Plateforme de Streaming en Temps Réel pour la Gestion et l'Analyse d'Utilisateurs

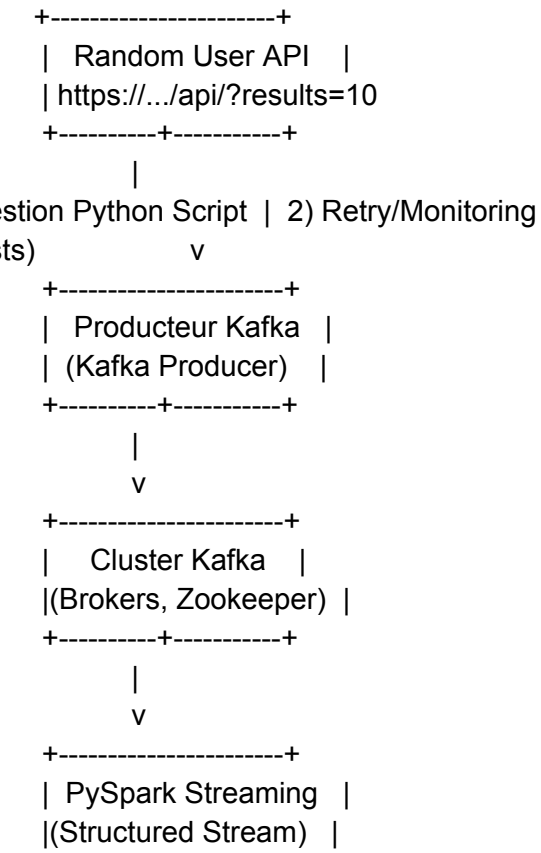
Ce projet propose de créer une pipeline de bout en bout pour ingérer, traiter et visualiser un flux de données en temps quasi réel. En s'appuyant sur l'API Random User pour collecter par lots des profils d'utilisateurs fictifs, vous apprendrez à :

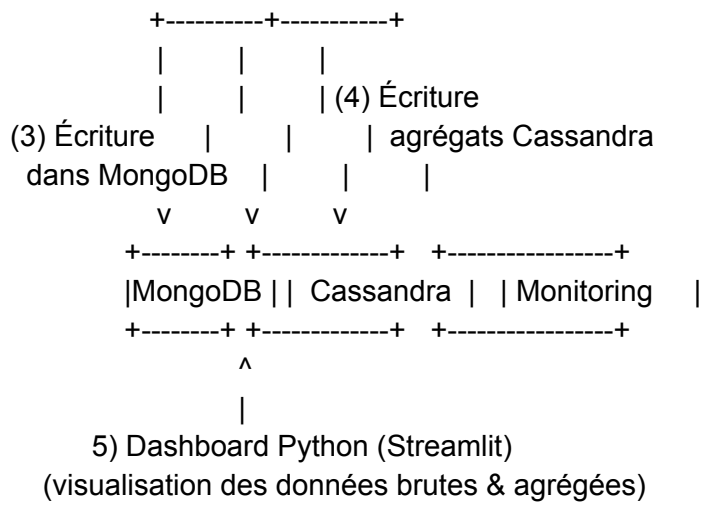
- Déployer et configurer un cluster Kafka pour la réception et la distribution des messages.
- Mettre en œuvre PySpark Structured Streaming afin d'extraire, transformer et agréger les informations (ex. âge moyen, répartition par pays).
- Stocker les données brutes et enrichies dans MongoDB et Cassandra, offrant à la fois de la flexibilité et de la haute performance.
- Créer un tableau de bord Python (Streamlit) pour afficher en continu les derniers utilisateurs et les statistiques agrégées, avec possibilité de représentations graphiques et de rafraîchissement automatique.

L'objectif est de maîtriser les concepts clés du streaming de données (scalabilité, résilience, agrégations en temps réel) et de construire une architecture complète, depuis la collecte en amont jusqu'à la visualisation finale.

