Documentation du Projet : Mise en place d'une plateforme météorologique Analytique avec Kafka, Cassandra/MongoDB, Docker, Docker Compose, Python, et Kafka

Hajar EL HAFA
Nada BEN BRAHIM
Meryem OUJA
Moundji BELHANNACHI
Reda KAMEL
Wildried KAMAHA MONKAM

Introduction

Ce projet a pour objectif de construire une plateforme automatisée pour la collecte, le traitement, le stockage et la visualisation des données météorologiques. En utilisant des technologies telles que Kafka, Cassandra/MongoDB, Docker, et Python, cette plateforme est capable de scraper des données météorologiques en temps réel et de les afficher via un tableau de bord interactif.

A- Étapes de Réalisation

A. Sélectionner les données selon les villes

1. **Récupération des données de l'API :**
    - Nous avons utilisé l'API OpenWeatherMap pour obtenir les données météorologiques des villes de France. Les identifiants des villes ont été extraits d'un fichier JSON fourni par OpenWeatherMap.
2. **Vérification des résultats des requêtes vers l'API :**
    - Une vérification systématique a été mise en place pour s'assurer que les données récupérées de l'API sont valides. Cela inclut la gestion des erreurs pour les requêtes qui échouent ou retournent des données invalides.
3. **Transformation des données :**
    - Les données récupérées ont été transformées pour s'assurer qu'elles sont dans un format cohérent et utilisable pour le stockage et l'analyse. Cela comprend la conversion des formats de date et l'ajustement des unités de mesure.

B. Développement du Consumer Python sur un conteneur

1. **Consommation des messages Kafka :**

- Un consommateur Kafka a été développé en Python pour traiter les messages en temps réel. Ce consommateur est configuré pour fonctionner en continu et écouter les messages Kafka, les traitant dès qu'ils sont reçus.

2. **Communication entre le consommateur Kafka et Streamlit :**
   - Les données consommées de Kafka sont stockées dans une structure de données partagée, permettant à Streamlit d'accéder aux données en temps réel et de les afficher sur un tableau de bord interactif.
3. **Récupération des données historiques depuis Cassandra :**
   - Le consommateur Kafka récupère également les données historiques stockées dans Cassandra, permettant une visualisation des données passées sur le tableau de bord Streamlit.

C. Choix d'une clé appropriée pour l'organisation des données

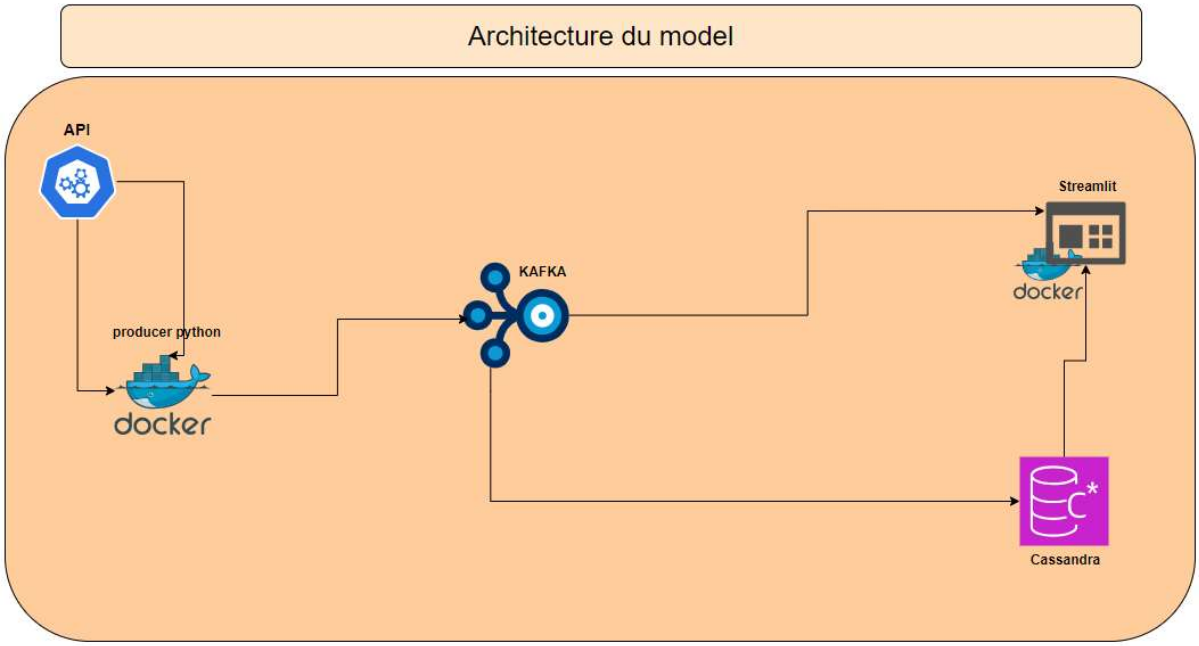
1. **Publication des données sur Kafka :**
   - Les données sont publiées sur Kafka avec une clé de partitionnement appropriée pour garantir une répartition équilibrée des messages et faciliter leur récupération.
2. **Définition d'une fonction callback pour la publication Kafka :**
   - Une fonction callback a été implémentée pour gérer les succès et les erreurs lors de l'envoi de messages à Kafka, assurant ainsi une robustesse accrue du système.

