Institut National des Sciences Appliquées et de Technologie

UNIVERSITÉ DE CARTHAGE

Projet de Fin d'année

Filière : **GL**

Data Fusion: Un système Big Data pour l'optimisation de l'expérience du conducteur dans la voiture

Présenté par

Chaker Iyadh
Bouchoucha Rached
Abrougui Baha Eddine

Encadrant INSAT : **Mme Sfaxi Lilia**
Encadrant ENTREPRISE : **Mme Besbes Nour**

Présenté le : **Juin 2020**

Année Universitaire : 2019/2020

Remerciements

On a envie d'adresser nos sincères remerciements à ceux qui ont contribué à l'élaboration de ce projet. Je tiens tout particulièrement à remercier Mme Sfaxi Lilia, qui nous a soutenu, encouragé tout au long du semestre, et qui nous a fournit cette opportunité. Nous remercions enfin Faurecia pour nous avoir proposé un sujet assez intéressant grâce à lequel on a pu immerger dans le monde de Big Data et pouvoir participer au développement d'un système dès le début.

Table des Matières

Liste des Figures

Résumé

Notre projet consiste à développer une architecture pour un pipeline de Big Data qui prend des décisions intelligentes en temps réel pour améliorer l'expérience d'un utilisateur de voiture intelligente en traitant, stockant et effectuant une analyse prédictive des flux de données provenant de plusieurs capteurs à l'intérieur de la voiture. Le projet proposé par Faurecia doit résoudre le problème de fusion de données qui présente un besoin fondamental dans le projet de la voiture intelligente en cours de développement par Faurecia puisqu'on a plusieurs capteurs différents et indépendants qui émettent des données variées et à des fréquences élevées et qui doivent passer dans le pipeline pour obtenir un résultat qui permet d'améliorer les conditions de l'utilisateur dans la voiture.

Abstract

Our project is to develop an architecture for a Big Data pipeline that makes intelligent decisions in real time to improve the experience of a smart car user by processing, storing and performing predictive analysis of data flows from several sensors. inside the car. The project proposed by Faurecia must solve the problem of data fusion which presents a fundamental need in the smart car project being developed by Faurecia since we have several different and independent sensors which transmit varied data and at high frequencies. and who must pass through the pipeline to obtain a result which improves the conditions of the user in the car.

Introduction Générale

La voiture qui nous offrait, jusqu'à ces derniers temps, un certain refuge se transforme rapidement en un centre d'information sur roues. De plus en plus, les automobiles sont équipées d'un réseau sophistiqué d'ordinateurs inter-reliés qui sont connectés avec Internet. Certains véhicules surveillent et dressent un rapport sur leurs systèmes internes et l'utilisation du véhicule. D'autres systèmes facilitent le pilotage de certaines fonctions, par exemple la direction et les freins. D'autres encore font partie des systèmes intégrés de navigation, de communication, d'information et de distraction.

Cependant, ces voitures intelligentes reposent principalement sur les données générées par les capteurs, et une bonne gestion et exploitation de ces données est indispensable pour garantir le bon fonctionnement et la minimisation des risques. En effet, les capteurs génèrent une quantité colossales de données et ceci toutes les millisecondes, on peut s'attendre à ce qu'une dizaines de voitures pourra générer ainsi des zétabytes de données dans quelques jours. Plus de leurs tailles et fréquences énormes, ces données sont de différents formats (images, textes, vidéos, audio, etc..), et ici apparait la limite des systèmes de gestion de données classiques, qui se sont rendu utiles pendant tout ce temps là. Ainsi on a eu recours à des systèmes puissants pouvant gérer cette grande quantité de données avec ses formats hétérogènes, et donc répondant à notre besoin, les systèmes Big Data. Ces systèmes assurent la bonne gestion de données et garantissent d'en tirer de la valeur.

C'est dans ce cadre, en collaboration avec la société française Faurecia, que nous avons développé un système Big Data, capable de gérer les données des différents capteurs dans une voiture intelligente, leur exploitation et analyse, et la prise de décisions se basant sur ces résultats afin d'optimiser l'expérience du conducteur et des passagers et ceci en temps réel.

Dans ce rapport, nous allons d'abord aborder la partie théorique expliquant le système et l'architecture choisi, et puis on va immerger dans les détails techniques et discuter le choix des technologies ainsi que l'implémentation du système.

Chapter I

Immersion dans le contexte du Projet

Plan

Introduction

Ce chapitre vise à décrire notre projet d'une façon plus détaillé ainsi à introduire la problématique qui nous a poussé à réaliser ce projet pour remédier à certaines difficultés et de préciser notre contribution.

1 Description de l'entreprise

Faurecia est un groupe français d'ingénierie et de production d'équipements automobiles fondée en 1997.

L'entreprise développe, fabrique et commercialise des équipements destinés aux constructeurs automobiles : sièges, systèmes d'intérieur, technologies de contrôle des émissions.
Couramment, Faurecia travaille sur un projet pour l'optimisation de l'expérience conducteur dans la voiture intelligente, et elle veut faire face au challenge de Data Fusion.

2 Problématique

Le volume et la vélocité des données émises par les capteurs de la voiture sont très importants, ils envoient des valeurs dans de petits intervalles de temps et à des fréquence élevées (par exemple les capteurs de siège envoient plus que 360 mesures par seconde), en plus les données émergeantes sont très variées et hétérogènes. Ces caractéristiques représentent un problème pour les méthodes et les outils classiques de traitement et de stockage de données qui ont montré des lacunes face à des données de plus en plus massives, complexes et a haute vélocité. C'est là que les solutions Big Data peuvent être très utiles. En fait, ces solutions ont prouvé leur efficacité face à d'énormes quantités de données variées et en constante augmentation ainsi que leur fiabilités de traitement des données en temps réel. Le défi est, donc, de trouver une architecture Big Data qui satisfait les exigence précises et qui permet de stocker, traiter et interpréter les données émergentes des différents capteurs de la voiture afin de générer une alertes ou action embarquées en temps réel et à faible latence.

3 Contribution

Faurecia, face au challenge de Data Fusion, qui est devenu de plus en plus difficile à vaincre, puisque les sources de données ne cessent d'augmenter et de manière diverse, a décidé de mettre en jeu un système assez puissant pour gérer ses données massives et diverses, ainsi, dans ce projet on va développer un prototype de l'architecture de ce système, qui va acquérir les données venant des capteurs se trouvant dans les voitures intelligentes, analyser et en trier de la valeur (détection de stress, éventuelle agression, etc) et tout cela en temps réel. Le but est l'optimisation de l'expérience humaine en voiture par l'automatisation des différentes tâches afin de résoudre les problèmes détectés.

4 Étude des besoins

4.1 Besoins fonctionnels

Notre système doit faire un bon usage des données des capteurs afin de détecter les éventuels problèmes et enfin exécuter la bonne action embarquée pouvant remédier à ces problèmes (activer le climatiseur, mettre de la musique, etc) ou bien afficher le bon message suggérant une action (cas de somnolence).

4.2 Besoins non fonctionnels

Dans notre cas, il s'agit d'un système temps réel, il doit ainsi garantir les besoins non fonctionnels de base, la performance ainsi que la disponibilité, de plus le but principal est d'optimiser l'expérience du conducteur, d'où une bonne ergonomie est indispensable, de même que la fiabilité. Comme il s'agit d'un système Big Data, des données massives seront mises en jeu, et on exiges toujours la protection de nos données des éventuelles attaques, d'où on tire que la sécurité est aussi un besoin très important. Pour finir, la scalabilité horizontale tel que verticale doit être assurée par notre système afin de garantir le bon fonctionnement lors de l'augmentation des charges. Ces besoins non fonctionnels seront assurées par les techniques suivantes :

- La performance : Toutes les technologies qui vont être utilisées reposent sur la notion du parallélisme, ainsi si nous disposons d'un cluster composé de plusieurs noeuds, les traitements de notre pipeline seront faits de manière parallèle, d'où on peut exécuter des jobs complexes en un très peu de temps.
- La disponibilité : Le fait que notre cluster est composé de plusieurs noeuds assura la disponibilité permanente de notre système, ainsi si l'un des noeuds tombe en panne, les technique de distribution, réplication et calcul en parallèle permettra le fonctionnement normal du système jusqu'au dépannage du noeud en panne.
- L'ergonomie : Une dashboard devrait être présent dans la voiture avec un design simple et user-friendly.
- La fiabilité : La disposition d'un grand nombre de donnée doit garantir la bonne précision de nos modèle et déduire à des prédictions correctes.
- La sécurité : Toutes les technologies utilisées garantissent la sécurité, on peut même ajouter une autre couche de sécurité et ceci en activant les connexion SSL et mettre en oeuvre un projet Kerberos.

- La scalabilité : Notre système sera fait sur une base d'architecture scalable, qui permet l'augmentation des nombre des noeuds dans un cluster de manière facile et efficace, les technologies utilisées assureront la facilité de la scalabilité horizontale, de même pour la scalabilité verticale.

5 Méthodologie :

Pour ce projet on a adopté la méthodologie Agile Scrum.

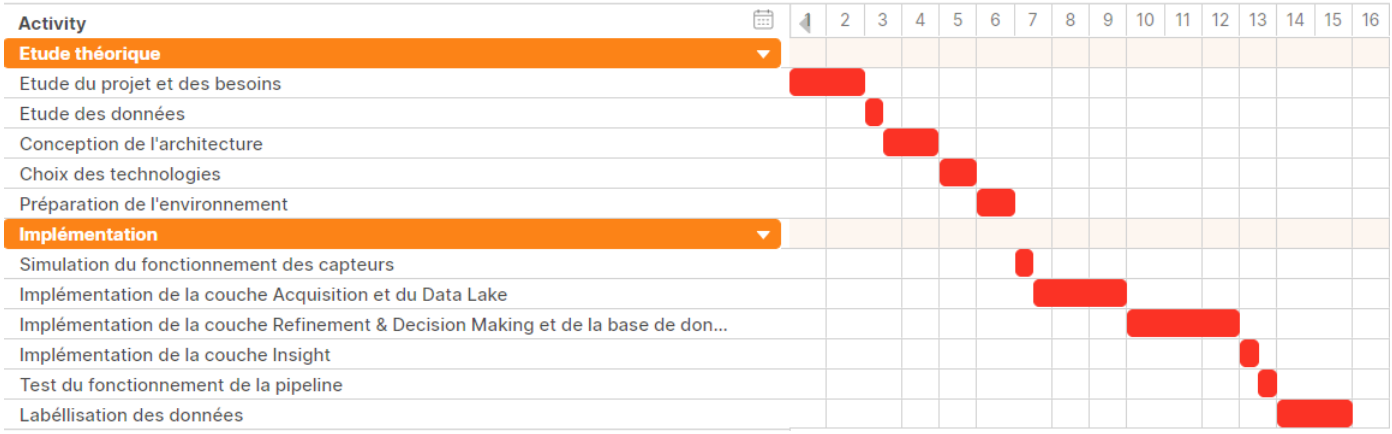

Figure I.1 – Sprints sous forme de diagramme de Gantt (1 carreau représente 1 semaine)

Conclusion

Dans ce chapitre, on a présenté le projet d'une façon claire et détaillé. Nous allons ensuite immerger dans le monde de Big Data et faire une étude théorique du projet.

Chapter II

Étude théorique et architecture

Plan

Introduction

Dans Cette nous allons éclaircir le besoin du Big Data, faire une étude des besoins ainsi qu'une analyse détaillé qui a par suite mené à la conception de l'architecture du projet.

1 Émergence du Big Data

1.1 Comment est-on arrivé là ?

Très souvent les données de l'entreprise proviennent des PGI (Progiciels de données de gestion), GRC (gestion de relation client), commerce électronique (E-Commerce), GCL (gestion de la chaîne logistique), des entrepôts de données (Data Warehouse en anglais) et des bases de données issues d'un développement applicatif interne. La majorité de ces données sont souvent stockées dans un système de gestion de bases de données relationnelles (SGBDR).

Les SGBDR sont toujours au cœur du système d'information des entreprises et les investissements lourds de ces dernières dans ces solutions de stockage rendent leurs remplacements problématiques, et toutes les tentatives ont échoué (à titre d'exemple les bases orientées objet), devant leurs puissances, parce qu'ils ont d'abord de très nombreux avantages que nous allons expliquer dans la partie limitations des bases de données relationnelles , de plus ils sont devenus un référentiel et un mécanisme d'intégration pour la majorité des DSI.

Aujourd'hui, avec l'IOT ainsi que le Web, nous sommes confrontés à une croissance incomparable de la quantité de données que nous traitons, et qui augmente très rapidement. Le volume des données atteindra certainement l'exabytes ou yottabytes, comme le montre la figure ci-dessous.

