

Mini Projet :

Créer une pipeline de projet Data avec Apache Airflow



Réalisée par :

- Hind El Ouahabi
- Mohcine Boudenjal

Encadré par :

- Yasyn EL YUSUFI

Introduction:

L'objectif principal de ce projet est de concevoir une architecture de pipeline de données automatisée pour l'ingestion, la transformation et le chargement des données provenant de diverses sources (API, bases de données, fichiers, etc.) vers un entrepôt de données (Data Warehouse). Ce pipeline permettra de centraliser les données dans un environnement structuré et optimisé pour la prise de décision. En complément, un tableau de bord sera développé pour visualiser et analyser les données à travers des graphiques, indicateurs et rapports.

Outils et technologies:

Pour ce projet de pipeline de données, plusieurs outils et technologies sont utilisés pour construire et orchestrer les étapes de traitement de données, de leur ingestion jusqu'à leur visualisation.

Python est un langage de programmation de haut niveau, largement utilisé en science des données et pour les pipelines de traitement des données grâce à sa flexibilité, sa simplicité et la richesse de son écosystème.

Apache Airflow est un outil open-source conçu pour automatiser et orchestrer des workflows complexes et reproductibles. utilisé pour Définir et planifier des workflows (DAGs) permettant d'automatiser l'ingestion, la transformation et le chargement des données à intervalles réguliers. et pour superviser l'exécution des tâches et relancer automatiquement les tâches en cas d'échec.

MySQL est un système de gestion de bases de données relationnelles (SGBDR) qui offre robustesse et performance pour le stockage et la gestion de grands volumes de données. Dans le cadre de ce projet, MySQL est utilisé comme entrepôt de données (Data Warehouse) et aussi comme une des sources de données.

Flask est un micro-framework Python léger et flexible, idéal pour le développement de petites applications web et de tableaux de bord. Pour ce projet, Flask est utilisé pour créer une application web qui servira d'interface utilisateur pour visualiser les données stockées dans MySQL (Entrepôt de données).

Chart.js, une bibliothèque JavaScript de visualisation de données, est intégrée à Flask pour rendre les graphiques plus interactifs et esthétiques.

Data Source:

Tout d'abord, il est essentiel de choisir un thème de données pertinent pour notre projet. Après réflexion, nous avons opté pour la météo en raison de son impact global et de la richesse des analyses possibles : température, humidité, pression atmosphérique, et autres indicateurs météorologiques permettent de générer des rapports et des comparaisons utiles. Ces données peuvent servir à analyser les variations climatiques selon les villes, les continents, les saisons, etc.

Une fois le thème de données choisi, nous avons sélectionné des sources fiables pour obtenir des informations complètes et variées :

API de WeatherStack.com : Cette API nous fournit des données météorologiques en temps réel, accessibles pour plusieurs villes dans le monde. Elle est particulièrement utile pour obtenir des mesures actualisées à tout moment.

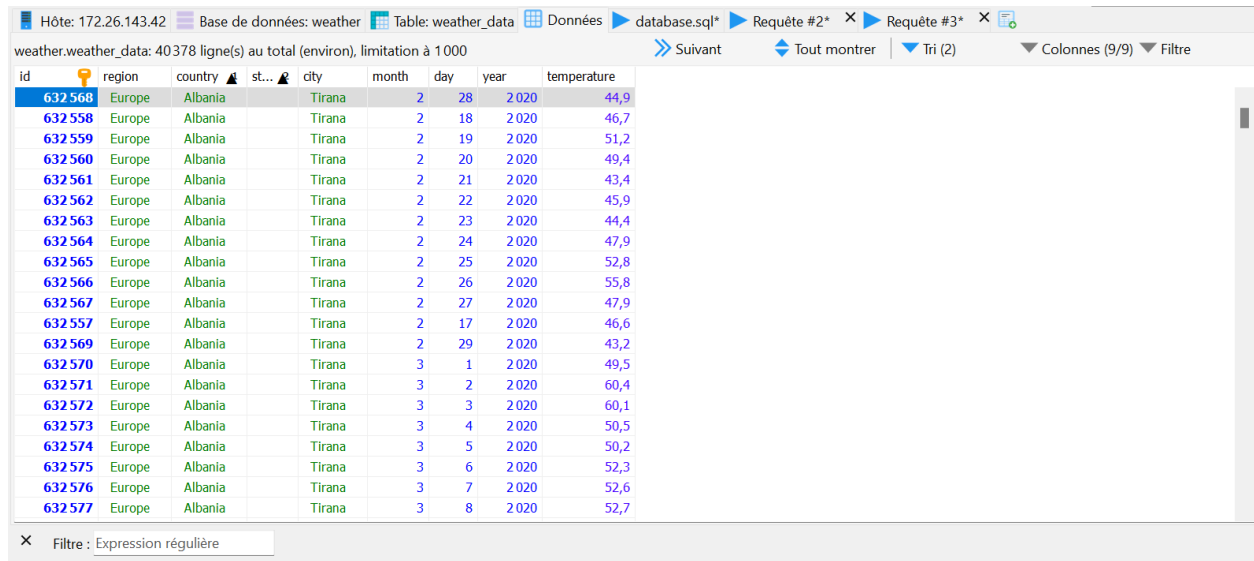
Exemple data:

```
//https://api.weatherstack.com/current?access_key=c6e808e073d4c8cfa219679c148ad912&query=New%20York
```

```
{
  "request": {
    "type": "City",
    "query": "New York, United States of America",
    "language": "en",
```

```
    "unit": "m"
  },
  "location": {
    "name": "New York",
    "country": "United States of America",
    "region": "New York",
    "lat": "40.714",
    "lon": "-74.006",
    "timezone_id": "America/New_York",
    "localtime": "2024-10-31 16:26",
    "localtime_epoch": 1730391960,
    "utc_offset": "-4.0"
  },
  "current": {
    "observation_time": "08:26 PM",
    "temperature": 23,
    "weather_code": 116,
    "weather_icons": [
"https://cdn.worldweatheronline.com/images/wsymbols01\_png\_64/wsymb01\_0002\_sunny\_intervals.png"
    ],
    "weather_descriptions": [
      "Partly cloudy"
    ],
    "wind_speed": 18,
    "wind_degree": 226,
    "wind_dir": "SW",
    "pressure": 1018,
    "precip": 0,
    "humidity": 62,
    "cloudcover": 75,
    "feelslike": 25,
    "uv_index": 6,
    "visibility": 16,
    "is_day": "yes"
  }
}
```

