables (examples below) - Clear grain definition (e.g., city-day) - Business-friendly column names

Example Gold tables (choose at least one): - Daily weather summary per city - Hourly temperature trends - Weather condition frequency by city

---

Data Warehouse

Load the **Gold layer** into a data warehouse.

Accepted options: - PostgreSQL / DuckDB / BigQuery / Snowflake (or similar)

Requirements: - Use proper table schemas - Data should be easily queryable with SQL - Demonstrate at least 3 analytical queries

---

Pipeline Execution

You must demonstrate how the pipeline is run.

Minimum expectations: - A single entry point (script or command) - Clear separation between ingestion, transformation, and loading - Idempotent or safely repeatable runs

Bonus (optional): - Scheduling logic (cron / Airflow / Prefect) - Incremental loads

---

Code & Tooling

You may use any reasonable DE tooling, including: - Python - Pandas / PySpark - SQL - dbt (optional) - GitHub Copilot (allowed)

Expectations: - Code must be readable and structured - Functions/modules should have clear responsibility - Configuration should be separated from logic

---

Project Structure (Example)

```
weather-to-warehouse/
│
├── ingestion/
├── bronze/
├── silver/
├── gold/
├── warehouse/
├── config/
├── scripts/
├── README.md
```

You may adapt this structure if justified.

Documentation (Required)

Your `README.md` must include:

1. Project overview
2. Architecture diagram (can be ASCII or image)
3. Data flow explanation
4. Schema definitions for Bronze, Silver, Gold
5. How to run the pipeline
6. Example SQL queries
7. Design decisions and trade-offs

Clarity matters.

Evaluation Criteria

You will be evaluated on:

- Correct use of Bronze / Silver / Gold concepts
- Data modeling quality
- Pipeline clarity and robustness
- Code organization and readability
- Ability to explain decisions
- Real-world practicality

Not evaluated: - UI - Fancy dashboards - Over-engineering

Submission

Submit: - GitHub repository link - Clear instructions to run the project

The project should be **complete, reproducible, and understandable** by another engineer.

Time Expectation

This assignment is expected to take **8–12 hours** for a strong DE candidate.

Completeness and clarity are more important than feature count.

Good luck — treat this as a real production pipeline, not a toy exercise.