Pipeline de Collecte et Traitement de Données Météorologiques

Projet Fin de Semestre

16 janvier 2026

Table des matières

1 Introduction

Ce rapport présente l'architecture et l'implémentation d'un pipeline de traitement de données météorologiques en temps quasi-réel. Le système intègre plusieurs services cloud et outils de big data pour récupérer, transformer et stocker des données météorologiques provenant de l'API OpenWeather.

L'objectif principal est de construire une infrastructure robuste et économique capable de :

— Collecter des données météorologiques de façon fiable et régulière
— Respecter les contraintes du tier gratuit d'OpenWeather
— Traiter les données en streaming avec Kafka
— Persister les données brutes dans AWS S3 avec une politique de rétention
— Préparer les données pour des analyses ultérieures

2 Architecture Globale

2.1 Vue d'ensemble

Le pipeline suit une architecture classique ETL (Extract-Transform-Load) :

1. **Extract** : Récupération des données via l'API OpenWeather
2. **Streaming** : Envoi des données vers Kafka (topic `weather_stream`)
3. **Storage** : Persistance dans AWS S3 avec partitionnement
4. **Retention** : Gestion automatique du cycle de vie des données (30 jours)

2.2 Composants Principaux

OpenWeather API Source de données météorologiques
Kafka Message broker pour le streaming temps réel
AWS S3 Stockage objet pour l'archivage et la persistance
Lifecycle Manager Gestion automatique de la rétention

3 Objectifs du Pipeline

3.1 Objectifs Fonctionnels

Le pipeline doit :

1. Récupérer régulièrement les données météorologiques via l'API OpenWeather
2. Envoyer les événements dans le topic Kafka `weather_stream`
3. Stocker les données en format JSON dans AWS S3
4. Organiser les fichiers par structure de partitionnement (date/heure/ville)
5. Maintenir les données pendant 30 jours minimum

3.2 Contraintes Non-Fonctionnelles

1. **Quota API** : 1000 appels par jour (tier gratuit)
2. **Coûts** : Minimisation des coûts AWS (stockage et transferts)
3. **Fiabilité** : Garantie de livraison des données
4. **Disponibilité** : Service disponible 24/7
5. **Performance** : Latence acceptable pour applications temps réel

4 Gestion du Quota OpenWeather

4.1 Clarification du Modèle de Quota

Une question fondamentale s'est posée lors de la conception : *Si j'utilise tous mes 1000 appels aujourd'hui, demain mes données historiques disparaissent-elles ?*

Réponse : Non, le quota reset mais l'historique persiste.

4.1.1 Quota Reset

Le quota de 1000 appels se réinitialise quotidiennement. Une fois les 1000 appels consommés, les appels supplémentaires sont rejetés jusqu'à la réinitialisation du jour suivant.

4.1.2 Persistance des Données

— **AWS S3** : Les objets stockés persistent indéfiniment tant qu'ils ne sont pas explicitement supprimés. La lifecycle rule appliquée supprime automatiquement les objets après 30 jours.

— **Kafka** : Les messages sont retenus selon la politique de rétention configurée (typiquement 7 jours ou basé sur la taille).

4.2 Calcul et Validation du Quota

4.2.1 Paramètres Choisis

Paramètre	**Valeur**
Fréquence de collecte	600 secondes (10 minutes)
Nombre de villes	5
Appels par jour par ville	144
Total d'appels par jour	720
Quota limite	1000

4.2.2 Formule de Calcul

$$\text{Appels par jour} = \frac{86400 \text{ sec/jour}}{600 \text{ sec/appel}} \times \text{Nombre de villes}$$

$$\text{Appels par jour} = \frac{86400}{600} \times 5 = 144 \times 5 = 720 \text{ appels/jour}$$

Validation : $720 < 1000$ ✓(Marge de 280 appels)

4.3 Sélection des Villes

Les 5 villes suivantes ont été choisies pour couvrir plusieurs continents et régions climatiques :

1. **Fez, Maroc (Fez,MA)** – Climat méditerranéen/semi-aride, référence locale
2. **Athènes, Grèce (Athens,GR)** – Climat méditerranéen (remplace Metaxourgio pour meilleure stabilité)
3. **Dubaï, Émirats Arabes Unis (Dubai,AE)** – Climat désertique

4. **New York, États-Unis (New York,US)** – Climat tempéré/continental
5. **Paris, France (Paris,FR)** – Climat tempéré océanique

Rationnelle : Cette sélection permet de capturer la variabilité climatique globale tout en respectant le quota API.