Base de données MySQL avec des températures historiques : Nous avons importé un jeu de données depuis Kaggle contenant les températures moyennes journalières pour plus de 300 villes à travers le monde, de 1995 jusqu'à mai 2020 (soit plus de 2 millions de lignes). Pour des raisons de performance, nous avons limité notre analyse aux données de 2020.



The screenshot shows a MySQL database interface with the following details:

- Host: 172.26.143.42
- Base de données: weather
- Table: weather_data
- Données: database.sql*
- Requête #2* (selected)
- Requête #3*

The table view displays 40,378 lines (limited to 1000). The columns are: id, region, country, st..., city, month, day, year, and temperature. The data shown is for Tirana, Albania, in February 2020.

id	region	country	st...	city	month	day	year	temperature
632568	Europe	Albania		Tirana	2	28	2020	44,9
632558	Europe	Albania		Tirana	2	18	2020	46,7
632559	Europe	Albania		Tirana	2	19	2020	51,2
632560	Europe	Albania		Tirana	2	20	2020	49,4
632561	Europe	Albania		Tirana	2	21	2020	43,4
632562	Europe	Albania		Tirana	2	22	2020	45,9
632563	Europe	Albania		Tirana	2	23	2020	44,4
632564	Europe	Albania		Tirana	2	24	2020	47,9
632565	Europe	Albania		Tirana	2	25	2020	52,8
632566	Europe	Albania		Tirana	2	26	2020	55,8
632567	Europe	Albania		Tirana	2	27	2020	47,9
632557	Europe	Albania		Tirana	2	17	2020	46,6
632569	Europe	Albania		Tirana	2	29	2020	43,2
632570	Europe	Albania		Tirana	3	1	2020	49,5
632571	Europe	Albania		Tirana	3	2	2020	60,4
632572	Europe	Albania		Tirana	3	3	2020	60,1
632573	Europe	Albania		Tirana	3	4	2020	50,5
632574	Europe	Albania		Tirana	3	5	2020	50,2
632575	Europe	Albania		Tirana	3	6	2020	52,3
632576	Europe	Albania		Tirana	3	7	2020	52,6
632577	Europe	Albania		Tirana	3	8	2020	52,7

Fichier JSON extrait d'OpenWeatherMap : Ce fichier contient des données météorologiques spécifiques (température, humidité, pression, etc.) collectées le 21 octobre 2024 pour plus de 300 villes.



```
31_10_2024_14_00_openweathermap_weather_cities.json > {} Algiers
1  {
2    "Algiers": {
3      "coord": {
4        "lon": 3.042,
5        "lat": 36.7525
6      },
7      "weather": [
8        {
9          "id": 801,
10         "main": "Clouds",
11         "description": "few clouds",
12         "icon": "02d"
13       }
14     ],
15     "base": "stations",
16     "main": {
17       "temp": 24.9,
18       "feels_like": 24.83,
19       "temp_min": 24.9,
20       "temp_max": 24.9,
21       "pressure": 1017,
22       "humidity": 53,
23       "sea_level": 1017,
24       "and_level": 1011
```

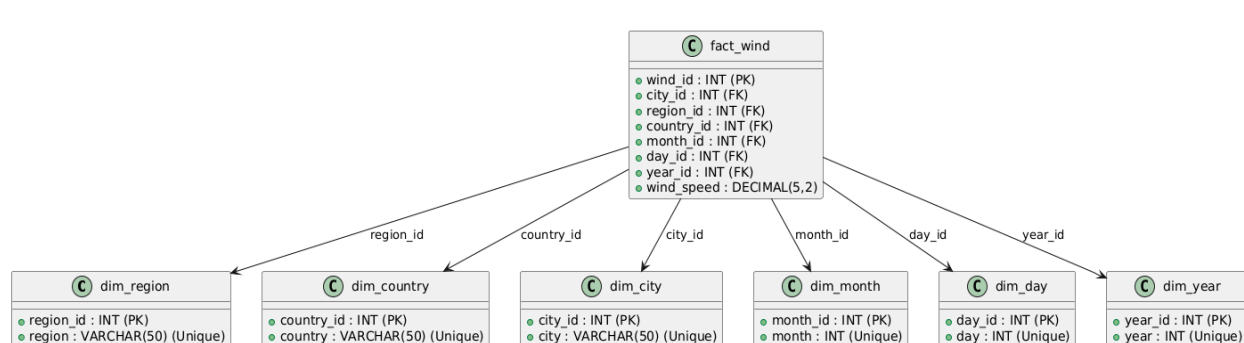
Conception:

Les données sont maintenant prêtes, et il est temps de réfléchir à la structuration du **data warehouse**. Un data warehouse bien conçu doit cibler un ou plusieurs *faits* (mesures clés) en lien avec plusieurs *dimensions* (attributs de contexte). Dans notre projet, nous avons choisi de suivre quatre faits principaux : **température**, **humidité**, **pression** et **vent**.

En ce qui concerne les dimensions, nous avons sélectionné les six dimensions les plus pertinentes pour notre analyse : **ville**, **région** (continent), **pays**, **mois**, **jour**, et **année**. Ces dimensions permettent une analyse des faits en fonction de différents contextes, tels que la comparaison par ville ou par période temporelle.

