Définition du processus de batch

Context :

Le processus de batch consiste à extraire les données existantes de MariaDB et à les insérer dans ElasticSearch.

Méthode de batch initialement imaginée :

Extraire la donnée de chaque table pour chaque base de données présente dans MariaDB et l'insérer dans la nouvelle base de données.

Identification du problème :

MariaDB est une Base de données relationnelle, l'information y est donc répartie en plusieurs tables et reliée par la présence de clés. Ce n'est pas le cas dans ElasticSearch qui est une base de données orientée document dans laquelle les données sont stockées en une seule et même place (absence de relation). Cette absence de relation et la nécessité de stocker la donnée dans un même document **contraint à joindre les tables préalablement.**

Nous détaillons ci-après les solutions imaginées pour l'extraction et l'insertion des données existantes ainsi que les avantages et inconvénients de chacune d'entre elles. La solution B (connecteur JDBC configuré dans Logstash) est présentée à titre informatif mais est définitivement exclue du choix au vu des risques qu'elle présente.

Solutions :

A. Réalisation d'un script en python
B. Utilisation d'un connecteur JDBC avec Logstash
C. Utilisation d'un connecteur JDBC avec Kafka Connect
D. Bilan et justification du choix

Mots-clés :

Connecteur JDBC (Java DataBase Connectivity) : Le connecteur d'événement JDBC reçoit les résultats de requêtes SQL reçues en interrogeant les bases de données relationnelles définies comme connexion. Le connecteur d'action JDBC envoie des actions de modification des tables de base de données relationnelle en utilisant des instructions SQL de base (SELECT, INSERT, UPDATE, DELETE).

Documentation IBM - Connecteur JDBC

Logstash : Outil visant à simplifier l'insertion de données dans ElasticSearch.

Site Elasticsearch - Logstash

Définition du processus de batch

A. Réalisation d'un script en python

Le script en python comprendrait :

- la connexion à MariaDB ;
- la détection des Bases de données présentes ;
- la détection des tables dans chaque bases de données ;
- l'extraction des données de chaque table*[1] ;
- la jointure des données récupérées ;
- (la division des enregistrements en groupe réduits pour l'insertion) ;
- la conversion des données au format ndjson ;
- la connexion à ElasticSearch ;
- l'insertion des données dans ElasticSearch.

Le principal avantage de cette solution est sa simplicité d'implémentation. Elle nécessite pas de configurer des services supplémentaires. L'inconvénient est lié à l'éventuelle complexité du code à réaliser qui est à considérer mais ne doit pas pour autant conduire à rejeter cette solution.

*[1] l'extraction ne devra concerner que les enregistrements antérieurs à la mise en place du 'streaming'. Cela implique la mise en place d'une cause WHERE sur l'identifiant de la table mise à jour. Cette identifiant pourra dans le meilleur des cas être saisi dans un fichier lors de la mise en place de la pipeline de streaming afin d'être lu par le script ou devra être récupéré « manuellement » en allant constater dans Elastic la première valeur saisie.

B. Utilisation d'un connecteur JDBC avec Logstash

Guide Elastic pour la configuration d'un connecteur JDBC dans logstash

Cette option consiste à simplement configurer un connecteur JDBC dans logstash. Cette solution est plus simple à mettre en place car elle nécessite d'implémenter un connecteur de cette manière :

```
input {
  jdbc {
    jdbc_driver_library => "mysql-connector-java-5.1.36-bin.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://localhost:3306/mydb"
    jdbc_user => "mysql"
    parameters => { "favorite_artist" => "Beethoven" }
    schedule => "* * * * *"
    statement => "SELECT * from songs where artist = :favorite_artist"
  }
}
```

On aura donc un code de ce type pour chaque Base de données à transférer ; on modifiera la valeur du statement afin qu'elle exécute la requête SQL souhaitée en fonction de la base de données que l'on cherche à transférer.

Le principal avantage provient de la simplicité de mise en place : un code par base de données et l'insertion dans ElasticSearch est prise en charge par Logstash Il n'est donc pas nécessaire de s'interroger sur la structure à donner à la donnée. Qui plus est cela permettra de toujours tenir à jour ElasticSearch (synchronisation d'ElasticSearch avec les données présentes dans MariaDB).