---

1) Comprendre l'Architecture Générale





L'architecture se décompose en **cinq** grandes parties :

1. **Producteur (Ingestion)** : Un script Python qui appelle régulièrement l'API Random User (avec `?results=10` pour obtenir 10 utilisateurs par requête).
2. **Kafka** : La plateforme de streaming qui reçoit ces lots de données et les met à disposition de consommateurs.
3. **PySpark Streaming** : Un job de streaming qui consomme les messages de Kafka, effectue des **transformations** et des **agrégations** (par pays, etc.), et envoie les résultats vers MongoDB et Cassandra.
4. **Bases de données** :
    - **MongoDB** pour stocker les données brutes, de manière flexible (format document).
    - **Cassandra** pour stocker à la fois les données « transformées » (table des utilisateurs) et les agrégats (table des statistiques par pays).
5. **Dashboard** : Un outil (ex. Streamlit) qui va interroger Cassandra pour :
    - Afficher les **derniers utilisateurs** arrivés (vue brute).
    - Afficher des **statistiques agrégées** (comptage par pays, âge moyen, etc.) et des **visualisations** (graphiques).

---

2) Mise en Place de Kafka et du Topic

1. **Installation** : Vous avez besoin d'installer Kafka (ou de l'exécuter via Docker).
2. **Démarrage** : Lancez Zookeeper (généralement nécessaire en local) puis Kafka.
3. **Topic** : Créez un topic (ex. `random_user_data`), qui va recevoir les messages JSON envoyés par le producteur.

Pourquoi Kafka ?

- Kafka agit comme un "bus" permettant de **découpler** la source de données (le script Python) de la partie traitement (PySpark).

- Il gère automatiquement la rétention des messages et l'évolutivité (vous pouvez multiplier les partitions si le volume de données augmente).

---

3) Le Producteur Python (Ingestion en Batch)

1. **Appel de l'API** :
   - Utilisez la bibliothèque `requests` pour faire un `GET` sur l'URL : `https://randomuser.me/api/?results=10`.
   - Cela renvoie un **JSON** contenant un tableau `results` de 10 utilisateurs.
2. **Envoi vers Kafka** :
   - Récupérez la réponse JSON.
   - Sérialisez-la (souvent en JSON) et envoyez-la dans le topic Kafka en tant que "message".
3. **Boucle de Récupération** :
   - Faites une boucle pour appeler l'API toutes les X secondes (ex. 10s).
   - Si l'API est indisponible, implémentez une logique de **retry** (ex. attendre 60s avant de réessayer).

Détails à surveiller

- **Fréquence d'envoi** : De 10 à 15 secondes suffit pour tester un flux régulier.
- **Gérer les erreurs** : Vérifier les codes HTTP et traiter les exceptions `requests.exceptions`.
- **Message** : Chaque message contiendra **un** batch de 10 utilisateurs (pas 1, mais 10), ce qui augmente le débit de données injectées.

---

4) Traitement Streaming avec PySpark

C'est la partie la plus importante, car vous allez :

1. **Vous connecter** au topic Kafka pour lire en continu les messages.
2. **Parser** chaque message JSON.
3. **« Exploser » le tableau** de 10 utilisateurs (car chaque message contient plusieurs utilisateurs).
4. **Enrichir** chaque utilisateur (ex. créer un champ `full_name`, extraire `age`, `country`, etc.).
5. **Calculer des agrégations** (par pays, par genre, etc.).
6. **Écrire** les données transformées dans deux cibles différentes : MongoDB et Cassandra.

a) Connexion à Kafka

- Configurez PySpark Structured Streaming pour lire depuis Kafka : en général, via `.format("kafka")` + options telles que `.option("subscribe", "random_user_data")`.
- Précisez l'option `startingOffsets` (souvent `latest` pour ne pas relire tout l'historique).

b) Schéma des Données

- Les données Random User contiennent un champ `results` qui est un **tableau**.
- Chaque "élément" du tableau a diverses sous-structures (ex. `name.first`, `location.country`, `dob.age`, `login.uuid`, etc.).
- Définissez (ou inférez) un **schéma** Spark qui reflète la structure JSON.