5 Configuration des Composants

5.1 Kafka - Topic Weather Stream

5.1.1 Configuration du Topic

```
kafka-topics --create \
  --topic weather_stream \
  --bootstrap-server localhost:9092 \
  --partitions 3 \
  --replication-factor 2 \
  --config retention.ms=604800000
```

Listing 1 – Creation du topic Kafka

5.1.2 Paramètres

Partitions 3 partitions pour paralléliser la consommation (par ville)

Replication Factor 2 pour la haute disponibilité

Retention 7 jours (604800000 ms) pour l'archivage intermédiaire

5.2 AWS S3 - Stockage des Données Brutes

5.2.1 Structure de Partitionnement

Les données sont organisées selon la structure suivante :

```
s3://weather-bucket/weather/raw/
            year=2025/
                  month=01/
                        day=15/
                              hour=14/
                                    city=Fez/
                                          event_2025-01-15_14-30-45.
   json
                                    city=Athens/
                                        event_2025-01-15_14-30-50.json
                              hour=15/
                                  ...
```

Listing 2 – Schema de partitionnement S3

5.2.2 Avantages du Partitionnement

— Requêtes optimisées par date et ville (partition pruning)
— Suppression facile des données expirées
— Parallélisation de la lecture lors des analyses
— Gestion granulaire de la rétention

5.3 Lifecycle Policy S3

5.3.1 Configuration de la Rétention

Une règle de cycle de vie AWS S3 a été configurée pour automatiser la suppression des données :

5.3.2 Paramètres de la Règle

Paramètre	Valeur
ID de la règle	`weather-retention-30days`
Etat	Activée
Portée	Préfixe `weather/raw/`
Action	Expirer les versions actuelles d'objets
Durée	30 jours
Nettoyage des marqueurs	Oui

5.3.3 Comportement

— Les objets créés il y a 30 jours seront automatiquement supprimés
— AWS effectue le nettoyage de manière asynchrone (pas immédiat)
— La suppression s'effectue généralement dans les 24-48 heures suivant le délai
— Aucun coût supplémentaire pour les opérations de nettoyage

Important : Cette action n'est pas instantanée. AWS supprime les objets selon son processus interne, généralement autour du délai spécifié.

6 Format et Structure des Données

6.1 Schéma d'un Événement JSON

Chaque événement collecté contient les champs suivants :

```
{
  "event_time_utc": "2025-01-15T14:30:45Z",
  "city_query": "Fez,MA",
  "city": "Fez",
  "country": "MA",
  "coord": {
    "lat": 34.0331,
    "lon": -5.0033
  },
  "weather_main": "Clear",
  "weather_desc": "clear sky",
  "temp_c": 18.5,
  "feels_like_c": 17.8,
  "temp_min_c": 15.2,
  "temp_max_c": 21.3,
  "pressure_hpa": 1013.25,
  "humidity_pct": 65,
  "wind_speed_ms": 3.5,
  "wind_deg": 180,
```

```
  "cloudiness_pct": 10,
  "visibility_m": 10000,
  "rain_mm": 0,
  "snow_mm": 0,
  "timezone_offset_sec": 3600,
  "source": "openweather"
}
```

Listing 3 – Exemple d'événement météorologique

6.2 Description des Champs

Champ	Type	Description
event_time_utc	ISO 8601	Timestamp de collecte (UTC)
city_query	String	Requête envoyée à l'API
city	String	Nom de la ville
country	String	Code pays ISO
coord.lat, coord.lon	Float	Latitude et longitude
weather_main	String	Condition principale (Clear, Rainy, etc.)
weather_desc	String	Description détaillée
temp_c	Float	Température en Celsius
feels_like_c	Float	Température ressentie
temp_min_c, temp_max_c	Float	Min/Max température
pressure_hpa	Float	Pression atmosphérique
humidity_pct	Int	Humidité relative (%)
wind_speed_ms	Float	Vitesse du vent (m/s)
wind_deg	Int	Direction du vent (degrés)
cloudiness_pct	Int	Couverture nuageuse (%)
visibility_m	Int	Visibilité (mètres)
rain_mm, snow_mm	Float	Précipitations
timezone_offset_sec	Int	Offset UTC (secondes)

