

**Rapport**

Master Big Data et Aide à la Décision

Présenté par :

**Bella Abedlouahab**

**Boulanouar Mohamed Amine**

**Système de détection d'intrusions basé sur le clustering utilisant Apache Spark**

Remerciement

Avant de plonger dans le détail de ce travail académique, nous souhaitons prendre un moment pour exprimer notre gratitude envers ceux qui ont rendu ce parcours à la fois possible et fructueux.

Tout d'abord, nous tenons à adresser nos remerciements les plus sincères à l'ENSA Khouribga, pour nous avoir offert un environnement d'apprentissage stimulant et propice à l'épanouissement intellectuel. Les ressources mises à disposition et le cadre académique exceptionnel ont été des piliers fondamentaux dans la réalisation de ce travail.

Nous sommes particulièrement reconnaissants envers Pr. Nassima SOUSSI, notre superviseure, pour son accompagnement précieux, ses conseils éclairés et son soutien sans faille tout au long de ce parcours. Sa passion pour le sujet, son expertise et sa capacité à motiver ses étudiantes et étudiants ont été une source d'inspiration constante pour nous.

Nous souhaitons aussi exprimer notre gratitude à notre famille et à nos amis pour leur amour, leur soutien inconditionnel et leurs encouragements. Leur foi en nos capacités et leur présence rassurante ont été des atouts indispensables pour surmonter les défis rencontrés et atteindre nos objectifs.

Résumé

Face à l'explosion des données mondiales, le Big Data offre une solution aux limites des méthodes traditionnelles d'analyse. Ce projet vise à développer un système de détection d'intrusions en réseau utilisant le clustering et Apache Spark pour traiter et analyser de grandes quantités de données en temps réel. L'objectif est de tirer parti du traitement distribué et de l'analyse de données massives pour détecter efficacement les comportements anormaux dans les flux réseau.

Mots clés :

Big Data, Détection d'intrusions, Clustering, Apache Spark, Traitement distribué, Analyse de données, Hadoop, RDD.

Abstract

In the face of the explosion of global data, Big Data offers a solution to the limitations of traditional analysis methods. This project aims to develop a network intrusion detection system using clustering and Apache Spark to process and analyze large amounts of data in real time. The objective is to leverage distributed processing and big data analytics to efficiently detect abnormal behaviors in network traffic.

Keywords:

Big Data, Intrusion Detection, Clustering, Apache Spark, Distributed Processing, Data Analysis, Hadoop, RDD

Table des matières

[Les concepts fondamentaux du Big Data 7](#_Toc1564529189)

[Définition 7](#_Toc1533229158)

[Les 5V du Big Data 8](#_Toc833661214)

[MapReduce 9](#_Toc2037852115)

[Définition 9](#_Toc1048123863)

[Pourquoi MapReduce ? 10](#_Toc1751196351)

[L'architecture du modèle MapReduce 10](#_Toc549287073)

[La plateforme Hadoop 11](#_Toc1204452034)

[Définition 11](#_Toc1851768187)

[Caractéristiques essentielles d’Hadoop 12](#_Toc2113074115)

[La plateforme Spark 13](#_Toc1556088681)

[Définition 13](#_Toc176493562)

[Les composants de spark 14](#_Toc918910382)

[RDD (Resilient Distributed Data) 15](#_Toc1310324774)

[Les transformations 16](#_Toc1366195519)

[Les actions 16](#_Toc1638195849)

[Le DAG 16](#_Toc23214188)

[L’algorithme k-means 17](#_Toc2035261951)

[Définition 17](#_Toc1013463822)

[Avantage 17](#_Toc1550006140)

[Inconvénients 18](#_Toc1477345576)

[Figure 1 3 Vs du Big Data 8](#_Toc186979132)

[Figure 2 5 Vs du Big Data 9](#_Toc186979133)

[Figure 3 Architecture du MapReduce 11](#_Toc186979134)

[Figure 4 Écosystème Hadoop 13](#_Toc186979135)

[Figure 5 Les composants de Spark 14](#_Toc186979136)

[Figure 6 **Les transformations du RDD** 16](#_Toc186979137)

[Figure 7 Fonctionnement de K-Means 18](#_Toc186979138)

Introduction Générale

À l'heure actuelle, des trillions d'octets de données (Big Data) sont générés chaque jour. Ces données proviennent de sources variées, telles que les capteurs utilisés pour collecter des informations climatiques, les messages publiés sur les réseaux sociaux, les images numériques, et les vidéos mises en ligne.

