MISE EN PLACE D'UN OUTIL D'ANALYSE DE L'HISTORIQUE DE NAVIGATION DE MOZILLA FIREFOX

PROJET DE FIN DE MASTER 2 BI & BIG DATA

Stephanya CASANOVA MARROQUIN
Encadrée par
OMAR BOUSSAID [1]
1: Laboratoire ERIC, Institut de Communication, Université de Lyon 2, France

M2 Big Data & BI
Université Lumière Lyon II
Lyon - FRANCE
2021

Résumé

Ce travail s'inscrit dans le cadre d'un projet de validation du master en informatique spécialisée en BI & Big Data, sous la responsabilité du professeur Omar BOUSSAID.
L'objectif de ce stage était la mise en place d'un outil d'analyse de l'historique de navigation d'un utilisateur sur l'application Mozilla Firefox en incluant les étapes d'ingestion, préparation, raffinement et visualisation de données et en considérant également les contraintes de traitement en temps réel et de volume des données ont été aussi incluses dans le projet.

Le prototype développé facilite la récupération, le traitement et l'analyse d'un grand volume d'historiques de navigation sur Firefox en temps réel grâce à l'utilisation de technologies tels que Kafka, Spark, Cassandra.

Le projet a été divisé en 3 parties: la première partie a consisté à examiner l'ensemble de données des historiques de Firefox avec l'objectif d'établir quels seraient les besoins de traitement (format de données) et les axes d'analyse.
La deuxième partie a été consacrée à la conception de l'architecture du prototype. Dans cette partie différentes technologies ont été analysées afin de sélectionner lesquelles pourraient répondre aux contraintes du prototype.
Finalement, la troisième partie a été dédiée à l'analyse de données et la visualisation des résultats.

Le document est organisé en 6 sections. La section 1 comprend l'introduction à la problématique du projet. La section 2 présente le contexte et les prérequis nécessaires à la compréhension du travail effectué. La section 3 décrit l'état de l'art . La section 4 présente l'approche du stage et les méthodologies employées. La section 5 décrit le travail technique réalisé pendant le stage et la section 6 présente les conclusions de ce travail.

Mots clés: *Analyse textuelle, analyse en temps réel, Angular, base de données non-relationnelle, big data, broker de messages, Cassandra, Char.js, dataviz, datawarehouse, ingestion de données, Kafka, Spark.*

Sommaire

I. Introduction

Avoir connaissance de la qualité de données qui sont produits dans une entreprise peut être considéré comme un avantage non négligeable par rapport à la concurrence.
Les journaux des serveurs Web, des applications, les différents historiques d'utilisation et de navigation des utilisateurs sont des sources précieuses d'intelligence opérationnelle, révélant des opportunités de revenus potentielles et aidant à réduire les résultats négatifs.

Dans le passé, il était coûteux de capturer la plupart des données produites d'un système d'informations, sans parler de mettre en œuvre des systèmes qui agissent intelligemment sur celles-ci en temps réel.

Récemment de nombreuses avancées technologiques permettent maintenant de répondre aux problématiques évoquées précédemment. Aujourd'hui, il existe de nombreux outils comme par exemple Apache Hadoop, Kafka, MongoDB, Cassandra et Spark qui permettent de capturer des événements en temps réel, de les traiter en utilisant des algorithmes d'intelligence artificielle et enfin de visualiser les résultats ou analyses obtenus de manière à faciliter la prise de décisions.

Dans ce projet, nous avons mis en œuvre un système capable de récupérer l'historique de navigation d'un utilisateur sur Mozilla Firefox en temps réel ; les différentes composantes de cet historique (sites visités, recherches effectuées etc) sont ensuite transformées, indexées et rendues disponible à la visualisation.

Cette implémentation est basée sur des composants open source tels que Apache Kafka, Spark, Cassandra, Spring, Angular et chart.js.

2. Contexte et Pré-requis

2.1 Définition de la Problématique

Les historiques de navigation sur firefox, chrome ou autre navigateur renferment des informations clés sur nos besoins, nos intérêts, nos envies dans notre vie en général. Mettre en place des outils qui explorent et exploitent ces données telles que la fréquence, le type de navigation, le site visité ou encore le temps passé sur un site web est aujourd'hui un enjeu fondamental pour toute type d'organisation. L'extraction, la transformation et l'analyse de ces données peuvent permettre par exemple d'identifier des comportements récurrents pouvant être automatisés, de personnaliser des suggestions etc.. Les principales difficultés de mise en place de ces outils sont le volume de données à traiter, l'identification des axes d'analyse et enfin le traitement en temps réel de ces données.

2.2 Objectifs

2.2.1 Objectif Général

- Mettre en place un outil d'analyse de l'historique de navigation de Mozilla Firefox en considérant les étapes d'ingestion, préparation, raffinement et visualisation de données et les contraintes de traitement en temps réel et volume de données.

2.2.2 Objectifs spécifiques

- Identifier les axes d'analyse sur les données de navigation de Mozilla Firefox.
- Proposer une architecture qui satisfait les contraintes de traitement en temps réel et d'un grand volume de données.
- Mettre en place une application web qui permet de réaliser la "chaîne complète" : depuis la récupération des données jusqu'à la visualisation des résultats.

2.3 Les attentes et les contraintes

Les attentes de ce projet sont au nombre de deux. La première est la **montée en compétences sur plusieurs technologies big data telles que Kafka, Spark et Cassandra**, la deuxième est de **mettre en application les connaissances théoriques apprises lors du master en BI et Big data sur une véritable application.**

En ce qui concerne les contraintes du travail, celles-ci sont liées principalement à 2 variables : **la récupération et le traitement de données en temps réel** et **le traitement d'un volume de données important** lié à la navigation sur Mozilla firefox.

3. Cadre Théorique

La révision bibliographique s'est concentrée principalement sur 3 axes: le premier axe est lié aux différents types de pipelines entre le traditionnelle *ETL* et les modernes: *Cloud et ELT.* Le deuxième est consacré à 3 types d'architectures de base pour la mise en place d'un pipeline. Le dernier axe est lié aux techniques d'IA pour l'analyse textuelle de données.

3.1.1 ETL, Cloud, ELT – Les différents types de Pipeline de données [CARTELIS, 2021]

3.1.1.1 L'approche traditionnelle : l'ETL

L'ETL est l'approche classique pour la construction d'un pipeline de données. Le nom de cette famille d'outils décrit le cheminement de la donnée : **Extract – Transform – Load**.
Un ETL est un outil qui permet d'extraire les données à partir des sources de données, tels que des fichiers, des bases des données, des web services ou d'autres sources de données en utilisant divers connecteurs. Ces données sont ensuite transformées et, finalement, sont chargées dans un entrepôt de données (Data warehouse).
Historiquement, les ETL ont été utilisés pour charger des données en batch (sous forme de lots) dans les bases de destination, souvent à large échelle, cependant, dans l'actualité, il y a des outils qui permettent de charger la donnée en temps réel ou en quasi temps réel. C'est ce que l'on appelle le « streaming de données ».

Le schéma ci-dessous présente les étapes à suivre pour construire un pipeline de données ETL:

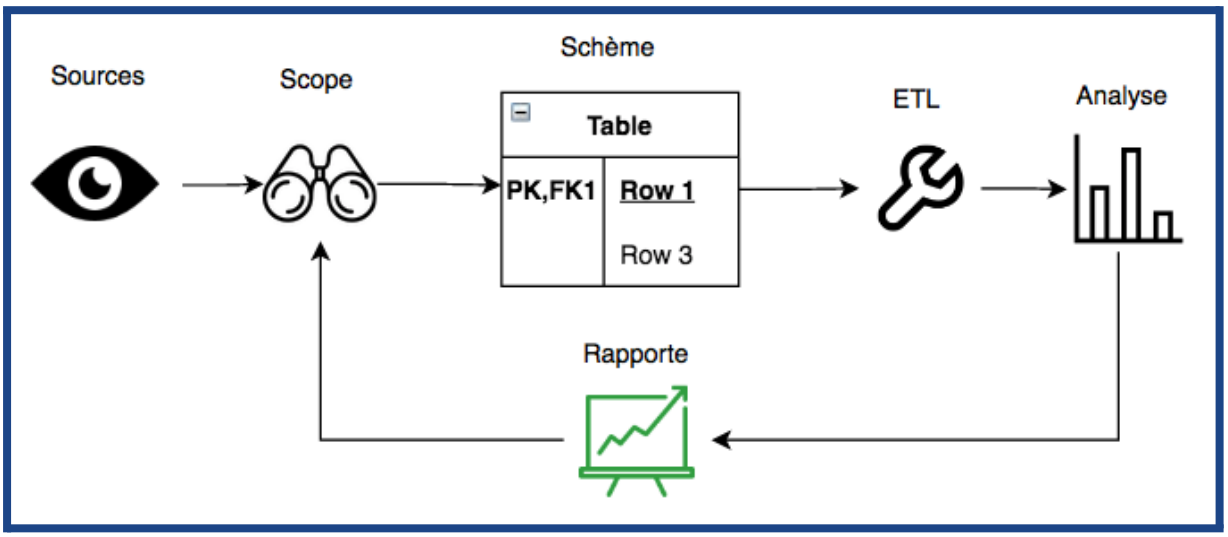

Figure 1. Workflow de data integration et analyse

Le premier pas pour construire un pipeline de données ETL pour faire de la BI est l'identification des sources de données à utiliser et les objectifs du reporting. Ensuite, il faut définir un modèle de données, qui détermine la manière dont ces données apparaîtront dans le reporting. De ce modèle vont

sortir les transformations à opérer sur les données. Après, il faut configurer le pipeline de données ETL pour qu'il réalise les transformations nécessaires sur les données. Ces données sont chargées dans la base de données (de type Data Warehouse), à laquelle est connectée un outil de BI qui permet de créer des visualisations de données (reporting).