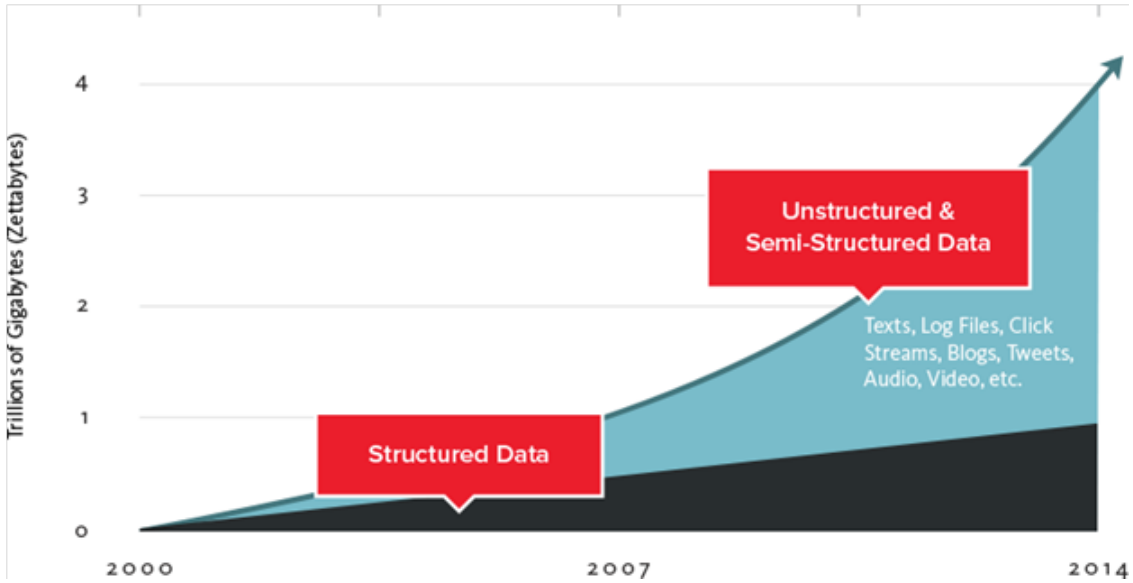

Figure II.1 – Quantité et structure des donnéesen 2014 [5]

Avec les nouvelles sources de données tels que les capteurs, les satellites, les réseaux sociaux, des flux de données énormes et continus, semi-structurés et non structurés sont générés, nécessitant d'être stockés et traités et avec une faible latence, chose ne pouvant pas être réalisée par les SGBDR classiques.

1.2 Les succès de Big Data

La grande majorité des entreprises déclarent que les avantages du Big Data sont considérables. Dans l'enquête "NewVantage Partners", 73,2% des dirigeants ont déclaré avoir vu des résultats commerciaux mesurables de leurs efforts. En outre, les dirigeants qui ont répondu non pensent qu'il est trop tôt pour dire quel impact ces investissements auront sur leur entreprise. Parmi

ces avantages on cite: meilleur prise de décision, augmentation de la productivité, détection de fraude, meilleur revenu, meilleur innovation, meilleur "time de market", etc.

1.3 Les enjeux du Big Data et cas d'usage

Désormais dans la société de la connaissance (Knowledge society), les entreprises doivent être proactives et agiles dans un monde instable au lieu d'être réactives au détriment des enjeux économiques et concurrentiels ardus. Or les lourdeurs des outils classiques (SGBDR, BI) ne permettent pas de suivre les nouveaux besoins du marché.

On pense modestement que l'enjeu majeur pour les entreprises est un enjeu économique, car les entreprises stockent plusieurs données de leurs clients qui ne sont pas exploitées. Ces données engorgent de l'argent, et si les entreprises arrivent à proposer des services innovants et personnalisés pouvant répondre aux besoins de clients, ils peuvent représenter une source de valeur. On va illustrer cela à travers quelques cas d'usage du big data, pour créer de la valeur. Notant que toutefois le big data est utilisé dans plusieurs autres contextes.

- La grande distribution : Fait usage du big data pour faire le suivi des émissions des tickets de caisse, le flux de marchandises entre ses fournisseurs, entrepôts et magasins, et le parcours utilisateur sur son site e-commerce .

- Open data : le gouvernement met en avant des banques de données, qui permettent à des entreprises de proposer des services innovants, à titre d'exemple l'application citymapper , qui utilise des données fournies par l'agglomération lyonnaise, pour proposer des services dans le domaine du transport , et qui donne des résultats très satisfaisants.

Le tableau suivant dresse d'autres utilisations du Big Data, ainsi que les gains.

Gains	Entreprises ayant constaté un gain
Meilleure habilité à prendre des décisions stratégiques	69 %
Meilleure gouvernance opérationnelle	54 %
Meilleure connaissance et amélioration de l'expérience utilisateur	52 %
Réduction des coûts	47 %
Accélération des décisions	44 %
Développement d'un nouveau produit/service	43 %
Meilleure connaissance du marché et des concurrents	41 %
Développement d'un nouveau business model	38 %
Augmentation des revenus	35 %
Automatisation des décisions	24 %

Figure II.2 – Gains constatés par la mise en place des solutions big data [5]

2 Les systèmes Big Data

2.1 Processus de traitement des applications Big Data

Les applications Big Data appliquent un processus de traitement sur les données passant par différentes phases présentées de manière abstraite dans la figure ci-dessous. Nous allons décrire succinctement chacune de ces étapes dans ce qui suit.

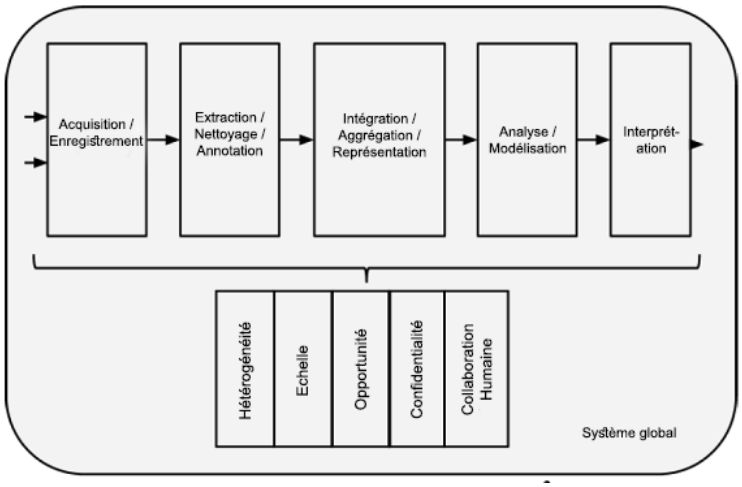

Figure II.3 – Différentes stages de traitement des données

- **Acquisition/Enregistrement** : correspond à la procédure d'acquisition des données Big Data à l'instar de la capture de la température ou l'estimation du taux de pollution dans l'air a travers les objets connectées. Compte tenu du gros volume des données recueillies, cette phase devra éliminer certaines données inutiles grâce à des filtrages et des compressions. Seulement, elle devra faire attention a ce que des informations significatives ne soient pas écarté telles que les données aberrantes qui puissent refléter des pannes ou des fraudes. Ce stage devra également assurer la génération des méta-données sur la structure et la provenance des données, mais également sur les détails de l'opération de capture. Les méta-données auront une importance capitale pour la suite des phases, plus particulièrement, l'analyse des données

  .

- **Extraction/Nettoyage/Annotation** : souvent, les données capturées se trouvent dans un format inadapté à l'analyse. Cette phase s'occupe de corriger leur structure et d'extraire l'information significative, mais également d'éliminer les données potentiellement erronées. En effet, le critère de véracité du Big Data stipule que les données sont parfois indignes de confiance et doivent être épurées avant l'analyse.

- **Intégration/Agrégation/Représentation** :les analyses à grande échelle font appel à des ensembles de données différents en structure et en taille. Un défi important correspond à trouver la représentation la plus adéquate pour les stocker et à intégrer ces ensembles entre eux de façon à conduire une analyse globale.

- **Analyse/Modélisation** : il s'agit de l'analyse des données afin de déceler des modèles intrinsèques, d'extraire des relations et des connaissances, mais aussi de corriger les erreurs et d'éliminer les ambiguïtés.

- **Interprétation** : les décideurs doivent interpréter les résultats d'une analyse Big Data. Cette interprétation est obligatoire, car les données et par conséquent l'analyse elle-même ne sont pas exemptes d'erreurs. De plus, la plupart des modèles et théorèmes appliqués se basent sur des hypothèses qui ne sont pas toujours vérifiables. Les décideurs devront valider les résultats en retraçant les opérations effectuées. Des outils doivent être mis en place afin de faciliter ce processus. Ils doivent offrir des visualisations interactives des données, permettre de retracer leur provenance et d'appliquer des modifications dessus puis voir l'impact sur les résultats en temps réel.

2.2 Propriétés

- **La Scalabilité** : Dans leur tentative d'aborder le Big Data, les nouvelles technologies s'efforcent à satisfaire une propriété primordiale qui est la scalabilité. On entend par cela la capacité d'un système à améliorer ses performances en augmentant la taille ou le nombre de ses ressources lorsqu'il fait face à une charge plus grande. En pratique, on retrouve deux approches dites scalabilité verticale et son analogue horizontale. La première est réalisée en augmentant la taille du système et la puissance de ses composants (RAM, CPU...). Par contre, la scalabilité horizontale se manifeste sous la forme d'un Cluster. Il s'agit d'un système distribué composé de plusieurs machines de capacité modérée appelées nœuds. Ces machines ou nœuds communiquent dans le but de réaliser certaines opérations et manipuleront chacune une partie de la charge imposée au système adoptant ainsi la politique diviser pour régner . Ladite charge peut représenter une problématique de stockage d'une grande masse de données ou leur traitement. La figure ci-dessous résume le concept.

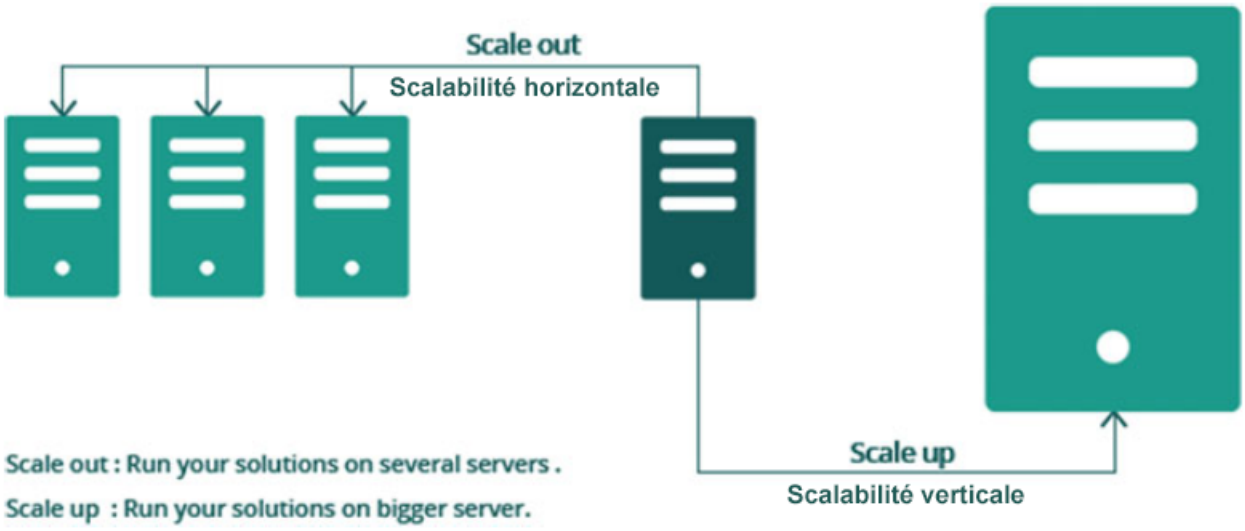

Figure II.4 – Scalabilité verticale et horizontale [5]

- **Théorème CAP** : Même si les systèmes distribués, qu'ils soient ou non hébergés sur le cloud, semblent être la solution idéale pour gérer les applications Big Data, ils apportent un lot de contraintes énoncées dans le théorème CAP (Consistency, Availability, Partition tolrance). Celui-ci déclare qu'il est impossible pour un système distribué de garantir les trois propriétés suivantes simultanément.

- Consistance : le système retourne la bonne réponse à chaque requête. À comprendre, une réponse sans incohérence ni erreurs. La définition exacte de la correction de la réponse dépend du service rempli par le système.
- Disponibilité : le système est tout le temps disponible et répond à tout moment à ses utilisateurs. En d'autres mots, toutes les opérations sont exécutées avec succès au bout d'un temps fini.
- Tolérance aux partitions réseau : dans ce genre de systèmes avec des machines à puissance moyenne, les pannes réseau sont inévitables. Lorsqu'elles occurrent, elles créent des partitions de telle sorte que les machines à l'intérieur d'une même partition peuvent communiquer entre elles, mais sont isolées des autres. Cette propriété stipule que l'existence d'un tel partitionnement ne doit pas empêcher ou altérer le bon fonctionnement du système.

2.3 Les notions conceptuelles

- Le paradigme MapReduce : MapReduce est un paradigme de programmation qui permet de distribuer des traitements parallèles sur des volumétries de données dépassant typiquement 1To, dans un cluster composé de centaines, voire de milliers de nœuds, avec une architecture de type maître/esclave, grâce à la séparation des données et des traitements. Il est composé de 4 grandes phases, "split", "map", "shuffle and sort" et "reduce".