7 Flux de Données

7.1 Cycle de Vie d'un Événement

1. **Collection** (t+0s) : Collecteur HTTP interroge OpenWeather API pour chaque ville
2. **Enrichissement** (t+0.5s) : Données transformées en événement JSON standardisé
3. **Streaming** (t+1s) : Événement publié dans topic Kafka `weather_stream`
4. **Persistance** (t+2s) : Événement consommé et écrit dans S3 selon structure partitionnée
5. **Rétention** (0-30j) : Données accessibles pour consultation
6. **Expiration** (30j+) : Lifecycle policy supprime automatiquement les objets

7.2 Fréquence de Collecte

$$\text{Intervalle} = 600 \text{ secondes} = 10 \text{ minutes}$$

Par jour :

$$N_{\text{collectes/jour}} = \frac{86400}{600} = 144 \text{ collectes}$$

Pour 5 villes :

$$N_{\text{appels API/jour}} = 144 \times 5 = 720 \text{ appels}$$

8 Considérations Opérationnelles

8.1 Monitoring et Alerting

— Surveiller le taux d'erreur API OpenWeather
— Monitorer la disponibilité de Kafka
— Vérifier le ratio de réussite des écritures S3
— Alerter si les quotas approchent les 1000 appels/jour

8.2 Récupération en Cas d'Erreur

1. **Retry Logic** : Réessais exponentiels pour les appels API
2. **Dead Letter Queue** : Kafka DLQ pour les événements non traitables
3. **Checkpointing** : Sauvegarde des offsets Kafka pour éviter les doublons

8.3 Sécurité

— Clés API OpenWeather stockées dans AWS Secrets Manager
— Credentials AWS restrictifs (principe du moindre privilège)
— Chiffrement en transit (HTTPS) et au repos (S3 SSE)
— Contrôle d'accès S3 via IAM policies

Cette phase constitue le cœur analytique du projet. Elle assure le passage d'un stockage brut et non structuré dans le Data Lake S3 vers une base de données relationnelle hautement optimisée pour le requêtage.

8.4 Connexion Snowflake et AWS S3

La première étape consiste à établir un pont sécurisé entre AWS et Snowflake. Pour éviter de manipuler des identifiants statiques sensibles, nous avons configuré une **Storage Integration**.

Configuration de l'intégration de stockage

L'intégration d'objets de stockage permet à Snowflake de déléguer la gestion des accès à AWS IAM. Comme le montre la capture d'écran suivante, nous avons lié Snowflake au rôle ARN spécifique de notre infrastructure AWS :

Cette commande SQL permet de définir les emplacements autorisés (`STORAGE_ALLOWED_LOCATIONS`), restreignant ainsi Snowflake à ne lire que les données du dossier `/weather/` de notre bucket. Grâce à cela, Snowflake peut créer un **External Stage**, agissant comme une fenêtre directe sur nos fichiers JSON sans nécessiter de déplacement physique des données.

8.5 Modélisation et Transformation avec dbt

Une fois les données accessibles dans Snowflake sous forme brute (type `VARIANT`), nous utilisons **dbt (Data Build Tool)** pour orchestrer les transformations via du code SQL modulaire.