Quelques inconvénients des pipeline de données ETL :

- **Un pipeline basé sur un dispositif ETL + Data Warehouse est rigide:** ce qui implique que le changement d'un élément dans le processus d'extraction, de transformation ou dans le processus de chargement dans la base cible devient en une opération complexe et coûteuse.
- **Les pipelines de données ETL sont assez complexes et coûteux à mettre en œuvre**, puisque ils sont construits à base de routines dictées par des besoins spécifiques, des transformations spécifiques. Ces tâches ont besoin d'un personnel spécialisé pour en assurer la maintenance.

3.1.1.2 L'approche moderne : Cloud

Le Cloud désigne l'utilisation de ressources informatiques à distance, via internet, sur un mode décentralisé.
De plus en plus d'applications et de services sont accessibles dans le Cloud puisqu' il permet de se libérer des contraintes des infrastructures physiques et résout la plupart des problématiques d'échelle, d'accessibilité et de coût.
Les organisations n'ont plus besoin de discriminer les sources de données pour optimiser la vitesse de traitement et peuvent multiplier les analyses, en plus, les entreprises peuvent ajouter ou supprimer des ressources de stockage en quelques clics et les utilisateurs peuvent accéder à des tableaux de bord et des rapports via des interfaces web.

3.1.1.3 L'approche moderne : l'ELT

Les outils ELT (Extract-Load-Transform) mettent en œuvre une nouvelle approche en matière de pipeline de données. L'idée consiste à streamer la donnée et de la charger dans la base de destination sans la transformer. Autrement dit, on inverse la séquence utilisée dans les pipelines de données classiques.
Le développement des outils ELT est inséparable de l'essor du Cloud. Les outils ELT sont par construction des outils Cloud, c'est-à-dire des technologies SaaS. Seul le Cloud permet en effet un streaming temps réel. Correctement implémenté, un pipeline de données ELT permet d'intégrer la donnée en continu, avec le minimum d'intervention manuel et de codes custom.

L'inversion des étapes de chargement et de transformation résout les principales problématiques des ETL, c'est-à-dire :

- **Le problème de la complexité:** Un pipeline de données ELT est un pipeline simplifié. Il permet de charger les données dans la base de manière simple, puisqu'il n'y a pas de transformations. Dans les infrastructures ETL classiques, c'est sur les data engineers que repose tout le travail de construction et de maintenance du pipeline. Dans une infrastructure ELT, ce sont les analystes, les data scientists, qui transforment la donnée stockée dans la base en utilisant du SQL.

- **Le problème de la rigidité:** Un pipeline ELT est plus résilient, plus facile à faire évoluer puisque les transformations ne sont pas faites en amont (en phase de design du pipeline). Les transformations sont effectuées après que les données aient rejoint la base dans leur forme brute. Typiquement, l'ajout d'une nouvelle source de données n'oblige pas à re-designer le pipeline et de faire appel aux data engineers.
- **Le problème du coût:** Un pipeline de données ELT est moins coûteux car plus simple à maintenir. Dans les faits, la maintenance du pipeline de données (des phases « Extraction » & « Chargement ») est souvent externalisée et automatisée. Un ELT permet à l'entreprise de concentrer ses efforts sur ce qui importe le plus : la transformation des données et, derrière, leur exploitation en BI ou en activation.

3.1.2 3 Types d'architectures de pipeline de données [CARTELIS, 2021]

3.1.2.1 Architecture Simple

Dans cette architecture simple, le pipeline de données est basé sur un process en batch. Il est construit selon le processus ETL classique. Au début de la chaîne se trouve la source de données qui génère un grand nombre de data points qui seront après poussez dans un Data Warehouse ou une base de BI suite à plusieurs opérations de transformation.

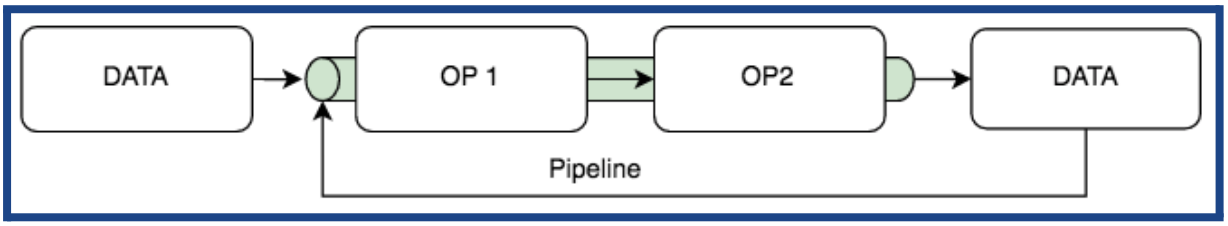

Figure 2. Pipeline

3.1.2.2 Architecture streaming

Dans ce modèle d'architecture, la donnée extraite est processée en même temps qu'elle est générée. La donnée récupérée des sources est délivrée en continue et en quasi temps-réel aux outils d'analyse et d'activation. C'est l'opposé de l'approche batch qui consiste à basculer les données extraites des sources dans la base sous forme de lots, de manière discontinue donc.
Les enjeux autour du temps réel rendent l'architecture streaming de plus en plus incontournable. Les pipelines de données construits sur cette approche utilisent plusieurs applications, contrairement au pipeline de données ETL simple, qui ne repose que sur une technologie : l'ETL.

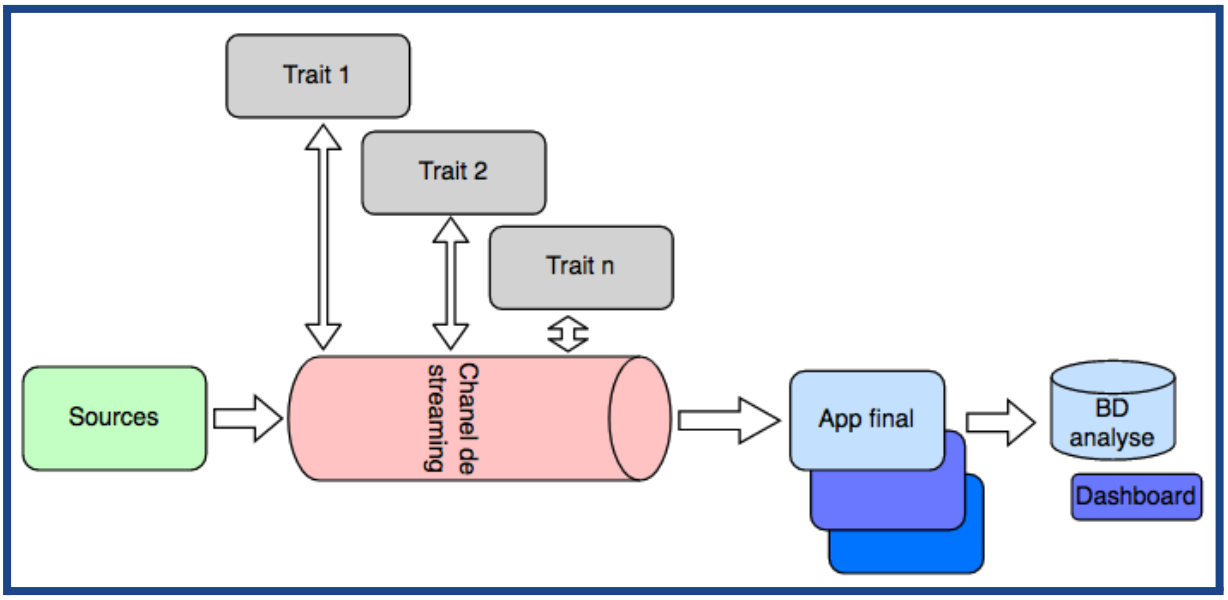

Figure 3. Architecture streaming

3.1.2.3 Architecture Lambda

Cette architecture combine l'approche batch et l'approche streaming. L'architecture Lambda est populaire dans les environnements Big Data puisqu'elle permet aux développeurs de tenir compte à la fois des cas d'usage du streaming en temps réel et des cas d'usage historique du batch processing. L'une des clés de cette architecture est qu'elle encourage le stockage des données dans leur format brut, ce qui facilite l'évolution du pipeline de données.