D. Configuration de la base de données Cassandra

1. **Déclaration de la base de données Cassandra :**
   - Cassandra a été configurée pour stocker un historique complet des données météorologiques, en utilisant des clés de partitionnement basées sur le temps pour optimiser les requêtes par date.
2. **Exécution dans un conteneur Docker :**
   - Cassandra est déployée dans un conteneur Docker séparé, permettant une isolation et une gestion faciles de la base de données dans l'environnement de développement et de production.

B- Architecture de la plateforme météorologique

Le diagramme ci-dessus représente l'architecture de la plateforme météorologique analytique que nous avons développée.



API :

- La plateforme commence par collecter des données météorologiques. Ces données sont récupérées à partir du site openweathermap avec une API pour différentes villes de France à intervalle régulier.

Producteur Python dans un conteneur Docker :

- Un script Python, encapsulé dans un conteneur Docker, est utilisé pour interroger l'API et extraire les informations météorologiques pertinentes.
- Le producteur transforme ces données, si nécessaire, avant de les envoyer au consommateur.

Kafka :

- Kafka est utilisé comme système de streaming de données. Il reçoit les messages produits par le producteur kafka et les distribue aux différents consommateurs.
- Cela permet de gérer efficacement le flux de données en temps réel et de s'assurer que toutes les informations météorologiques sont transmises aux services appropriés.

Consumer Python dans un conteneur Docker :

- Un consommateur Python, également exécuté dans un conteneur Docker, est responsable de la consommation des messages de Kafka.

- Ce consommateur traite les données reçues et les stocke dans une base de données Cassandra pour un accès historique.

Base de données Cassandra :

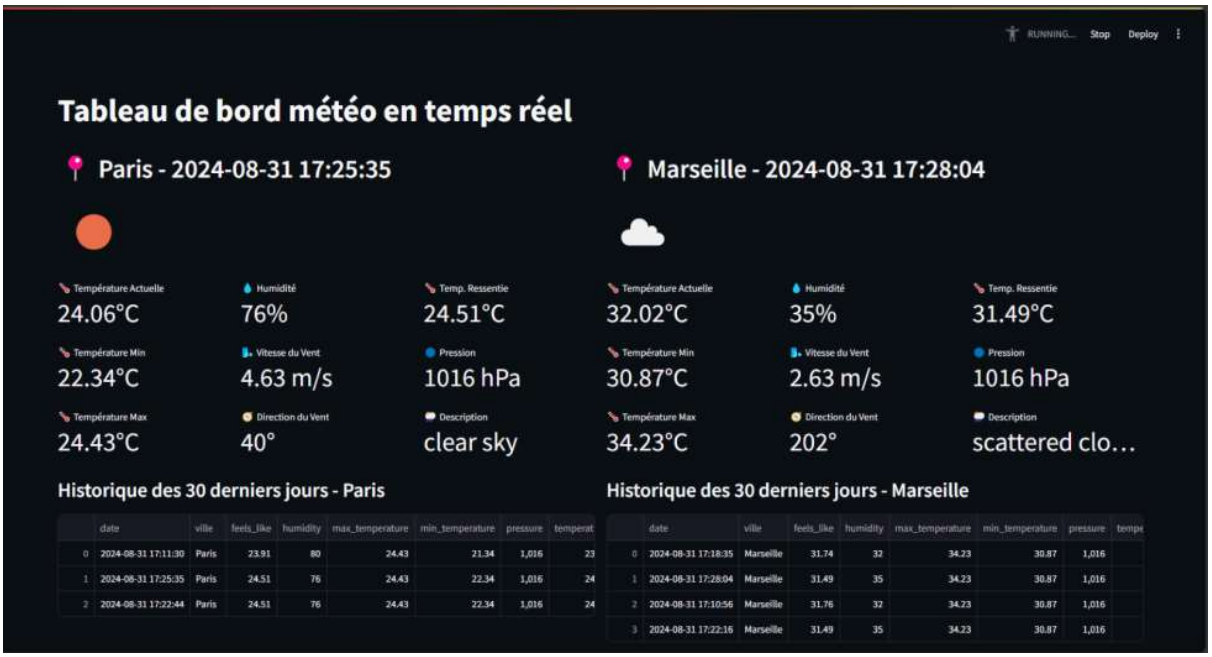
- Cassandra est utilisée pour stocker les données météorologiques historiques. La base de données est configurée pour optimiser les requêtes de données par ville et par date, offrant ainsi une récupération rapide des données historiques.