Conception du data warehouse





Base de données:

Nom	Lignes	Taille	Créé le	Mis à jour	Moteur	Commentaire	Type
dim_city	311	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
dim_country	123	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
dim_day	31	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
dim_month	7	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
dim_region	6	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
dim_year	2	32,0 KiB	2024-11-02 19:10...		InnoDB		Table
fact_humidity	639	160,0 KiB	2024-11-02 19:10...		InnoDB		Table
fact_pressure	666	160,0 KiB	2024-11-02 19:10...		InnoDB		Table
fact_temp	34 509	9,6 MiB	2024-11-02 19:10...		InnoDB		Table
fact_wind	606	160,0 KiB	2024-11-02 19:10...		InnoDB		Table

AirFlow Pipelines:

Nous avons les données nécessaires, mais pour qu'elles soient utilisables et analysables dans notre **data warehouse**, elles doivent passer par un processus de transformation ETL (Extract, Transform, Load). Ce processus consiste à extraire les données de différentes sources (API, fichiers, bases de données), les transformer pour qu'elles correspondent à la structure de notre **data warehouse**, puis les charger dans la base de données.

Le rôle d'**Apache Airflow** dans ce projet est d'orchestrer l'ensemble de ces étapes chaque jour. Il permet de planifier et d'exécuter automatiquement les tâches nécessaires pour récupérer les données de différentes sources, les transformer et les insérer dans notre **data warehouse**.

Voici un exemple de tâche:

```
GNU nano 7.2 extract_weather_data_from_json.py

def load_weather_data_from_file(file_path):
    """Load weather data from a JSON file."""
    with open(file_path, 'r') as file:
        return json.load(file)

def get_or_create_dimension_id(hook, table_name, column_name, value):
    """Check if the value exists in the dimension table and return the ID, or create it.
    try:
        sql_check = f"SELECT {column_name}_id FROM {table_name} WHERE {column_name} = %s"
        result = hook.get_first(sql_check, parameters=(value,))

        if result is None:
            # Insert into the dimension table if it does not exist
            sql_insert = f"INSERT INTO {table_name} ({column_name}) VALUES (%s)"
            hook.run(sql_insert, parameters=(value,))
            # Retrieve the newly created ID
            result = hook.get_first("SELECT LAST_INSERT_ID()")

    """
```

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location
^X Exit	^R Read File	^_ Replace	^U Paste	^J Justify	^_ Go To Line

Pour chaque source de données, on a créé son propre job, le code tout simplement Il vérifie si les identifiants des dimensions existent déjà dans la base de données, ou les crée si nécessaire. Ensuite, il insère les données dans les tables de faits correspondantes (pression, humidité, vent) en associant les identifiants des dimensions pertinentes. Ce processus permet de maintenir un entrepôt de données actualisé quotidiennement pour faciliter l'analyse des données météo.

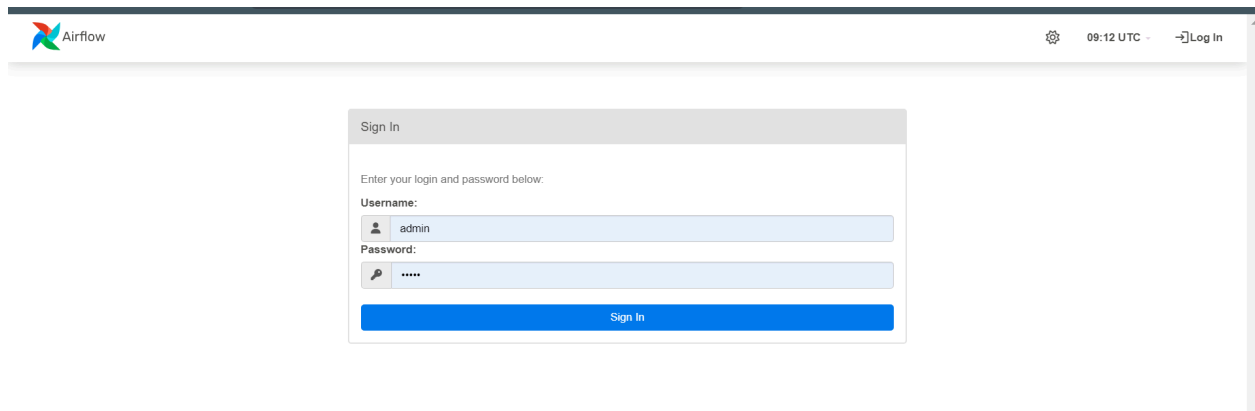
voici les autres jobs

```
__pycache__ extract_weather_data_from_json_v2.py
extract_weather_data_from_api.py weather_data_etl.py
```

Puis on va lancer le airflow avec les deux commandes:

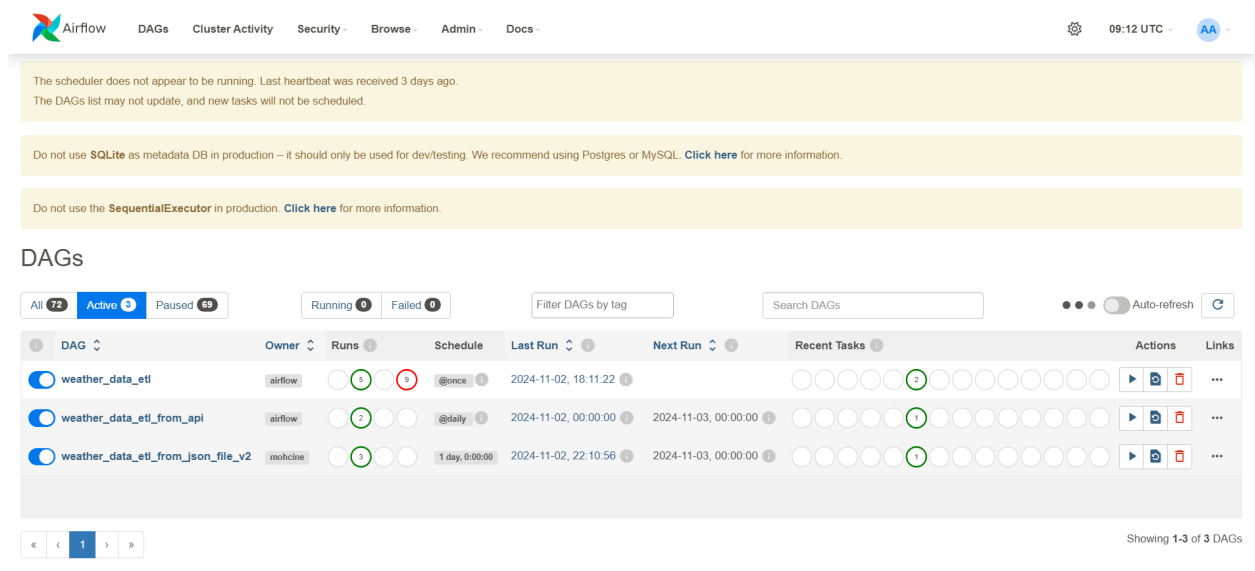
Lancer le serveur web d'Airflow: `airflow webserver --port 8080`