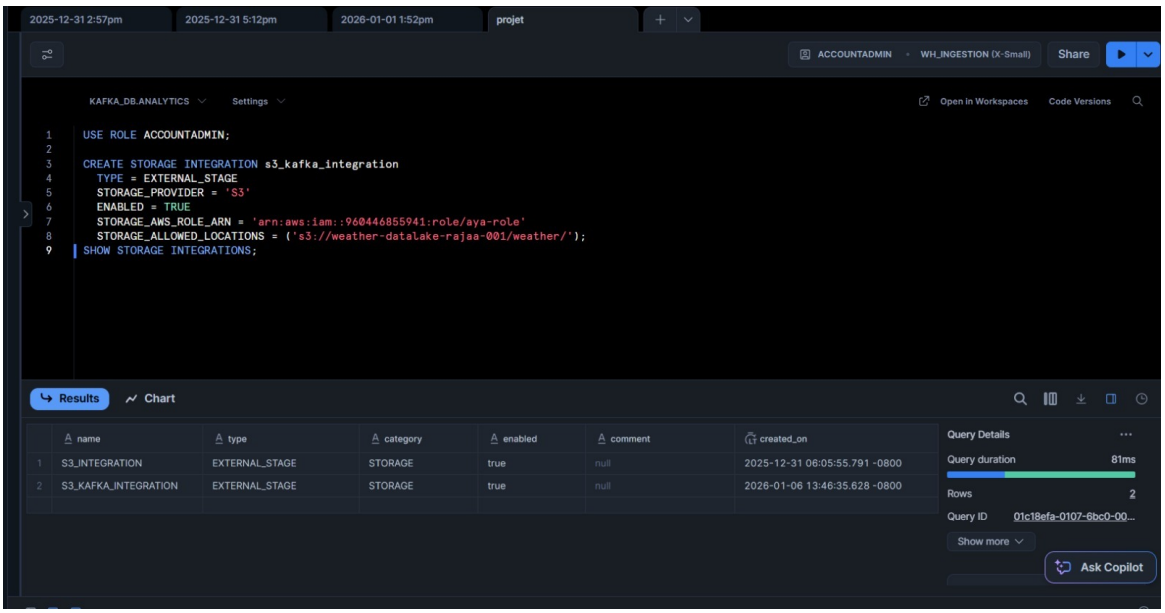

FIGURE 1 – Création de la Storage Integration $\texttt{s3}_k afka_i ntegration dans l'interface Snowflake$.

Analyse du Lignage des Données (Lineage Graph)

Le processus de transformation suit une architecture en couches (Medallion Architecture) pour garantir la qualité de la donnée finale. Le graphe de lignage généré par dbt illustre cette dépendance :

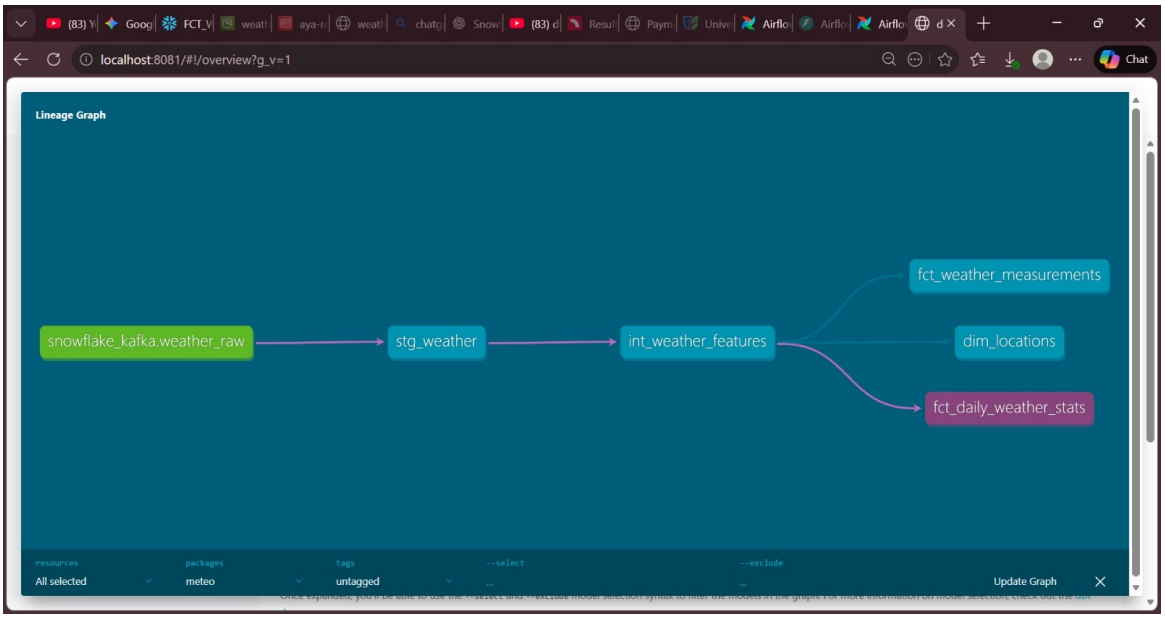

FIGURE 2 – Graphe de lignage dbt montrant le flux de transformation des données brutes vers les modèles finaux.

