

Rapport de projet tutoré

Réalisation d'un pipeline Big data pour un système de surveillance du conducteur

Réalisé par:

Nouha ZOUAGHI & Wacef CHACHIA

Supervisé par:

Mr. Mohamed AYADI & Mr. Alaeddine AZIZ

Introduction générale

Le calcul des tarifs des assurances des véhicules continue à être une tâche très délicate pour les assureurs et les payeurs. En effet depuis son apparition, les modèles de tarification utilisés n'ont pas cessé d'évoluer.

Dans ce contexte se situe ce projet qui consiste à développer une plateforme qui permet d'aider au calcul de tarification des assurances et décompensation des bons conducteurs.

Ce projet tutoré est élaboré en coopération avec la société Hydati dont le domaine d'activités est la science des données et le développement des solutions pour les comptes des banques et des assurances.

La plateforme consiste tout d'abord à concevoir et fabriquer un dispositif de surveillance des conducteurs à l'aide des objets connectés.

Ce dispositif permet de collecter des informations qui vont être stockées dans un espace Cloud et utilisées pour élaborer des modèles d'apprentissage pour le calcul d'un score servant à la fixation de la prime optimale à payer.

Une interface Web mobile sera développée pour assurer le suivi, la configuration ainsi que la consultation.

Table de matières

I.	Contexte général.....	5
	I.I. Présentation de la société.....	5
	I.II. Description du projet.....	5
	I.III. Les tâches du projet.....	5
II.	Concepts de base.....	6
	II.I. Le pipeline de données.....	6
	II.II. Kafka.....	6
	II.III. Spark.....	7
	II.IV. PostgreSQL.....	7
	II.V. Thingsboard.....	7
III.	Le cas pratique.....	8
	III.I. Introduction.....	8
	III.II. Étapes du travail.....	8
VI.	Conclusion.....	12

Table de figures

figure 1: logo Hydatis

figure 2: le pipeline big data

figure 3: logo de kafka

figure 4: les composantes de Apache Spark

figure 5: logo de PostgreSQL

figure 6: l'interface de thingsboard

figure 7: plugin thingsboard kafka

figure 8: règles de transfert de données

figure 9: Fonction tqui stocke les données dans PostgreSQL

I. Contexte général

I.I. Présentation de la société

Hydatis est une entreprise qui cherche à concevoir des solutions qui rendent ses clients plus innovants et productifs, des solutions qui changent leurs usages et leur apportent un réel avantage concurrentiel, mais aussi et surtout des solutions qui respectent les personnes et l'environnement dans lequel elles vivent.



Figure1 : logo Hydatis

I.II. Description du projet

L'Internet des objets est le phénomène de mode actuel, mais avec un grand nombre d'appareils intelligents générant une énorme quantité de données, il serait idéal d'avoir un système de big data détenant l'historique de ces données.

C'est ce qu'on va réaliser dans ce projet qui était proposé par l'entreprise Hydatis.

Il s'agit principalement d'un pipeline big data qui va stocker et traiter la quantité énorme de données générées par des capteurs, situés dans une véhicule.

Ces capteurs mesurent plusieurs paramètres tel que la vitesse, l'ivresse etc..

Ces données vont être stockés afin de fabriquer un dispositif de surveillance des conducteurs qui attribue un score à chaque conducteur servant à la fixation d'une prime optimale à payer.

I.III. Les tâches du projet

Dans un premier temps, on s'est documenté sur les architectures des pipelines big data afin de déterminer l'architecture la plus adéquate à notre projet.

Puis, on a fait quelques petites manipulations sur la plateforme IoT thingsboard pour se familiariser avec elle.

Enfin, on a commencé à développer le pipeline big data en utilisant Java comme langage de programmation.

II. Concepts de base :

II.I. Le pipeline de données :

Un pipeline de données englobe une série d'actions qui débute avec l'ingestion de l'ensemble des données brutes issues de n'importe quelle source, pour les transformer rapidement en données prêtes à être exploitées.

On a implémenté un pipeline qui va stocker et traiter des données en temps réel générés par la plateforme IoT thingsboard.

Ce pipeline big data est composé de kafka et spark streaming et postgresSQL comme le montre cette figure.