Les Big Data se caractérisent par leur volumétrie (données massives) et leur diversité, tant en termes de formats que de structures nouvelles. Elles imposent également des exigences strictes en matière de rapidité de traitement. À ce jour, il n'existe pas de logiciel capable de gérer efficacement les multiples types et formes de données en constante augmentation. Ainsi, la problématique du Big Data est devenue omniprésente dans notre quotidien, nécessitant des solutions de plus en plus sophistiquées pour gérer cette immense quantité de données dans des délais réduits.

Le calcul distribué se rapporte au traitement de grandes quantités de données. Ce type de traitement ne peut être réalisé avec les paradigmes traditionnels, et nécessite donc l'utilisation de plateformes distribuées. Plusieurs solutions existent dans la littérature pour implémenter ce paradigme. Un exemple notable est celui de Google, qui a développé le modèle de programmation MapReduce, particulièrement adapté pour le traitement de Big Data. Ce modèle est utilisé sur diverses plateformes, notamment Hadoop. Cependant, Hadoop souffre de problèmes de latence, ce qui a conduit au développement d'alternatives visant à améliorer la performance du traitement. Parmi ces alternatives, la plateforme Spark se distingue par sa puissance, sa flexibilité et sa rapidité par rapport à Hadoop MapReduce.

Ce rapport a pour objectif d'explorer l'utilisation d'Apache Spark dans la mise en place d'un système de détection d'intrusions distribué.

# Les concepts fondamentaux du Big Data

## Définition

Le Big Data désigne un volume massif de données, souvent hétérogènes, qui se présentent sous diverses formes et formats (texte, données de capteurs, audio, vidéo, journaux de navigation, etc.). Ces données peuvent être structurées, semi-structurées ou non structurées. Le Big Data est complexe et nécessite des technologies avancées ainsi que des algorithmes puissants pour son traitement et son stockage. En raison de cette complexité, il ne peut pas être géré efficacement avec des outils traditionnels comme les systèmes de gestion de bases de données (SGBD).

## Les 5V du Big Data

Le Big Data est généralement défini à travers les "3V", qui décrivent ses principales caractéristiques :

* **Vélocité** : Les données sont générées à une grande vitesse et doivent être traitées rapidement pour en extraire des informations pertinentes. Par exemple, Walmart génère plus de 2,5 pétabytes de données chaque heure grâce aux transactions de ses clients. YouTube est également un exemple de la rapidité de génération des données.
* **Variété** : Les données volumineuses proviennent de multiples sources et sont disponibles dans différents formats (vidéos, documents, commentaires, journaux, etc.). Elles incluent des données structurées et non structurées, publiques ou privées, locales ou distantes, complètes ou incomplètes.
* **Volume** : Cela fait référence à la quantité de données générées, stockées et analysées. Le volume des données est en constante augmentation, atteignant presque 800 000 pétaoctets aujourd'hui. Par exemple, Twitter génère plus de 7 téraoctets de données par jour, tandis que Facebook produit plus de 10 téraoctets de données quotidiennes.

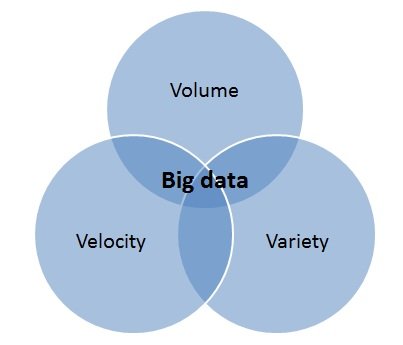


Figure 1 3 Vs du Big Data

En plus des trois dimensions initiales, deux autres dimensions ont été ajoutées, élargissant ainsi le concept du Big Data aux "5V" :

* **Véracité** : La véracité des données se réfère à leur fiabilité et leur exactitude. La confiance dans les données est essentielle pour leur adoption par les décideurs. Si la qualité ou la pertinence des données est mise en doute, il devient difficile d'en tirer parti efficacement.
* **Valeur** : La valeur des données est cruciale, car le traitement des Big Data doit permettre de générer des bénéfices concrets pour les entreprises et leurs clients, en atteignant des objectifs stratégiques.