S'authentifier:



The image shows the Airflow web interface's login page. At the top, there's a header with the Airflow logo, a settings icon, the time '09:12 UTC', and a 'Log In' link. The main content area features a 'Sign In' box. Inside this box, there's a prompt 'Enter your login and password below:'. Below this, there are two input fields: 'Username:' with the value 'admin' and 'Password:' with masked characters '.....'. A blue 'Sign In' button is at the bottom of the box.

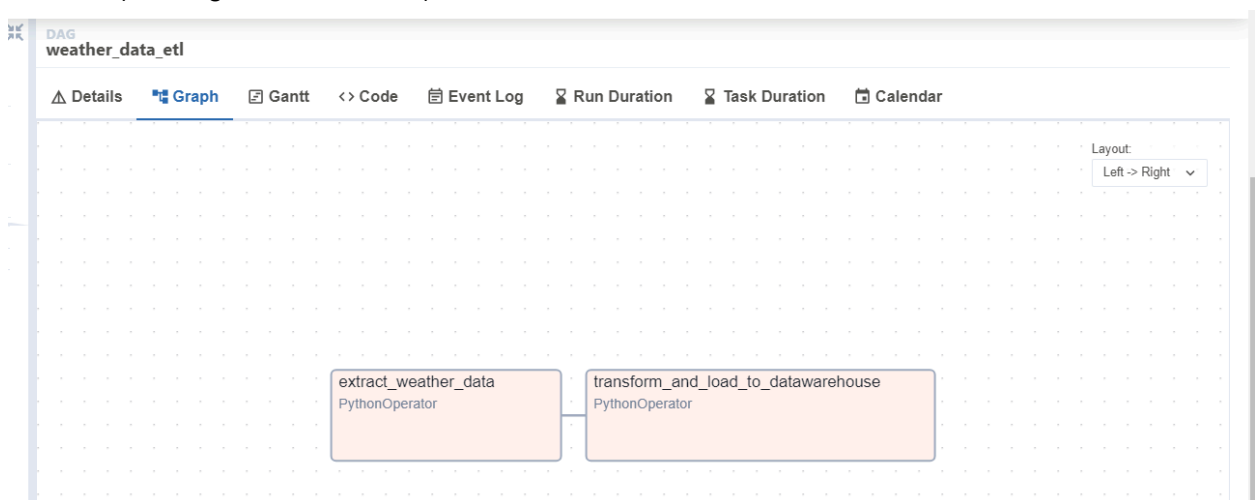
Voici les différents tâches qu'on a en mode active:



The image displays the 'DAGs' section of the Airflow web interface. At the top, there's a navigation bar with links for 'DAGs', 'Cluster Activity', 'Security', 'Browse', 'Admin', and 'Docs'. Below this, there are three yellow warning banners: 'The scheduler does not appear to be running. Last heartbeat was received 3 days ago.', 'Do not use SQLite as metadata DB in production - it should only be used for dev/testing.', and 'Do not use the SequentialExecutor in production'. The main section is titled 'DAGs' and contains a table of active DAGs. The table has columns for 'DAG', 'Owner', 'Runs', 'Schedule', 'Last Run', 'Next Run', 'Recent Tasks', 'Actions', and 'Links'. Three DAGs are listed: 'weather_data_etl', 'weather_data_etl_from_api', and 'weather_data_etl_from_json_file_v2'. Each DAG has a status indicator (a green circle with a white '1' or '2'), a schedule, and a 'Last Run' timestamp. The 'Recent Tasks' column shows a sequence of task status icons (green for success, red for failure, grey for pending). The 'Actions' column contains icons for running, refreshing, and deleting the DAG. At the bottom, there's a pagination bar showing 'Showing 1-3 of 3 DAGs'.

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
weather_data_etl	airflow	1	@once	2024-11-02, 18:11:22		1	[Run] [Refresh] [Delete]	...
weather_data_etl_from_api	airflow	2	@daily	2024-11-02, 00:00:00	2024-11-03, 00:00:00	1	[Run] [Refresh] [Delete]	...
weather_data_etl_from_json_file_v2	mohcine	1	1 day, 0:00:00	2024-11-02, 22:10:56	2024-11-03, 00:00:00	1	[Run] [Refresh] [Delete]	...

