Step-by-Step with Free Resources

1. Data Extraction (Python + OpenWeatherMap)

* OpenWeatherMap: Free tier allows 60 API calls/minute, 1M/month. More than enough for demo.
* Python: Use requests for API calls (open-source).

💡 No cost here.

2. Data Loading & Transformation (PostgreSQL + Python)

* PostgreSQL:
  + Install locally (free, open source).
  + Or use ElephantSQL free tier (20MB) / Supabase free tier (500MB DB).
* Python: Use psycopg2 or SQLAlchemy for loading/transformation.

💡 For demo → local PostgreSQL is enough.

3. Data Quality Checks (SODA)

* Soda Core: Open-source, free to use locally.
* Run scans with a YAML config, no need for Soda Cloud (paid).

💡 100% free with Soda Core.

4. Functional & Unit Testing (pytest + unittest + Airflow)

* pytest/unittest: Both free & open source.
* Airflow: Open-source, can be installed locally with pip install apache-airflow.
  + For POC, run locally in SequentialExecutor (no need for paid Composer/Managed Airflow).

💡 Entire testing stack is open-source.

5. CI/CD (GitHub Actions)

* GitHub Actions: Free for public repos; for private repos, 2000 free minutes/month.
* Enough to demo pipelines with tests + SODA scans.

💡 Free tier is sufficient.

🔹 Optional (Visualization)

If you want to show results visually:

* Metabase (free, open-source BI tool).
* Superset (free, open-source).
* Or even Power BI Desktop (free, though Service is paid).

✅ Answer: Yes, you can build your DataOps Quality Framework POC with zero cost, using free/open-source + free cloud tiers.

dataops-quality-framework/

│── extraction/ # Python scripts for API calls

│ └── fetch\_weather.py

│

│── transform/ # SQL / Python transformations

│ └── transform\_weather.py

│

│── dq\_checks/ # Data Quality checks (Soda configs, rules)

│ └── weather\_checks.yml

│

│── tests/ # Unit + functional tests

│ ├── test\_extraction.py

│ └── test\_transformation.py

│

│── dags/ # Airflow DAGs

│ └── weather\_pipeline\_dag.py

│

│── .github/

│ └── workflows/ # GitHub Actions CI/CD pipelines

│ └── ci.yml

│

│── requirements.txt # Python dependencies

│── README.md # Documentation

# Git Commands to Create This Structure

# 1. Create a new project folder

mkdir dataops-quality-framework

cd dataops-quality-framework

# 2. Initialize git repo

git init

# 3. Create subfolders

$folders = "extraction","transform","dq\_checks","tests","dags",".github",".github\workflows"

foreach ($f in $folders) { mkdir $f }

# 4. Create placeholder files

New-Item extraction\fetch\_weather.py -ItemType File

New-Item transform\transform\_weather.py -ItemType File

New-Item dq\_checks\weather\_checks.yml -ItemType File

New-Item tests\test\_extraction.py -ItemType File

New-Item tests\test\_transformation.py -ItemType File

New-Item dags\weather\_pipeline\_dag.py -ItemType File

New-Item .github\workflows\ci.yml -ItemType File

New-Item requirements.txt -ItemType File

New-Item README.md -ItemType File

5.

mkdir C:\Users\BJeyshree\WeatherDataOps

cd C:\Users\BJeyshree\WeatherDataOps

# 6. Move your project folders inside it

move C:\Users\BJeyshree\extraction C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\transform C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\dq\_checks C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\tests C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\dags C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\.github C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\requirements.txt C:\Users\BJeyshree\WeatherDataOps\

move C:\Users\BJeyshree\README.md C:\Users\BJeyshree\WeatherDataOps\

#7. Add a .gitignore file

# Ignore Windows system files

\*.log

\*.tmp

Thumbs.db

Desktop.ini

# Ignore Python cache

\_\_pycache\_\_/

\*.pyc

# Ignore virtual environments

venv/

.env/

# Ignore VS Code settings

.vscode/

# Ignore sensitive files

\*.key

\*.env

# 8. Stage & commit

git add .

Step 1: Data Extraction – OpenWeatherMap + Python

1. Get an API Key

* Sign up at OpenWeatherMap.
* You’ll get a free API key (valid for current weather, forecast, air pollution, etc.).