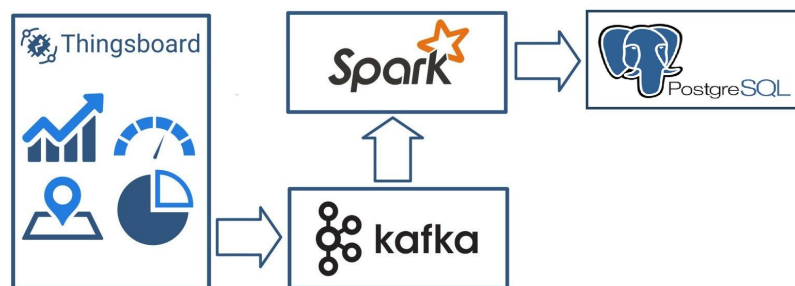


figure2: le pipeline big data

Dans la suite, on va parler des composantes principales du pipeline.

II.II. Kafka :

Apache Kafka est [une plateforme de streaming de données open source](#).

Cette plateforme est utilisée pour traiter des données volumineuses à cause de sa capacité à exécuter des traitements simultanés et à déplacer de grandes quantités de données rapidement.

Les composantes principales de Kafka sont :

- L'événement: C'est un message contenant des données.
- Le producteur (publisher): Il écrit des événements et les transmettent à Kafka (il peut s'agir d'un appareil IoT, un serveur web etc.)
- Le consommateur (subscriber): Il reçoit les données écrites par le producteur et les utilise.

En résumé, Les producteurs [publient les événements sur des " topics "](#) Kafka. Les consommateurs peuvent s'abonner pour obtenir l'accès aux données dont ils ont besoin. Les topics sont des séquences d'événement, et chacun peut servir des

données à de multiples consommateurs. Ainsi, les producteurs sont parfois appelés " publishers " et les consommateurs " subscribers " .



figure3 : logo de kafka

II.III. Spark :

Apache Spark est un moteur d'analyse unifié et ultra-rapide pour le traitement de données à grande échelle. Il permet d'effectuer des analyses par le biais de machines de Clusters.

Spark Streaming a étendu le concept d'Apache Spark de traitement par lots en streaming en décomposant le flux en une série continue de mini batches, qui pouvaient ensuite être manipulées à l'aide de l'API Apache Spark.

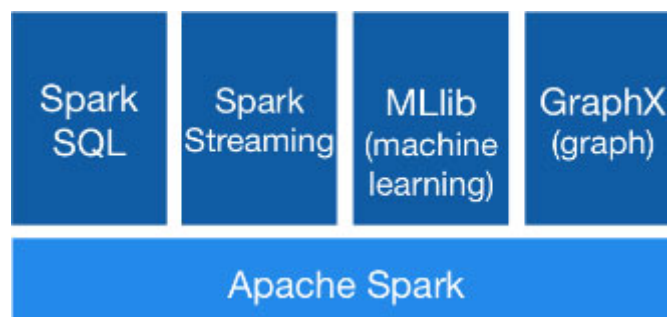


figure4: les composantes de Apache Spark

II.VI. PostgreSQL :

PostgreSQL est un SGBDR (système de gestion de base de données relationnelle) fonctionnant sur des systèmes de type UNIX.



figure3 : logo de PostgreSQL

II.V. Thingsboard :

ThingsBoard est une plateforme IoT open-source qui permet le développement, la gestion et la mise à l'échelle rapides de projets IoT.

Elle fournit des solutions IoT en nuage ou sur site qui permettront de mettre en place une infrastructure côté serveur pour les applications IoT.

III. Le cas pratique :

IV.I. Introduction

- Comme expliqué dans la partie précédente, nous avons utilisé 4 composants essentiels : ThingsBoard, Apache Kafka, Spark Streaming, Postgresql.
- Les données envoyées par les capteurs à la plateforme « ThingsBoard » seront routées par les outils « Kafka » et « Spark Streaming » vers une base de données locale « Postgresql » afin qu'elles soient accessibles à l'équipe IA qui traitera ces données afin de pour créer un modèle de calcul des scores des différents pilotes et de leurs récompenses.
- Les principales tâches réalisées :
 - Acheminez les données de l'appareil de télémétrie de «Thingsboard» vers «Kafka topic» à l'aide du plug-in intégré.
 - Agréger les données de plusieurs appareils à l'aide d'une simple application "Apache Spark".
 - Pousser les résultats vers la base de données locale "Postgresql".
- Dans le cas pratique nous avons utilisé 2 capteurs, le premier mesure la vitesse du véhicule, le deuxième mesure la température de l'air. Nous pouvons ajouter tout autre capteur pouvant rendre des informations utiles à notre modèle (nombre d'heures de conduite, sommeil, ivresse, etc...).

IV.II. Étapes du travail

- Le travail comporte plusieurs étapes commençant par les installations, les configurations, le développement du code et se terminant par les exécutions et les tests. Nous citerons les tâches principales :

- **Installation de ThingsBoard sur CentOS** : Cette tâche consiste à installer Java, configurer la base de données Postgresql et enfin à configurer Kafka en tant que service de file d'attente.
-La figure ci-dessous montre le lancement du service ThingsBoard localement sur le port 8080.