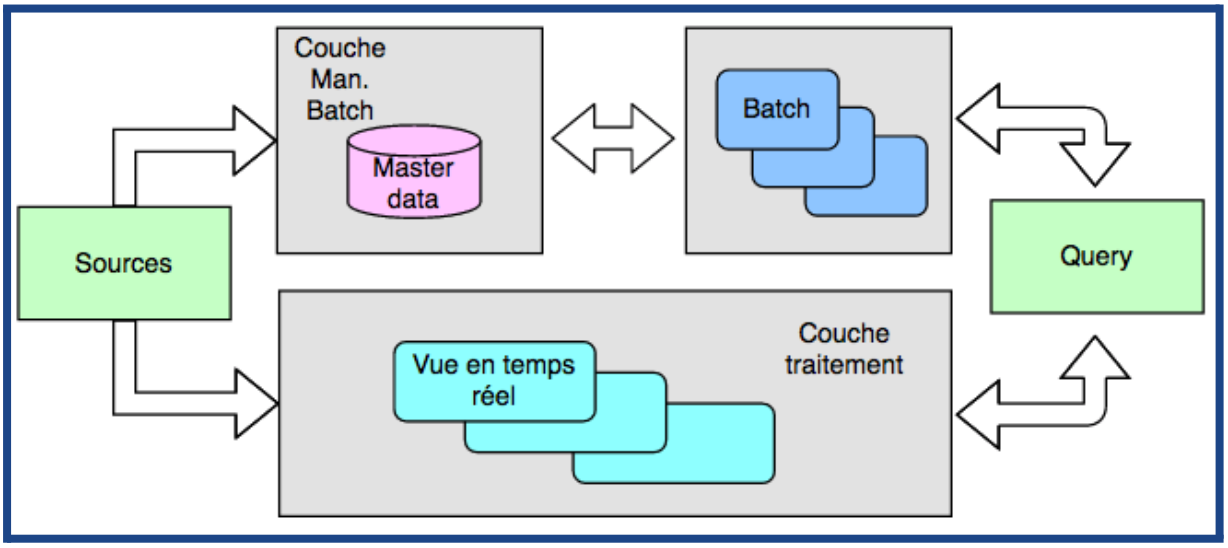

Figure 4. Architecture lamba

3.1.3 Approches dans l'exploration de données textuelles

L'Analyse de Données Textuelles regroupe les méthodes qui visent à découvrir l'information essentielle contenue dans un texte. Ce besoin d'analyser les document provient principalement de 2 groupes :

- D'une part, les entreprises, qui peuvent aujourd'hui collecter très facilement une grande quantité de textes avec Internet (articles, brevets, rapports, études, mais aussi e-mails, messages de forums, enquêtes clients, fiches de centres d'appel, descriptifs de produits...).
- Et d'autre part une demande des chercheurs, qui ont besoin d'une alternative soit à de traditionnelles analyses de contenu jugées trop subjectives, soit à de simples analyses par mots-clés jugées trop pauvres.

Pour les entreprises, l'intérêt est s'agit d'organiser automatiquement les contenus, d'extraire de l'information à partir d'un ensemble hétérogène de textes peu structurés. Ici, nous parlerons de text mining (ou cartographie web) et l'indexation automatique de documents (capitalisation des connaissances avec des outils statistiques) comme les techniques plus utilisées pour les organisations.

Pour les chercheurs, son travail se centre à explorer les méthodologies qualitatives « assistées » par des outils quantitatifs basées sur le Web sémantique et le T.A.L. (Traitement Automatique des Langues).

Entres les approches qui nous identifions pour l'analyse textuelle, il y a **(FALLERY, RODHAIN, 2017)**:

- *L'analyse lexicale* qui s'est fondée sur les proximités entre les mots employés et la statistique fréquentielle.
- *L'analyse linguistique* qui repose sur l'idée qu'il existe des connections entre système linguistique et système cognitif, et il s'agit alors de prendre en charge à la fois les aspects liés à la cohérence référentielle (ce à quoi le texte se réfère : des substantifs, signes linguistiques qui renvoient à une réalité extra linguistique) et aussi ceux relatifs au contexte d'énonciation (comment est-ce dit : des verbes, des adverbes, des conjonctions, des connecteurs.. qui servent à traduire la relation du locuteur à la situation, son point de vue et ses jugements).
- *La cartographie cognitive* ou carte cognitive qui s'agit d'une représentation matérielle graphique des représentations mentales d'un ou plusieurs sujets à un moment donné, est généralement obtenue à partir d'une représentation discursive exprimée dans un texte ou un entretien.
- *L'analyse thématique qui* prendre en compte les processus interprétatifs dans la construction de la donnée, mais avec la possibilité d'augmenter la validité des analyses de contenu « classiques » qui ne proposaient qu'une approche méthodique fondée sur l'explicitation des règles de lecture, d'interprétation et de codage.

4. Méthodologie

4.1 Méthodologie pour <u>Objectif Spécifique 1:</u>

«Identifier les axes d'analyse sur les données de navigation sur Firefox.»
Pour le premier objectif, nous avons suivi:

1. Révision webographie sur le modèle de données proposé par firefox pour stocker les données de navigation.
2. Sélection de relations qui génèrent un axe d'analyse

4.2 Méthodologie pour <u>Objectif Spécifique 2:</u>

«Proposer une architecture qui satisfait les contraintes de traitement en temps réel et un grand volume de données.»
Pour le deuxième objectif, nous avons :

1. Révision webographie des plusieurs technologies big data.
2. Filtrage de technologies selon la pertinence avec le projet
3. Recherche des exemples de mise en place et utilisation

4.3 Méthodologie pour l'<u>objectif Spécifique 3:</u>

«Mise en place d'une application web qui permet de réaliser la chaîne complète de récupération, traitement et visualisation de données base sur les historiques de navigation sur firefox.»
Finalement, pour la mise en place de l'application nous avons fait:

1. Définition des spécifications du backend
2. Conception de l'architecture et technologies liées
3. Installation des technologies big data choisis
4. Utilisation du framework jhipster pour la génération de code de base.
5. Mise en place de l'étape d'ingestion de données avec kafka et sqlite (firefox)
6. Mise en place de l'étape de traitement données en temps réel avec Spark, spark sql, java
7. Stockage des traitements sur cassandra
8. Mise en place de l'étape de raffinage avec spark sql, cassandra
9. Mise en place des services/calculs pour la récupération de donnés à visualiser
10. Définition des spécifications de frontend
11. Mise en place de l'interface web avec Angular et chart.js

5. Travail effectué

Le diagramme ci-dessous illustre la configuration que nous avons suivie pour la mise en place du projet d'analyse de l'historique de navigation **[BAGHEL, 2016][MCDONALD, 2019]**. Les données brutes proviennent de l'historique de navigation sur firefox qui sont stockés dans une base de données relationnelle SQLITE: *places*.
Ces données sont ensuite stockées dans différents topics en utilisant le connecteur sqlite-kafka. Enfin ces données sont récupérées, en temps réel, par des consommateurs Spark qui les traitent, les transforment et les stockent dans un datawarehouse construit à l'aide de Cassandra.
Ces données sont alors disponibles pour l'application web. La partie visualisation a été réalisée en utilisant Angular et chart.js.
Le projet à été construit en utilisant le framework Spring(java) pour le backend et Angular pour le frontend.

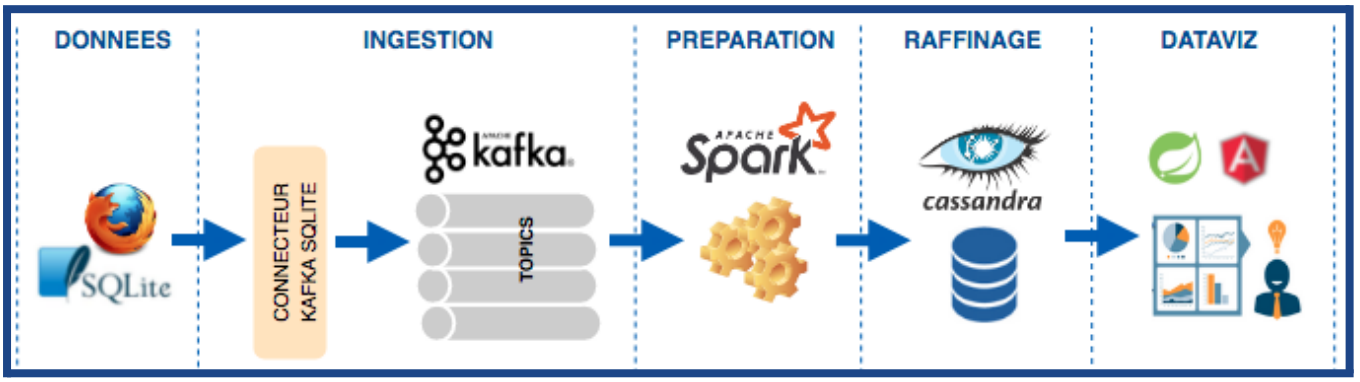

Figure 5 Architecture

5.1 Exploration des données et Définition du modèle d'analyse

Le schéma de la base de données *places.sqlite* est illustré sur la figure 6.