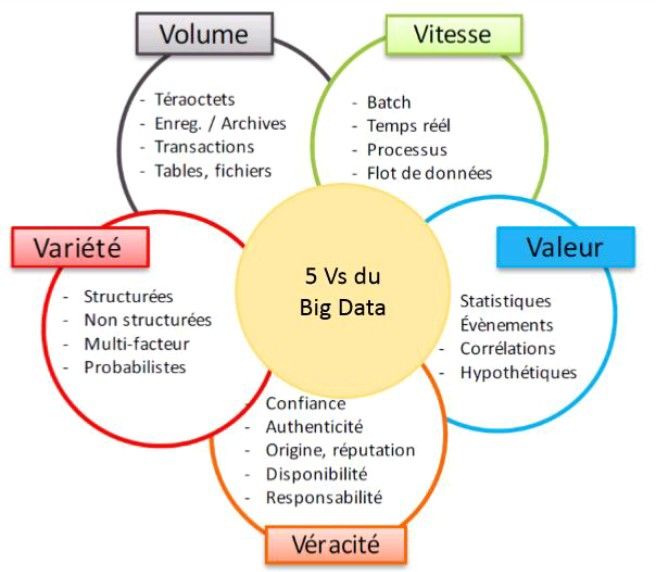


Figure 2 5 Vs du Big Data

# MapReduce

## Définition

Le modèle MapReduce a été conçu dans les années 2000 par les ingénieurs de Google. Il s'agit d'un modèle de programmation destiné à traiter plusieurs téraoctets de données sur des milliers de nœuds dans un cluster. MapReduce permet de traiter de manière plus rapide et efficace des téraoctets et des pétaoctets de données. En conséquence, il a rapidement gagné en popularité et est devenu un outil essentiel dans de nombreux secteurs. Ce modèle offre une plateforme hautement efficace pour l'exécution parallèle des applications, la distribution des données dans des systèmes de bases de données distribuées, et la communication réseau tolérante aux pannes. L'objectif principal de MapReduce est de faciliter la parallélisation des données, leur distribution et l'équilibrage de la charge à travers une bibliothèque simple.

## Pourquoi MapReduce ?

Les systèmes traditionnels des entreprises reposent généralement sur un serveur centralisé pour le stockage et le traitement des données. Ce modèle classique n'est pas adapté pour traiter de grands volumes de données évolutives, et les serveurs de bases de données standards ne sont pas capables de les gérer efficacement. De plus, le système centralisé crée souvent des goulots d'étranglement lors du traitement simultané de multiples fichiers. Pour résoudre ce problème, Google a développé le modèle MapReduce, qui permet d'éviter ces limitations.

## L'architecture du modèle MapReduce

MapReduce a été conçu pour traiter de grandes quantités de données non structurées ou semi-structurées, telles que des documents ou des journaux de requêtes sur les pages Web, sur de vastes clusters de nœuds. Ce modèle permet de générer différents types de données, comme des indices inversés ou des fréquences d'accès aux URL. L'architecture de MapReduce comprend trois composants principaux : le *Master*, la fonction *Map*, et la fonction *Reduce*.

* **Le Master** : Il est responsable de la gestion des fonctions *Map* et *Reduce*, de la mise à disposition des données et des procédures nécessaires, et de l'organisation de la communication entre les mappeurs et les réducteurs.
* **La fonction Map** : Elle est appliquée à chaque enregistrement d'entrée et produit une liste d'enregistrements intermédiaires sous forme de paires clé/valeur.
* **La fonction Reduce** : Elle est appliquée à chaque groupe d'enregistrements intermédiaires ayant la même clé, et génère une valeur finale à partir de ces groupes.