Détail des couches de transformation

— **Source (`snowflake_kafka.weather_raw`)** : Il s'agit de la table "Landing" où le JSON brut arrive. Elle contient une colonne unique contenant l'intégralité du payload météo.
— **Staging (`stg_weather`)** : Cette couche effectue le "casting" des données. On extrait les champs JSON (température, humidité, vent) pour les transformer en colonnes SQL

typées. C'est ici que nous gérons le nettoyage initial.
— **Intermediate (`int_weather_features`)** : Cette étape prépare les calculs complexes. Elle centralise les transformations communes afin d'éviter la duplication de code dans les modèles finaux.
— **Marts (Couche de sortie)** :
  — `fct_weather_measurements` : Table de faits contenant l'historique complet de toutes les mesures à la minute.
  — `dim_locations` : Table de dimension regroupant les informations géographiques des villes (latitude, longitude, pays).
  — `fct_daily_weather_stats` : Modèle agrégé calculant les indicateurs clés (min, max, moyenne) par jour, idéal pour les rapports de tendances à long terme.

9 Visualisation avancée et dashboarding opérationnel

La dernière étape de ce projet consiste à rendre la donnée accessible et actionnable pour les utilisateurs métiers, techniques ou non, via des outils modernes de data visualisation. Nous avons choisi d'utiliser deux solutions : **Grafana** pour le monitoring et la BI temps réel, et **Streamlit** pour l'exploration interactive sur-mesure.

9.1 Pourquoi Grafana et Streamlit ?

Ces outils sont complémentaires :
— **Grafana** : reconnu pour sa robustesse, son interface temps réel connectée à plusieurs data sources (Snowflake, PostgreSQL, Prometheus...), son système de panels dynamiques et d'alerting (très adapté au monitoring, mais aussi à la BI météo opérationnelle).
— **Streamlit** : très utilisé pour développer rapidement des web apps de data science, prototypage, exploration, ou partages de résultats personnalisés sans avoir à déployer une infrastructure lourde.

Les deux approches couvrent l'ensemble des besoins utilisateur : monitoring quotidien, analyse multi-villes, visualisation ad hoc, génération de rapports, exploration "no code", prise de décision rapide.

9.2 Dashboard météo avec Grafana

Prise en main et Connexion

Grafana a été relié directement à la base Snowflake via le connecteur natif. Cela permet :
— L'accès en temps réel à toutes les tables transformées (modèle en étoile optimisé pour l'analyse)
— La possibilité d'exploiter les macros temporelles avancées (filtres de période dynamiques)
— La gestion de droits selon les utilisateurs (authentification sécurisée)

Authentification : Grafana offre nativement différents modes d'authentification (compte utilisateur, LDAP, Google, etc.) permettant de garantir la sécurité et la traçabilité des accès.

Construction du Dashboard

— Création d'un dashboard principal multi-panels, où chaque panel = un angle d'analyse métier

— Sélection des sources : requêtes SQL sur les tables météo (`fct_weather_measurements`, `fct_daily_weather_stats`, etc.), souvent jointes avec `dim_locations`
— Configuration des filtres globaux (sélecteur de ville, période, type de métrique)

Types de visualisations utilisés :

1. **Cartes géographiques (World Map Panel)** : localisation des villes suivies, couleur selon la température, taille par force du vent
2. **Bar charts et classements** : comparaison directe entre villes pour la température, l'humidité, l'amplitude thermique, le score de confort
3. **Courbes temporelles (Time Series)** : suivi évolutif par heure/jour
4. **Pie charts** : distribution des conditions météo
5. **Heatmaps** : relations humidité/température et répartition "heures les plus pluvieuses"
6. **Panels Stat/alertes** : indicateurs clés affichés avec changement de couleur dès que des seuils sont atteints (alerte canicule, pluie persistante, vent fort, etc.)

