Simple Weather ETL (Step-by-Step, with Code)

A copy-pasteable, beginner-friendly guide to build a **non-automated** ETL pipeline that pulls **current weather** from OpenWeather, stores raw JSON locally, transforms it to a tidy table, and loads it into **MySQL** (plus an optional CSV fallback). You can run the pipeline manually from your terminal—no schedulers.

What you'll build

- **Extract**: Call OpenWeather's Current Weather endpoint for a list of cities.
- **Transform**: Normalize JSON → a clean tabular schema.
- **Load**: Insert rows into a MySQL table (or write to CSV if you don't have MySQL yet).
- **Artifacts**: Raw JSON in `data/raw/…`, processed CSV/Parquet in `data/processed/…`.

---

0) Prerequisites (10–20 min)

- **Python 3.9+**
- **MySQL Server** (8.x recommended) + **MySQL Workbench** (optional)
- An **OpenWeather API key** (free): create an account → User dashboard → *API keys*. We'll use the **Current Weather Data** endpoint.

If you can't use MySQL yet, you can still complete this guide using the **CSV fallback** and add MySQL later.

---

1) Project Structure

```
weather-etl/
├─ README.md
├─ requirements.txt
├─ .gitignore
├─ config/
│  ├─ config.sample.yaml
│  └─ config.yaml            # your real config (not committed)
├─ data/
│  ├─ raw/                   # raw JSON dumps
│  └─ processed/             # cleaned CSV/Parquet
├─ etl/
│  ├─ extract.py
│  ├─ transform.py
│  ├─ load.py
│  └─ pipeline.py
```

```
└─ sql/
   └─ create_tables.sql
```

Create these folders/files as you go. We'll fill them in.

---

2) Create & Activate a Virtual Environment

```
# macOS/Linux
python3 -m venv .venv
source .venv/bin/activate

# Windows (PowerShell)
python -m venv .venv
.venv\Scripts\Activate.ps1
```

---

3) Dependencies

`requirements.txt`

```
requests
pandas
PyYAML
SQLAlchemy
PyMySQL
python-dateutil
pyarrow                # for Parquet (optional but nice)
```

Install:

```
pip install -r requirements.txt
```

---

4) Git Hygiene

`.gitignore`

```
# venv
.venv/
```

```
# Python
__pycache__/
*.pyc

# Data & secrets (never commit!)
config/config.yaml
data/
logs/
```

---

5) Configuration

`config/config.sample.yaml` (commit this)

```
openweather:
  api_base: "https://api.openweathermap.org/data/2.5/weather"
  api_key: "REPLACE_ME"
  units: "metric"        # metric | imperial | standard

run:
  cities: ["Kathmandu,NP", "Pokhara,NP"]  # city, country_code
  save_parquet: true

mysql:
  enabled: true
  host: "localhost"
  port: 3306
  user: "weather_user"
  password: "REPLACE_ME"
  database: "weather"
  table: "weather_observations"
```

Make a copy named `config.yaml` and put your real values there. Don’t commit it.

---

6) MySQL Setup (5–10 min)

In MySQL Workbench or CLI, create a database, a user, and the table.

Create DB & user

```sql
CREATE DATABASE IF NOT EXISTS weather CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci;
CREATE USER IF NOT EXISTS 'weather_user'@'%' IDENTIFIED BY
'YOUR_STRONG_PASSWORD';
GRANT ALL PRIVILEGES ON weather.* TO 'weather_user'@'%';
FLUSH PRIVILEGES;
```

Table schema (`sql/create_tables.sql`)

```sql
CREATE TABLE IF NOT EXISTS weather_observations (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  city_id BIGINT NULL,
  city_name VARCHAR(128) NOT NULL,
  country_code VARCHAR(8) NULL,
  lat DECIMAL(9,6) NULL,
  lon DECIMAL(9,6) NULL,
  observation_time_utc DATETIME NOT NULL,
  temp_c DECIMAL(6,2) NULL,
  feels_like_c DECIMAL(6,2) NULL,
  temp_min_c DECIMAL(6,2) NULL,
  temp_max_c DECIMAL(6,2) NULL,
  pressure_hpa INT NULL,
  humidity_pct INT NULL,
  wind_speed_ms DECIMAL(6,2) NULL,
  wind_deg INT NULL,
  clouds_pct INT NULL,
  weather_main VARCHAR(64) NULL,
  weather_desc VARCHAR(256) NULL,
  rain_1h_mm DECIMAL(6,2) NULL,
  snow_1h_mm DECIMAL(6,2) NULL,
  source VARCHAR(64) NOT NULL,
  ingested_at_utc DATETIME NOT NULL,
  UNIQUE KEY uq_city_time (city_id, observation_time_utc)
) ENGINE=InnoDB;
```

Run it once:

```bash
# if you have mysql client
mysql -u weather_user -p -h 127.0.0.1 weather < sql/create_tables.sql
```

> **Note**: We keep both an `AUTO_INCREMENT id` and a unique key on `(city_id, observation_time_utc)` to prevent duplicates when you re-run the same fetch.