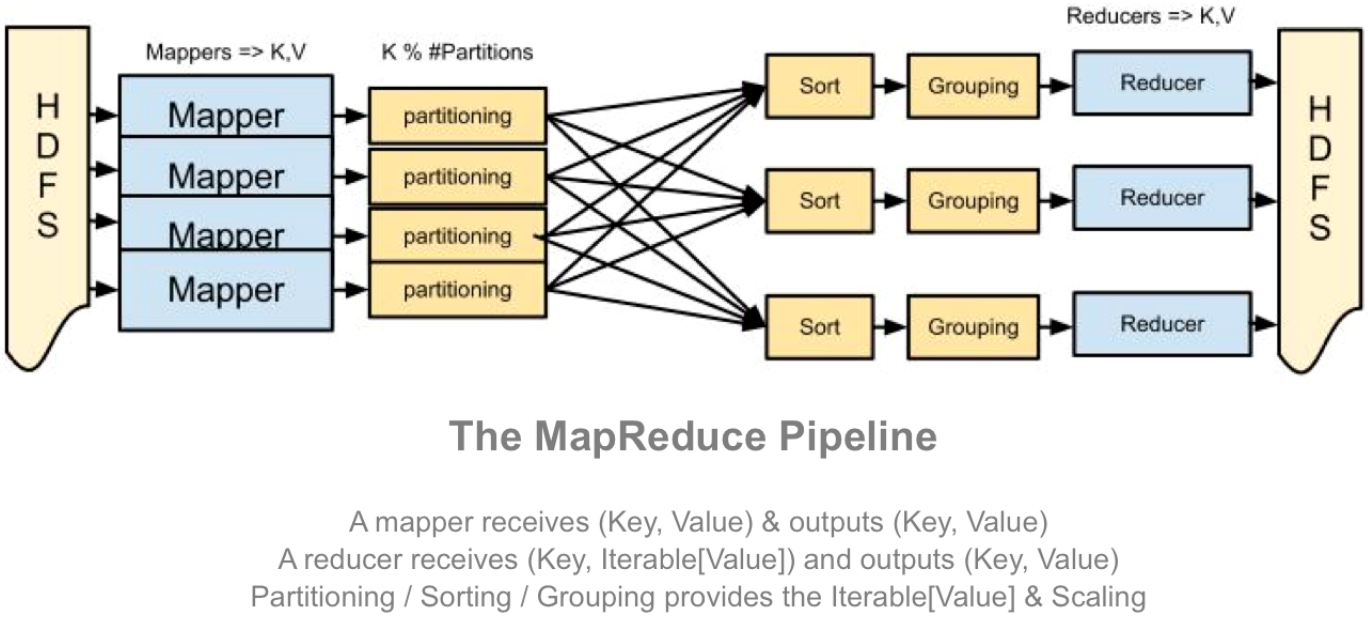

Figure II.5 – Framework MapReduce [5]

- Le traitement par lot : Le traitement par lots (Batch) est un traitement automatique et sans intervention humaine, avec un temps d'exécution élevé.

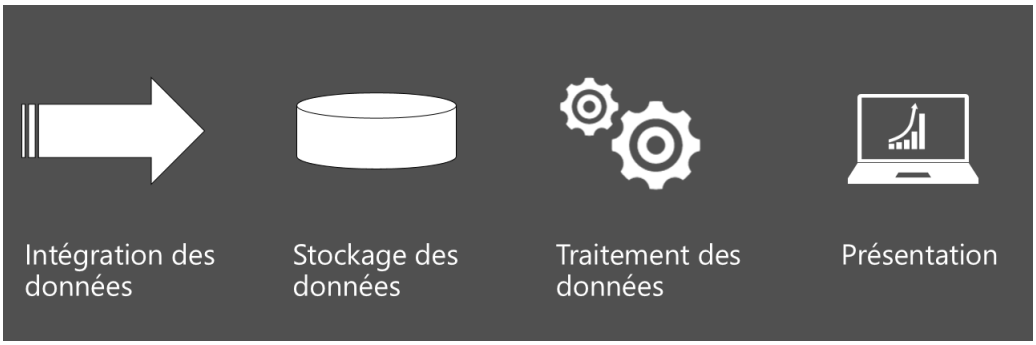

Figure II.6 – Traitement par lot [5]

- Le traitement temps réel : Le traitement temps réel (streaming) est capable de prendre en compte les contraintes temporelles, pour délivrer des résultats exacts avec le respect d'une latence très faible.

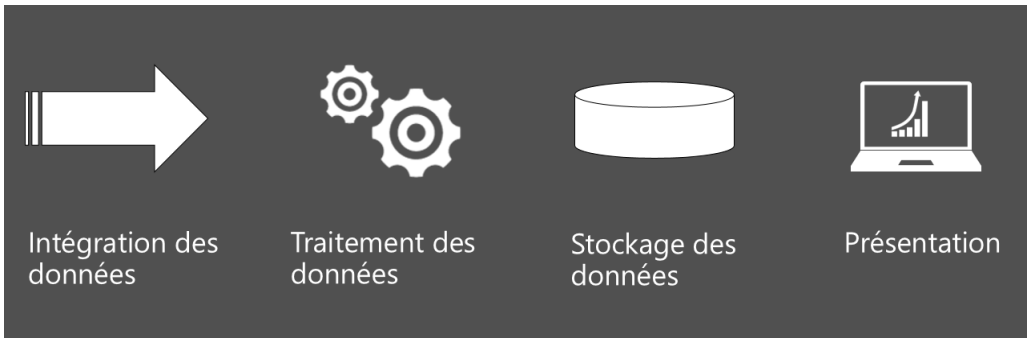

Figure II.7 – Traitement temps réel [5]

2.4 Architectures existantes

Il existe aujourd'hui un nombre important d'architectures big data, l'architecture Lambda, l'architecture Kappa ou l'architecture Zeta.

- L'architecture Lambda : L'architecture Lambda est indépendante de la technologie, et se base sur le précalcul des résultats, puis à les récupérer dans une base et les envoyant au demandeur. Elle est composée de trois couches : Couche Batch, Couche de vitesse ou temps réel (Speeding) et couche de Service (Serving).

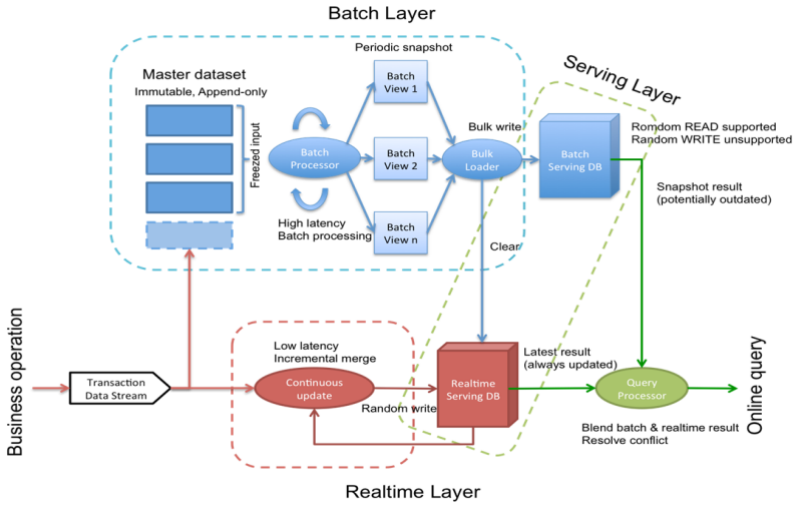

Figure II.8 – Architecture Lambda [5]

- L'architecture Kappa : L'architecture Kappa, permet de simplifier l'architecture Lambda, en fusionnant la couche batch et la couche Speeding. L'architecture Kappa n'est pas destinée au stockage des données, mais uniquement à leur traitement, comme le montre le schéma suivant :

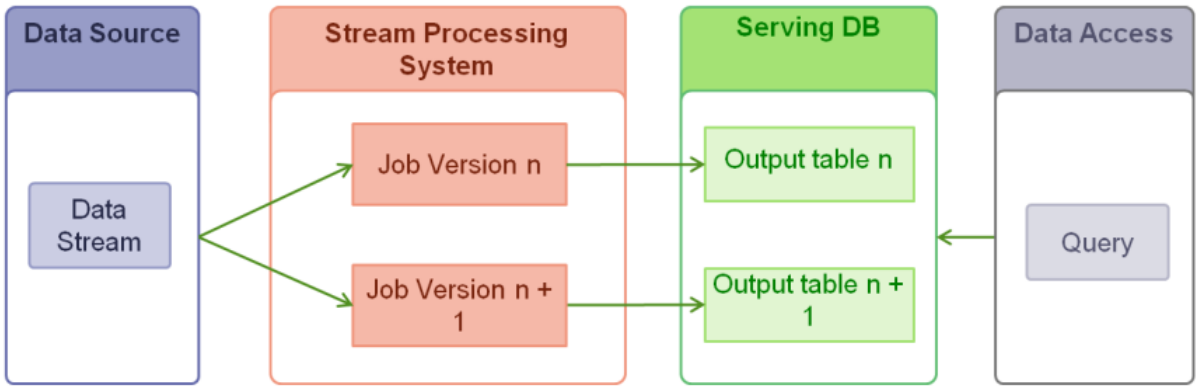

Figure II.9 – Architecture Kappa [5]

3 Les différentes critères de choix de l'architecture

En général, il faut suivre les règles suivantes :

- **Le type de traitement :**
  Le type de traitement dans ce projet est principalement un traitement en streaming, les données émises par les capteurs doivent être interprétées en temps réel, càd avec faible latence afin d'éviter les éventuels risques. Cependant, un traitement par lots (Batch processing) est indispensable, et ce dans la partie amélioration de la précision du modèle. (exemple : labélisation du Data Lake et training). Ainsi une architecture se basant sur l'architecture Lambda sera mise en œuvre.

- **L'utilisateur final des données :**
  L'utilisateur sera l'être humain se trouvant dans la voiture, conducteur ou bien passager, ainsi il faut accorder une grande importance à la visualisation des données, car l'être humain comprend mieux le graphique que le texte.

- **La source des données :**
  La source de données sera principalement les différents capteurs se trouvant dans la voiture. On pourra avoir recourt à une autre source de données pour améliorer le training des modèles, mais pour le moment on se limitera aux données provenant des capteurs.

- **Format du contenu :**
  Le format peut être soit structuré, semi-structuré, non structuré ou bien une combinaison de ces trois. Ceci sera décidé lorsque les données seront disponibles.

- **Type de données :**
  Les données à manipuler seront principalement des données physiologiques.

- **Fréquence et taille de données :**
  Les données seront générées en temps réel, on pourra s'attendre à un grand flux de données arrivant toutes les secondes. La taille de données dépendra du nombre de voitures en production.

- **Méthodologie de traitement des données :**
  On aura recourt à une analyse prédictive des données, de plus à leurs exploitation pour le training afin d'obtenir des modèles plus fiable et robuste, avec une génération d'action.

- **Choix du matériel :**
  Ce type d'architecture ne requiert pas un type de matériel particulier, et les machines bon marché sont bien adaptées. On doit choisir le matériel et le système d'exploitation en fonction de vos préférences et des coûts.

4 Architecture proposée

4.1 Aperçu

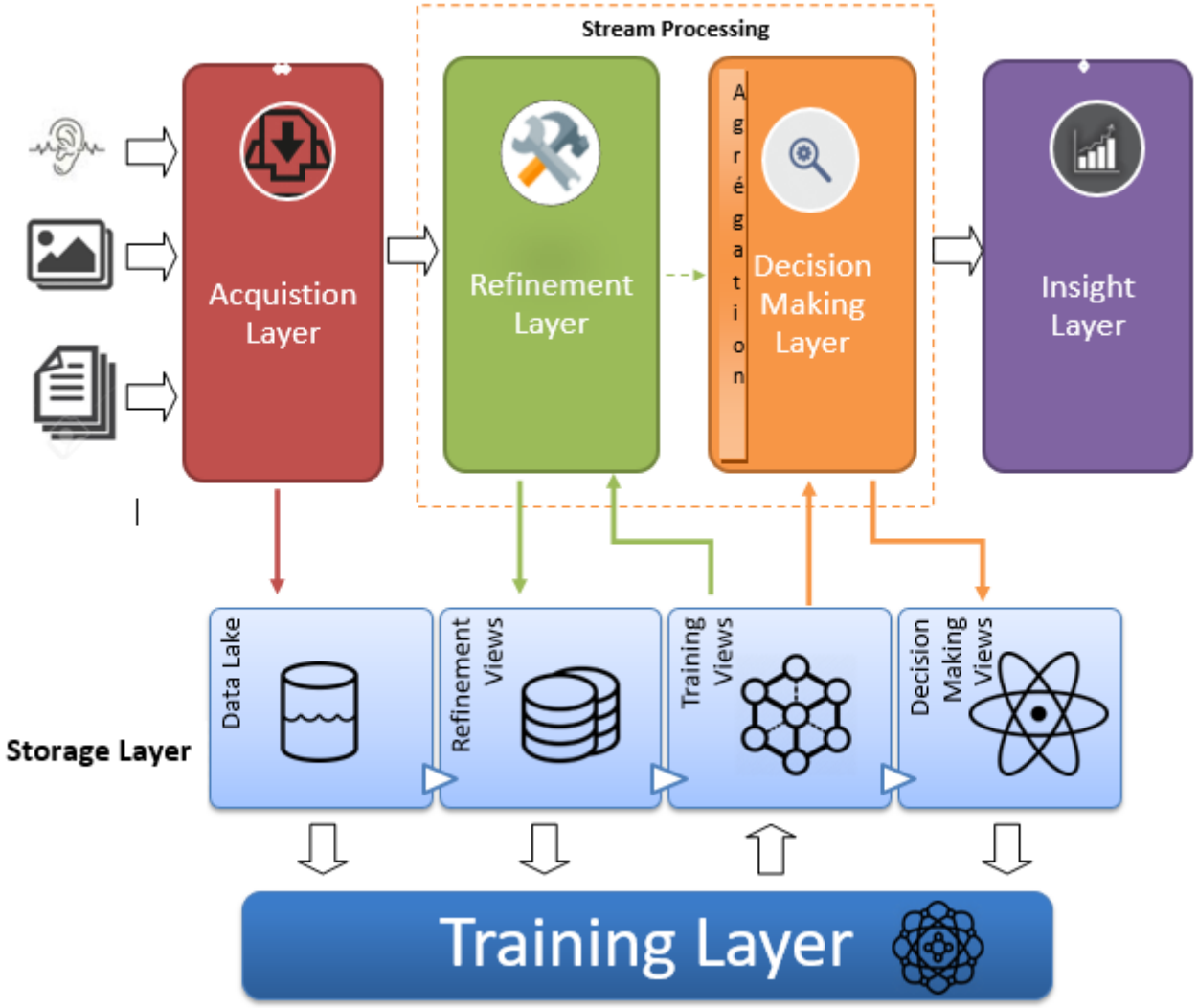

Figure II.10 – L'architecture proposée

4.2 Explication des différentes couches

- **La couche "Acquisition" :** C'est la couche reliée directement aux capteurs, elle se chargera de recevoir les différentes données, et de diriger ce flux de données vers la couche adéquate, que soit la couche de refinement ou bien celle de stockage. La couche devra traiter l'acquisition des différents formats de données, et ceci en temps réels. Elle les convertit aussi à un format précis pour faciliter le stockage ou bien l'exploitation

- **La couche "Refinement" :** C'est dans cette couche que le pre et post processing auront lieu. Dans une première partie, un simple pre-processing sera mise en œuvre afin de transformer les données vers le format adéquat pour leur utilisation pour le post-processing , qui s'agit d'exploiter les modèles résultants de training afin de faire des traitement complexes de machine learning afin de générer les inputs nécessaire pour la couche décision making. De plus, la couche refinement va s'occuper aussi de stocker les résultats obtenus dans les refinement views, et ceci pour une future utilisation pour l'amélioration des modèles.

