KAFKA

Exercice 1 : Mise en place de Kafka et d'un producteur simple

1. Créez un topic Kafka nommé `weather_stream`.
2. Envoyez un message statique, par exemple :

```
{"msg": "Hello Kafka"}
```

Exercice 2 : Écriture d'un consommateur Kafka

1. Créez un script Python consommateur qui lit les messages depuis un topic Kafka passé en argument.
2. Affichez les messages reçus en temps réel.

Exercice 3 : Streaming de données météo en direct

1. Écrivez un producteur `current_weather` qui interroge l'API Open-Meteo pour une **latitude/longitude** passée en argument.
2. Envoyez les données reçues dans le topic Kafka `weather_stream`.

Exercice 4 : Transformation des données et détection d'alertes

1. Traitez le flux `weather_stream` en temps réel avec **Spark**.
2. Produisez un topic Kafka `weather_transformed` contenant :
   - **Alertes de vent** :
     - Vent faible (< 10 m/s) → `level_0`
     - Vent modéré (10–20 m/s) → `level_1`
     - Vent fort (> 20 m/s) → `level_2`
   - **Alertes de chaleur** :
     - Température normale (< 25°C) → `level_0`

    - Chaleur modérée (25–35°C) → `level_1`
    - Canicule (> 35°C) → `level_2`
3. Ajoutez les colonnes suivantes :
    - `event_time` (timestamp)
    - `temperature` et `windspeed` transformés si nécessaire
    - `wind_alert_level` et `heat_alert_level`

Exercice 5 : Agrégats en temps réel avec Spark

Goal : Calculer des agrégats en temps réel sur des fenêtres glissantes ou fixes.

1. Implémentez un **sliding window** (1 ou 5 minutes) sur le flux `weather_transformed`.
2. Calculez des métriques comme :
    - Nombre d'alertes `level_1` ou `level_2` par type d'alerte (vent/chaleur)
    - Moyenne, min, max de la température
    - Nombre total d'alertes par ville ou pays

Exercice 6 : Extension du producteur

1. https://open-meteo.com/en/docs/geocoding-api?name=paris
2. Modifiez les producteurs pour accepter **la ville et le pays** comme arguments
3. Chaque message produit doit inclure ces informations afin de permettre le partitionnement par HDFS et les agrégats par région.

Exercice 7 : Stockage dans HDFS organisé

1. Écrivez un consommateur Kafka qui lit `weather_transformed`.
2. Sauvegardez les alertes dans **HDFS** :
    - Structure : `/hdfs-data/{country}/{city}/alerts.json`

Exercice 8 : Visualisation et agrégation des logs météo

Consommer les logs HDFS et Implémentez les visualisations suivantes :

- Évolution de la température au fil du temps
- Évolution de la vitesse du vent
- Nombre d'alertes vent et chaleur par niveau
- Code météo le plus fréquent par pays

Exercice 9 : Récupération de séries historiques longues

- Téléchargez des **données météo sur 10 ans** pour une ville donnée (ex. Paris) via l'API archive (https://archive-api.open-meteo.com/)
- Stockez les données brutes dans Kafka (`/hdfs-data/{country}/{city}/weather_history_raw`) et sauvegardez dans HDFS.

Exercice 10 : Détection des records climatiques locaux

- Écrivez un job Spark qui analyse HDFS pour chaque ville :
  - Le **jour le plus chaud** et **le plus froid** de la décennie.
  - La **rafale de vent la plus forte**.
  - La **période la plus pluvieuse**.
- Émettez ces "records" dans Kafka (`/hdfs-data/{country}/{city}/weather_records`) pour exploitation par un dashboard plus tard

---

Exercice 11 : Climatologie urbaine (profils saisonniers)

- Avec Spark, regroupez les données historiques par **mois** pour chaque ville.
- Calculez :
  - Température moyenne par mois (profil saisonnier).
  - Moyenne de vitesse du vent.
  - Probabilité d'alerte (`level_1` ou `level_2`).
- Stockez les résultats dans HDFS sous `/hdfs-data/{country}/{city}/seasonal_profile/` .

Exercice 12 : Validation et enrichissement des profils saisonniers

- Vérifier que chaque ville/pays possède un profil complet sur **12 mois**.

- Détecter les valeurs manquantes (par exemple, température moyenne de janvier manquante).
- Vérifier que les valeurs sont réalistes :
  - température entre `-50°C` et `+60°C`,
  - vent entre `0` et `60 m/s`.

1. **Calcul de statistiques de dispersion**
   - Ajouter l'**écart-type** (`std`) pour température et vent.
   - Ajouter les **valeurs min et max observées** par mois.
   - Cela permettra ensuite de détecter des anomalies basées sur la variabilité, pas seulement sur la moyenne.
2. **Enrichissement du profil**
   - Calculer la **médiane**
   - Ajouter des **quantiles** (ex. Q25, Q75) pour définir des seuils dynamiques.
3. **Sauvegarde dans HDFS**
   - Stocker les profils validés/enrichis sous :

     ```
     /hdfs-data/{country}/{city}/seasonal_profile_enriched/{year}/profile.json
     ```

Exercice 13 : Détection d'anomalies climatiques (Batch vs Speed)

- **Ingestion des données de comparaison**
  - Lire en temps réel les messages du topic Kafka `weather_transformed`.
  - Charger depuis HDFS les profils saisonniers (par ville, pays, mois).
    - Exemple : température moyenne, vent moyen, probabilité d'alerte.
- **Jointure Batch vs Speed**
  - Associer chaque mesure temps réel à la ligne de référence historique correspondante :
    - clé = `{country, city, month}`

        - colonnes de référence = `temp_mean`, `wind_mean`, `alert_probabilities`.
- **Définition des seuils d'anomalie**
    - Température : considérer comme anomalie si l'écart entre `temp_realtime` et `temp_mean` est supérieur à un seuil (ex. ±5 °C).
    - Vent : anomalie si `windspeed_realtime` dépasse de 2 écarts-types la moyenne historique.
    - Alertes : détecter un écart de fréquence significatif entre alertes réelles et probabilité historique.
- **Détection en streaming avec Spark**
    - Implémentez une logique Spark Structured Streaming pour calculer ces écarts en temps réel.
    - Produisez un champ booléen ou textuel : `is_anomaly`, `anomaly_type`.
    - Ajoutez métadonnées utiles : `event_time`, `city`, `country`, `variable`, `observed_value`, `expected_value`.
- **Publication des anomalies**
    - Émettre les résultats dans un topic Kafka `weather_anomalies`.
    - Exemple de message JSON :

```
{
  "city": "Paris",
  "country": "France",
  "event_time": "2025-09-23T15:00:00Z",
  "variable": "temperature",
  "observed_value": 30,
  "expected_value": 18,
  "anomaly_type": "heat_wave"
}
```

- **Sauvegarde et exploitation**
    - Sauvegarder également les anomalies dans HDFS : `/hdfs-data/{country}/{city}/anomalies/{year}/{month}/anomalies.json`
    - Ces anomalies seront utilisées par les dashboards du frontend

---

Exercice 13 : Frontend global de visualisation météo

Créer une interface web qui regroupe **tous les dashboards et visualisations** issus des exercices précédents (temps réel, historique, comparaisons et anomalies).