Documentation et Guide d'Utilisation – WeatherPipeline

Groupe 4:

- ✓ **Paulin NGUEMO**
- ✓ **LAHMAR Omar**
- ✓ **Bintou MAIGA**
- ✓ **Andy KOUTON**
- ✓ **Namy Romual**

1. Introduction

WeatherPipeline est une application conçue pour collecter, stocker et visualiser des données météorologiques en temps réel. Cette solution repose sur des technologies modernes telles que Flask, MongoDB, et Dash, et peut être étendue pour inclure des fonctionnalités avancées telles que des notifications météo et des prédictions basées sur l'intelligence artificielle.

Les objectifs principaux sont :

- Faciliter la collecte de données météo pour plusieurs villes.
- Stocker ces données dans une base de données MongoDB.
- Fournir un tableau de bord interactif pour explorer et analyser ces données.

---

2. Étapes d'installation

2.1 Prérequis

- Python 3.9 ou version ultérieure.
- MongoDB Community Edition.
- Node.js (optionnel, pour un front-end React si vous souhaitez l'ajouter).
- Docker (optionnel, pour conteneuriser l'application).

2.2 Installation

1. **Cloner le projet depuis GitHub** :

```
bash
Copier le code
```

```
git clone https://github.com/<votre-repo>/WeatherPipeline.git
cd WeatherPipeline
```

2. **Installer les dépendances Python** :
   Assurez-vous que vous êtes dans l'environnement virtuel ou global approprié, puis exécutez :

   ```
   bash
   Copier le code
   pip install -r requirements.txt
   ```

3. **Configurer MongoDB** :
   - **Linux** :

     ```
     bash
     Copier le code
     sudo systemctl start mongodb
     sudo systemctl enable mongodb
     ```

   - **Windows** : Lancez MongoDB Compass et configurez une instance locale.
4. **Configurer Flask** :
   Lancez le serveur Flask pour les API backend :

   ```
   bash
   Copier le code
   python run.py
   ```

5. **Lancer le tableau de bord Dash** :
   Exécutez :

   ```
   bash
   Copier le code
   python run_dashboard.py
   ```

6. **Tester l'installation** :
   - Accédez à l'API à l'adresse : http://127.0.0.1:5000.
   - Accédez au tableau de bord Dash : http://127.0.0.1:8050.

---

3. Architecture du pipeline

WeatherPipeline suit une architecture modulaire et évolutive. Voici une vue simplifiée des composants clés :

1. **API OpenWeather** :
   - Source des données météorologiques.

    - Requêtes effectuées via l'API REST pour récupérer les conditions météo d'une ville.
2. **Backend Flask** :
    - Implémentation des API REST pour interagir avec les données.
    - Sauvegarde des données dans MongoDB.
3. **Base de données MongoDB** :
    - Stockage persistant des données météo.
    - Accessible via PyMongo pour des opérations CRUD.
4. **Tableau de bord Dash** :
    - Visualisation interactive des données météo.
    - Graphiques dynamiques créés avec Plotly.
5. **Docker (optionnel)** :
    - Facilite le déploiement et l'exécution des services dans des conteneurs.

Schéma simplifié de l'architecture :

```
css
Copier le code
[OpenWeather API] -> [Flask Backend] -> [MongoDB] -> [Dash Frontend]
```

---

4. Exemples de requêtes API

4.1 Récupérer les données météo d'une ville

- **Requête** :

  ```
  http
  Copier le code
  GET /weather?city=Paris
  ```

- **Exemple de réponse** :

  ```
  json
  Copier le code
  {
      "city": "Paris",
      "temperature": 15,
      "humidity": 80,
      "weather": "Clear Sky"
  }
  ```

4.2 Vérifier les conditions météo extrêmes

- **Requête** :

  ```
  http
  Copier le code
  POST /check_weather
  ```

- **Corps JSON** :

```
json
Copier le code
{
    "main": {"temp": 42},
    "weather": [{"description": "clear sky"}]
}
```

- **Réponse attendue** :

```
json
Copier le code
{
    "message": "Notification envoyée"
}
```

4.3 Sauvegarder des données dans MongoDB

- **Requête** :

```
http
Copier le code
POST /save_weather
```

- **Corps JSON** :

```
json
Copier le code
{
    "city": "Paris",
    "temperature": 15,
    "humidity": 80,
    "weather": "Clear Sky"
}
```

- **Réponse** :

```
json
Copier le code
{
    "message": "Data saved",
    "id": "64b10f48c45e5a4b128a"
}
```

---

5. Utilisation du tableau de bord interactif

5.1 Accéder au tableau de bord

- Lancez l'application Dash avec :

```
bash
Copier le code
python run_dashboard.py
```

- Ouvrez un navigateur et accédez à http://127.0.0.1:8050.

5.2 Fonctionnalités principales

1. **Visualisation des températures** :
   - Graphique à barres affichant les températures pour les villes collectées.
   - Les données sont récupérées dynamiquement depuis MongoDB.
2. **Filtrage interactif** :
   - Ajoutez des options de filtrage (villes, plages de températures) pour personnaliser l'affichage.
3. **Extension possible** :
   - Intégrez des cartes interactives ou des prédictions météo basées sur machine learning.

---

6. Conseils pour le déploiement

1. **Utiliser Docker** :
   Conteneurisez vos services Flask, MongoDB, et Dash en créant un fichier `docker-compose.yml` :

   ```
   yaml
   Copier le code
   version: '3.8'
   services:
     flask_app:
       build: .
       ports:
         - "5000:5000"
     mongodb:
       image: mongo
       ports:
         - "27017:27017"
     dash_app:
       build: .
       ports:
         - "8050:8050"
   ```

2. **Déployer sur un serveur cloud** :
   Utilisez des services comme AWS, GCP ou Azure pour héberger vos conteneurs.

Conclusion

Le projet **WeatherPipeline** offre une architecture flexible et extensible pour la collecte et l'analyse de données météorologiques. En suivant cette documentation, vous pourrez installer et utiliser l'application, tout en explorant des possibilités d'extension comme les notifications ou l'apprentissage automatique.