- **La couche "Decision Making" :** Cette couche va rassembler les différents résultats de la couche précédente (il s'agit d'une agrégation), et ce pour leurs exploitation dans traitement qui peuvent être simple (des règles d'inférence de type si A et B alors C) ou bien complexes (qui exigent l'utilisation des modèles de deep learning ou autre) afin de détecter l'action à prendre dans les différentes situations. La couche de decision making va aussi stocker les résultats dans les vues adéquates, et ceci, toujours, pour une amélioration de nos modèles.

- **La couche "Insight" :** La couche insight communique directement avec l'utilisateur, par un messages (alertes, images, son, etc) ou par une prise d'action au sein de la voiture (freinage, ouverture de porte, etc). Pour obtenir des meilleurs modèles intelligents, on peut mettre en place un système de feedback, qui apparait quelques minutes après la prise d'action demandant à l'utilisateur de confirmer l'alerte (si la baisse de température était favorable, s'il s'agit de fausse alerte, etc). Les résultats de feedback, vont être agrégés au résultat stockés par les différentes couches.

- **La couche "Storage" :** C'est la couche dans laquelle les données seront présente

  - *Data Lake :* Stocke les données brutes telles qu'elles sont venant de la couche d'acquisition, et un traitement par lots sera lancé périodiquement pour la labélisation de ces données. (NB : la labélisation peut aussi se faire d'une manière manuelle).

  - *Refinement Views :* Les vues dans les quelles seront stockées les données résultant de la couche refinement .
  - *Training Views :* Un contexte abstrait, stockant les modèles intelligents résultant du training.
  - *Decision Making Views :* Les vues dans les quelles seront stockées les données résultant de la couche decision making.

- **La couche "Training" :** C'est une couche travaillant en mode batch, elle exploite les différentes données labélisées, afin de produire des modèles intelligents fiable qui vont servir pour la prédiction dans les différentes couches.

5 Flux des données dans le système

Dans cette partie on va représenter les donner entrant et sortant de chaque couche ainsi que leur type :

- **La couche "Acquisition" :**
  - **Flux entrant :**

CAPTEUR	DESCRIPTION	FREQUENCE
AirQuality	Donne des informations sur la qualité de l'aire dans la voiture par exemple : Taux d'humidité, Taux de CO2 ...	1 ou 2 mesures par 2 secondes
AW	C'est l'ensemble des capteurs du siège	Environ 360 mesures par secondes
Camera	Mesures physiologique du conducteur par exemple : Ouverture des yeux, Direction de tête/yeux, Dilatation des pupilles ...	1 mesure à chaque frame 30 frames par secondes
EMPATICA	Mesures physiologique du conducteur	6 capteurs Entre 1 et 64 Hz
Zéphyr	Mesures physiologique du conducteur	4 capteurs Entre 1 et 250 Hz

Figure II.11 – Flux des données entrant à la couche Acquisition

  - **Flux sortant :**
    - Stream des mesures vers la couche Refinement .
    - Accumulation des données pour stockage dans le Data Lake.

- **La couche "Refinement" :**
  - **Flux entrant :**
    - Stream des mesures des capteurs.
    - Modèles d'IA pour la détection des caractéristiques.
  - **Flux sortant :**
    - Données prétraitées, nettoyées, synchronisées et classifiées détectant les éventuelles caractéristiques (Stress, confort, angoisse, etc.) vers les vues de Refinement .
    - Les caractéristiques détectées vers la couche Decision Making .

- **La couche "Decision Making" :**
  - **Flux entrant :**
    - Caractéristiques détectées (Stress, confort, etc...)
    - Modèles d'IA pour l'agrégation et la prise de décisions.
  - **Flux sortant :**
    - Caractéristiques agrégées et labélisées avec la décision prise pour stockage dans les vues de Decision Making .
    - Décision pouvant être le code d'une action embarquée à exécuter ou bien un simple message vers la couche Insight .

- **La couche "Insight" :**
  - **Flux entrant :**
    - Décision prise.
  - **Flux sortant :**
    - Interface utilisateur / Alerte ou action embarquée.

- **La couche "Storage" :**
  - **Flux entrant :**
    - Données brutes vers le Data Lake

    * Données prétraitées, nettoyées, synchronisées et classifiées suivies des caractéristiques détectées vers les vues de Refinement.
    * Caractéristiques agrégées et labélisées avec la décision prise pour stockage dans les vues de Decision Making .
    * Modèles d'IA vers les vues de Training.
  - **Flux sortant :**
    * Modèles AI vers la couche "Refinement".
    * Modèles AI vers la couche "Decision Making".
    * Données traitées et mises dans le format adéquat vers la couche Training (Vu qu'on se dispose d'un stockage "schema-on-read"

- **La couche "Training" :**
  - **Flux entrant :**
    * Données traitées et labélisées.
  - **lux sortant :**
    * Modèles d'IA.

Conclusion

On a pu citer les notions importantes du Big Data, fait une analyse détaillée du projet et par suite expliqué le choix et la conception de l'architecture qui sera adaptée tout le long du projet.

Chapter III

Étude technique et implémentation

Introduction

Dans cette partie on va étudier en premier lieu le noyau et l'écosystème Hadoop dans laquelle on va présenter aussi les différentes distributions HADOOP existantes et faire une comparaison entre eux pour expliquer notre choix, puis on va analyser en détail l'implémentation et le fonctionnement ainsi que les technologies utilisé dans chaque couche de l'architecture et les résultats obtenus après les tests.

1 La solution Big Data: Hadoop

Dans cette partie nous allons présenter les différentes solutions techniques pour le big data.

1.1 Le noyau d'Hadoop

Hadoop est la plateforme logicielle open source de la fondation Apache permettant de répondre aux besoins du big data, à savoir la volumétrie, la véracité et vélocité des données, ainsi qu'au paradigme de traitement MapReduce pour avoir une meilleure vitesse d'exécution d'algorithme. Elle permet de stocker et d'effectuer des traitements sophistiqués sur des données de différents types et différentes provenances à grande échelle et à moindre coût.
Grâce aux différentes contributions, Hadoop est aujourd'hui un écosystème complexe. En effet le cœur (Kernel) de Hadoop est basé sur le système de fichier HDFS (Hadoop Distributed File System) et le paradigme MapReduce.
On a également une multitude d'outils autour de ce noyau tels que : Zookeeper, Hive, Pig, etc.

1.2 L'écosystème d'Hadoop

1.2.1 Les différents outils de l'écosystème

L'écosystème Hadoop est une plate-forme ou un framework qui aide à résoudre les problèmes de Big Data. Il comprend différents composants et services (ingestion, stockage, analyse et maintenance) à l'intérieur. La plupart des services disponibles dans l'écosystème Hadoop sont

destinés à compléter les quatre principaux composants de base d'Hadoop, notamment HDFS, YARN, MapReduce et Common.

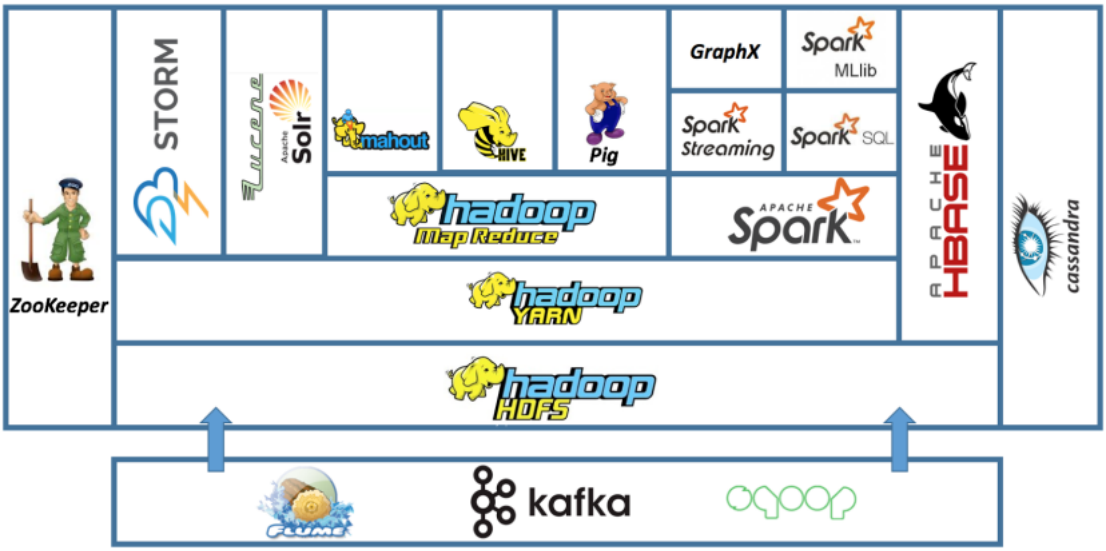

Figure III.1 – L'écosystème HADOOP [5]

1.2.2 Les distributions Hadoop

1.2.3 Distributions existantes

Hadoop est un système complet, il comporte plusieurs briques logicielles, plusieurs contributeurs et plusieurs mainteneurs, ce qui entraine plusieurs cycles de version pour les outils. En conséquence une maitrise des versions compatibles entre elles est très complexe, cela nécessite également une configuration entre les composants, également très complexe et présente de grandes difficultés à l'installation.
De ce fait, il existe aujourd'hui plusieurs distributions qui se positionnent (comme Hortonworks ou HDP, MapR ou Cloudera), afin de faciliter le déploiement de Hadoop. Une distribution est un ensemble de briques d'Hadoop prépackagées, mises ensemble, et configurées pour fonctionner ensemble. L'intérêt d'un package complet c'est qu'il permet de démarrer plus facilement, plus rapidement et d'avoir un support. Parmi ces distribution on peut présenter :

- **Cloudera** ou **CDH**
  a été créée par divers experts de différents acteurs du web (Oracle, Facebook, Google, Yahoo), elle a une seule version qui est libre et ils ont développé quelques composants spécifiques qui ont été donnés à la fondation Apache notamment Cloudera Search, Hue

et Impala. Cloudera est spécialisée dans les offres commerciales de support, formations et certifications.

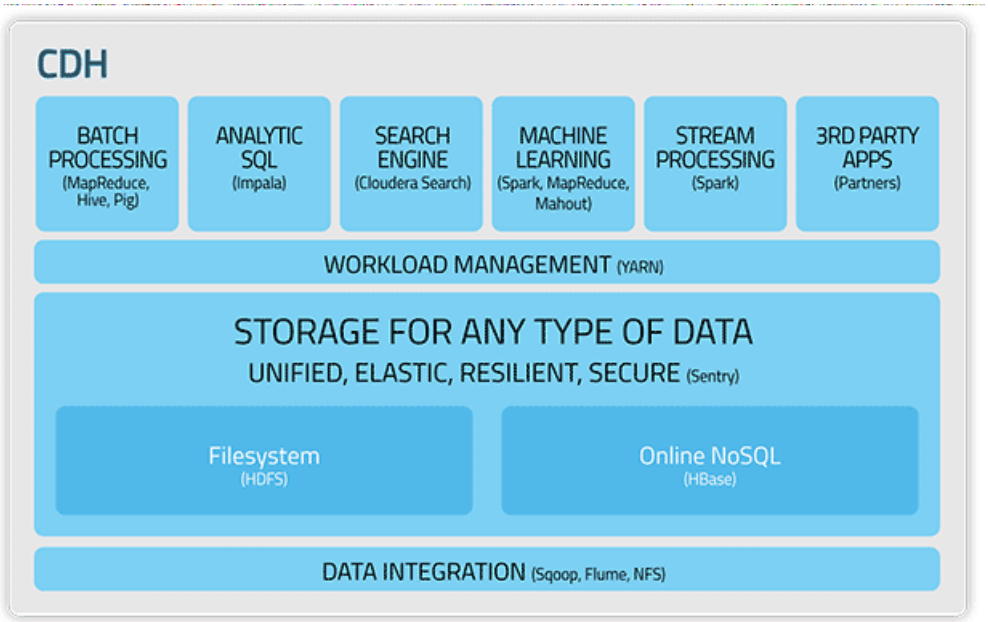

Figure III.2 – Distribution Cloudera [5]