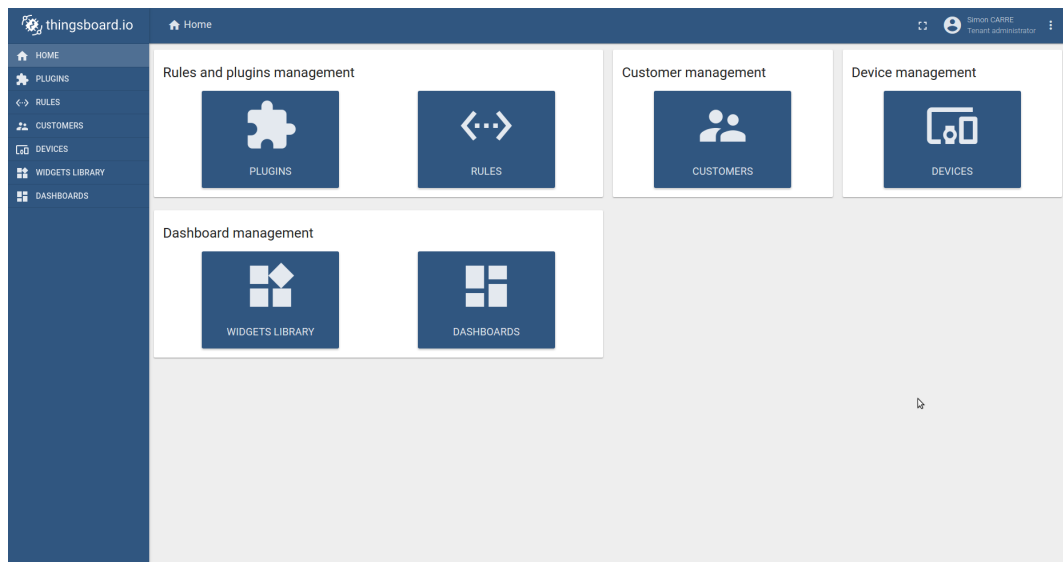


figure4 : l'interface de thingsboard

- **Configuration du plugin Thingsboard Kafka** : Le plugin Kafka sera utilisé pour transmettre les données de télémétrie à Kafka. Il s'attend à ce que Kafka s'exécute sur l'hôte local avec le port 9092.

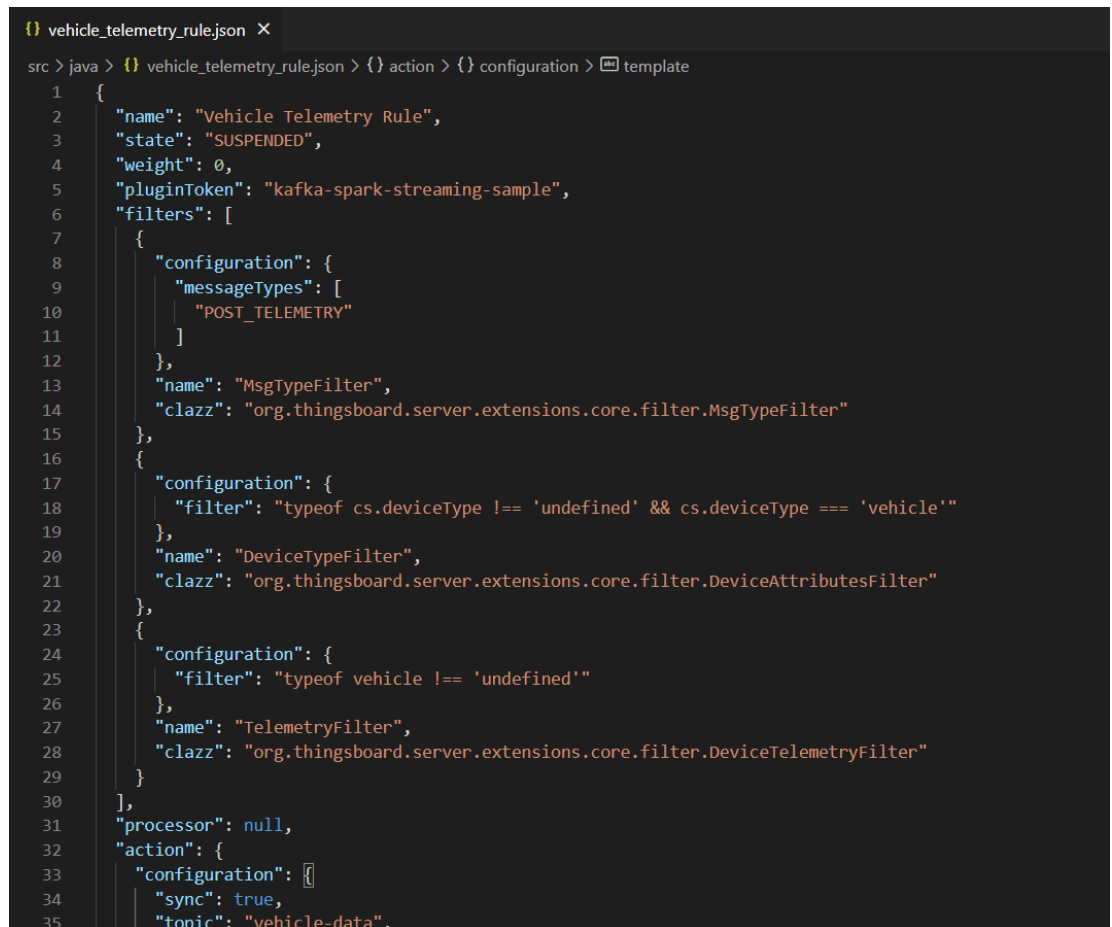
```

kafka_plugin_for_spark_streaming_sample.json ×
src > java > {} kafka_plugin_for_spark_streaming_sample.json > ...
1  {
2    "apiToken": "kafka-spark-streaming-sample",
3    "name": "Kafka Plugin for Spark Streaming Sample",
4    "clazz": "org.thingsboard.server.extensions.kafka.plugin.KafkaPlugin",
5    "publicAccess": false,
6    "state": "SUSPENDED",
7    "configuration": {
8      "bootstrapServers": "localhost:9092",
9      "batchSize": 16384,
10     "bufferMemory": 33554432,
11     "acks": -1,
12     "keySerializer": "org.apache.kafka.common.serialization.StringSerializer",
13     "valueSerializer": "org.apache.kafka.common.serialization.StringSerializer"
14   },
15   "additionalInfo": null
16 }

