Challenge Data Engineer Adot

Le challenge se compose de plusieurs exercices:

- Traitement de données géospatial en Python
- Serving des données géospatial via une API
- Ingestion de données en Streaming
- Data processing

Pour réaliser les 4 exercices suivants, vous êtes libre quant aux choix des versions de Python, Scala, Spark et autres librairies ainsi que sur l'organisation du projet (script Python, docker, fat jar, spark-shell, ...). L'essentiel étant d'avoir votre solution et votre cheminement pour résoudre ces exercices ainsi qu'un README détaillé nous permettant de tester facilement.

La rigueur et la propreté du livrable seront prises en compte. Aussi, pour rappel, le volume de données des exercices suivants ne représente qu'une infime partie de ce que nous ingérons chaque jour, il est donc important de veiller à prendre en compte ces problématiques de volumétrie dans vos réalisations.

Si vous bloquez sur un exercice, ce n'est pas grave, pensez simplement à bien documenter tout ce que vous avez réalisé dans le cadre de l'exercice, code même si non fonctionnel, vos recherches, les erreurs ou encore les points de blocage.

Pour simplifier le processus de rendu, il vous est demandé de nous partager un **repository github privé** auquel vous donnez accès aux usernames suivant:

- maxpoulain
- Tyron2308

1. Traitement de données geospatial en Python

L'objectif est d'insérer un fichier de type Geojson contenant les parcelles du département 75 dans une base de données Postgres/Postgis. Puis de relier des events (impressions publicitaires et/ou clics) à la parcelle la plus proche.

Données:

Vous disposez de 2 fichiers:

- **events.csv** contenant les informations sur les events avec les colonnes suivantes:
  - type: le type d'event
  - lat: latitude de l'event
  - lon: longitude de l'event
  - ts: timestamp en millisecondes

- **cadastre-75-parcelles.json** contenant toutes les parcelles du département 75 avec les champs suivant:
  - Type: type de geometry
  - Id: id de la parcelle
  - Geometry:
    - Type: type de geometry
    - Coordinates: coordonnées de la géométrie de la parcelle
  - Properties:
    - Id: id de la parcelle
    - Commune: code insee de la parcelle
    - Prefixe: prefixe de la parcelle
    - Section:section de la parcelle
    - Numero: numero de la parcelle
    - Contenance: aire de la parcelle
    - Arpente: information sur l'arpentage d'une parcelle
    - Created: creation de la parcelle
    - Updated: mise à jour de la parcelle

Exercice:

Comme indiqué, l'exercice doit être réalisé en Python.

Les différentes étapes à réaliser sont :

1. Lancer la base de données Postgres/Postgis en utilisant l'image suivante: postgis/postgis (https://registry.hub.docker.com/r/postgis/postgis/)
2. Créer une table cadastre avec le schéma suivant:

        i. Id: varchar
        ii. Commune: varchar
        iii. Geometry: geometry
3. Lire le fichier “cadastre-75-parcelles.json”
4. Insérer les données dans la base de données
5. Relier chaque event à la parcelle la plus proche. Le format attendu est un dictionnaire comme suivant dans un fichier:
    a. {“id_parcelle1”: {“imp”: <NB_EVENT>, “clic”: <NB_EVENT>}, “id_parcelle2”: {“imp”: <NB_EVENT>, “clic”: <NB_EVENT>}}

2. Serving des données géospatial via une API

Le but est de créer une API en Python (et optionnellement en Scala) qui permettra de “servir” les données géospatial ingérées dans la base de données dans l'exercice n°1.

Exercice

L’API doit implémenter les 4 routes suivantes :

1. Une route **GET** qui renvoie toutes les informations de la parcelle la plus proche à partir d’une latitude et d’une longitude données en “query parameter”. (Ex: /parcelle?lat=<LAT>&lon=<LON>)

2. Une route **POST** qui permet d’insérer une nouvelle parcelle dans la base de données. (ex: /parcelle/<ID> et en body les informations de la parcelle Commune & Geometry)

3. Une route **DELETE** qui permet de supprimer une parcelle de la base de données. (ex: /parcelle/<ID>)

4. Une route **PATCH** qui permet d'updater une ou plusieurs colonnes d’une parcelle dans la base de données. (ex: /parcelle/<ID> et en body les informations de la parcelle à mettre à jour)

3. Ingestion de données en Streaming

Le but est de créer une application qui utilisera Spark Streaming afin de consommer un topic Kafka. Dans le cadre de cet exercice nous allons ingérer les données de visites de site web de plusieurs utilisateurs. Ainsi, une ligne correspond à une visite.