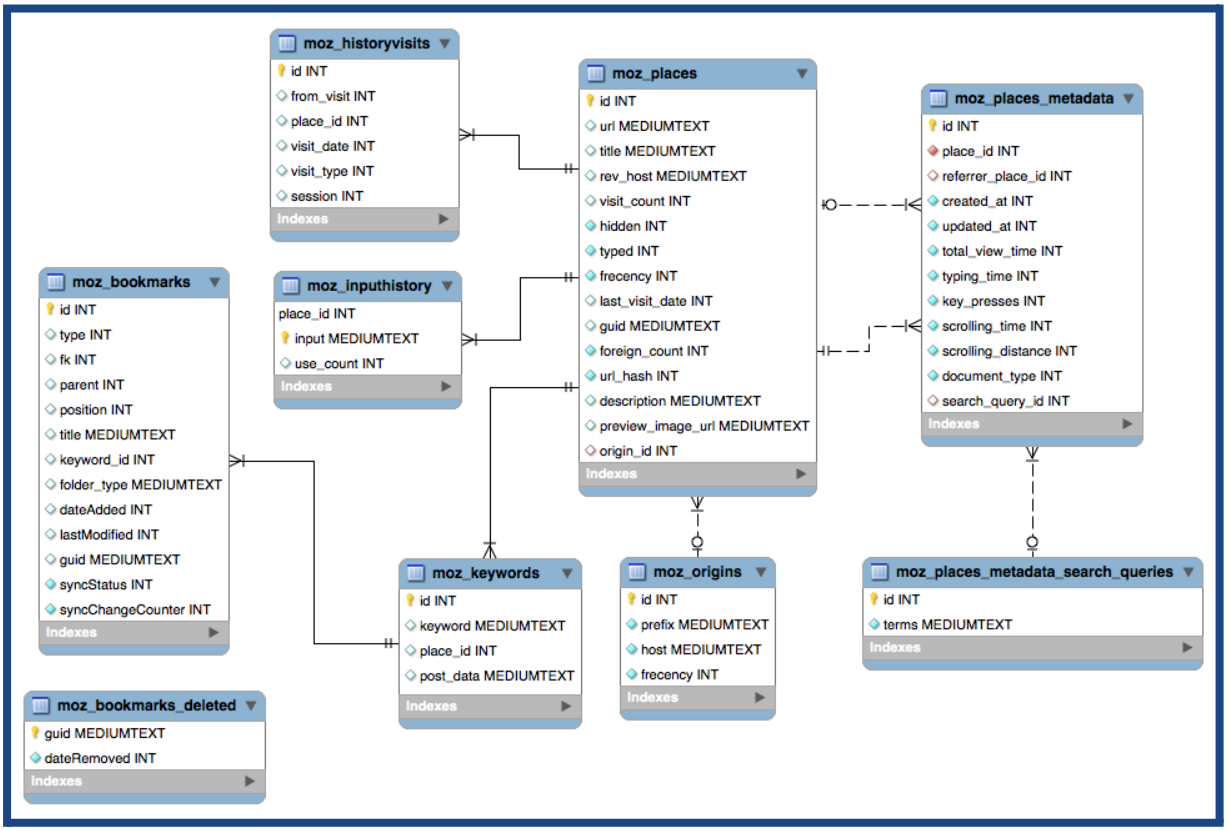

Figure 6 Diagramme Entité-relation places

Les tables à partir desquelles nous recueillons actuellement des données sont [] :

- *(places)* **moz_places :** fournit des informations sur les URL visités. Voici les colonnes de cette table :

ATTR	TYPE	DESCRIPTION
ID	INTEGER PRIMARY KEY	Entier qui indique la clé primaire de la base de données
URL	LONGVARCHAR	Url site
TITLE	LONGVARCHAR	Titre de la page tel qu'il apparaît dans le navigateur
REV_HOST	LONGVARCHAR	L'inverse du nom d'hôte qui a été visité.
VISIT_COUNT	INTEGER DEFAULT 0	Compteur pour le site

FRECUENCY	INTEGER DEFAULT -1 NOT NULL	Combinaison de fréquence et de récence, utilisé pour calculer quels sites apparaissent en haut de la liste de suggestions lorsque les URL sont saisies dans la barre d'adresse.
HIDDEN	INTEGER DEFAULT 0 NOT NULL	0 ou 1. si l'URL est masquée, l'utilisateur n'y a pas accédé directement, indique généralement une page intégrée utilisant quelque chose comme un iframe
TYPED	INTEGER DEFAULT 0 NOT NULL	Indique si l'utilisateur a tapé l'URL directement dans la barre d'adresse

- *(places)* **moz_historyvisits :** contient l'historique de recherche. Il renvoie vers des lieux de ***moz_places*** et fournit des informations sur le référent, la date de la visite, le type de visite (c'est-à-dire une connexion directe, une redirection de page, etc.), et la session de navigation.

ATTR	TYPE	DESCRIPTION
ID	INTEGER	Entier qui indique la clé primaire
FROM_VISIT	INTEGER	Visit d'origen
PLACE_ID	INTEGER	Correspond à une place dans la table moz_places
VISIT_DATE	TIMESTAMP	Date de la visite
VISIT_TYPE	INTEGER	Représente le type de visit entre 7 différents : 1-TRANSITION_LINK, 2-TRANSITION_TYPED, 3-TRANSITION_BOOKMARK, 4-TRANSITION_EMBED, 5-TRANSITION_REDIRECTED_PERMANENT, 6-TRANSITION_REDIRECTED_TEMPORARY, 7-TRANSITION_DOWNLOAD, 8- TRANSITION_FRAMED_LINK, 9-TRANSITION_RELOAD

- *(places)* **moz_origins :** contient les différents hosts qui hébergent les sites visités pour l'usager.

ATTR	TYPE	DESCRIPTION
ID	INTEGER	Entier qui indique la clé primaire
PREFIX	LONGVARCHAR	Protocole

HOST	LONGVARCHAR	Nom du host
FRECUENCY	INTEGER DEFAULT -1 NOT NULL	Combinaison de fréquence et de récence, utilisé pour calculer quels sites apparaissent en haut de la liste de suggestions lorsque les URL sont saisies dans la barre d'adresse.

A partir de ces données brutes, nous avons construit un modèle de datawarehouse où nous allons sauvegarder des informations qui nous seront utiles pour analyser le comportement de navigation de l'usager sur le web :

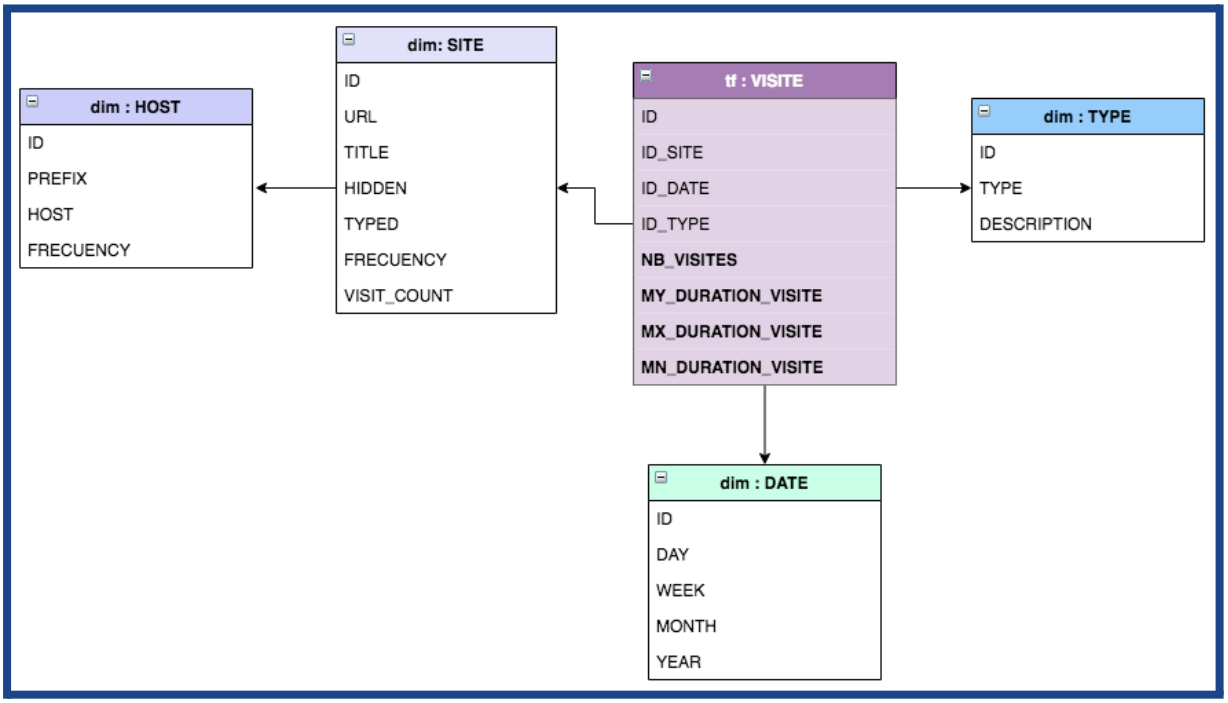

Figure 7 Modèle datawarehouse

Les **VISITES** ou recherches sur firefox sont ce sur quoi nous allons porter l'analyse et les axes d'analyse sont : **PLACE (web site), le TYPE de visite et la DATE**.

La mesure **NB_VISITES** correspond au nombre de visites réalisées à une date donnée, sur un site (site) spécifié et pour un type donné. **MY_DURATION_VISITE** est le temps moyen passé sur un site. **MX_DURATION_VISITE** est le temps maximum passé sur un site et **MN_DURATION_VISITE** est le temps minimum passé sur un site.

5.2 Ingestion des Données

La première étape opérative de ce projet est l'ingestion de données. Pour ce faire, nous avons choisi Kafka, il s'agit d'une plateforme de diffusion en continu distribuée open source qui est utile pour créer des pipelines de données en temps réel et des applications de traitement de flux. Dans notre cas nous avons utilisé Kafka pour récupérer les données de la base de données *places.sqlite* et pour les exposer sur plusieurs topics qui seront ensuite consommés par une application Spark (java).

Nous avons utilisé Kafka connect, un composant de Kafka qui fonctionne comme un hub de données centralisé pour une intégration simple des données entre les bases de données, les magasins de valeurs-clés, les index de recherche et les systèmes de fichiers.