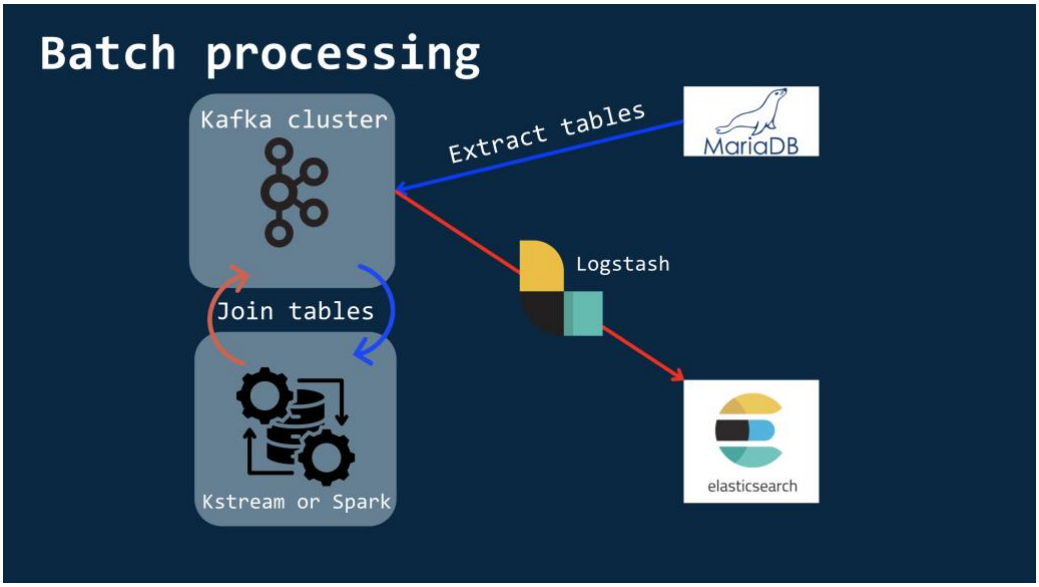
L'inconvénient majeur et non négligeable provient du problème énoncé au départ : étant donné qu'il faut joindre les tables, cela va solliciter MariaDB outre mesure ce qui risque d'être très long voire de ne pas aboutir et potentiellement rendre MariaDB « injoignable ».

Ce problème nous conduit à rejeter la solution B et à étudier la solution C.

C. Utilisation d'un connecteur JDBC avec Kafka Connect

La troisième solution étudiée consiste à utiliser un connecteur JDBC mais par l'intermédiaire de Kafka cette fois et non plus de Logstash. La donnée de chaque table est extraite de MariaDB par l'intermédiaire du connecteur et est envoyée à Kafka. Cette donnée peut ensuite être récupérée par un outil de traitement des données comme Kafka stream ou Apache Spark. Les opérations de jointure seraient réalisées avec ces outils sans conséquence pour MariaDB qui n'est plus sollicitée qu'avec des requêtes SQL du type « SELECT * FROM <TABLE> WHERE … » bien moins gourmandes en termes de ressources. La donnée une fois regroupée est ensuite renvoyée dans Kafka puis insérée dans ElasticSearch par l'intermédiaire de Logstash.

Ci-dessous un schéma de la solution décrite :



Cette solution présente de multiples avantages. Tout comme la solution précédente, l'insertion des données par l'intermédiaire de Logstash limite la complexité associée à la structuration de la donnée pour l'insertion. De plus, la pipeline de « streaming » censée transmettre les données de Kafka à Elastic ne serait plus nécessaire car là encore, Elastic pourraient être alimentée directement depuis Mariadb. Contrairement à la solution précédente, cette solution ne sollicite pas MariaDB pour les jointures ce qui garantit une meilleure vitesse d'exécution des requêtes et assure de ne pas surpasser les capacités de MariaDB.

Les inconvénients de cette solution sont :

- la relative complexité de l'architecture associée ;
- la nécessité de configurer un outil de traitement de façon anticipée ;
- et la multiplication des échanges de données entre les différents services qui augmenteraient les conséquences de la solution sur l'environnement.

Définition du processus de batch

D. Bilan et justification du choix

	Script python	JDBC - Kafka	JDBC - Logstash
Complexité du code	Intermédiaire - Elevée	Basse - Intermédiaire	Basse
Implémentation	Faible	Elevé	Intermédiare
Architecture	« Aucune »	Complexe	Simple
Faisabilité			Risqué
Automatisation	Jointure (eventuellement clause WHERE)	Jointure « à la main »	Jointure « à la main »
Eco-conception		X	

Nous suggérons de réaliser la pipeline de transfert des données de Mariadb à ElasticSearch par l'intermédiaire d'un script en python.

La solution avec Logstash est rejetée ; La solution incluant Kafka est plus complexe en termes d'architecture finale et présente des conséquences plus élevées sur l'environnement.

En ce qui concerne l'automatisation chacune des solutions implique de réaliser « à la main » la requête SQL pour la jointure des tables et ce pour chacune des bases de données à transférer. La solution avec Kafka gère la synchronisation entre les Bases de données par l'intermédiaire du connecteur JDBC, il ne serait pas nécessaire de mettre en place une clause « WHERE » pour récupérer uniquement les enregistrements antérieurs à la mise en place de la pipeline de streaming puisque tout serait réalisé par l'intermédiaire du connecteur. Le script python peut automatiser cela mais uniquement si un fichier contenant la valeur de la clé est renseigné lors de la mise en place du streaming.