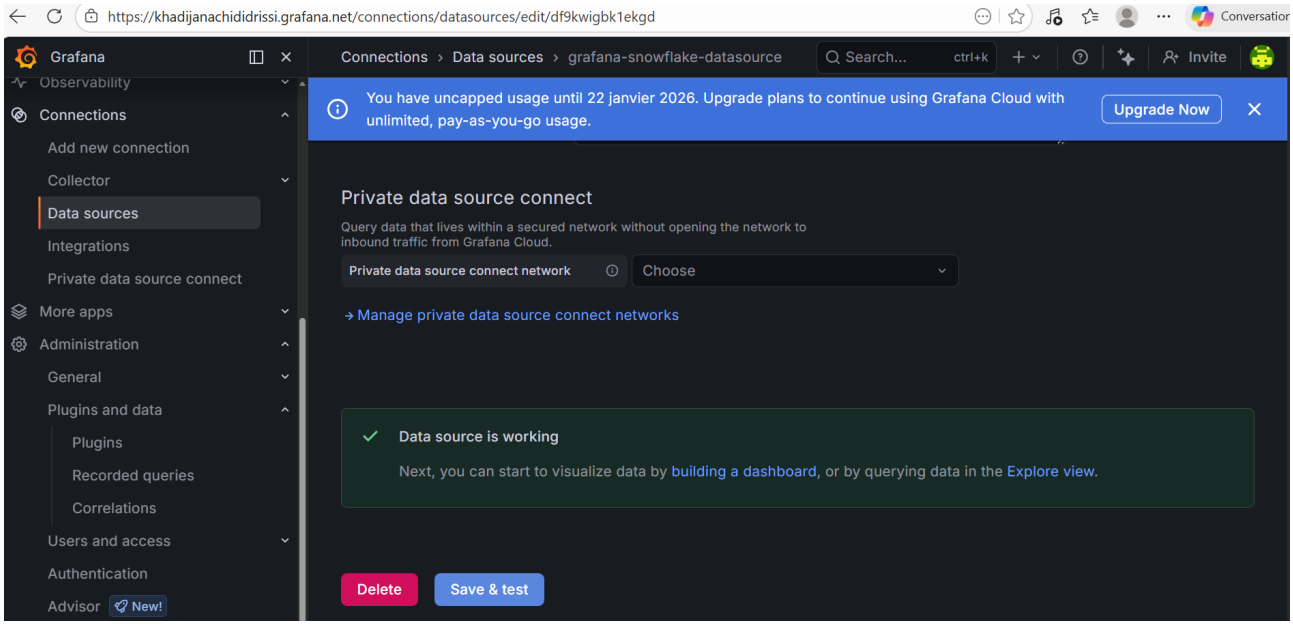

FIGURE 3 – Intégration de Grafana avec la stack Snowflake/DBT

Points forts de Grafana :
— Visualisation ultra-rapide, actualisation quasi-instantanée (idéal pour du monitoring météo)
— Panels très variés pour répondre à tous les cas d'analyse
— Interface de filtres très ergonomique
— Système d'alerting natif

Limites :
— Personnalisation limitée des interactions (vs. une vraie webapp)
— Peu adapté aux analyses "exploratoires" non prévues à l'avance

9.3 Applications et exploration interactive avec Streamlit

Contexte et déploiement

Streamlit a été choisi pour :