```

figure5: plugin thingsboard kafka

- **Configuration des règles de transfert de données** : ces règles seront utilisées pour transmettre les données du véhicule du Thingsboard à Kafka.



```

1  {
2    "name": "Vehicle Telemetry Rule",
3    "state": "SUSPENDED",
4    "weight": 0,
5    "pluginToken": "kafka-spark-streaming-sample",
6    "filters": [
7      {
8        "configuration": {
9          "messageTypes": [
10           "POST_TELEMETRY"
11         ]
12       },
13       "name": "MsgTypeFilter",
14       "clazz": "org.thingsboard.server.extensions.core.filter.MsgTypeFilter"
15     },
16     {
17       "configuration": {
18         "filter": "typeof cs.deviceType !== 'undefined' && cs.deviceType === 'vehicle'"
19       },
20       "name": "DeviceTypeFilter",
21       "clazz": "org.thingsboard.server.extensions.core.filter.DeviceAttributesFilter"
22     },
23     {
24       "configuration": {
25         "filter": "typeof vehicle !== 'undefined'"
26       },
27       "name": "TelemetryFilter",
28       "clazz": "org.thingsboard.server.extensions.core.filter.DeviceTelemetryFilter"
29     }
30   ],
31   "processor": null,
32   "action": {
33     "configuration": {
34       "sync": true,
35       "topic": "vehicle-data",

```

figure6: règles de transfert de données

-Ces règles vous permettent de vérifier le type de capteurs à partir desquels les données sont reçues. Ils sont utiles dans le cas de deux ou plusieurs capteurs pour éviter toute confusion entre eux.

-Les principales règles utilisées sont : Attributes filter pour différencier les capteurs, Timeseries data filter qui nous permet de manipuler des sous-ensembles de données, Kafka plugin action.

- **Développement de code** : Notre projet a l'architecture suivante :

```

src ____>kafka_plugin_for_spark_streaming_sample.json
|____>RestClient.java
|____>Sensor1Data.java
|____>Sensor2Data.java
|____>SparkKafkaStreamingDemoMain.java
|____>vehicle_telemetry_rule.json
resources ____>logback.xml
pom.xml

```

Rôle des fichiers : Les rôles dans l'ordre indiqué dans l'architecture :

[kafka_plugin_for_spark_streaming_sample.json](#): Il sera utilisé pour transmettre les données de télémétrie à Kafka

[RestClient.java](#): Cette classe nous permet de nous connecter à la plateforme ThingsBoard, de nous connecter à notre compte utilisateur, envoyer et recevoir des messages avec MQTT, etc...