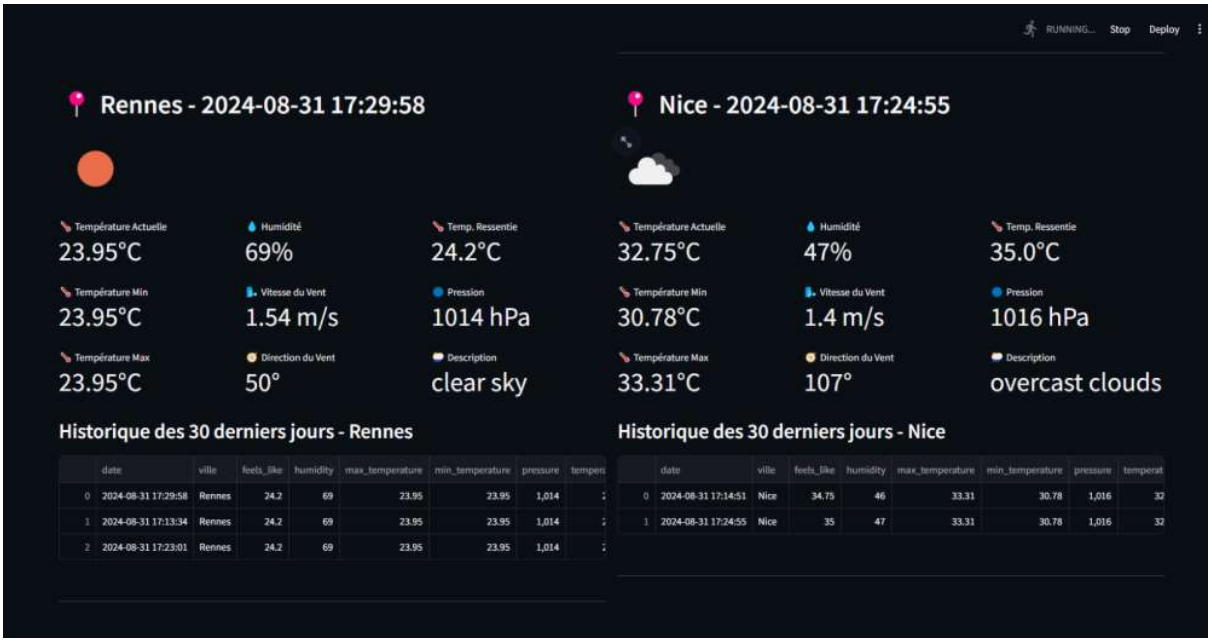
Streamlit dans un conteneur Docker :

- Streamlit, une bibliothèque Python, est utilisée pour développer une interface utilisateur interactive.
- Ce service consomme les données en temps réel provenant de Kafka et les données historiques de Cassandra pour afficher un tableau de bord interactif.
- Le tableau de bord permet aux utilisateurs de visualiser les conditions météorologiques actuelles et passées, avec une fréquence de rafraîchissement élevée.

C- Dashboard Temps Réal avec Streamlit

Cette plateforme nécessite Docker en cours d'exécution et peut être lancée avec la commande `docker compose up -d`.
Pour vérifier le tableau de bord Streamlit, l'adresse est la suivante : ``https://localhost:8501``
Remarque Importante : **Il faut attendre quelques instants avant que les informations s'affiche dans le dashboard !**





Le tableau de bord météorologique en temps réel a été développé à l'aide de **Streamlit** et affiche les conditions météorologiques actuelles pour plusieurs villes (Paris, Marseille, Rennes, Nice). Pour chaque ville, les informations suivantes sont présentées de manière claire et organisée :

- **Température actuelle** : La température enregistrée à l'instant dans la ville.
- **Humidité** : Le pourcentage d'humidité de l'air.
- **Température ressentie** : La température perçue, qui peut différer de la température réelle en fonction de facteurs comme le vent et l'humidité.
- **Températures minimale et maximale** : Les températures les plus basses et les plus élevées enregistrées récemment.

- **Vitesse et direction du vent** : Informations sur la force et la direction du vent.
- **Pression atmosphérique** : Mesure de la pression barométrique actuelle.
- **Description du ciel** : Une brève description des conditions météorologiques actuelles, par exemple "ciel dégagé" ou "nuages dispersés".

En dessous de ces informations en temps réel, un tableau récapitulatif présente l'historique des conditions météorologiques des 30 derniers jours pour chaque ville, incluant des données telles que la température, l'humidité, et la pression.

Conclusion

Ce projet a permis de mettre en place une solution complète pour la collecte, le traitement, le stockage et la visualisation des données météorologiques. Grâce à l'utilisation de Kafka pour le streaming de données et de Docker pour la conteneurisation, la plateforme est prête pour une mise en production scalable.