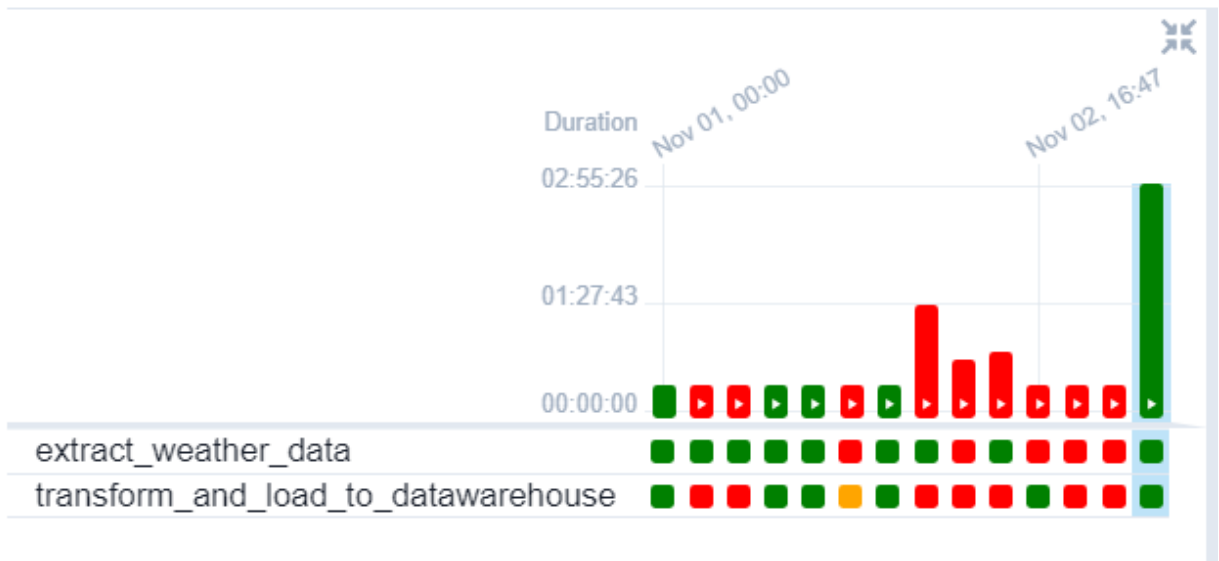
Par exemple on peut visualiser les étapes de notre pipeline sous forme de diagramme et voir son état (running, error, success)



Des informations détaillées (durée de la tâche, statuts , les dates de début et fin)

DAG weather_data_etl / Run ▶ 2024-11-02, 18:11:22 UTC		Clear ▾	Mark state as... ▾
Details Graph Gantt Code Event Log			
Dag Run Details			
Status	■ Success		
Run ID	manual__2024-11-02T18:11:22.728226+00:00 🔗		
Run type	▶ manual		
Run duration	02:55:26		
Last scheduling decision	2024-11-02, 21:06:50 UTC		
Queued at	2024-11-02, 18:11:22 UTC		
Started	2024-11-02, 18:11:24 UTC		
Ended	2024-11-02, 21:06:50 UTC		
Data interval start	2024-11-02, 18:11:22 UTC		
Data interval end	2024-11-02, 18:11:22 UTC		
Externally triggered	True		

Un autre diagramme qui visualise historique des lancements, avec les états et durée:



Après lancement des pipeline, on a fait charger data d'une manière structurée dans notre entrepôt de données sous forme des **schémas en étoile**:

Exemple de table fait de température:

datawarehouse.fact_temp: 34 509 ligne(s) au total (environ), limitation à 1 000

>> Suivant

Tout montrer

Tri

temp_id	region_id	country_id	city_id	month_id	day_id	year_id	temperature
1	1	1	1	1	1	1	46,3
2	1	1	1	1	2	1	45,4
3	1	1	1	1	3	1	48,0
4	1	1	1	1	4	1	53,4
5	1	1	1	1	5	1	47,3
6	1	1	1	1	6	1	45,9
7	1	1	1	1	7	1	47,0
8	1	1	1	1	8	1	53,2
9	1	1	1	1	9	1	52,9
10	1	1	1	1	10	1	52,4
11	1	1	1	1	11	1	49,7
12	1	1	1	1	12	1	49,8
13	1	1	1	1	13	1	50,6
14	1	1	1	1	14	1	46,7
15	1	1	1	1	15	1	45,7
16	1	1	1	1	16	1	46,6
17	1	1	1	1	17	1	48,7
18	1	1	1	1	18	1	50,1
19	1	1	1	1	19	1	53,5
20	1	1	1	1	20	1	54,3
21	1	1	1	1	21	1	59,3

X Filtrer les données

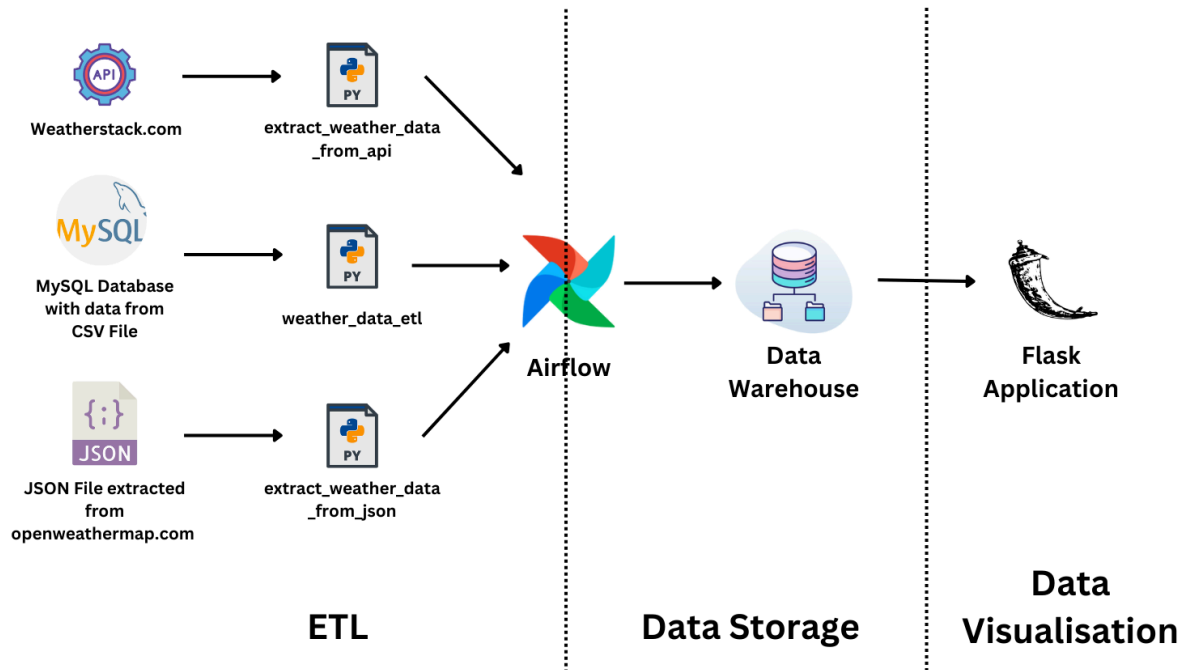
Exemple de table dimension des villes:

