Data Engineering

PROJET M2

Université Gustave Eiffel

Table des matières

Introduction

Ce projet fait suite au projet d'IA symbolique, qui sert de support de données.
On démarre donc avec un graphe RDF contenant des données de LUBM, auxquelles sont générées aléatoirement des nouvelles informations sur des personnes (nom, ville, année de naissance, …) ainsi qu'un type de vaccin contre le Covid-19.

L'objectif de ce projet est de rajouter en plus de ces données des effets secondaires sur les personnes vaccinées.
On a donc 2 sources de données dont chaque enregistrement possède un identifiant, qui est partagé entre ces 2 sources. Le but est de joindre chaque paire d'enregistrements dont l'identifiant est le même pour former un enregistrement complet (une personne du LUBM vaccinée avec un effet secondaire).

Pour cela on utilise Apache Kafka qui est un système distribué de messages. On crée des flux de données qui peuvent transformer les messages, les filtrer, les joindre avec d'autres flux etc…
Dans ce projet on crée une topologie de flux dont les deux qui correspondent aux entrées sont les deux sources de données évoquées.

Par la suite on utilisera le flux de sortie pour alimenter une base de données InfluxDB, qui servira de support pour Grafana, un outil de visualisation de données en temps réel.
On peut parler de Metrics By Design, ou Metrics Driven Development, le fait d'agréger des données depuis des applications, de les stocker dans éventuellement un moteur de recherche puis d'en exposer les métriques.

Membres du groupe

- Amany MILED – amiled@etud.u-pem.fr
- Stéphane SANCHEZ FERNANDES – ssanches@etud.u-pem.fr
- Samy WADAN – swadan@etud.u-pem.fr
- Axel ZEMMOUR – azemmour@etud.u-pem.fr

Création des données

Les données proviennent du graphe LUBM, on ajoute avec Java-Faker des données aléatoires mais réaliste (par exemple un étudiant n'a pas plus de 30 ans et un enseignant n'est pas en dessous de 20 ans), la proportion d'homme et de femmes, de vaccins administrés, du nombre de vaccins et de la proportion de chacun d'entre eux est paramétrables dans les classes **FakeData** et **LUBM1person**.

La première source de données, que l'on va appeler partie statique, correspond aux personnes de ce graphe (avec toutes les infos correspondantes). (**LUBM1person**)
La deuxième source de données, que l'on va appeler partie dynamique, correspond à ensemble d'enregistrements générés à partir de chaque personne vaccinée et possédant un code d'effet secondaire. (**SideEffectRecord**).

Topics et Topologie

Un topic est un ensemble de partitions dans lesquels vont circuler les enregistrements.
On démarre ici avec 2 topics, un pour chaque source de données. Le topic correspondant à la partie dynamique contient des informations (prénoms et noms) dont on ne souhaite pas pour anonymiser le flux. On va donc rajouter créer un nouveau topic qui est le même mais où seuls les enregistrements anonymisés circuleront, et relier ces 2 topics avec un nœud de filtrage.

Le topic des données dynamiques anonymisées va ensuite se joindre au topic des données statiques pour ne former plus qu'un topic de sortie.