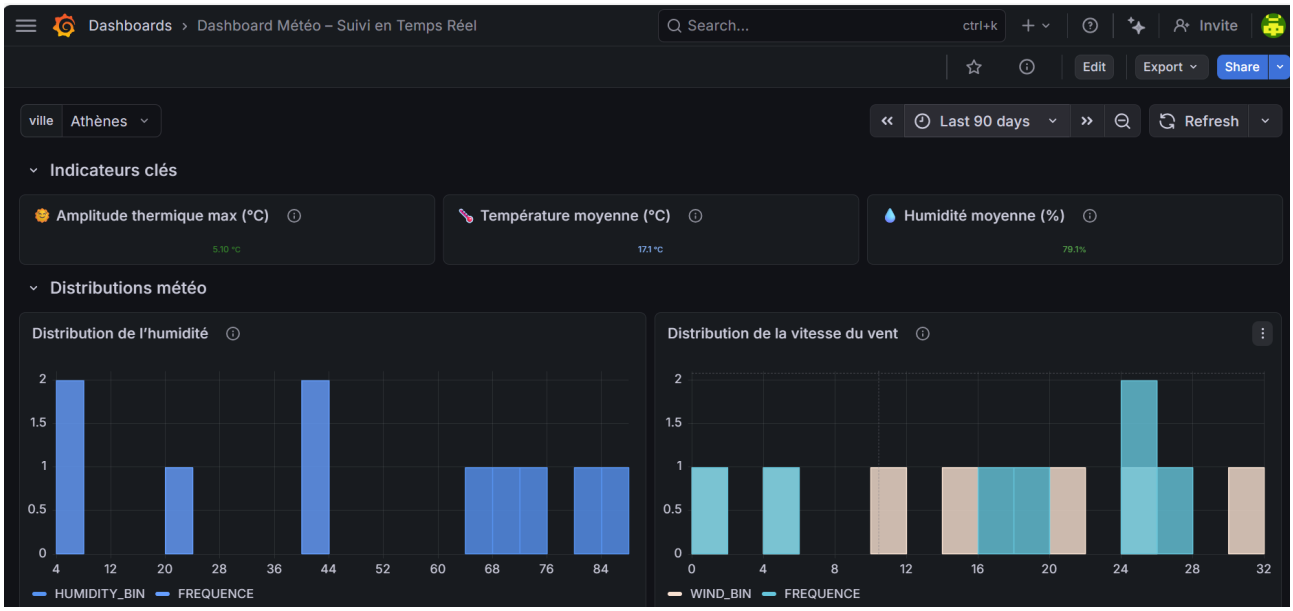

FIGURE 4 – Dashboard météo Grafana : Indicateurs clés avec filtres multi-villes et plages temporelles

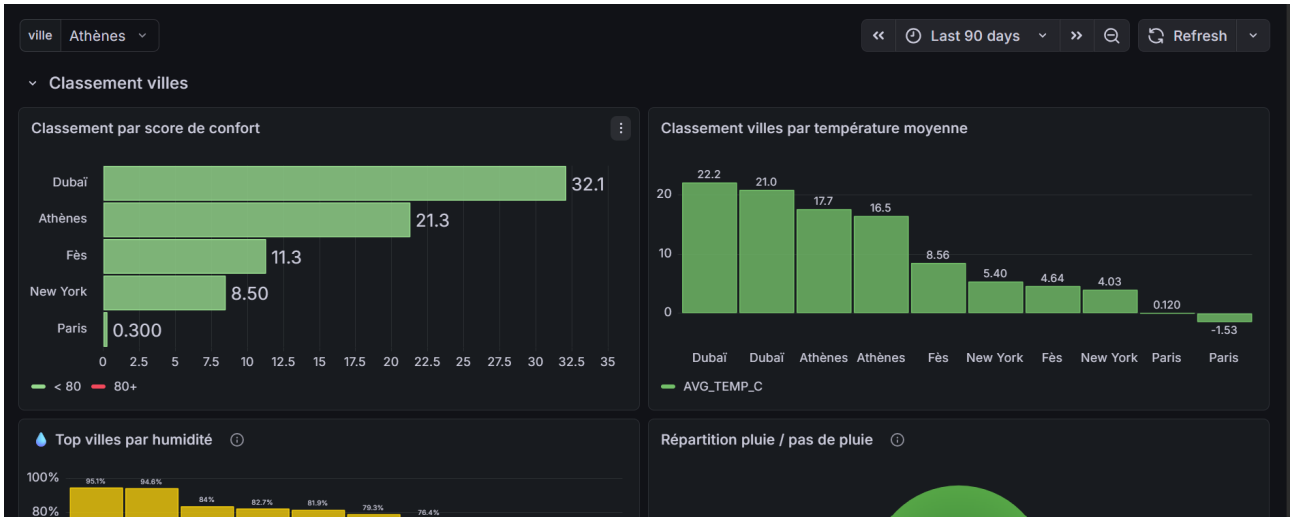

FIGURE 5 – Classements villes, pie chart des conditions météo

— Offrir une interface data-scientist "low code" : construction de rapports interactifs, analyse sur mesure, partage rapide de prototypes.
— Permettre à des profils métiers non techniques de "jouer" avec les données météo : sélection dynamique des villes/périodes, génération de courbes ou téléchargement des résultats.
— Tester rapidement différentes méthodes de visualisation ou corrélation (par exemple : météo vs pollution si extension du projet).

Déploiement : Utilisé en local (localhost) ou hébergé sur Streamlit Cloud suivant les besoins. Application accessible via une simple URL et partagée très facilement entre membres de l'équipe.