1.2.4 Distribution choisie

Comparatif des distributions des principaux pure player d'Hadoop			
	Cloudera	Hortonworks	MapR
Année de création	2008	2011	2009
Positionnement	Premier à avoir dégainé, Cloudera propose à côté du package Hadoop des outils pour faciliter configuration et administration.	Hortonworks a fait le choix du "tout open source". Sa distribution ne comprend que des composants Apache.	Distribution la plus éloignée du projet Apache, qui s'appuie sur son propre système de gestion de fichiers
Points forts	N°1 mondial, composants premium, configuration du cluster par cas d'usage	Documentation riche, communauté dynamique, support de Windows, outil de sécurité performant	Rapidité, robustesse et performance de la plateforme
Points faibles	Plus lent que MapR, orientation grands comptes	Plus lent que MapR, et manque de stabilité	Coût relativement élevé, interface austère, absence d'outil de sécurité
Offres et tarifs	Une édition gratuite, quatre versions payantes. Licence tarifée au nœud. De 4 000 à 10 000 dollars par an.	HDP (Hortonworks Data Platform) et HDF (Hortonworks DataFlow). Seul le support est payant.	Une édition gratuite (Converged Community), une payante (Converged Enterprise). Licence tarifée au nœud.
Partenaires	Oracle, HPE, NetApp, Intel et Cisco	Red Hat, Microsoft, SAP et Teradata	EMC, Google, Cisco et Amazon Web Services
Références en France	Axa, PMU, Faurecia, Solocal, Saint-Gobain	EDF, BNP Paribas, Société Générale, Banque de France	Darty, Crédit Agricole, Cdiscount

Figure III.3 – Comparaison entre Cloudera, HortonWorks et MapR [5]

Selon les comparaisons on peut remarquer que Cloudera est un pionnier dans le monde de Big Data. Il est la distribution Hadoop la plus utilisée dans le monde et c'est à grâce à son architecture stable et complémentaire qui comporte des outils comme Cloudera Manager qui automatise et facilite le déploiement et la supervision des clusters Hadoop ainsi que ces services

1.2.5 Dockerisation

Afin de garantir une monté en charge matérielle facile et efficace, on a eu recourt à la solution Docker, nous avons fait de sorte que les différents noeuds (machines) composant notre cluster Big Data seront des conteneurs Docker, ainsi la montée en charge se manifestera par un simple clonage d'une image Docker, par exemple si on a besoin d'augmenter le nombre des "Data Nodes" Hadoop, un simple clonage d'un noeud déjà existant fera l'affaire. Pour le moment, nous ne disposons que d'un seul conteneur, ainsi un seul "Data Node" Hadoop, puisque nous somme entrain de développer un prototype. Lors du déploiement, une augmentation du nombre de conteneurs et une séparation des couches sera mise on oeuvre.
L'environnement dans lesquel les conteneurs seront déployé sera une machine Linux, ceci est dû au fait de la présence d'un nombre minimal de couches et donc meilleures performances.
Voici une image justifiant le choix du docker au lieu d'une machine virtuelle classique et de la distribution Linux.

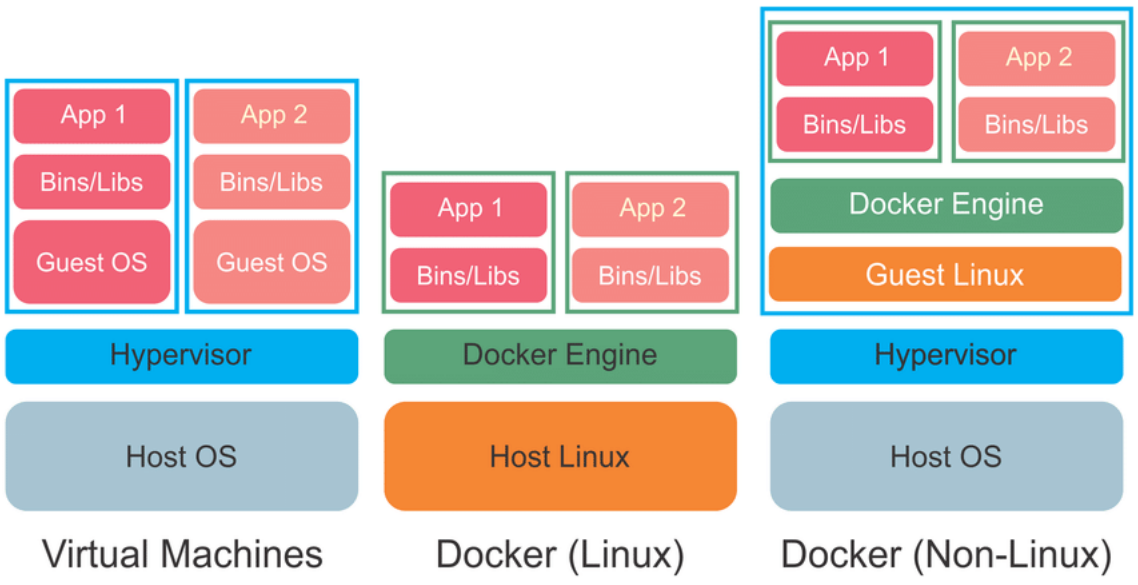

Figure III.4 – VM vs Docker(Linux) vs Docker(non-Linux) [5]

2 Descriptions détaillée de l'implémentation de l'architecture

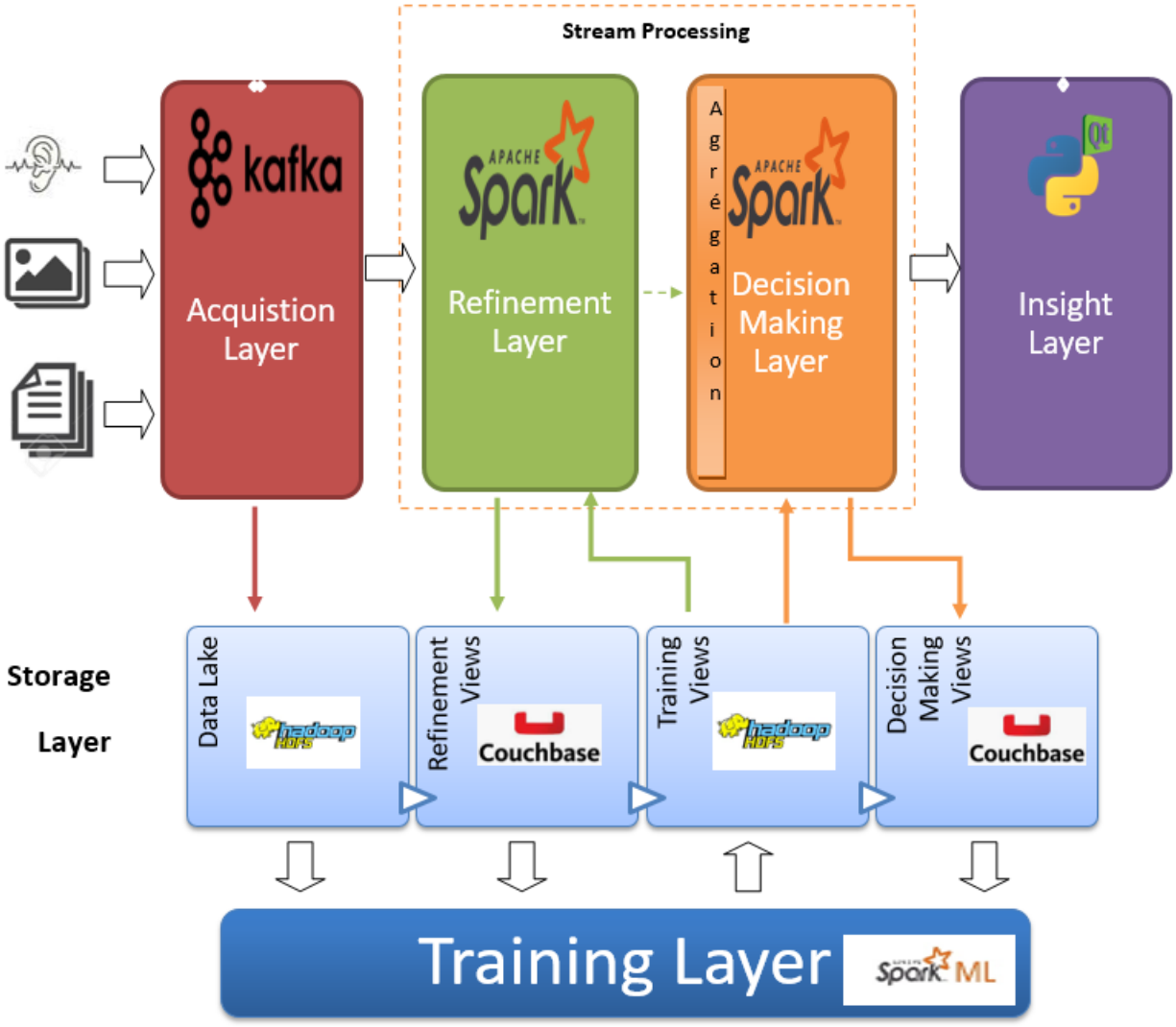

Figure III.5 – Technologies utilisées dans l'architecture

2.1 Simulation du fonctionnement des capteurs

Les capteurs se trouvant dans les voitures doivent être connecté via Internet à des serveurs chez Faurecia, ils doivent envoyer des données en temps réel tant que la voiture est en marche, cependant, vue que cette étape est indispensable pour le test, et qu'on dispose pas réellement des voitures pour tester, on fait une simulation des données collectées qu'on se dispose.
Il s'agit d'un projet python, chaque classe python est un producteur Kafka d'un certain capteur,

cette classe se charge de lire les données collectées, qui peuvent être sous forme de csv ou bien edf (European Data Format) comme le montre la figure ci dessous, puis de les distribuer dans la partition adéquate du topic Kafka, et ceci en respectant les mêmes délais de l'émission des capteurs des voitures.

	Time	Sequence	Year	Month	Day	Hour	Minutes	Seconds	Engine_Load_[
2	1581078213	0	2020	02	07	13	23	33	0
3	1581078213	0	2020	02	07	13	23	33	0
4	1581078213	0	2020	02	07	13	23	33	0
5	1581078213	0	2020	02	07	13	23	33	0
6	1581078213	0	2020	02	07	13	23	33	0
7	1581078213	0	2020	02	07	13	23	33	0
8	1581078213	0	2020	02	07	13	23	33	0

Figure III.6 – Exemple de données csv

```
# EdfVersion=4.0
# Date=2020-02-07T13:26:24.588002+01:00
# ApplicationName=ControlCenter
# ApplicationVersion=1.18.0
# FilePath=C:\Users\Operator\data_logging\2020-02-07_13-26-04-SGP30_8373181.edf
# HostName=HVXXF12
# OperatingSystem=Windows-7-6.1.7601-SP1
# SensorId=8373181
# Type=float64,Unit=s,Format=.1f        Type=string
Type=float,Signal=Ethanol_Signal,Device_Serial_Number=EKS2-340074000F51363132363937,Sensor_Serial
Type=float,Signal=H2_Signal,Device_Serial_Number=EKS2-340074000F51363132363937,Sensor_Serial_Numb
Type=float,Signal=CO2_eq,Device_Serial_Number=EKS2-340074000F51363132363937,Sensor_Serial_Number=
Type=float,Signal=TVOC,Device_Serial_Number=EKS2-340074000F51363132363937,Sensor_Serial_Number=83
Epoch_UTC       Local_Date_Time SOUT_ETOH_SGP30_8373181 SOUT_H2_SGP30_8373181
C_CO2eq_SGP30_8373181   C_TVOC_SGP30_8373181
1581078365.3    2020-02-07T13:26:05.300000+01:00        15730.0 11754.0 400.0   0.0
1581078366.3    2020-02-07T13:26:06.300000+01:00        16112.0 12316.0 400.0   0.0
```

Figure III.7 – Exemple de données edf

Voici la structure du projet de simulation des données des capteurs.