Dans notre cas, nous avons utilisé le plugin du connecteur ***sqlite-Kafka*** **[KAFKA][NEWTON, 2019]** pour exposer chacune des données stockés dans les tables *places, origins et visithistory* dans un topic différent :
firefox-moz_origins,firefox-moz_historyvisits,firefox-moz_places

- **moz_places** : firefox-moz_places
- **moz_historyvisits :** firefox-moz_historyvisits
- **moz_origins :** firefox-moz_origins

Sur les figures suivantes, on peut trouver les requêtes HTTP POST pour créer la connexion à chacune des tables :

```
{"name": "sqlite-kafka-firefox-places",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "2",
    "connection.url": "jdbc:sqlite:/Users/sim/Library/Application
Support/Firefox/Profiles/arh8ticx.default-release-1627228538856/places.sqlite",
    "table.whitelist": "moz_places",
    "mode": "timestamp+incrementing",
    "incrementing.column.name": "id",
    "topic.prefix": "firefox-",
    "validate.non.null": "false",
    "name": "sqlite-kafka-firefox-places"
  },
  "tasks": [],
  "type": "source"}
```

```
{"name": "sqlite-kafka-firefox-historyvisits",
  "config": {
     "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
     "tasks.max": "2",
     "connection.url": "jdbc:sqlite:/Users/sim/Library/Application
Support/Firefox/Profiles/arh8ticx.default-release-1627228538856/places.sqlite",
     "table.whitelist": "moz_historyvisits",
     "mode": "timestamp+incrementing",
     "incrementing.column.name": "id",
     "topic.prefix": "firefox-",
     "validate.non.null": "false",
     "name": "sqlite-kafka-firefox-historyvisits"
  },
  "tasks": [],
  "type": "source"}
```



```
{"name": "sqlite-kafka-firefox-origin",
  "config": {
     "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
     "tasks.max": "2",
     "connection.url": "jdbc:sqlite:/Users/sim/Library/Application
Support/Firefox/Profiles/arh8ticx.default-release-1627228538856/places.sqlite",
     "table.whitelist": "moz_origins",
     "mode": "timestamp+incrementing",
     "incrementing.column.name": "id",
     "topic.prefix": "firefox-",
     "validate.non.null": "false",
     "name": "sqlite-kafka-firefox-origin"
  },
  "tasks": [],
```

```
"type": "source"}
```

Postérieurement, nous pouvons observer les topics créés :

```
→  kafka_2.13-2.8.0 bin/kafka-topics.sh --list --bootstrap-server localhost:9092
__consumer_offsets
connect-configs
connect-offsets
connect-status
firefox-moz_formhistory
firefox-moz_historyvisits
firefox-moz_origins
firefox-moz_places
→  kafka_2.13-2.8.0
```

Figure 8 : Liste de topics créés : firefox *. Chaque topic corresponds à une table *places, origins ou historyvisits*

Afin de valider la création de ces connecteurs, nous pouvons créer un consommateur dans Kafka et observer les données qui sont stockées dans les topics :

```
Last login: Wed Aug 25 14:42:55 on ttys006
→  kafka_2.13-2.8.0 bin/kafka-console-consumer.sh --topic firefox-moz_places --from
-beginning --bootstrap-server localhost:9092
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"}
,{"type":"string","optional":true,"field":"url"},{"type":"string","optional":true,"
field":"rev_host"},{"type":"int32","optional":true,"field":"visit_count"},{"type":"
int32","optional":true,"field":"hidden"},{"type":"int32","optional":true,"field":"t
yped"},{"type":"int32","optional":true,"field":"frecency"},{"type":"int32","optiona
l":true,"field":"last_visit_date"},{"type":"string","optional":true,"field":"guid"}
,{"type":"int32","optional":true,"field":"foreign_count"},{"type":"int32","optional
":true,"field":"url_hash"},{"type":"string","optional":true,"field":"description"},
{"type":"string","optional":true,"field":"preview_image_url"},{"type":"int32","opti
onal":true,"field":"origin_id"}],"optional":false,"name":"moz_places"},"payload":{"
id":1,"url":"https://www.mozilla.org/privacy/firefox/","rev_host":"gro.allizom.www.
","visit_count":1,"hidden":1,"typed":0,"frecency":20,"last_visit_date":-1417447786,
"guid":"PtGkE1e-Iqdf","foreign_count":0,"url_hash":101683833,"description":null,"pr
eview_image_url":null,"origin_id":1}}
```

Figure 9 : Consommateur pour le topic *firefox-moz_places*

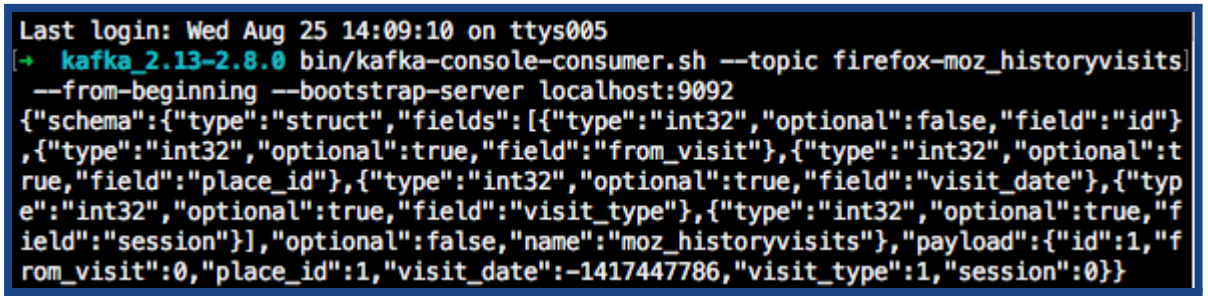

Figure 10 : Consommateur pour le topic *firefox-moz_historyvisits*

Sur les figures 9 et 10, nous pouvons observer que les messages récupérés par les consommateurs contiennent le schéma de la table *places et historyvisits* mais également les données nous intéressant dans la partie “payload”. Chacun de ces messages correspond à une “visite” d’un utilisateur.

La figure 11 est un extrait de code réalisé à l’aide de Spark (java) pour consommer de manière parallèle et distribuée les messages stockés dans les topics kafka. La tâche de consommation des topics reste active de manière permanente :

```
private void processSpark(final List<String> topics) throws
InterruptedException {
   JavaInputDStream<ConsumerRecord<String, String>> messages =
        KafkaUtils.createDirectStream(
            this.streamingContext,
            LocationStrategies.PreferConsistent(),
            ConsumerStrategies.Subscribe(topics, consumerProps));
   topics.forEach(topic -> saveInformationTemporarily(messages,
topic));
   this.streamingContext.start();
   this.streamingContext.awaitTermination();}
```

Figure 11 : Code en Spark (java) pour consommer les topics Kafka

Sur la figure 12, nous pouvons observer le journal du consommateur pour les 3 topics. Un offset représente la position du curseur du consommateur qui, dans le cas où l'application est arrêtée, permet à celui-ci de ne pas charger de nouveau l’intégralité des données mais de reprendre à cette position (position de lecture).

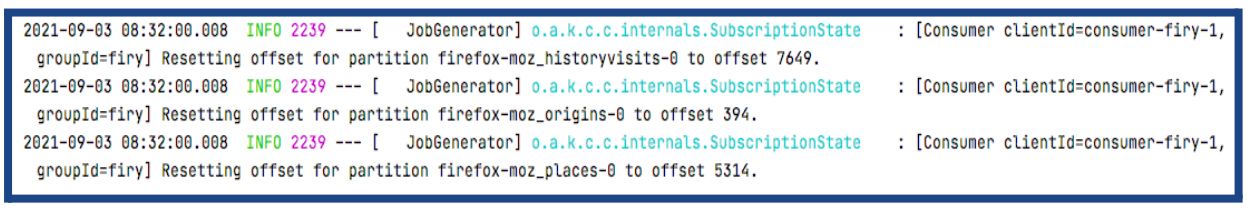

Figure 12 : Logs de consommateurs de topics

5.3 Préparation des données

Dans cette étape, nous avons réalisé les tâches de *filtrage, de calcul des nouvelles valeurs et de stockage dans des tables intermédiaires* (pré-raffinage).

La tâche de filtrage est liée à la sélection des attributs qui nous intéressent dans chaque table. Pour les tables *historyvisits et origin*, nous avons gardé tous les attributs.
Pour place, nous avons gardé 10 des 15 attributs :

```
{id,url,title,rev_host,visit_count,hidden,typed,frecency,description,guid,preview_image_url,last_visit_date,url_hash,origin_id,foreign_count} =>
```

```
{id,url,title,visit_count,hidden,typed,frecency,description,last_visit_date,origin_id}
```

La tâche de calcul de nouvelles valeurs correspond à la transformation de plusieurs dates en timestamp vers un format *'YYYY-MM-DD'* qui est plus lisible pour l'être humain.

```
this.last_visit_date_simple =
timestamp.toLocalDateTime().toLocalDate();
```

Figure 13 : Calcul de temps passé dans un page : *duration*

La deuxième valeur calculée est le temps passé sur une page: "*duration*". Pour ce calcul, nous avons calculé la différence entre les dates de visite de deux visites consécutives.

```
private Dataset<Row> calculateDurationVisite(final Dataset<Row>
visits) {
   WindowSpec windowSpec = Window.orderBy("visit_date");
   return visits.withColumn("duration",
                            visits.col("visit_date")
.minus(when((lag("visit_date",1).over(windowSpec)).isNull(), 0)
.otherwise(lag("visit_date", 1).over(windowSpec))));}
```

Figure 14 : Calcul de temps passé dans un page : *duration*

La dernière tâche de cette étape, correspond au stockage de ces données traitées. La figure 15 illustre la création des tables intermédiaires :

```
CREATE TABLE host(id int PRIMARY KEY, prefix text, host text, frecency int);
```

```
CREATE TABLE site(id int PRIMARY KEY, url text, title text, hidden int, typed int, frecency int,
visit_count int, rev_host text, description text, last_visit_date bigint, last_visit_date_simple date, origin_id
int);
CREATE TABLE visit(id int PRIMARY KEY, from_visit text, place_id text, visit_date bigint,
visit_date_simple date, visit_type int, duration bigint);
```