Fonctionnalités principales développées

— Tableau de bord "full custom" : une tuile/graffe par indicateur clé

FIGURE 6 – Les tendances

— Sélecteur de ville, période, ou condition météo
— Génération de courbes, histogrammes, et heatmaps dynamiques selon le choix de l'utilisateur
— Options d'export des graphes au format image ou CSV
— Possibilité d'évoluer vers du machine learning (détection automatique d'anomalies météo, prévisions, etc.)

FIGURE 7 – Interface Streamlit personnalisée : sélection dynamique des villes et analyses météo

Forces de Streamlit :
— Liberté totale sur la forme des dashboards et l'interactivité des analyses
— Adaptabilité pour des besoins exploratoires très spécifiques
— Facilité de déploiement et de partage

Limites :
— Moins instantané que Grafana pour le suivi temps réel (rafraîchissements à la demande)
— Moins adapté au monitoring/alerting automatisé

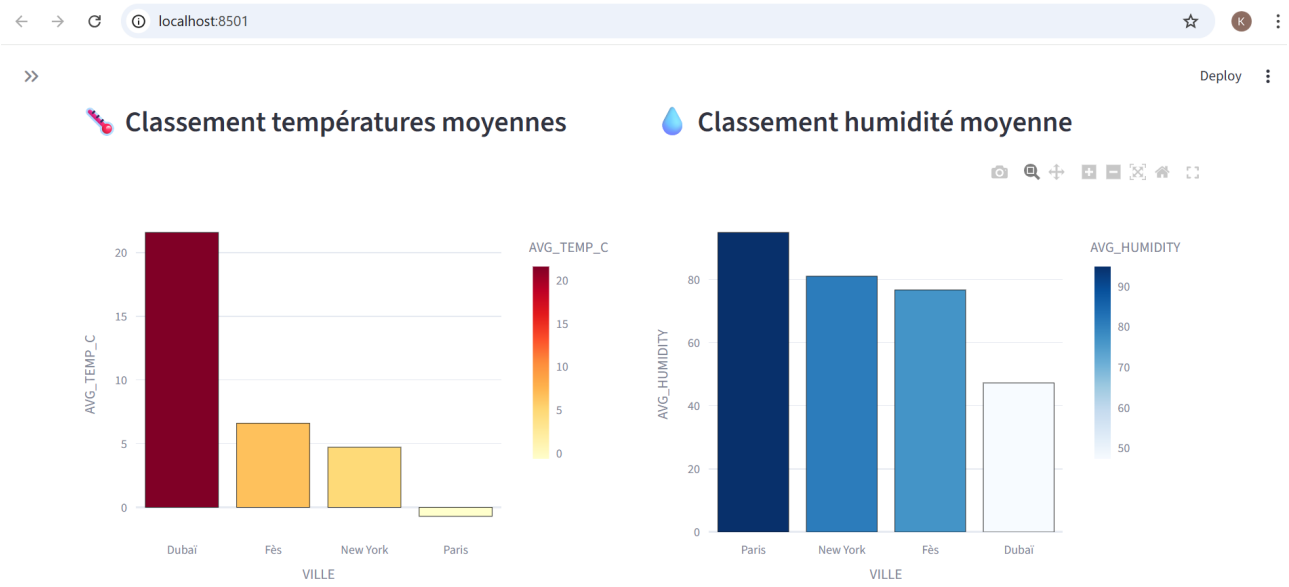

FIGURE 8 – Exemples de graphes interactifs et d'export dans Streamlit

9.4 Comparatif et enrichissement métier

Grâce à la complémentarité entre Grafana et Streamlit, nous pouvons :

— **Monitorer** en temps réel tous nos indicateurs météo sur l'ensemble des villes surveillées
— **Explorer** les données historiques ou zoomer/interroger des périodes ou phénomènes spécifiques (pluie exceptionnelle, canicule, etc.)
— **Industrialiser** la diffusion des dashboards avec un effort minimal grâce à l'infrastructure cloud (Snowflake, DBT, S3)
— **Adapter** rapidement l'interface aux nouvelles exigences métier ou scientifiques (ajout de villes, nouvelles métriques, etc.)

Perspectives : Ces interfaces rendent le pipeline very "data-driven" et prêt à être étendu, par exemple :

— Ajout d'intelligence artificielle pour la prévision ou l'alerte précoce
— Brancher d'autres bases (pollution, réseau transport, sinistres climatiques...)
— Connecter Grafana à d'autres outils d'alerting (mails, Slack, SMS...)