---

7) Extract

`etl/extract.py`

```python
from __future__ import annotations
import json
from pathlib import Path
from typing import Dict, List, Any
from datetime import datetime, timezone
import time
import requests

RAW_DIR = Path("data/raw")


def _timestamp() -> str:
    return datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%SZ")


def fetch_current_weather_city(api_base: str, api_key: str, city: str, units:
str = "metric") -> Dict[str, Any]:
    """Fetch current weather for a single city using q=city,country.
    Returns parsed JSON (dict). Raises for non-200.
    """
    params = {"q": city, "appid": api_key, "units": units}
    resp = requests.get(api_base, params=params, timeout=20)
    resp.raise_for_status()
    return resp.json()


def extract_current_weather(api_base: str, api_key: str, cities: List[str],
units: str = "metric", pause_secs: float = 1.0) -> List[Dict[str, Any]]:
    """Fetch multiple cities with a small pause to be polite to the API.
    Saves each raw JSON to data/raw/YYYYMMDD/<city>_<timestamp>.json
    Returns the list of JSON dicts in memory as well.
    """
    day_dir = RAW_DIR / datetime.now(timezone.utc).strftime("%Y%m%d")
    day_dir.mkdir(parents=True, exist_ok=True)

    results: List[Dict[str, Any]] = []
    for city in cities:
        data = fetch_current_weather_city(api_base, api_key, city, units)

        # persist raw JSON
        city_sanitized = city.replace(",", "_").replace("/", "-")
        fname = f"{city_sanitized}_{_timestamp()}.json"
        with open(day_dir / fname, "w", encoding="utf-8") as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

        results.append(data)
        time.sleep(pause_secs)

    return results
```

Keep `pause_secs` to respect rate limits on free tiers.

---

8) Transform

`etl/transform.py`

```
from __future__ import annotations
from typing import List, Dict, Any
import pandas as pd
from datetime import datetime, timezone


# Utility to safely dig into nested dicts

def get(d: Dict[str, Any], path: str, default=None):
    cur = d
    for part in path.split("."):
        if not isinstance(cur, dict) or part not in cur:
            return default
        cur = cur[part]
    return cur


def normalize_current_weather(payloads: List[Dict[str, Any]]) -> pd.DataFrame:
    """Normalize OpenWeather current weather JSON payloads to a tidy DataFrame.
    Assumes units=metric unless noted otherwise.
    """
    rows = []
    now_utc = datetime.now(timezone.utc).replace(microsecond=0)

    for p in payloads:
        rows.append({
            "city_id": get(p, "id"),
            "city_name": get(p, "name"),
            "country_code": get(p, "sys.country"),
            "lat": get(p, "coord.lat"),
            "lon": get(p, "coord.lon"),
            # dt is Unix UTC timestamp of the observation
            "observation_time_utc": datetime.utcfromtimestamp(get(p, "dt",
0)).replace(microsecond=0),
            "temp_c": get(p, "main.temp"),
            "feels_like_c": get(p, "main.feels_like"),
            "temp_min_c": get(p, "main.temp_min"),
            "temp_max_c": get(p, "main.temp_max"),
            "pressure_hpa": get(p, "main.pressure"),
            "humidity_pct": get(p, "main.humidity"),
            "wind_speed_ms": get(p, "wind.speed"),
            "wind_deg": get(p, "wind.deg"),
            "clouds_pct": get(p, "clouds.all"),
            "weather_main": (get(p, "weather") or [{}])[0].get("main"),
            "weather_desc": (get(p, "weather") or [{}])[0].get("description"),
            "rain_1h_mm": get(p, "rain.1h", 0.0),
            "snow_1h_mm": get(p, "snow.1h", 0.0),
            "source": "openweather_current",
            "ingested_at_utc": now_utc,
        })

    df = pd.DataFrame(rows)

    # Enforce dtypes where reasonable
    dtype_map = {
        "city_id": "Int64",
        "city_name": "string",
        "country_code": "string",
        "lat": "float64",
        "lon": "float64",
        "temp_c": "float64",
        "feels_like_c": "float64",
        "temp_min_c": "float64",
        "temp_max_c": "float64",
        "pressure_hpa": "Int64",
        "humidity_pct": "Int64",
        "wind_speed_ms": "float64",
        "wind_deg": "Int64",
        "clouds_pct": "Int64",
        "weather_main": "string",
        "weather_desc": "string",
        "rain_1h_mm": "float64",
        "snow_1h_mm": "float64",
        "source": "string",
    }

    for col, dt in dtype_map.items():
        if col in df.columns:
            df[col] = df[col].astype(dt)
    # Datetime columns
    for col in ["observation_time_utc", "ingested_at_utc"]:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], utc=True)

    return df
```