Figure 15 : Création des tables

Code pour sauvegarder les données traitées dans les tables temporaires **[BORSOS, 2017]**:

```
private void saveTemporarily(final JavaDStream<ConsumerRecord<String,
String>> messages,final Class mapperClass, final String nameTable) {
ObjectMapper objectMapper = new ObjectMapper();
messages.map(msg -> ((JSONObject)
JSONValue.parse(msg.value())).get("payload"))
.foreachRDD(javaRdd -> {
List<Object> list = new ArrayList<>();
for (Object record : javaRdd.collect()) {
String value = record.toString();
Object temp = objectMapper.readValue(value, mapperClass);
list.add(temp);}
if (!list.isEmpty()) {
JavaRDD<Object> rdd =
this.streamingContext.sparkContext().parallelize(list);
javaFunctions(rdd).writerBuilder("cassandrafiry", nameTable,
mapToRow(mapperClass)).saveToCassandra();}});}
```

Figure 16 : Code Spark pour sauvegarder de manière temporaire les données traitées

En figure 17, nous observons un exemple de ces données stockés dans les tables temporaires :
c

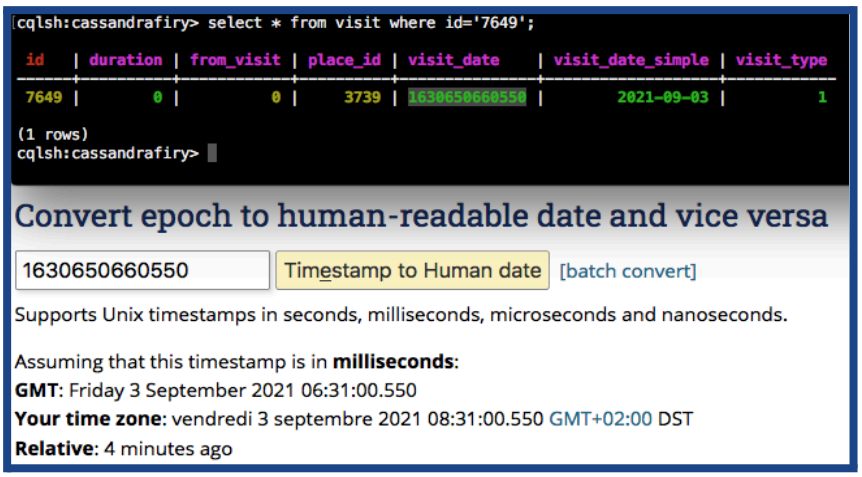

Figure 17 : Exemple de données brutes stockées

id	duration	from_visit	place_id	visit_date	visit_date_simple	visit_type
4317	0	4312	10	1630661933524	2021-09-03	1
3372	0	3371	2475	1630662079972	2021-09-03	1
1584	0	1583	1229	1630575498954	2021-09-02	1
7034	0	0	4888	1630748299090	2021-09-04	1

Figure 18 : Exemple de données stockées dans la table temporaire visit

id	title	frecency	origin_id	visit_count
3472		null	1718	3	1
2690		null	1675	3	1
6443	(3) Always (Album) - Julian Lennon 'Everything Changes' The Videos - YouTube	100	4	1
2166		null	84	16	1
2621		null	361	227	4

Figure 19 : Exemple de données stockées dans la table temporaire site

id	frecency	host	prefix
97	6095	particuliers.societegenerale.fr	https://
165	200	news.fr.shop-canda.com	https://
301	600	community.cloudera.com	https://
192	302	www.isere-tourisme.com	https://

Figure 20 : Exemple de données stockées dans la table temporaire host

5.4 Raffinage des données

Après avoir transformé les données brutes, nous devons définir et calculer les mesures que nous voulons analyser. Les mesures définis dans le projet sont :

- Nombre de visites par jour, site et type

```
visits.groupBy("place_id", "visit_type", "visit_date_simple")
            .agg(count("id").alias("nb_visits"));
```

- Temps passé moyen par jour, site et type

```
visits.groupBy("place_id", "visit_type", "visit_date_simple")
            .agg(avg("duration").alias("dur_mean_vis"));
```

- Temps passé minimum par jour, site et type

```
visits.groupBy("place_id", "visit_type", "visit_date_simple")
            .agg(min("duration").alias("dur_min_vis"));
```

- Temps passée maximum par jour,site et type

```
visits.groupBy("place_id", "visit_type", "visit_date_simple")
            .agg(max("duration").alias("dur_max_vis"));
```

Ensuite nous stockons ces informations dans notre data warehouse créé sur cassandra. Pour le projet, nous avons défini les 5 tables conformes à notre modèle de datawarehouse (fig 7) :

```
CREATE TABLE f_visit(id int PRIMARY KEY, place_id int, date_id int, type_id int, visit_date_simple date, nb_visits int, dur_mean_vis double, dur_max_vis int, dur_min_vis int, host text);
CREATE TABLE d_site(id int PRIMARY KEY, url text, title text, hidden int, typed int, frecency int, visit_count int, description text, last_visit_date bigint, origin_id int);
CREATE TABLE d_host(id int PRIMARY KEY, prefix text, host text, frecency int);
CREATE TABLE d_date(id int PRIMARY KEY, day int, week int, month int, year int);
CREATE TABLE d_type(id int PRIMARY KEY, description text,type text, );
```

Figure 21 : Création des tables du modèle de dw

Ensuite, le code pour le stockage de chacune des tables de type dimension et de la table de fait:

```
Dataset<Row> hosts = this.sparkSession.read(
.format("org.apache.spark.sql.cassandra")
.option("keyspace", "cassandrafiry")
.option("table", "host")
.load();
if (!hosts.isEmpty()) {
JavaRDD<DimHost>rdd=this.streamingContext.sparkContext()
.parallelize(hosts.as(Encoders.bean(DimHost.class)).collectAsList());
javaFunctions(rdd).writerBuilder("cassandrafiry",
"d_host",mapToRow(DimHost.class)).saveToCassandra();}
```

Figure 22 : Dimension Host

```
Dataset<Row> sites = this.sparkSession.read()

.format("org.apache.spark.sql.cassandra")
.option("keyspace", "cassandrafiry")
.option("table", "site")
```

```
.load()
.drop("rev_host", "last_visit_date");
if (!sites.isEmpty()) {
   List<DimSite> list = new ArrayList<>();
   for (Row site : sites.collectAsList()) {
       Integer id = (Integer) site.get(0);
       String url = (String) site.get(8);
       String title = (String) site.get(6);
       Integer visit_count = (Integer) site.get(9);
       Integer hidden = (Integer) site.get(3);
       Integer typed = (Integer) site.get(7);
       Integer frecency = (Integer) site.get(2);
       String description = (String) site.get(1);
       Date last_visit_date_simple = (Date) site.get(4);
       Integer origin_id = (Integer) site.get(5);
           list.add(new DimSite(id, url, title, visit_count, hidden,
typed, frecency, description, last_visit_date_simple, origin_id));}
JavaRDD<DimSite>                              rdd                              =
this.streamingContext.sparkContext().parallelize(list);
javaFunctions(rdd).writerBuilder("cassandrafiry",
"d_site",mapToRow(DimSite.class)).saveToCassandra();}
```

Figure 23: Dimension Site

```
if (!dates.isEmpty()) {
List<DimDate> list = new ArrayList<>();
for (Row date : dates.collectAsList()) {
Integer day = date.getDate(0).toLocalDate().getDayOfMonth();
WeekFields weekFields = WeekFields.of(Locale.getDefault());
Integer                                  week                                 =
date.getDate(0).toLocalDate().get(weekFields.weekOfWeekBasedYear());
Integer month = date.getDate(0).toLocalDate().getMonth().getValue();
Integer year = date.getDate(0).toLocalDate().getYear();
list.add(new DimDate((Integer) date.get(1), day, week, month, year));}
JavaRDD<DimDate>                              rdd                              =
this.streamingContext.sparkContext().parallelize(list);
javaFunctions(rdd).writerBuilder("cassandrafiry","d_date",mapToRow(DimD
ate.class)).saveToCassandra();}
```

Figure 22 : Dimension Date

```
Dataset<Row>f_visits_final =f_visits_with_id.join(dates,
dates.col("visit_date_simple")

.equalTo(f_visits_with_id.col("visit_date_simple")))
                    .join(sites, sites.col("id")
                           .equalTo(f_visits_with_id.col("place_id")))
                    .join(hosts, hosts.col("id")
                           .equalTo(sites.col("origin_id")))
                    .select(f_visits_with_id.col("id"),
                            f_visits_with_id.col("place_id"),
                            f_visits_with_id.col("type_id"),
                            dates.col("id").alias("date_id"), //date_id
                           f_visits_with_id.col("visit_date_simple"),
                            f_visits_with_id.col("nb_visits"),
                            f_visits_with_id.col("dur_mean_vis"),
                            f_visits_with_id.col("dur_min_vis"),
                            f_visits_with_id.col("dur_max_vis"),
                            hosts.col("host"));
List<FactVisit> list = new ArrayList<>();
for (Row visit : f_visits_final.collectAsList()) {
   Integer id = (Integer) visit.get(0);
   Integer place_id = (Integer) visit.get(1);
   Integer type_id = (Integer) visit.get(2);
   Integer date_id = (Integer) visit.get(3);
   Integer nb_visits_ = (Integer) visit.get(5);
   Double dur_mean_vis = (double) visit.get(6);
   Integer dur_min_vis =
Long.valueOf(visit.get(7).toString()).intValue();
   Integer dur_max_vis =
Long.valueOf(visit.get(8).toString()).intValue();
   Date visit_date_simple = (Date) visit.get(4);
   String host = visit.get(9).toString();
list.add(new FactVisit(id, place_id, date_id, type_id, nb_visits_,
dur_max_vis, dur_mean_vis, dur_min_vis, visit_date_simple, host));}
JavaRDD<FactVisit>rdd=this.streamingContext.sparkContext().parallelize
(list);
javaFunctions(rdd).writerBuilder("cassandrafiry", "f_visit",
mapToRow(FactVisit.class)).saveToCassandra();
```

Figure 23 : Table Fait Visit

5.5 Visualisation

La data visualisation est la dernière étape de la chaîne d'analyse des historiques de navigation sur firefox. Nous avons généré les diagrammes suivants :

- *Nombre de visites par jour :* montre le nombre total de visites par jour. Nous pouvons observer que sur une période de 4 jours il y a entre 2000 et 3000 visites par jour sur firefox.