datawarehouse.dim_city: 320 ligne(s) au total (environ)

city_id	city
49	Manila
50	Singapore
51	Seoul
52	Colombo
53	Taipei
54	Dusanbe
55	Ashabad
56	Tashkent
57	Hanoi
58	Brisbane
59	Canberra
60	Melbourne
61	Perth
62	Sydney
63	Auckland
64	Tirana
65	Vienna
66	Minsk
67	Brussels
68	Sofia
69	Zagreb

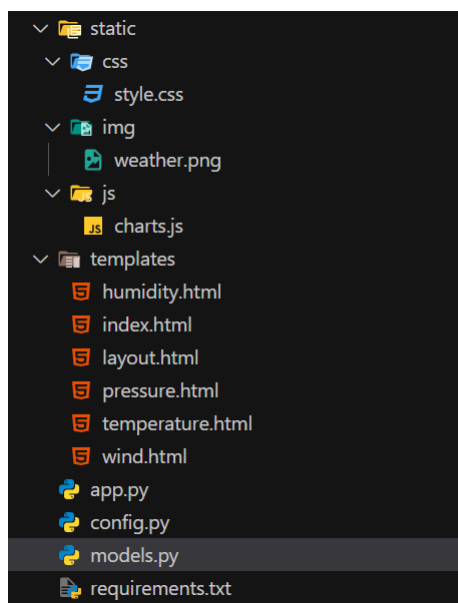
Architecture du projet:

Voici architecture générale de notre projet:



Visualisation:

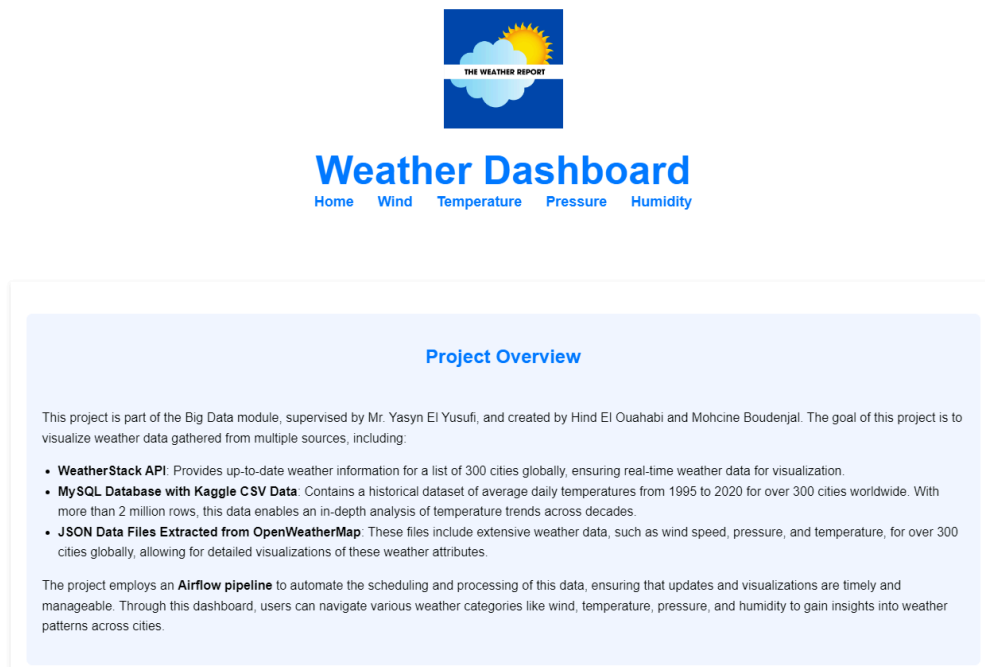
Après avoir élaboré notre entrepôt de données, la prochaine étape a consisté à créer la partie de visualisation à l'aide de Flask. Voici l'architecture du projet que nous avons mise en place :



Nous avons conçu notre modèle de données en utilisant SQLAlchemy pour établir une connexion avec notre entrepôt de données (data warehouse). Grâce à cette connexion, nous avons pu interagir efficacement avec la base de données afin de manipuler, insérer et extraire les données nécessaires pour l'analyse. Ensuite, nous avons développé des API qui exposent des vues permettant de récupérer les données pertinentes pour nos rapports et graphiques. Pour la visualisation de ces données, nous avons intégré la bibliothèque Chart.js, qui offre une grande flexibilité pour créer des graphiques interactifs et dynamiques.