c) Explosion du Tableau

- Après avoir converti la colonne "value" (string JSON) en structure (`from_json`), vous obtenez quelque chose comme `data.results`, qui est un **Array** d'objets.
- Utilisez la fonction Spark `explode` pour transformer cet array en plusieurs lignes (1 ligne par utilisateur).

d) Enrichissements

- Créez un champ `full_name` (concaténation de `first_name` et `last_name`).
- Extrayez l'`age` depuis `dob.age`.
- Récupérez le `country`, le `gender`, etc.
- Ajoutez un champ `ingestion_time` (par exemple via `current_timestamp()`) pour savoir quand l'utilisateur est arrivé dans le pipeline.

e) Calcul des Agrégations

- Dans la même étape (ou dans un foreachBatch), faites un `groupBy("country")` pour calculer :
  - Le **nombre d'utilisateurs** par pays (ex. `count(*)`).
  - L'**âge moyen** par pays (ex. `avg("age")`).
- Vous pouvez faire des agrégations plus poussées, par exemple par genre, ou un top N pays.
- Stockez ces résultats de manière à pouvoir les visualiser facilement (par exemple, une table Cassandra où `country` est la clé primaire, et vous mettez à jour `count_users`, `avg_age`, `last_update`, etc.).

f) Écriture dans MongoDB et Cassandra

- **MongoDB** :
  - Stockez toutes les données brutes (ou enrichies) pour avoir un historique complet.

    - Avantage : vous pouvez facilement requêter en JSON, faire des analyses ponctuelles, etc.
- **Cassandra** :
    - Créez une table pour stocker **chaque utilisateur** (ex. `random_user_table`) avec une clé primaire `(user_id, ingestion_time)`.
    - Créez une table pour les agrégations (ex. `country_stats`) où chaque ligne correspond à un **pays**, avec `count_users`, `avg_age`, etc.
    - Lors de l'écriture, vous ferez de l'"upsert" (Cassandra met à jour la ligne si la clé primaire existe déjà).

---

5) Mise en Place des Bases de Données

a) MongoDB

- Vous pouvez créer une base `randomUserDB` et, dans votre job Spark, paramétrer la cible pour écrire dans `randomUserCollection`.
- Vous n'avez pas forcément besoin de créer manuellement la collection au préalable : **Mongo** la créera si elle n'existe pas quand vous faites un insert.

b) Cassandra

- Dans la console `cqlsh`, créez un **keyspace** (ex. `random_user_keyspace`) avec un `replication_factor=1` (pour du test).
- Créez :
    1. **`random_user_table`** pour stocker chaque utilisateur (brut/enrichi).
    2. **`country_stats`** pour les agrégations par pays.
- Vérifiez bien la structure des champs (types "text", "int", "double", etc.), et la **clé primaire**.

---

6) Construction du Dashboard (ex. Streamlit)

a) Récupération des Données Brutes

1. Connectez-vous à Cassandra via le driver Python (`cassandra-driver`).
2. Dans la table `random_user_table`, vous pouvez faire un `SELECT * ORDER BY ingestion_time DESC LIMIT 10` pour prendre les 10 derniers utilisateurs arrivés.
3. Affichez les résultats dans un tableau ou un simple listing.

b) Lecture des Statistiques Agrégées

1. Dans la table `country_stats`, vous avez, pour chaque pays, le `count_users`, `avg_age`, etc.
2. Récupérez ces données et convertissez-les en DataFrame (ou en dictionnaire) pour Streamlit.
3. Affichez un bar chart du **nombre d'utilisateurs par pays**, un graphe de l'**âge moyen** par pays, etc.

c) Rafraîchissement en Continu

- Dans Streamlit, vous pouvez utiliser une boucle ou un "timer" qui va relancer la requête toutes les X secondes.
- Il est également possible d'employer `st.experimental_rerun()` pour forcer un rechargement de la page.