Figure 24 : Nombre de visites par jour

- *Nombre de visites par type :* Dans ce graphique nous présentons la distribution de visites selon le type de visites. On peut observer que les visites sont principalement effectuées en suivant un lien hypertexte (type 1) ou en effectuant une recherche typée (2).

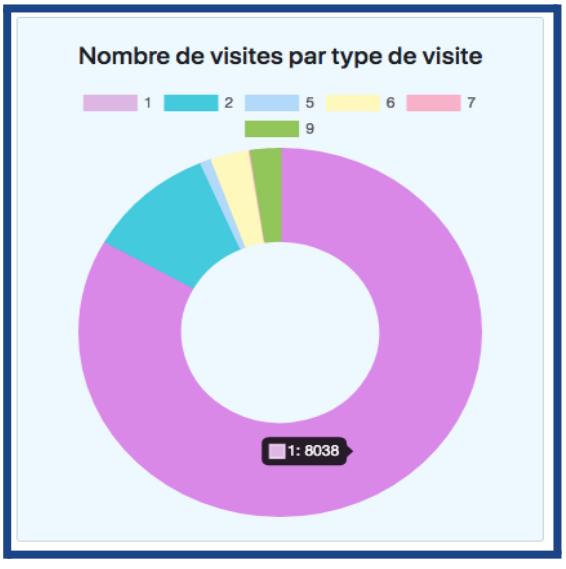

Figure 25 : Nombre de visites par type

- *Mots plus fréquents sur la navigation :* Nous avons choisi les 10 mots les plus fréquents dans les recherches sur firefox. Dans notre cas, nous avons trouvé que la plupart des visites se font sur les mots : *google, google maps, youtube, google traductor.*

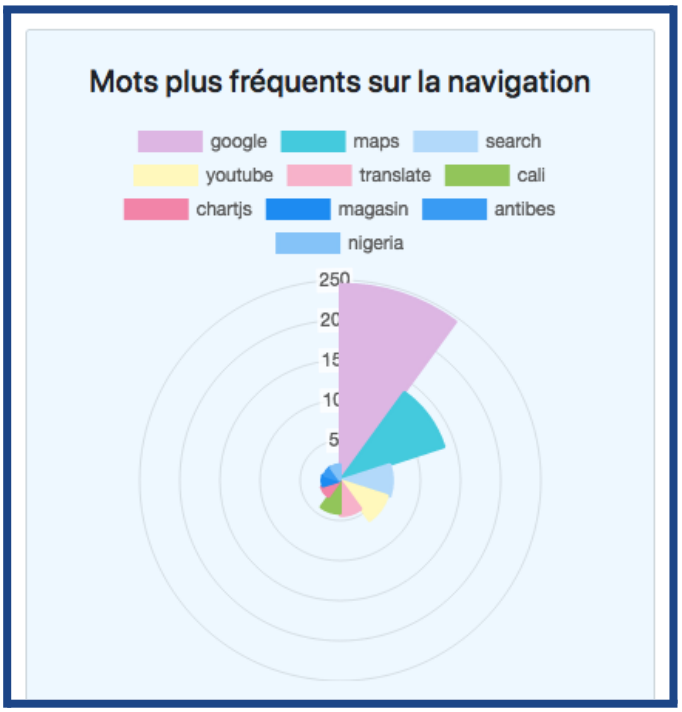

Figure 26 : 10 mots les plus fréquents recherche sur firefox

- *Temps de visite moyen par site :*

Figure 27 : 15 sites avec le temps de visite moyen plus haut

Les graphiques ont été générés en utilisant **chart.js,** une bibliothèque JS open source pour la visualisation des données, qui prend en charge les types de graphiques : barre, ligne, zone, tarte (beignet), bulle, radar, polaire et nuage de points.

5.6 Application Web FIRY

FIRY, l'application d'analyse des historiques de navigation sur firefox, a été développé avec les suivants technologies :

- JHipster : Generateur de code à partir d'un modèle de données
- Spring : Framework JAVA - backend
- Kafka : Broker messages
- Postman : Cliente Web pour créer les connecteurs Kafka - Sqlite
- Spark: Framework pour traitement distribué
- Cassandra: Base de données NoSql
- Angular : Framework js (frontend)
- Chart.js : Library js

Voici les interfaces de l'application **[OZENERO, 2020]**:

- Page d'accueil

Figure 28 : Page d'accueil. 3 informations sur Firy: Quoi ? Comment ? et Pourquoi ?

- Page Dashboard

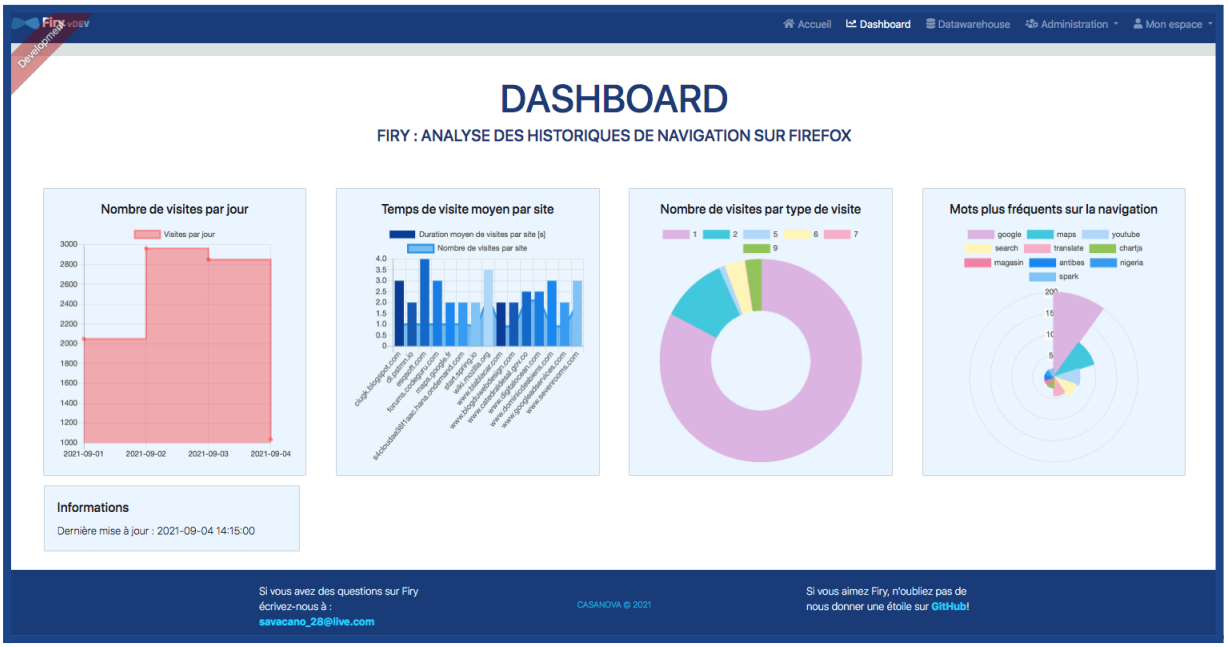

Figure 29 : Page du dashboard. 4 graphiques : nombre de visites par jour, temps de visite moyen par site, nombre de visites par type et 10 mots le plus fréquents dans la navigation.