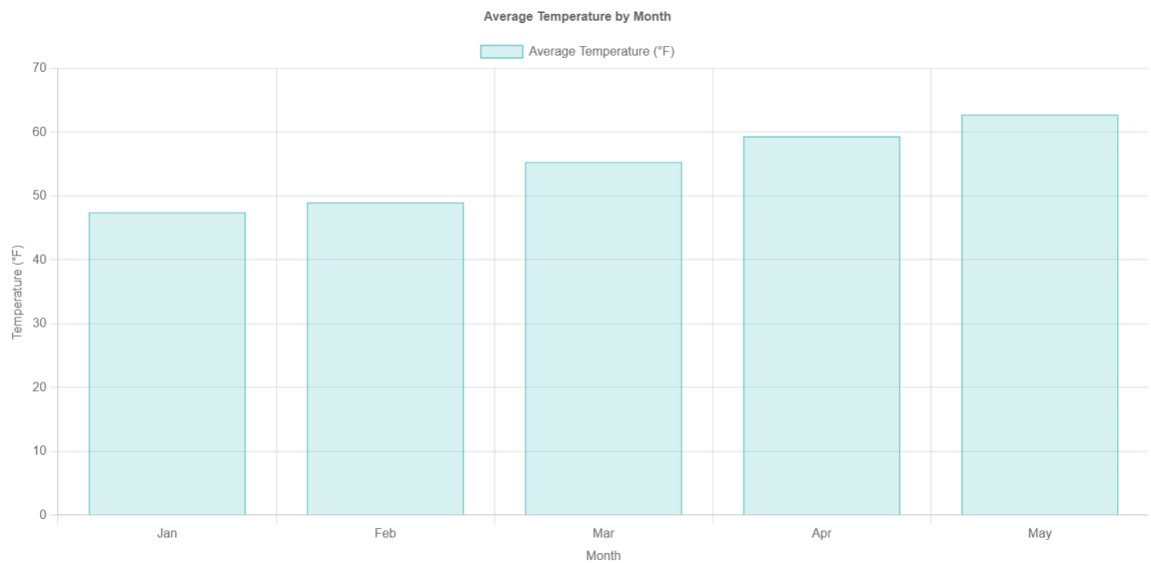
Voici les résultats obtenus :

Page d'accueil :

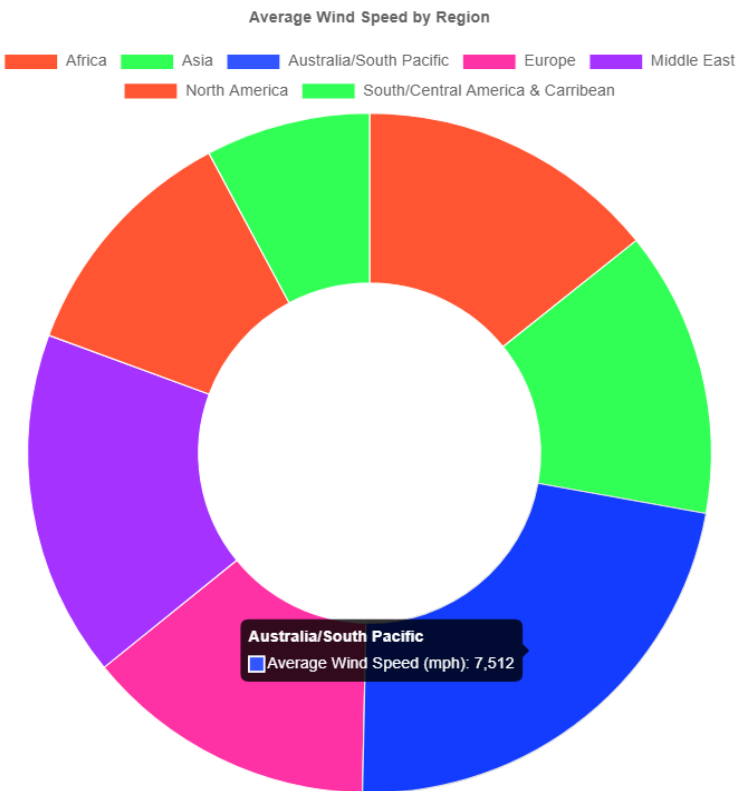


Les différents tableaux de bord pour chaque fait avec différents types :