2. API Endpoint Example

For current weather:

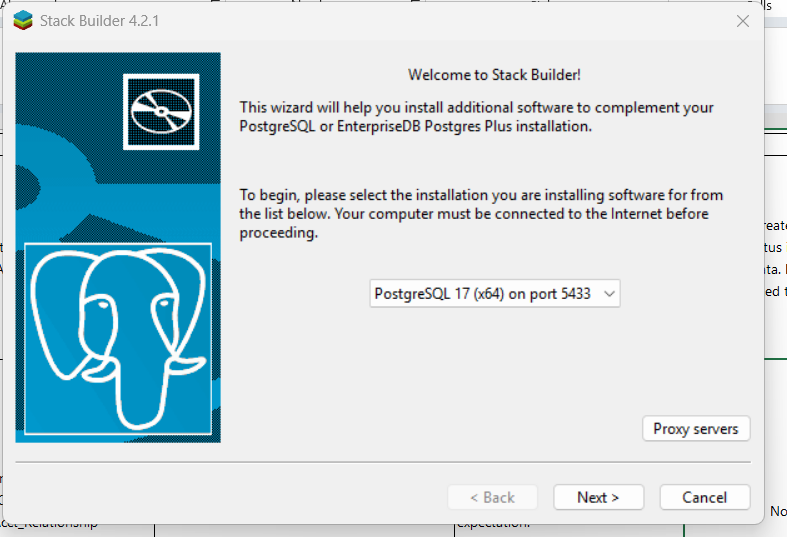
http://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR\_API\_KEY&units=metric

* q: City name (e.g., London, Chennai, New York).
* appid: Your API key.
* units=metric: Converts temperature to Celsius.

3. Python Script

Data loading and transformation:

1. Set up POSTgre sql - https://www.youtube.com/watch?v=4qH-7w5LZsA



🐘 PostgreSQL (Self-managed)

* What it is: Open-source relational database. You install and manage it on your machine, VM, or server.
* Pros:
  + Full control (extensions, tuning, backups, networking).
  + No vendor lock-in.
  + Production-grade features out of the box.
* Cons:
  + You are responsible for setup, scaling, upgrades, and backups.
  + More ops work.

When to use:

* Enterprise/prod workloads.
* You want full control and can manage infra.
* You need to go beyond “free-tier toy projects”.

🟦 ElephantSQL (Free 20MB Tier)

* What it is: Managed PostgreSQL service. Free tier gives only 20MB storage (very tiny).
* Pros:
  + Quick start — no install, just get a URL and connect.
  + Good for POCs, demos, or training.
* Cons:
  + 20MB is almost nothing (a few thousand rows).
  + Limited performance and features.
  + Not suitable for production.

When to use:

* Learning SQL/Postgres basics.
* Quick POC / sandbox for data quality tools (like Soda Core, Great Expectations).
* Teaching/demo environments.

🚀 Supabase (Free Tier)

* What it is: A Postgres + Backend-as-a-Service. Free tier includes:
  + PostgreSQL DB (500MB storage)
  + APIs (auto-generated REST & GraphQL on top of tables)
  + Authentication, row-level security, file storage.
* Pros:
  + Bigger free limit than ElephantSQL (500MB vs 20MB).
  + Built-in API & auth → makes it great for building apps quickly.
  + Good docs & active community.
* Cons:
  + Free tier has limits (50K monthly API requests, 500MB storage).
  + More “app dev focused” than pure data engineering.

When to use:

* Building full-stack apps quickly.
* You need Postgres + API + Auth without coding backend from scratch.
* Side projects, hackathons, MVPs.

For your data quality testing (Soda Core / pipelines):

* Use ElephantSQL if you just need a toy DB for testing logic.
* Use Postgres self-managed if you want a real pipeline simulation.
* Use Supabase only if you want to expose that data via API or link it with a frontend app.

# Create a table for weather data

CREATE DATABASE weatherdb;

\c weatherdb;

CREATE TABLE weather (

id SERIAL PRIMARY KEY,

city VARCHAR(100),

country VARCHAR(10),

latitude FLOAT,

longitude FLOAT,

temperature FLOAT,

humidity INT,

weather\_desc VARCHAR(255),

wind\_speed FLOAT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

# Install dependencies

In your project folder (WeatherDataOps), add these to requirements.txt:

psycopg2-binary

sqlalchemy

pandas

pip install -r requirements.txt

# Python script to load data