- Page Data Warehouse

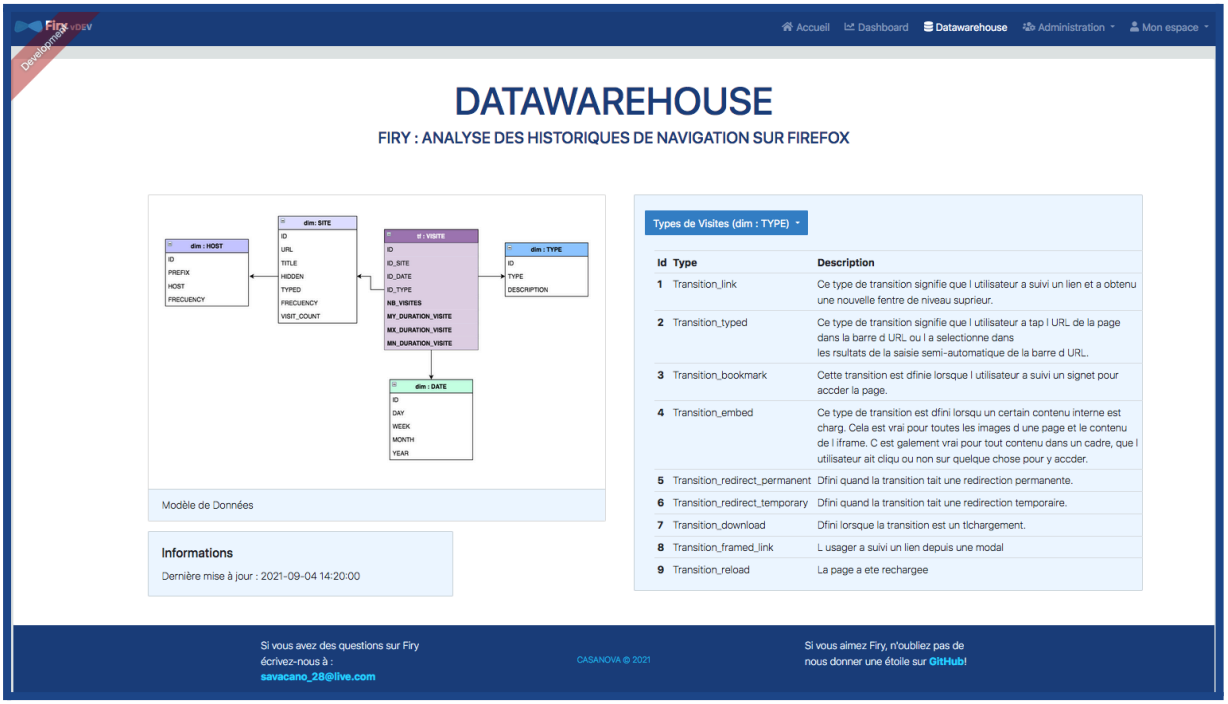

Figure 30 : Page pour visualiser des données depuis le datawarehouse

- API

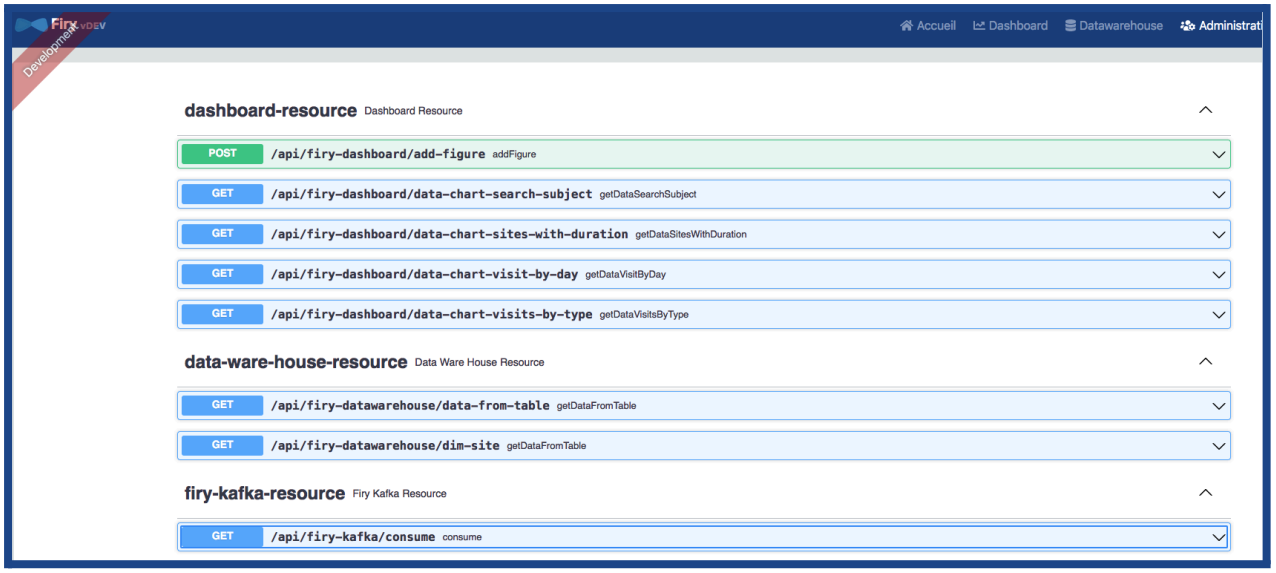

Figure 31 : Page pour appel aux services de FIRY - API

6. Conclusion

Dans ce projet un outil d'analyse en temps réel de l'historique de navigation sur Firefox a été développé. Afin de garantir le traitement en temps réel 3 technologies big data ont été utilisées : *Kafka, Spark et Cassandra.*

Ce projet m'a donc permis de me familiariser avec ces outils qui sont maintenant considérés comme incontournables dans le domaine de la BI / BIG DATA. En effet, ces solutions sont adaptées aux problématiques modernes du traitement / stockage de données : important volume de données, traitement en temps réel... Il s'agit d'outils performants et très puissants mais qui nécessitent un temps d'apprentissage conséquent.

De plus, j'ai également pu utiliser des outils plus classiques mais néanmoins indispensables pour présenter les résultats obtenus : Angular pour les interfaces et le framework Spring (JAVA) pour la partie serveur.

7. Glossaire

A

ANALYSE EN TEMPS RÉEL : Technologies et processus qui évaluent, gèrent et analysent les données que collecte votre entreprise au moment où elles intègrent votre base de données.

ANGULAR : Framework côté client, open source, basé sur TypeScript. Angular est une réécriture complète d'AngularJS. Il permet la création d'applications Web.

B

BI : Ensemble des technologies permettant aux entreprises d'analyser les données au profit de leurs prises de décisions.

BIG DATA : Ensembles de données collectées par les entreprises, pouvant être explorées et analysées afin d'en dégager des informations exploitables ou utilisées pour des projets de Machine Learning.

C

CASSANDRA : Système de gestion de base de données (SGBD) de type NoSQL conçu pour gérer des quantités massives de données sur un grand nombre de serveurs, assurant une haute disponibilité en éliminant les points de défaillance unique. Il permet une répartition robuste sur plusieurs centres de données 4, avec une réplication asynchrone sans nœud maître et une faible latence pour les opérations de tous les clients.

D

DATAVIZ : Outils pour mettre en forme de manière lisible un ensemble de données brutes.

E

ETL : Extract-transform-load. Il s'agit d'une technologie informatique intergicielle (comprendre middleware) permettant d'effectuer des synchronisations massives d'information d'une source de données (le plus souvent une base de données) vers une autre. Selon le contexte, on est amené à

exploiter différentes fonctions, souvent combinées entre elles : « extraction », « transformation », « constitution » ou « conversion », « alimentation ».

ELT : Extract, Load, Transform. L'ELT est une variante de l'Extract, Transform, Load (ETL), un processus d'intégration de données dans lequel la transformation a lieu sur un serveur intermédiaire avant d'être chargée dans la cible. En revanche, les ELT permettent de charger les données brutes directement dans la cible et de les transformer par la suite. Cette capacité est particulièrement intéressante pour le traitement des grands ensembles de données nécessaires à la Business Intelligence (BI) et à l'analyse de données volumineuses.

I

INGESTION DE DONNÉES : Ensemble des phases de recueil et d'importation des données pour utilisation immédiate ou stockage dans une base de données

J

JAVA : Langage de programmation orienté objet et compilé vers une représentation binaire intermédiaire qui peut être exécutée dans une machine virtuelle Java (JVM) en faisant abstraction du système d'exploitation.

JHIPSTER : Générateur d'applications libre et open source utilisé pour développer rapidement des applications Web modernes en utilisant Angular et le framework Spring.

K

KAFKA: Projet à code source ouvert d'agent de messages développé par l'Apache Software Foundation et écrit en Scala. Le projet vise à fournir un système unifié, en temps réel à latence faible pour la manipulation de flux de données. Sa conception est fortement influencée par les journaux de transactions

M

MÉTADONNÉES : Données sur une donnée. Plus précisément, c'est un ensemble structuré d'informations décrivant une ressource quelconque, numérique ou non.

MODÈLE DE DONNÉES : Modèle qui décrit la manière dont sont représentées les données dans une organisation métier, un système d'information ou une base de données.

R

RAFFINAGE DE DONNÉES :chaîne de caractères (littéral).

S

SPRING : Framework open source pour construire et définir l'infrastructure d'une application Java3, dont il facilite le développement et les tests.

SPARK : Framework open source de calcul distribué. Il s'agit d'un ensemble d'outils et de composants logiciels structurés selon une architecture définie. Développé à l'université de Californie à Berkeley par AMPLab3, Spark est aujourd'hui un projet de la fondation Apache. Ce produit est un cadre applicatif de traitements big data pour effectuer des analyses complexes à grande échelle.

8. Bibliographie

[BAGHEL, 2016] *Traffic Data Monitoring Using IoT, Kafka and Spark Streaming, https://www.infoq.com/articles/traffic-data-monitoring-iot-kafka-and-spark-streaming/*

[BORSOS, 2017] *Data Analytics using Cassandra and Spark, https://opencredo.com/blogs/data-analytics-using-cassandra-and-spark/*

[CARTELIS, 2021] *Pipeline de données – Qu'est-ce que c'est , et comment les mettre en place ? https://www.cartelis.com/blog/pipeline-de-donnees-definition-exemples/*

[FALLERY et RODHAIN, 2017] *Quatre approches pour l'analyse de données textuelles: lexicale, linguistique, cognitive, thématique.*

[KAFKA] *https://kafka.apache.org/documentation/#connect*

[MCDONALD, 2019] *Streaming ML Pipeline for Sentiment Analysis Using Apache APIs: Kafka, Spark, and Drill, https://dzone.com/articles/streaming-machine-learning-pipeline-for-sentiment-1*

[NEWTON, 2019] *Intro to Kafka, https://lankydan.dev/intro-to-kafka-consumers.*

[OZENERO, 2020] *Spring Boot + Angular 6 example | Spring Data + REST + Cassandra CRUD example, https://ozenero.com/spring-boot-angular-6-example-spring-data-rest-cassandra-crud-example.*

[QUINTANA, 2016] *Tutoriel pour intégrer Spark et Cassandra, https://zenika.developpez.com/tutoriels/nosql/integration-spark-cassandra/*