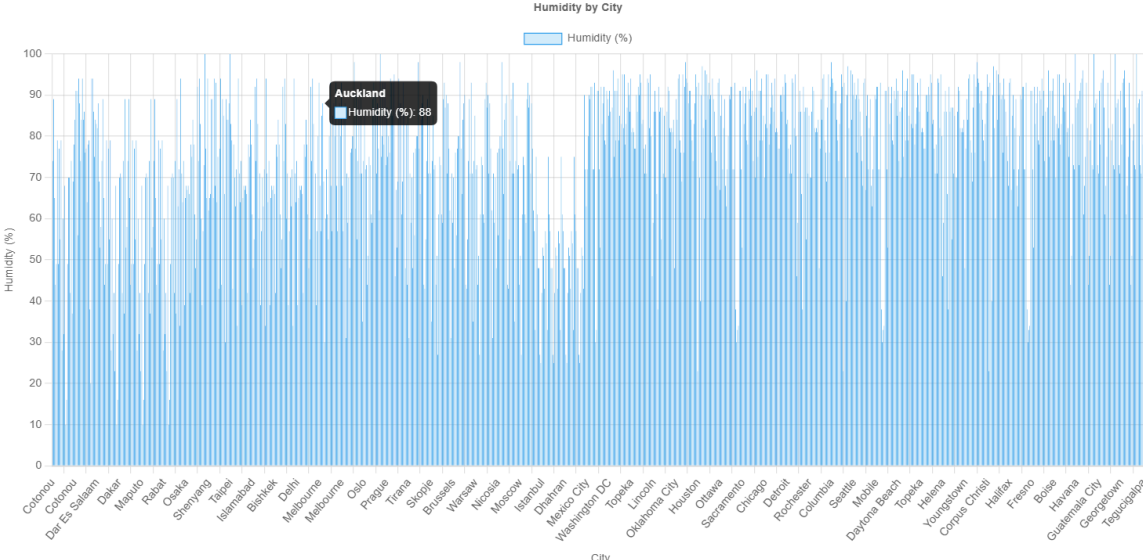
Average Temperature by Month



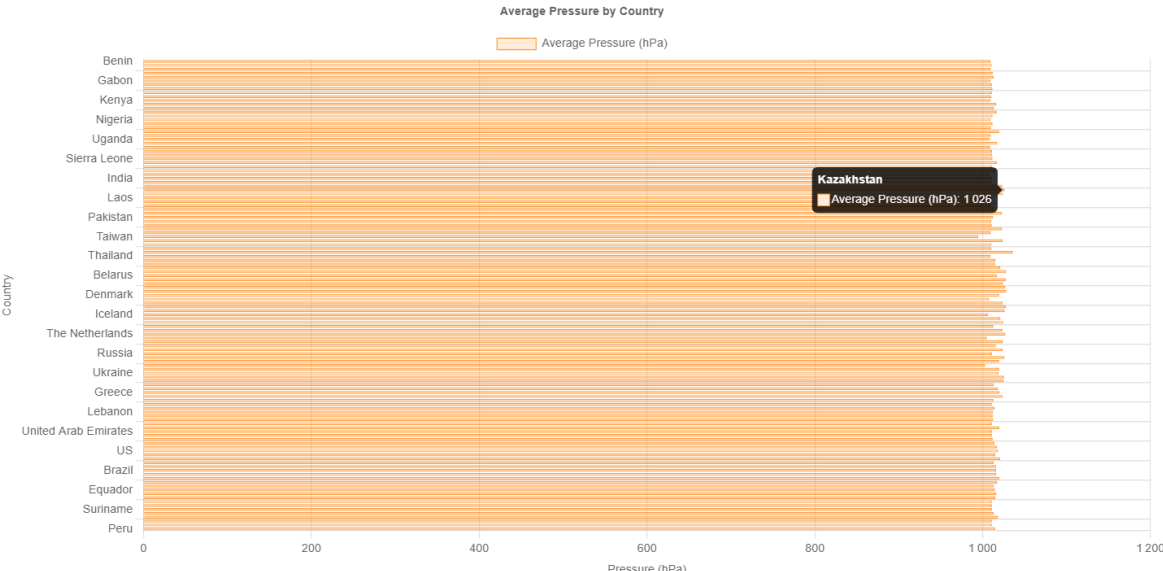
Wind Speed by Region

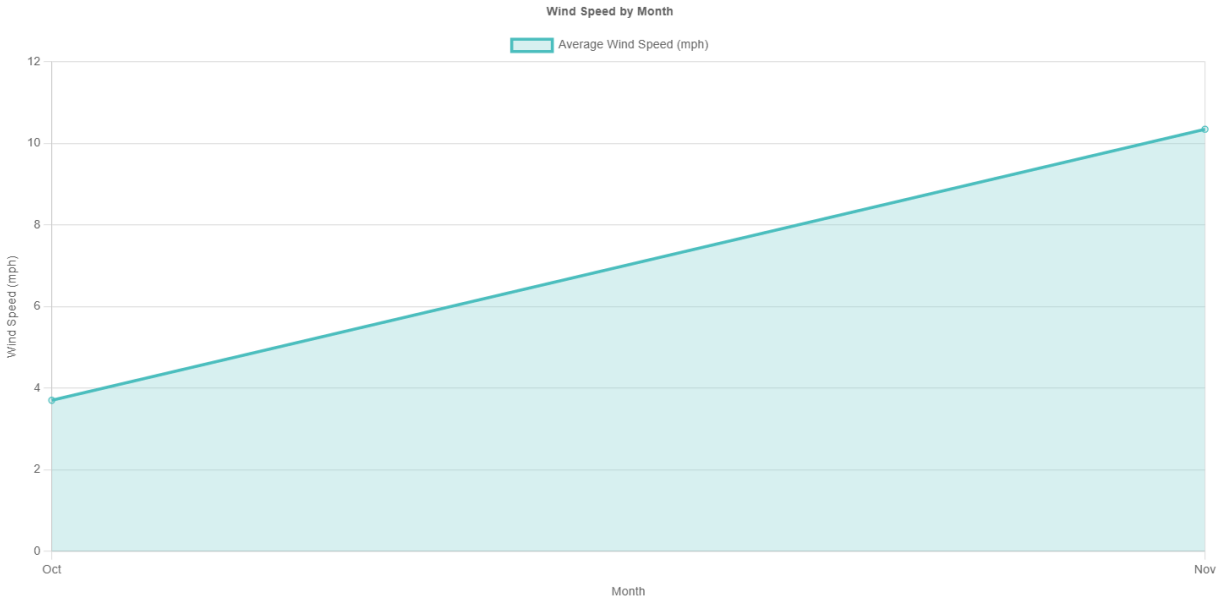


Humidity by City (Bar Chart)



Average Pressure by Country





Conclusion:

En conclusion, ce projet nous a permis de mettre en place une pipeline ETL robuste en utilisant Apache Airflow pour l'ingestion, la transformation et le chargement de données météorologiques dans un entrepôt de données. Grâce à l'utilisation de technologies telles que SQLAlchemy, MySQL, et Chart.js, nous avons pu créer une solution complète permettant la visualisation de données complexes sous forme de graphiques interactifs, facilitant ainsi l'analyse et la prise de décision. Nous tenons à remercier notre professeur, M. Yasyn El Yusufi, pour son soutien, ses précieux conseils et sa guidance tout au long de ce projet.

Bibliographie:

1. **Apache Airflow Documentation**

Apache Software Foundation. (2024). *Apache Airflow Documentation*. Retrieved from <https://airflow.apache.org/docs/>

2. **MySQL Documentation**

Oracle Corporation. (2024). *MySQL Documentation*. Retrieved from <https://dev.mysql.com/doc/>

3. **SQLAlchemy Documentation**

SQLAlchemy Project. (2024). *SQLAlchemy Documentation*. Retrieved from <https://www.sqlalchemy.org/docs/>

4. **Chart.js Documentation**

Chart.js. (2024). *Chart.js Documentation*. Retrieved from <https://www.chartjs.org/docs/latest/>