[Sensor1Data.java](#) et [Sensor2Data.java](#): Ces classes ainsi représentent les deux capteurs que nous allons utiliser et qui nous permettent de cartographier les données reçues du ThingsBoard.

[SparkKafkaStreamingDemoMain.java](#): Cette classe permet de configurer Kafka "Brokers" pour le connecter à "Spark Streaming" et recevoir des messages de données de celui-ci pour l'envoyer à la base de données locale 'Postgresql'.

[vehicle_telemetry_rule.json](#): Il sera utilisé pour transmettre les données du véhicule du ThingBoard à Kafka.

[logback.xml](#): Il enregistre les messages WARN et INFO et filtre les messages DEBUG.

[pom.xml](#): Il contient la majorité des informations nécessaires à la construction d'un projet.

Captures de code :

```
// ThingsBoard server URL
private static final String THINGSBOARD_REST_ENDPOINT = "http://localhost:8080";
// ThingsBoard Create Asset endpoint
private static final String CREATE_ASSET_ENDPOINT = THINGSBOARD_REST_ENDPOINT + "/api/asset";
// ThingsBoard Get Vehicle Assets endpoint
private static final String GET_ALL_TENANT_ASSETS_ENDPOINT = THINGSBOARD_REST_ENDPOINT + "/api/tenant/assets?limit=100&type=Vehicle" ;
// ThingsBoard Publish Asset telemetry endpoint template
private static final String PUBLISH_ASSET_TELEMETRY_ENDPOINT = THINGSBOARD_REST_ENDPOINT + "/api/plugins/telemetry/ASSET/${ASSET_ID}/timeseries/values";
private static final String ASSET_ID_PLACEHOLDER = "${ASSET_ID}";
private static final String VEHICLE = "Vehicle";    ///

// ThingsBoard User login
private static final String USERNAME = "tenant@thingsboard.org";
// ThingsBoard User password
private static final String PASSWORD = "tenant";
```

figure7: RestClient.java : Paramètres utilisées dans ThingBoard

```

// Kafka brokers URL for Spark Streaming to connect and fetched messages from.
private static final String KAFKA_BROKER_LIST = "localhost:9092";
// Time interval in milliseconds of Spark Streaming Job, 10 seconds by default.
private static final int STREAM_WINDOW_MILLISECONDS = 10000; // 10 seconds
// Kafka telemetry topic to subscribe to. This should match to the topic in the rule action.
private static final Collection<String> TOPICS = Arrays.asList("topic_sensor1","topic_sensor2"); //Topics
// The application name
public static final String APP_NAME = "Kafka Spark Streaming App";
// Misc Kafka client properties
private static Map<String, Object> getKafkaParams() {
    Map<String, Object> kafkaParams = new HashMap<>();
    kafkaParams.put("bootstrap.servers", KAFKA_BROKER_LIST);
    kafkaParams.put("key.deserializer", StringDeserializer.class);
    kafkaParams.put("value.deserializer", StringDeserializer.class);
    kafkaParams.put("group.id", "DEFAULT_GROUP_ID");
    kafkaParams.put("auto.offset.reset", "latest");
    kafkaParams.put("enable.auto.commit", false);
    return kafkaParams;
}

```

figure8: SparkKafkaStreamingDemoMain.java : Paramètres Kafka

```

private static void add_data_to_db(String table_name,String id,Double val)
{
    Connection c = null;
    Statement stmt = null;
    try {
        Class.forName("org.postgresql.Driver");
        c = DriverManager.getConnection("jdbc:postgresql://localhost:5432/vehicledb","wacef", "123456");
        c.setAutoCommit(false);
        System.out.println("Opened database successfully");

        stmt = c.createStatement();
        String sql = "INSERT INTO " + table_name + " (id,value) "
            + "VALUES (" + id + " , " + val.toString() + ")";
        stmt.executeUpdate(sql);
        stmt.close();
        c.commit();
        c.close();
    } catch (Exception e) {
        System.err.println( e.getClass().getName()+" : " + e.getMessage() );
        System.exit(0);
    }
    System.out.println("Records created successfully");
}

```

figure9: Fonction tqui stocke les données dans PostgreSQL

-Les données capturées dans les blocs RDD par Spark Streaming sont envoyées à la base de données Postgresql locale qui est lancée sur le port 5432.

IV. Conclusion

Ce projet nous a permis de participer à un projet innovant, acquérir des connaissances dans le domaine du Big Data, découvrir et utiliser de nouveaux outils, travailler en binôme et échanger des idées. Il présente une bonne expérience qui aura un impact sur nos carrières.