---

9) Load

`etl/load.py`

```
from __future__ import annotations
from typing import Optional
from pathlib import Path
import pandas as pd
from sqlalchemy import create_engine, text

PROCESSED_DIR = Path("data/processed")
PROCESSED_DIR.mkdir(parents=True, exist_ok=True)


def save_as_csv(df: pd.DataFrame, basename: str) -> str:
    csv_path = PROCESSED_DIR / f"{basename}.csv"
    df.to_csv(csv_path, index=False)
    return str(csv_path)


def save_as_parquet(df: pd.DataFrame, basename: str) -> str:
    pq_path = PROCESSED_DIR / f"{basename}.parquet"
    df.to_parquet(pq_path, index=False)
    return str(pq_path)


def make_mysql_engine(user: str, password: str, host: str, port: int, database:
str):
    url = f"mysql+pymysql://{user}:{password}@{host}:{port}/{database}?
charset=utf8mb4"
    return create_engine(url, pool_pre_ping=True)


def ensure_table(engine, table_name: str):
    """Run a minimal DDL to ensure the target table exists.
    This is a safety net in case you didn’t run sql/create_tables.sql.
    """
    ddl = f"""
    CREATE TABLE IF NOT EXISTS {table_name} (
      id BIGINT PRIMARY KEY AUTO_INCREMENT,
      city_id BIGINT NULL,
      city_name VARCHAR(128) NOT NULL,
      country_code VARCHAR(8) NULL,
      lat DECIMAL(9,6) NULL,
      lon DECIMAL(9,6) NULL,
      observation_time_utc DATETIME NOT NULL,
      temp_c DECIMAL(6,2) NULL,
      feels_like_c DECIMAL(6,2) NULL,
      temp_min_c DECIMAL(6,2) NULL,
      temp_max_c DECIMAL(6,2) NULL,
      pressure_hpa INT NULL,
      humidity_pct INT NULL,
      wind_speed_ms DECIMAL(6,2) NULL,
      wind_deg INT NULL,
      clouds_pct INT NULL,
      weather_main VARCHAR(64) NULL,
      weather_desc VARCHAR(256) NULL,
      rain_1h_mm DECIMAL(6,2) NULL,
      snow_1h_mm DECIMAL(6,2) NULL,
      source VARCHAR(64) NOT NULL,
      ingested_at_utc DATETIME NOT NULL,
      UNIQUE KEY uq_city_time (city_id, observation_time_utc)
    ) ENGINE=InnoDB;
    """
    with engine.begin() as conn:
        conn.execute(text(ddl))


def append_to_mysql(df: pd.DataFrame, engine, table_name: str):
    # Pandas will map dtypes reasonably; for production, define explicit
SQLAlchemy types
    df.to_sql(table_name, con=engine, if_exists='append', index=False,
method='multi', chunksize=1000)
```

10) Orchestrate the ETL (Manual Run)