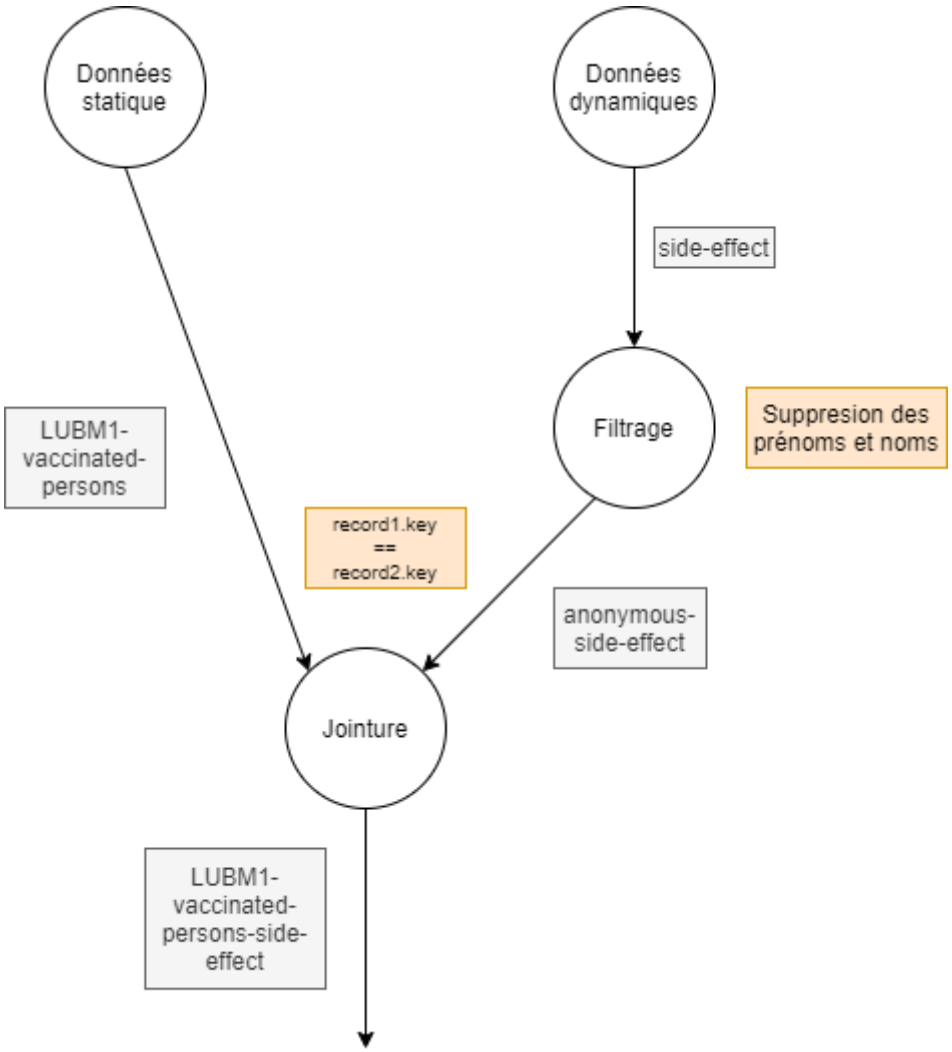

Figure 1 Topologie des flux

Chacun de ces topics possède autant de partitions qu'il y a de vaccins (**Vaccine**). Ainsi on peut associer une partition à un type de vaccin.
Notre groupe de 5 consumers (qui lisent le dernier topic) va donc pouvoir lire les enregistrements qui arrivent et effectuer des statistiques sans se soucier du type de vaccin car ce sera toujours le même dans une même partition.

Sérialisation / Désérialisation

Ce projet propose 2 versions différentes : une version où les enregistrements sont envoyés au format JSON, et une autre *(Git -> branche Kafka-avro)* où les enregistrements circulent sous forme de tableaux d'octets et sérialisés / désérialisés avec un schéma Avro. (Deux schéma pour les 2 types d'enregistrements de ce projet).
La (de)sérialisation JSON s'effectue avec FasterXML.Jackon, et celle avec Avro se fait Bijection de Twitter.
La solution optimale serait d'avoir utiliser l'API de Confluent mais celle-ci nécessite de gérer une Schéma Registry et est plus longue à mettre en place.
Un enregistrement Avro est plus léger qu'un enregistrement JSON et le passage par un schéma permet de beaucoup gagner en maintenabilité. Cependant après des tests il est a noté que la (de)sérialisation avec Avro (en utilisant Bijection) est beaucoup plus longue qu'avec le format JSON : pour environ 850 enregistrements, le temps d'exécution du programme (en passant par toute la topologie) est d'environ 1 seconde avec JSON et environ 20 secondes avec Avro.

Stockage des données dans une base de données

Les enregistrements en sortie du topic final (*LUBM1-vaccinated-persons-side-effect*) sont envoyés dans une base InfluxDB, c'est une base de données noSQL orientée séries temporelles, donc bien adaptée à un système de messages comme Kafka.
L'avantage d'utiliser cette BDD est l'outil Telegraf qu'elle propose. C'est un agent qui va lire une entrée et envoyer les données à la base.
Dans notre cas, un plugin *Kafka-consumer* existe et permet à l'agent de lire un topic et d'envoyer à intervalle régulier les messages dans la BDD, comme un consumer Kafka.

InfluxDB expose les concepts de Bucket, Tag et Field. Sans détailler un bucket est un espace disque dans lequel on stocker nos enregistrements, qui contiennent des fields et éventuellement des tags. On peut voir un Bucket comme une table en SQL, un tag comme un champ indexé et un field comme un champ non indexé.
Pour des raisons de simplicité, dans ce projet chacun des champs de nos enregistrements Kafka sont des tags (dont indexés, on peut voir ça comme des colonnes).