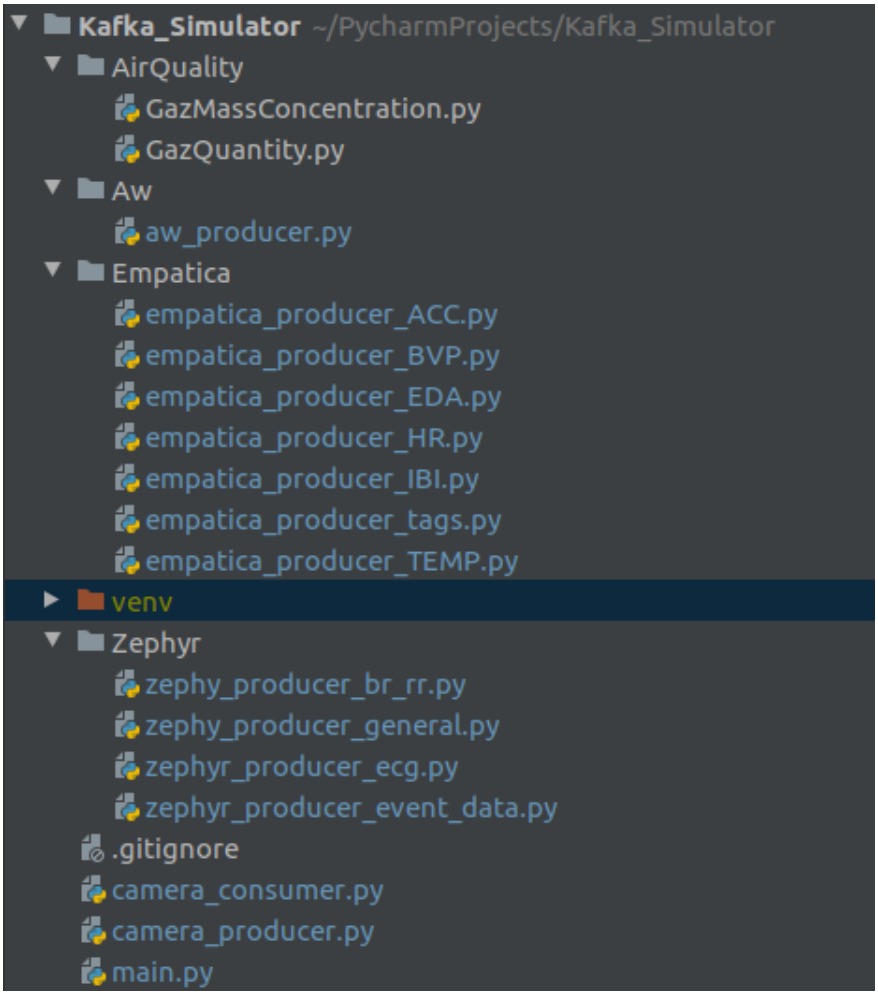

Figure III.8 – Structure du projet

2.2 La couche "Storage"

2.2.1 Le système de stockage de fichier HDFS

HDFS (Hadoop Distributed File System) est le système de stockage de fichiers d'Hadoop, basé sur le système de fichiers développé par Google, GDF (Google File System) et dont la théorie a été publiée dans des papiers de recherche de Google. HDFS a pour objectifs d'être :

- **tolérant aux pannes** d'une manière native (Fault tolerant) : c'est-à-dire qu'il peut résister à l'erreur, en partant du principe que la défaillance est la règle générale plutôt que l'exception, en premier lieu la panne matérielle et notamment le disque dur.

- **scalable** : il peut supporter une montée en charge horizontale, avec très peu de contraintes et de pertes de performance.
- **modèle d'accès immuable** : écriture unique (il n'est plus modifiable) et multiples lectures pour les fichiers, ce qui favorise la cohérence des données.
- **Déplacer les calculs vers les données** : parce que le déplacement des fichiers volumineux peut conduire à un goulot d'étranglement réseau et impacter les performances globales du système.
- **simple à mettre en place** : en assurant la portabilité sur différentes plateformes.

Il propose de nombreuses fonctionnalités :

- **gestion des fichiers par blocs** : chaque fichier sera découpé en blocs de 64 Mo.
- **réplication et distribution** : tous les blocs seront à la fois répliqués et distribués pour assurer un stockage fiable des fichiers de très grand volume.
- **gestion des droits** : les permissions en lecture et écriture et l'accès au répertoire à la fois pour le propriétaire et pour le groupe. L'authentification peut être soit absente, soit gérée par login et mot de passe ou bien par Kerberos.
- **accès aux données en continu (Streaming)** : il a un accès continu aux jeux de données, avec un haut débit d'accès au détriment de la faible latence, parce qu'il est adapté au traitement par lots.
- **stockage des grands jeux de données** : les fichiers stockés dans HDFS sont typiquement de plusieurs gigaoctets ou téraoctets, avec une bande passante de haut débit pour les différents nœuds du cluster.

L'architecture HDFS repose sur une architecture maître/esclave comme le montre le schéma. Elle est composée de :

- un serveur maître nommé NameNode , qui gère l'espace de noms des fichiers et répertoires ainsi que les droits d'accès.
- un certain nombre de serveurs esclaves nommés DataNodes , qui gèrent le stockage des fichiers divisés en plusieurs blocs répliqués et distribués.

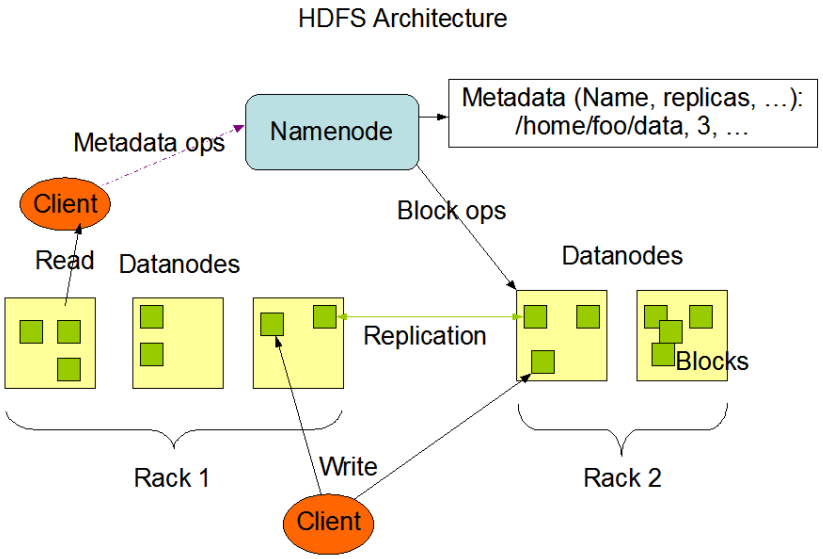

Figure III.9 – Architecture HDFS [5]

- Dans le contexte du projet :

Les données traitées par les consommateurs Kafka dans la couche 'Acquisition' seront enregistrées dans le système de fichiers HDFS selon la clé et le Topic : les topics ayant plusieurs partitions dont chaque partition contient un type de donnée c'est-à-dire des clés (keys) différents (chaque clé représente un CSV) seront enregistrés sous le répertoire du topic (qui représente le capteur). Par exemple pour le capteur EMPATICA on a 6 partitions, après la consommation des messages de différentes partitions du Topic d'Emaptica le consommateur écrit les données obtenues dans le fichier qui a pour nom la clé des messages de la partition traitée et le fichier obtenu sera de son tour enregistrer dans le répertoire d'Emaptica.

```
[root@quickstart /]# su hdfs
bash-4.1$ hadoop fs -ls
Found 5 items
drwxr-xr-x   - hdfs supergroup          0 2020-06-01 17:10 AW
drwxr-xr-x   - hdfs supergroup          0 2020-06-01 17:13 AirQuality
drwxr-xr-x   - hdfs supergroup          0 2020-06-01 17:10 Camera
drwxr-xr-x   - hdfs supergroup          0 2020-06-01 17:13 Empatica
drwxr-xr-x   - hdfs supergroup          0 2020-06-01 17:11 Zephyr
bash-4.1$ hadoop fs -ls Empatica
Found 7 items
-rw-r--r--   3 hdfs supergroup       3229 2020-06-06 11:16 Empatica/ACC.csv
-rw-r--r--   3 hdfs supergroup       1802 2020-06-06 11:16 Empatica/BVP.csv
-rw-r--r--   3 hdfs supergroup        531 2020-06-06 11:16 Empatica/EDA.csv
-rw-r--r--   3 hdfs supergroup       1499 2020-06-06 11:16 Empatica/HR.csv
-rw-r--r--   3 hdfs supergroup       6881 2020-06-06 11:16 Empatica/IBI.csv
-rw-r--r--   3 hdfs supergroup          0 2020-06-06 11:16 Empatica/TAGS.csv
-rw-r--r--   3 hdfs supergroup       2088 2020-06-06 11:16 Empatica/TEMP.csv
```

Figure III.10 – L'organisation des fichiers dans HDFS

2.2.2 CouchBase

Presentation de CouchBase:

Les bases de données orientés documents - en opposition à data - sont aujourd'hui la variante la plus populaire des bases de données NoSQL. Leur grande flexibilité les rend appropriées pour une large gamme d'applications, et leur nature objet s'accorde bien avec les pratiques de programmation actuelles.

CouchBase est donc une base de données NoSQL orientée document qui envoie et reçoit des données en s'appuyant sur le protocole JSON qui est très utile pour les applications qui envoient et reçoivent beaucoup de données périodiquement. Des entreprises comme le service de réservation de voyages Amadeus, PayPal, McGraw-Hill, Orbitz ou encore LinkedIn utilisent le système de gestion de base de données de CouchBase.

Comparaison entre CouchBase et MongoDB:

Les bases de données NoSQL orientée documents les plus populaires aujourd'hui sont CouchBase et MongoDB, dans notre cas nous avons choisi CouchBase en se basant sur ces 2 critères:

- La performance de MongoDB se dégradent rapidement lorsque la taille du cluster ou le nombre d'utilisateurs augmente par contre CouchBase possède une couche de mise en cache entièrement intégrée pour les données et les index, ce qui garantit une communication efficace et minimise le temps de latence.
- On a besoin d'une base de données qui assure la disponibilité (availability) pour qu'on puisse lire et écrire des données à tout moment et la tolérance au partitionnement afin que le système continuer à fonctionner et à répondre quand un nœud du cluster se bloque. Selon le théorème CAP CouchBase vérifie ces deux critères mais MongoDB est sur sur le côté CP (Cohérence et tolérance au partitionnement) et ne garantit pas la disponibilité.

CouchBase dans le contexte du projet:

Les données prétraitées dans la couche 'Refinement' sont enregistrées dans la base de données CouchBase. Lorsqu'on termine le nettoyage et la préparation des données dans la couche 'Refinement' on crée un Map qui possède comme clé la date à laquelle on a reçu les données et comme valeur des Map aussi qui contiennent les capteurs et leurs valeurs. Une fois le Map est près on le stocke dans un document qui sera enregistré dans la base de données en temps réel.

2.3 La couche "Acquisition"

2.3.1 Technologies utilisées

Apache Kafka : Apache Kafka est un broker de message, qui gère des évènements en se basant sur un modèle publish-subscribe. Donc on a des producteurs qui vont envoyer des évènements vers le cluster Kafka, ensuite le cluster Kafka va réceptionner ces évènements et va les mettre en queue, puis va permettre au consommateur d'évènements de souscrire des abonnements sur ces queues d'évènements pour être prévenu s'il y a un évènement qui les intéresse.Kafka se base sur ZooKeeper, pour assurer la scalabilité.

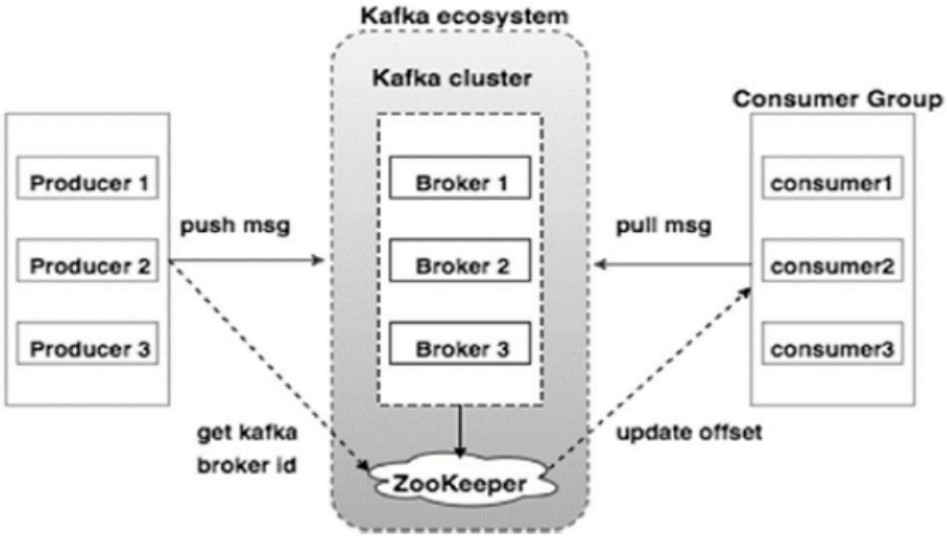

Figure III.11 – Kafka [5]

Apache ZooKeeper : Zookeeper est une base de données hiérarchisée pour la coordination distribuée pour les applications distribuées. Zookeeper est composé de nœud nommé ZNode, qui a comme caractéristique, une latence très faible avec des temps de réponse inférieur à la milliseconde, de ce fait un ZNode a pour but de stocker les données de configuration, de les sérialiser, de synchroniser les différents services Zookeeper, et doit avoir au maximum une taille de quelques kilooctets en données. Zookeeper est hautement disponible. Zookeeper peut être couplé à un Watcher, il permet de faire des notifications lors d'un changement dans un ZNode, Zookeeper est au cœur de Hadoop, pour les NameNode, YARN, HBase, Kafka, Storm.