`etl/pipeline.py`

```
from __future__ import annotations
import argparse
```

```python
from pathlib import Path
import yaml
from datetime import datetime, timezone

from extract import extract_current_weather
from transform import normalize_current_weather
from load import save_as_csv, save_as_parquet, make_mysql_engine, ensure_table,
append_to_mysql

CONFIG_PATH = Path(__file__).resolve().parents[1] / 'config' / 'config.yaml'


def run_pipeline():
    # 1) Load config
    with open(CONFIG_PATH, 'r', encoding='utf-8') as f:
        cfg = yaml.safe_load(f)

    api_base = cfg['openweather']['api_base']
    api_key = cfg['openweather']['api_key']
    units = cfg['openweather'].get('units', 'metric')

    cities = cfg['run']['cities']
    save_parquet_flag = bool(cfg['run'].get('save_parquet', True))

    mysql_cfg = cfg.get('mysql', {})
    use_mysql = bool(mysql_cfg.get('enabled', False))

    # 2) Extract
    payloads = extract_current_weather(api_base, api_key, cities, units)

    # 3) Transform
    df = normalize_current_weather(payloads)

    # 4) Load → CSV/Parquet (always), and optionally MySQL
    ts = datetime.now(timezone.utc).strftime('%Y%m%dT%H%M%SZ')
    base = f"weather_current_{ts}"

    csv_path = save_as_csv(df, base)
    print(f"Saved CSV → {csv_path}")

    if save_parquet_flag:
        pq_path = save_as_parquet(df, base)
        print(f"Saved Parquet → {pq_path}")

    if use_mysql:
        engine = make_mysql_engine(
            mysql_cfg['user'], mysql_cfg['password'], mysql_cfg['host'],
mysql_cfg['port'], mysql_cfg['database']
        )
        ensure_table(engine, mysql_cfg['table'])
        append_to_mysql(df, engine, mysql_cfg['table'])
        print("Appended rows to MySQL.")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run the simple Weather ETL
pipeline (manual)")
    parser.parse_args()  # placeholder for future args
    run_pipeline()
```

---

11) Run It Manually

1) Put your API key and settings in `config/config.yaml`.

2) (Optional) Ensure the MySQL table exists (Section 6), or rely on the `ensure_table()` fallback.

3) Run the pipeline:

```
# from project root
python -m etl.pipeline
```

You should see:

- Raw JSON files in `data/raw/YYYYMMDD/…`
- A processed CSV (and Parquet) in `data/processed/`.
- If MySQL is enabled, new rows in `weather.weather_observations`.

---

12) Validate the Load

MySQL sample queries

```
SELECT COUNT(*) FROM weather_observations;

SELECT city_name, observation_time_utc, temp_c, humidity_pct
FROM weather_observations
ORDER BY observation_time_utc DESC
LIMIT 10;
```

13) CSV-Only Fallback (No MySQL)

If you don't have MySQL available yet, set in `config.yaml`:

```
mysql:
  enabled: false
```

Your ETL will still generate `data/processed/*.csv` (and Parquet if enabled). You can load the CSV into Excel, pandas, or a BI tool.

14) Common Pitfalls & Fixes

- **401 Unauthorized**: API key missing or wrong; confirm it's enabled for the Current Weather endpoint.
- **429 Too Many Requests**: Add a bigger `pause_secs` in `extract_current_weather()`; reduce number of cities.
- **City not found**: Ensure format `City,CountryCode` (e.g., `Kathmandu,NP`).
- **Unicode errors on Windows**: Always open files with `encoding="utf-8"` (already in code).
- **Duplicate rows**: The unique key `(city_id, observation_time_utc)` prevents exact dupes; if you still get errors, deduplicate in pandas before load.

15) Optional Enhancements (try later)

- **Logging**: Add `logging` to write to `logs/pipeline.log` instead of `print`.
- **Config via `.env`**: Keep secrets in environment variables; read them in code.
- **Validation**: Use `pydantic` models to validate payloads.
- **Tests**: Add a `tests/` folder and write unit tests for transform logic.
- **PySpark version**: Swap pandas with PySpark DataFrame API once your dataset grows.
- **Scheduling**: Later add Airflow or cron for automation (outside this guide).

16) Learning & Reference Links

- OpenWeather product: *Current Weather Data* (check your account dashboard for exact URL and parameters)
- pandas docs: DataFrame, `to_csv`, `to_parquet`, `to_sql`
- SQLAlchemy docs: create engine strings for MySQL (dialect `mysql+pymysql`)
- MySQL docs: data types, indexes, and `UNIQUE KEY`

(These are generic docs you can easily search; this guide stays tool-agnostic and copy-pasteable.)

17) What To Submit (if this is for an internship task)

- A link to your GitHub repo with:
- The project scaffold and code above
- A **README** explaining: what it does, how to run it, and sample outputs
- A screenshot of your MySQL table with a few rows (or the generated CSV)
- Short notes on: what went well, what you’d improve (e.g., retries, logging, tests)

✅You now have a clean, non-automated ETL you can run by hand.

When ready, we can iterate to a PySpark version and/or add Airflow for scheduling.