Données:

Voici un exemple de message qui sera envoyé à Kafka:

{"ssp": "smart", "id": "f357f5b4-c62d-4448-b106-d71204516427", "deviceOs": "AND", "c": "FR", "supplyType": "SITE", "supplyTag": {"tags": [4], "domain": "20minutes.fr"}, "consentMetadata": {"userConsent": true}, "bidderVersion": "3.21", "ts_no_bid": 1655986657000}

Voici la description des champs:

- ssp: supply side demand
- id: identifiant de l'utilisateur
- deviceOs: os du device de l'utilisateur
- c: pays de provenance
- supplyType: type du "supply" (SITE ou APP)
- supplyTag:
    - tag: tag du "supply" (voir exercice 3)
    - domain: nom du domaine du "supply"
- consentMetadata:
    - userConsent: consentement de l'utilisateur à l'utilisation des données
- bidderVersion: version du bidder
- ts_no_bid: timestamp en millisecondes

La totalité des données est contenue dans le fichier **bidrequests_exercice.json**.

Vous pourrez, par exemple, envoyer les données dans Kafka à l'aide de la commande suivante:

```
while read line; do echo "$line"; sleep 1; done < bidrequests_exercice.json |
bin/kafka-console-producer.sh
--topic <topic_name> --broker-list <broker_list>
```

Exercice

Les données doivent être traitées en Spark Streaming par batch de 10 secondes, nettoyées (cf ci-dessous) et le résultat écrit dans un fichier au format Parquet.

Le cœur du projet doit être en Spark/Scala.

Processing des données:

- Les messages dont le champ “userConsent” est à False ne doit pas être conservé car l’utilisateur n’a pas donné son consentement pour l’utilisation de ces données.

- Les champs “nested” doivent être flatten selon la nomenclature suivante:
    - consentMetadata: { ‘userConsent’ : <VALUE> } doit devenir dans le fichier final “consentMetadata_userConsent”
    - De même pour les champs “supplyTag_tags” & “supplyTag_domain”

- Les colonnes/champs suivant devront être créés:
    - **“date”** au format yyyy-mm-dd doit être créé à partir du timestamp en millisecondes (“ts_no_bid”)(exemple: 2021-09-21).
    - **“y”** au format yyyy doit être créé à partir du timestamp en millisecondes (“ts_no_bid”)(exemple: 2021).
    - **“m”** au format mm doit être créé à partir du timestamp en millisecondes (“ts_no_bid”)(exemple: 09).
    - **“d”** au format dd doit être créé à partir du timestamp en millisecondes (“ts_no_bid”)(exemple: 21).
    - **“h”** au format HH doit être créé à partir du timestamp en millisecondes (“ts_no_bid”)(exemple: 09 pour 9h).

- Ainsi le schéma du fichier parquet attendu est le suivant:

    - ssp
    - id
    - deviceOs
    - c
    - supplyType
    - supplyTag_tags
    - supplyTag_domain
    - consentMetadata_userConsent
    - bidderVersion
    - ts_no_bid
    - date
    - y
    - m
    - d
    - h

4. Data processing

Après avoir ingéré la donnée pour chaque utilisateur, nous pouvons utiliser cette donnée pour créer une table agrégé permettant de répondre à un use case business: Connaître l'affinité de chaque utilisateur envers les différents sites web visités.

L'objectif de cet exercice est donc de manipuler de la donnée ingérée pour construire un dataframe agrégé pour chaque utilisateur avec Spark/Scala.

Données:

Les données sont celles qui ont été ingérées et écrites dans un fichier parquet précédemment.

En se basant sur le schéma de sortie des données de l'exercice précédent, nous allons utiliser uniquement les champs suivants:

- id
- supplyTag_tags

Le nom des thématiques est fourni ci-contre:

- Shopping : 1
- Programme tv: 2
- Sport: 3
- News: 4
- Santé 5
- Voyage: 6

Exercice

Pour chaque utilisateur un tag est associé à la thématique du site web visité (ex: Shopping, Sport, Santé, …). Le but est de répondre aux questions suivantes:

- Calculer pour chaque utilisateur un score d'affinité envers les différentes thématiques visitées dans les données.
- Calculer le classement des ssp par deviceOs.
- Calculer le top 1 des "domains" les plus visités par Country.
- Calculer le classement des SupplyType par ssp.

Bonne chance !