2.3.2 Préparation et configuration de l'environnement

- Tout d'abord on doit commencer par l'installation du service Apache Kafka au Cloudera QuickStart qui s'exécute sur un container Docker :
  - On commence par l'installation de Java 8 et sa configuration

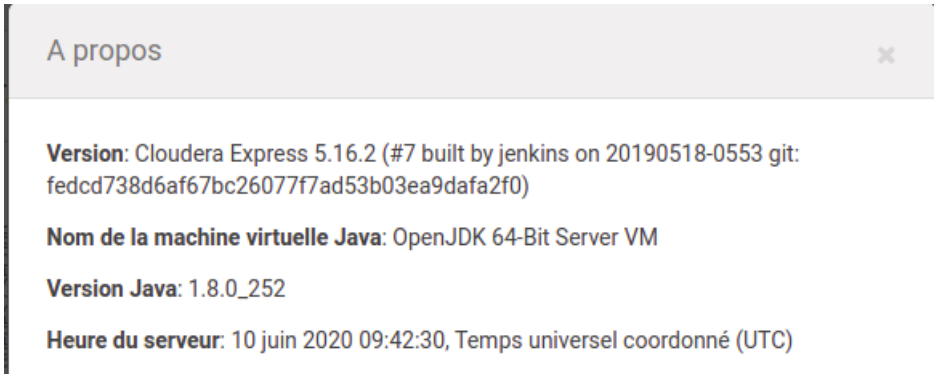

Figure III.12 – Java 8 installé et configuré

  - On ajoute la parcelle de Kafka à travers Cloudera Manager et on l'active

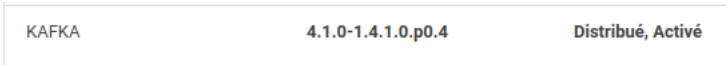

Figure III.13 – Kafka Parcel activé

  - Enfin on ajoute le service au cluster et on le configure en précisant :
    - Java Heap Size of Broker = 256 MiB

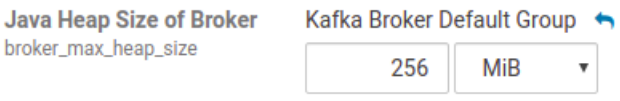

Figure III.14 – Java Heap Size of Broker

∗ Advertised Host = quickstart.cloudera

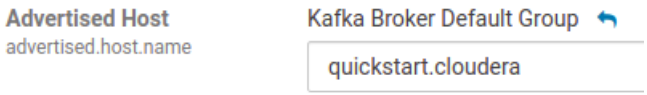

Figure III.15 – Advertised Host

∗ Inter Broker Protocol = PLAINTEXT

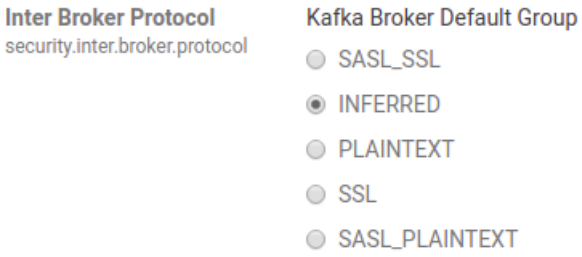

Figure III.16 – Inter Broker Protocol

∗ advertised.listeners=PLAINTEXT://172.17.0.2:9092 et offsets.topic.replication.factor=1

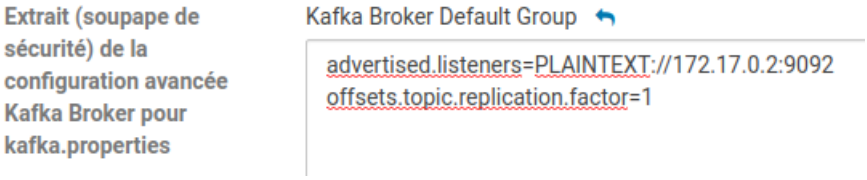

Figure III.17 – dvertised.listeners et offsets.topic.replication.factor

- Après l'installation et la configuration de Kafka on ajoute un Topic pour chaque capteur avec un nombre de partition égale au nombre de clés des messages envoyer par les producteurs
  - Pour Empatica on crée un Topic nomme Empatica qui a 6 partitions

- Pour Zephyr on crée un Topic nomme Zephyr qui a 4 partitions
- Pour AirQulity on crée un Topic nomme AirQuality qui a 2 partitions
- Pour Aw on crée un Topic nomme Aw qui a 1 partition
- Pour Camera on crée un Topic nomme Camera qui a 1 partition

2.3.3 Implémentation

Dans cette partie on va créer des consommateurs Kafka pour chaque topic, ces consommateurs vont consomme les données du topic concernée et selon la clé (Key) du message lus il l'enregistre dans un fichier spécifique dans HDFS.

Pour commencer on a configuré ces consommateurs en précisant :

- **BOOTSTRAP_SERVERS_CONFIG :** pour cette propriété on précise l'adresse de serveur Kafka (un pair host/port) que le consommateur utilise pour établir une connexion initiale au cluster Kafka.
- **GROUP_ID_CONFIG :** identifie le groupe du consommateur.
- **AUTO_OFFSET_RESET_CONFIG :** on choisit le paramètre Earliest afin que le nouveau groupe de consommateurs commence à chaque fois depuis le début.
- **ENABLE_AUTO_COMMIT_CONFIG :** permet de faire périodiquement le commit des Offset des messages déjà lus par le consommateur. Ce paramètre sera utilisé lorsque le processus échoue de trouver la position à partir de laquelle la consommation commencera.
- **KEY_DESERIALIZER_CLASS_CONFIG :** qui permet de définir la classe de désérialisation des clés des topics de KAFKA. Dans notre cas on a choisie StringDeserializer car les clés de nos messages sont des String
- **VALUE_DESERIALIZER_CLASS_CONFIG :** de même que KEY_DESERIALIZER_CLASS_C on précise la classe de désérialisation mais cette fois pour les messages reçus et qui va être StringDeserializer aussi puisque les messages qu'on a sont des String.
- **TOPIC :** le consommateur va s'abonner (par la méthode Subscribe) a ce topic.
- **FILEPATH :** représente le chemin dans lequel HDFS va enregistrer les fichiers (Filename). Ce paramètre varie d'un capteur a un autre.

- **FILENAME :** représente le nom du fichier dans lequel on va écrire les données reçues. Ce paramètre varie d'une partition a une autre.

Ensuite chaque consommateur qui représente dans notre cas un thread va lire les messages du topic concerné à travers la méthode poll qui renvoie les enregistrements extraits en fonction de l'offset de partition actuel, cette méthode est une méthode de blocage qui attend le temps spécifié en secondes (dans notre cas on a choisi 2 secondes). Si aucun enregistrement n'est disponible après la période spécifiée, la méthode poll renvoie un ConsumerRecords vide.

Enfin les données reçues seront écrites sans aucune modification dans des fichiers selon la clé du message reçue et seront enregistré après dans le système de fichier HDFS.

2.4 Les couches "Refinement" et "Decision Making"

2.4.1 Technologies utilisées

Apache Kafka : Dans cette partie, l'api consommateur Spark-Kafka a été utilisé. En effet chaque job Spark se compose d'abord d'un consommateur Kafka, celui qui s'occupe d'acquérir les données des différents topics, stocke chaque partition du topic dans une partition d'un RDD ou "Resilient Distributed Dataset" (nous allons parler de ces détails lorsqu'on présente Spark).
Apache Spark : C'est un moteur de traitement parallèle de données open source permettant d'effectuer des analyses de grande envergure par le biais de machines en clusters. Son principal avantage est sa vitesse, puisqu'il permet de lancer des programmes 100 fois plus rapidement que Hadoop MapReduce in-memory, et 10 fois plus vite sur disque. Spark présente le principe des RDD, c'est une collection d'objets distribués immuable. Chaque dataset dans un RDD est divisé en partitions logiques, qui peuvent être calculées sur différents nœuds du cluster.
Les composants de Spark utilisés dans cette couche sont :

- Spark Streaming : Spark Streaming est une extension de l'API Spark de base qui permet un traitement de flux évolutif, à haut débit et à tolérance de pannes de flux de données en direct. Les données peuvent être ingérées à partir de nombreuses sources telles que les sockets Kafka, Flume, Kinesis ou TCP, et peuvent être traitées à l'aide d'algorithmes complexes exprimés avec des fonctions de haut niveau comme Map, Reduce, Join et Window. Les traitement se font en des mini batch.
  Dans notre cas, Spark Streaming a été intégré avec Kafka.

Figure III.18 – Architecture de Spark Streaming

Figure III.19 – Flux de données dans Spark Streaming [5]

- SparkML : SparkML est aussi une extension de l'API Spark qui permet de faire des calculs complexe de Machine Learning de manière distribuée et efficace, cette extension a été utilisée pour faire des pre-processing sur les données acquis, les détails seront expliqués dans la partie implémentation.

Spark vs Storm vs Flink : Il existe deux type de traitement streaming, le streaming natif et le micro-batching. Storm et Flink reposent sur le concept de streaming natif, cela signifie que chaque enregistrement entrant est traité dès son arrivée, sans attendre les autres. Il existe certains processus en cours d'exécution (que nous appelons opérateurs / tâches / boulons en fonction du cadre) qui s'exécutent pour toujours et chaque enregistrement passe par ces processus pour être traité.

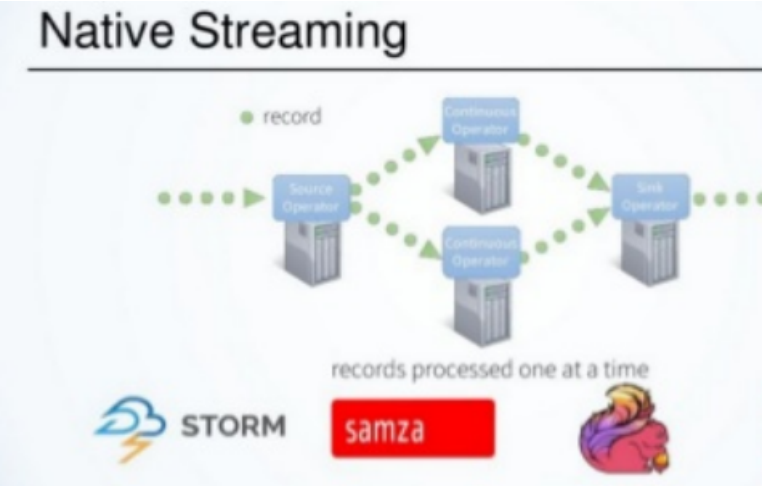

Figure III.20 – Streaming natif (cas de Storm et Flink) [5]

Spark repose sur le micro-batching, cela signifie que les enregistrements entrants en quelques secondes sont regroupés par lots puis traités en un seul mini-lot avec un délai de quelques secondes, ce qui idéal pour notre cas puisque on fait les prédictions toutes les 2 secondes et donc on travaille avec des micro-batch de 2 secondes.

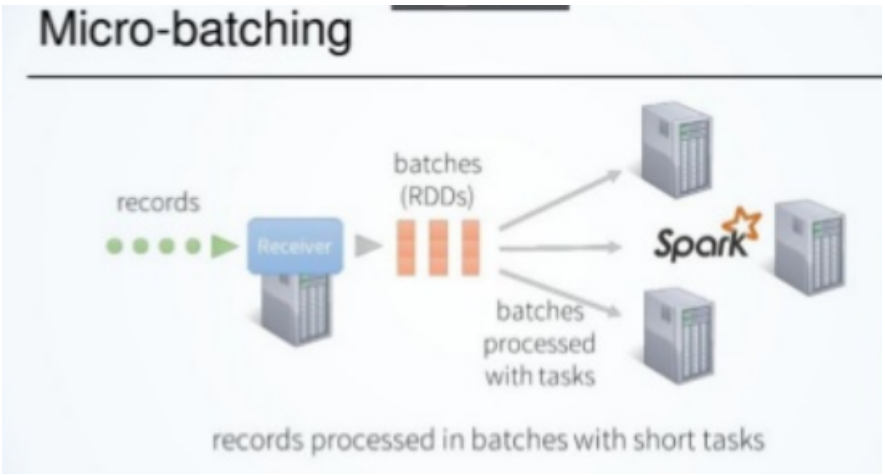

Figure III.21 – Streaming en micro-batch (cas de Spark) [5]

2.4.2 Préparation et configuration de l'environnement

- Dans une première partie, on a du configurer Spark pour fonctionner avec la version courante de Kafka broker qui est 0.10, ceci a été fait par l'intermédiaire de Cloudera Manager comme le montre la figure suivante.

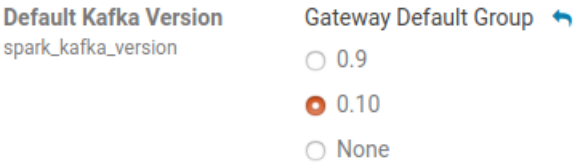

Figure III.22 – Choix du version du Broker Kafka

- Spark dispose de plusieurs modes de fonctionnement tels que Spark Standalone et Spark sur Yarn, dans notre cas on a opté pour Spark sur Yarn et ceci pour une meilleure gestion des ressources sur notre cluster Hadoop, de plus Spark sur Yarn permet l'intégration des structures complexes tels les files et les piles, chose qui serait utile lorsque la taille de cluster augmentera ainsi que les nombres des voitures et donc des jobs Spark. Pour que Spark fonctionne avec Yarn, des changement dans les fichiers de configuration de Spark on été faites.

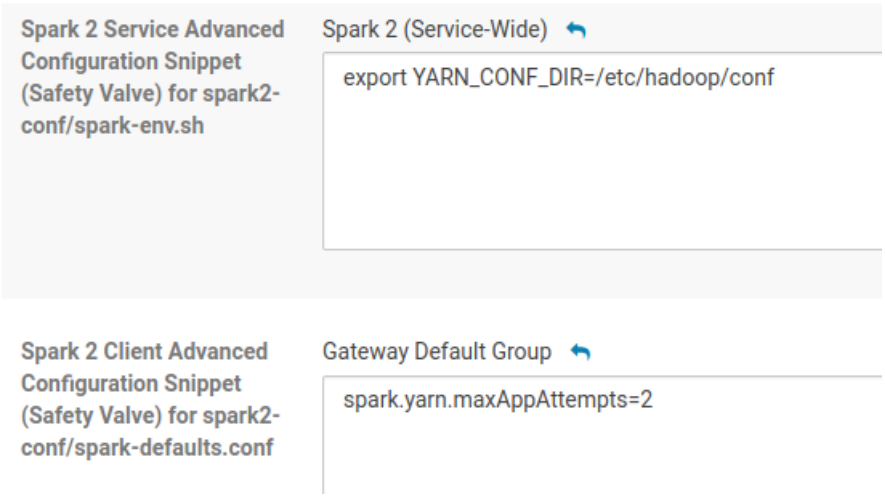

Figure III.23 – Configurations Spark-Yarn

2.4.3 Implémentation

La job Spark streaming a été écrite en Java 8, la job s'exécute en batch de 2 secondes, c'est à dire chaque 2 secondes :

- Le group consumer Spark souscrit au différentes topics Kafka, une fois les données acquis, il fait une opération Map afin de transfomer les records Kafka en des paires de la forme (clé, valeur).