InfluxDB propose également une interface dans lequel il est possible d'afficher nos données avec des graphiques (donc on peut même remettre en question l'utilisation de Grafana).
Depuis la version 2 d'InfluxDB (celle utilisée ici) le langage de requête est Flux, qui est une sorte de langage basé sur des pipes et des fonctions.

Grafana

La dernière étape de ce projet est de proposer une visualisation des données, avec Grafana.
L'outil a besoin d'un support pour aller chercher ses données, on utilise pour ça notre base InfluxDB, et avec son langage de requêtes, on prépare des données qui seront affichées sur des graphiques par exemple.

Malheureusement notre manque de maitrise de Grafana et le manque de temps ne nous a pas permis d'afficher de beaux graphiques, mais nous avons quand même un dashboard avec des tableaux, alimentés

et mis à jour en temps réel (il suffit d'exécuter le programme Kafka, qui déclenchera le service Telegraph, exportant les données vers InfluxDB, puis lu par Grafana).

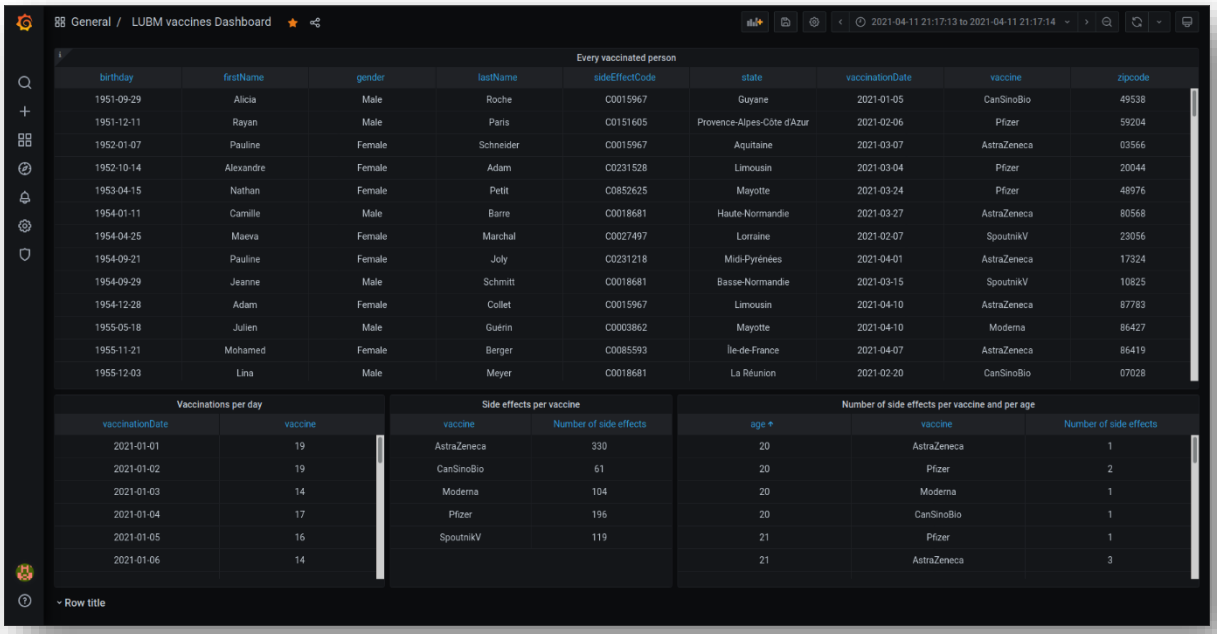

Figure 2 Dashbord Grafana

***Note :** Le rendu ne contient rien en ce qui concerne la mise en place d'InfluxDB et de Grafana, car ce sont des installations locales. Si besoin je peux faire une vidéo de présentation de toute cette mise en place, en incluant les fichiers de configurations.*

Utilisation du programme

Il est nécessaire de lancer Zookeeper avant d'exécuter le programme.

L'application utilise l'API Kafka Admin pour créer automatiquement les topics avec le nombre correspondant de partitions.

Le programme consiste à générer les données, les envoyés dans les topics créés, et à les consumer dans une boucle infinie, chaque consumer dans un thread séparé.
La console affiche les enregistrements consumés. Saisissez n'importe quelle valeur avec Entrer dans la console pour tuer les threads et arrêter le programme.