---

7) Stratégies de Monitoring et de Résilience

1. **Monitoring Kafka** :
    - Vérifiez que le topic reçoit bien les messages (Kafka CLI ou outils de monitoring).
    - Assurez-vous que votre job Spark ne rencontre pas d'erreurs de lecture.
2. **Monitoring Spark** :
    - Consultez le Spark UI pour voir si le streaming fonctionne (pas d'erreurs dans les micro-batchs).
3. **Retry & Alertes** :
    - Producteur : si l'API Random User est indisponible, loggez l'erreur et attendez avant de réessayer.
    - Consommation : si le job Spark plante (ex. si Cassandra est hors ligne), vous pouvez gérer des tentatives ou des alertes (logs, e-mail, etc.).

8) Plan de Gouvernance des Données

Pour vous assurer que les données circulent de manière fiable et sécurisée tout au long du pipeline, vous mettrez en place les actions suivantes :

1. **Classification et Politiques de Données**
    - Identifiez les types de données traitées (ex. informations personnelles, localisation, etc.).
    - Classez-les selon leur sensibilité (ex. public, interne, confidentiel).
    - Définissez des politiques (data policies) adaptées à chaque niveau de sensibilité : restriction d'accès, anonymisation si nécessaire, etc.
2. **Rôles et Responsabilités**
    - **Data Owner** : Personne responsable de la définition des règles métier et du niveau de qualité requis.

    - **Data Steward** : Garant de la mise en application des politiques, du suivi des indicateurs de qualité et du respect des normes.
    - **Data Engineer** : Chargé de la partie technique (ingestion, transformations, stockage, sécurité technique).
3. **Qualité et Validation des Données**
    - Mettez en place des contrôles de qualité dans le pipeline PySpark (vérification de types, champs obligatoires).
    - Gérez les anomalies (données incomplètes ou incohérentes) : loguez-les dans une table “anomalies” (MongoDB, par exemple) pour analyse et correction.
    - Instaurez des métriques de qualité (taux de complétude, duplication, etc.) remontées régulièrement.
4. **Sécurité et Conformité**
    - Restreignez l’accès aux tables Cassandra ou aux collections MongoDB via des rôles et des permissions adaptées (Auth, TLS si nécessaire).
    - Si les données contiennent des informations sensibles (p. ex. email personnel), assurez-vous de respecter la réglementation en vigueur (par ex. RGPD), notamment en gérant la suppression ou l’anonymisation si requis.
5. **Documentation et Mise à Jour**
    - Formalisez les politiques et procédures dans un document accessible à toutes les parties prenantes.
    - Prévoyez des revues périodiques pour adapter la gouvernance en fonction de l’évolution des données et des exigences légales ou métiers.

Modalités pédagogiques

Travail individuel
Temporalité :5 jours
Période : Du 03/03/2025 au 07/03/2025 à 18h00

Modalités d'évaluation

Mise en situation
Code review
Culture du projet
Archetecture et bon pratique du Data engineering

Livrables

Lien vers code source sur GitHub

Critères de performance

Évaluation de la compréhension des objectifs du projet et des besoins techniques.

Collecte de Données :
Capacité à utiliser efficacement l'API pour collecter les données nécessaires.
Vérification de la qualité et de l'intégrité des données.

Traitement en temps réel:
Évaluation de l'efficacité des scripts.
Pertinence des métriques analysées et de la logique d'agrégation.

Stockage des Résultats:
Conception appropriée du schéma MongoDB et Cassandra en fonction des besoins d'analyse.
Efficacité de l'insertion des résultats.

Analyse des Résultats :
Pertinence et profondeur des requêtes from DB pour extraire des insights.

Optimisation et Surveillance :
Améliorations apportées aux scripts pour optimiser les performances.
Mise en place de mécanismes de surveillance pour Kafka, Spark, mongoDB et cassandra.

Gestion des Droits et Conformité RGPD :
Mise en œuvre appropriée des permissions pour l'API.
Documentation complète et à jour des accès et des règles de partage.
Conformité avec les réglementations du RGPD, notamment en ce qui concerne le traitement des données.