- Une fois la transformation Map est faite, il faut nettoyer les données, certains capteur présente l'option de détecter si les données sont erronés, prenons l'exemple du capteur Caméra qui présente un attribut "Is_Face_Valid" et donc permettant de savoir si le record acquis sera utile pour les analyses ou pas. Cette opération est faite pour tous les capteurs.

- Une fois les données netoyées, un pre processing prends place, dans notre cas on a eu recours à un clustering assurée par l'algorithme d'apprentissage non supervisé DBSCAN. Son avantage est de détecter les "outliers" ou "noise", et c'est le but de notre pre processing. DBSCAN a été exécuté sur les capteurs se disposant d'un grand flux de données, de l'ordre de 500 records chaque 2 secondes. Voici une image décrivant le comportement de cet algorithme.

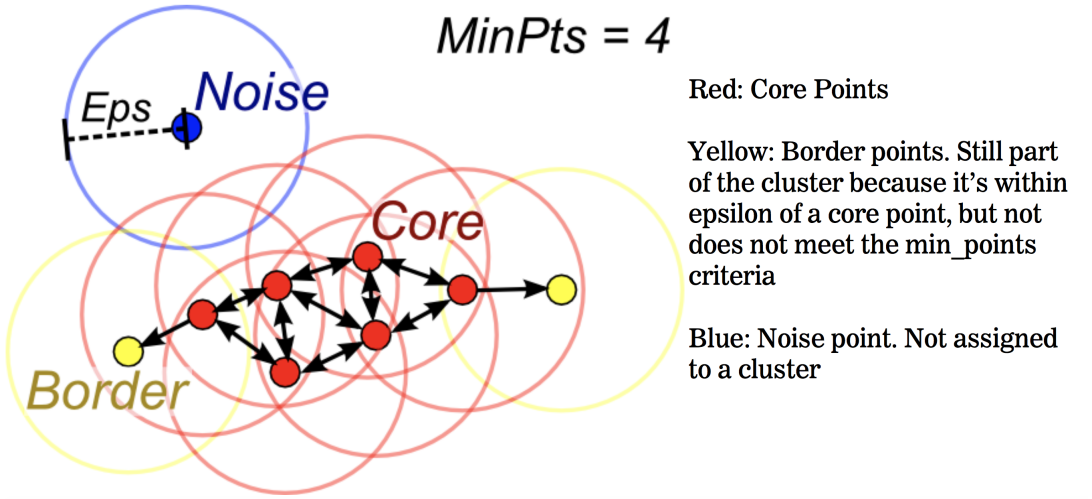

Figure III.24 – DBSCAN avec nombre de point minimum dans un cluster égale à 4 [5]

- Une fois les "outliers" détectés, on peut maintenant exploiter nos données, vu que pour faire les prédictions on a besoin d'un record de chaque capteur et que certains d'entre eux ont a flux de données élevés, on a opté pour faire la moyenne des valeurs acquis pour chaque capteurs, vu que ces valeurs sont proches l'une des autres. Dans le cas du capteur Caméra, on a décidé de passer une plage de valeur et de les utiliser tous pour faire les prédiction on essayant toutes les combinaisons possibles.

- Lorsque les données sont préparées, il sont stocké dans CouchBase puis passés au modèle AI pour faire la prédiction et détecter si il s'agit d'une personne stressée. Enfin, selon le résultat de la prédiction, un traitement adéquat sera mise en place et communiqué au dashboard.

2.5 La couche "Training"

2.5.1 Technologies utilisées

SparkML : C'est la librairie "Machine Learning" de l'écosystème Spark. Cette librairie nous permet de définir des 'pipelines' qui contiennent les traitements nécessaires pour l'apprentissage et la prédiction. Dans la couche training on définit des pipelines pour l'apprentissage qui seront exécuté chaque fois qu'un certain seuil de données est atteint. Les résultat de l'apprentissage sont les modèles SparkML qui seront déployés automatiquement dans les couches 'Refinement' et 'Decision making'. La persistance et transport de ses modèles est possible grâce a l'API haut niveau du SparkML. Un Pipeline est en faite un graphe orienté acyclique dont les noeud peut être des jobs 'transformer', 'estimator' ou 'evaluator'.

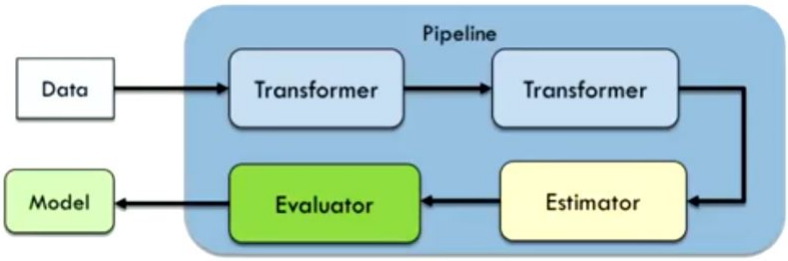

Figure III.25 – Pipeline SparkML [5]

Les noeuds 'transformer' effectuent les pré-traitements nécessaires sur les données. Exemples: la normalisation, Hashage, extraction de caractéristique, réduction du dimensionnalité.

Le noeud 'estimator' contient l'algorithme our l'ensemble d'algorithmes d'apprentissage choisit. Examples d'algorithmes de classification: 'Logistic Regression', 'Decision tree classifier'.

Le noeud 'evaluator' contient la métrique d'évaluation et la méthode de 'hyper-parameter tuning'. Examples des metriques: 'precision', 'Mean Square Error'. Examples de 'tuning': 'Cross Validation', 'Grid Search'.

2.5.2 Préparation et configuration de l'environnement

SparkML est un composant de l'environnement Spark et ne nécessite pas beaucoup de configurations. Il est possible de travailler avec SparkML en standalone avec la version prebuilt dont on a accès à une shell python ou scala.

2.5.3 Implémentation

Dans la couche Training, on fait du batch processing sur les données disponibles afin d'améliorer les algorithmes dans les autres couches. Dans ce projet on se concerne par un seul modèle, 'Stress Detection', qui sera déployé dans la couche 'Refinement'.

Puisque la collections des données n'est pas encore finalisé par Faurecia, nous disposons des données dont les 'timestamps' des labels ne correspondent pas aux 'timestamps' des données. Pour tester notre architecture on a opter pour la génération de labels aléatoires.

Le modèle choisi est un simple 'Binomial logistic regression' précédé par un simple transformateur de normalisation. La metrique d'evaluation est la precision definit par:

Precision=TruePositive/(TruePositive+FalsePositive)

2.6 La couche "Insight"

Cette couche dans devrait être modélisé soit par une action embarquée dans la voiture (Ajuster la température, allumer le radio, etc), soit par un message affiché dans le tableau de bord. Cependant puisqu'on ne dispose pas de voiture intelligente pour tester, on a opté pour un simple dashboard cré par Pyqt5, contenant un bouton "Start Car" qui active la simulation d'une voiture en marche entrain de générer des données via l'intermédiaire de ses capteurs, et un champ texte qui permet d'afficher si l'individu est stressé ou pas selon la prédiction faite sur les données acquises pendant le batch. Ceci est montré par la figure suivante.

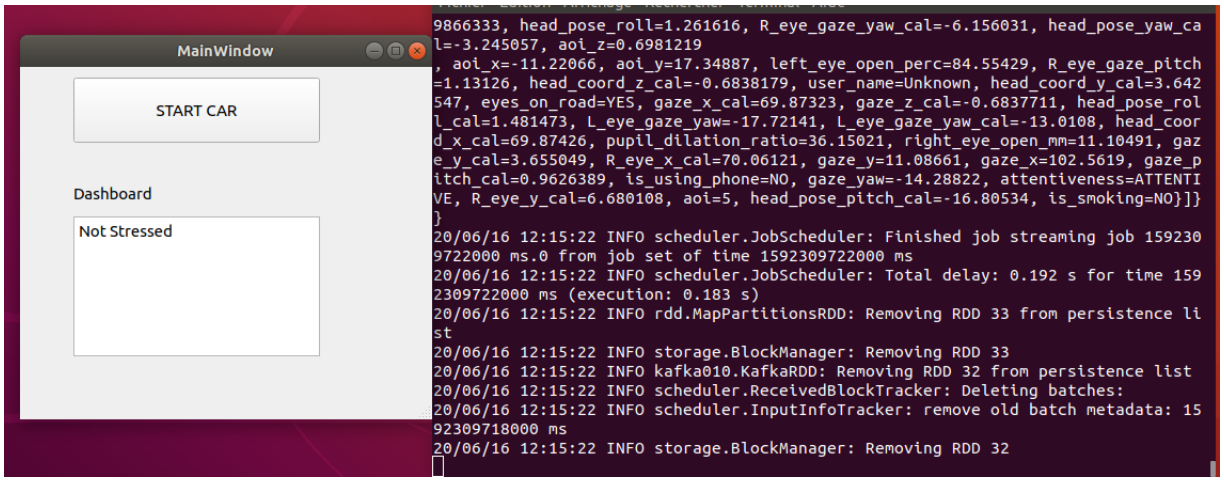

Figure III.26 – Dashboard avec l'application en arrière plan

Conclusion

Dans cette étape on a présenté tout d'abord les technologies qu'on a utilisées et la distribution Hadoop qu'on a choisi ainsi que la Dockerisation. Dans la deuxième partie on a analysé l'implémentation des différentes couches et leur mode de fonctionnement. On a expliqué aussi les configurations effectuer dans chaque couche qui représente une étape primordiale puisqu'elle diffère d'une architecture a une autre.

Conclusion Générale et Perspectives

Au cours de ce projet on a d'élaboré une architecture pour un pipeline Big Data qui répond au besoin de l'entreprise Faurecia de prendre des décisions en temps réel pour améliorer l'expérience d'un utilisateur de voiture intelligente en traitant, stockant et effectuant une analyse prédictive des flux de données provenant de plusieurs capteurs à l'intérieur de la voiture et qui résolve, en plus, le problème de 'Data Fusion' qui représente un très grand défi aujourd'hui puisque que les systèmes existants sont de plus en plus complex et le nombre des sources de données dans le même système continue à augmenter ce qui rend l'interprétation des données venant de plusieurs sources différents et indépendants plus difficile.

La première étape du projet était d'étudier les exigences du système et de bien comprendre les données venant des différents capteurs de la voiture, leurs fréquences d'envoi et la signification des données qu'elle émet. À la fin de l'analyse conceptuelle, on a conçu une architecture qui permet la gestion des données venant de plusieurs capteurs d'une façon fiable et efficace.

Après on s'est focalisé sur les technologies Big Data qu'on va choisir, et sur la comparaison des solutions existantes ainsi qu'aux bases de données et systèmes de stockage qui répondent à nos besoins non fonctionnels surtout.

La dernière étape, était l'implémentation qui a commencé au début par la configuration de l'image Docker qui contient la distribution Cloudera Quickstart qu'on a choisi parce qu'on voulait que le pipeline soit indépendant du système d'exploitation. Enfin on a pris le plus grand parti du temps à développer le code en Java qu'on a essayé d'assurer l'organisation, la modulaire et d'adopter les bonnes pratiques de développement afin de faciliter la compréhension du code surtout qu'il y a beaucoup d'interaction entre les classes et beaucoup de technologies utilisées.

À la fin du projet on a pu simuler le fonctionnement de la voiture qui émet des données de plusieurs capteurs dans la voiture, ces données vont être traitées par le pipeline qui fait un pré et un post processing aux donnes, les stocks, génère un résultat qui peut être soit stressé soit non stressé et envoi enfin le résultat obtenu à la voiture en temps réel.

Bibliographique

(1) Introduction to Big Data, by University of California San Diego - Ilkay Altintas et Amarnath Gupta : https://www.coursera.org/learn/big-data-introduction

(2) Big Data Modeling and Management Systems, by University of California San Diego - Ilkay Altintas et Amarnath Gupta : https://www.coursera.org/learn/big-data-management

(3) Big Data Integration and Processing, by University of California San Diego - Ilkay Altintas et Amarnath Gupta : https://www.coursera.org/learn/big-data-integration-processing

(4) Machine Learning With Big Data, by University of California San Diego - Ilkay Altintas et Mai Nguyen: https://www.coursera.org/learn/big-data-machine-learning

(5) Tutoriel pour apprendre à faire le choix d'une architecture big data - Naoufal BEN BOUAZZA : https://big-data.developpez.com/tutoriels/apprendre-faire-choix-architecture-big-data

(6) https://docs.cloudera.com/documentation/enterprise/5-6-x/topics/quickstart_docker_container.html [en ligne, consulté en Mars 2020]

(7) https://kafka.apache.org/documentation/ [en ligne, consulté en Avril 2020]

(8) La Collecte de Données avec le Bus Kafka- Lilia Sfaxi : https://insatunisia.github.io/TP-BigData/tp3/

(9) https://spark.apache.org/docs/2.4.5/quick-start.html [en ligne, consulté en Mai 2020]

(10) https://spark.apache.org/docs/2.4.5/rdd-programming-guide.html [en ligne, consulté en Mai 2020]

(11) https://www.tutorialspoint.com/couchdb/index.htm [en ligne, consulté en Juin 2020]