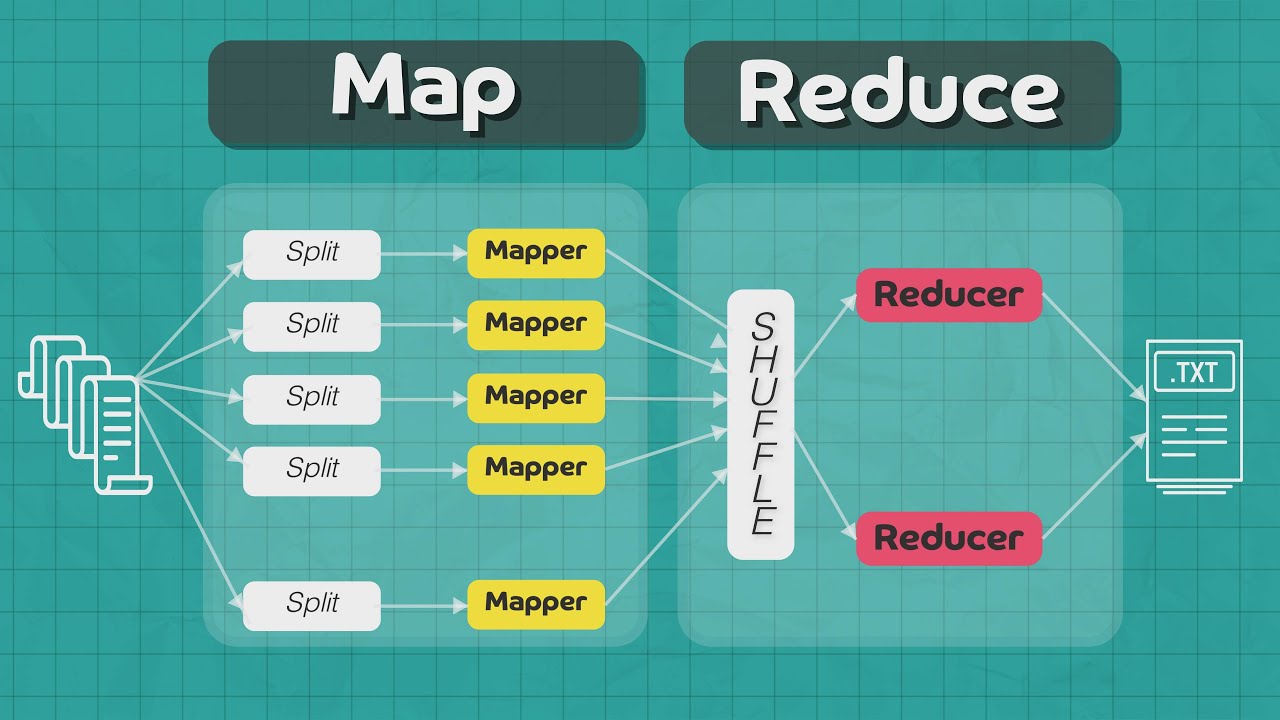


Figure 3 Architecture du MapReduce

Ainsi, le processus MapReduce suit les étapes suivantes :

1. Les données d'entrée sont divisées en enregistrements.
2. Les fonctions *Map* traitent ces données et produisent des paires clé/valeur pour chaque enregistrement.
3. Toutes les paires clé/valeur générées par la fonction *Map* sont fusionnées et regroupées par clé, puis triées.
4. Les résultats intermédiaires sont envoyés à la fonction *Reduce*, qui produit le résultat final.

# La plateforme Hadoop

## Définition

Hadoop est un framework open-source, écrit en Java, développé et distribué par la fondation Apache, conçu pour le traitement et la gestion de grandes quantités de données, pouvant atteindre des volumes de l’ordre des péta-octets et au-delà. Inspiré par plusieurs travaux techniques de Google, son objectif principal est de fournir un système de stockage et de traitement des données distribué, scalable et extensible. Hadoop prend en charge une grande variété de types de données, y compris les données non structurées. Il est souvent décrit comme un système non relationnel, plus général que NoSQL. Il permet notamment de stocker les données à l'aide de deux systèmes principaux : HDFS (Hadoop Distributed File System) et HBase, qui forment ensemble une base de données orientée colonnes, distribuée sur des serveurs en clusters.

Hadoop optimise le traitement des données en le parallélisant sur de nombreux nœuds répartis dans un cluster, ce qui accélère les calculs et réduit la latence des opérations d'entrée-sortie. Il inclut un système de fichiers distribué fiable qui assure la tolérance aux pannes par réplication des données.

Le framework Hadoop se compose principalement de deux composants : HDFS et MapReduce. Ces deux éléments sont liés au calcul distribué de la manière suivante :

* **HDFS** assure le stockage des données, tandis que **MapReduce** constitue le cœur du traitement parallèle, en utilisant les fonctions **Map** et **Reduce**.
* Sur le plan architectural, Hadoop distingue deux types de nœuds :
  + **Les nœuds maîtres** : le NameNode, le NameNode secondaire et le JobTracker.
  + **Les nœuds esclaves**, répartis dans le cluster : le DataNode et le TaskTracker.
* Les tâches sont exécutées par les **TaskTrackers**, qui sont déployés sur chaque machine du cluster, suivant les instructions données par le **JobTracker**.

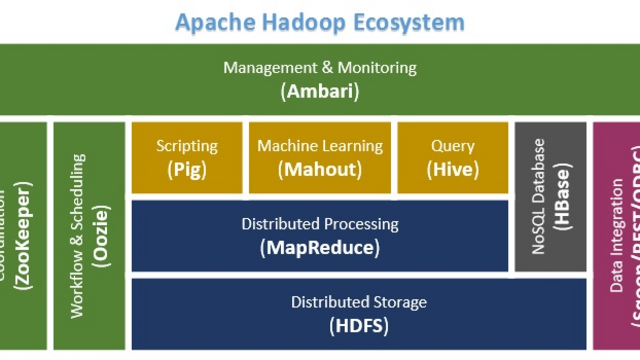


Figure 4 Écosystème Hadoop

## Caractéristiques essentielles d’Hadoop

Hadoop est une plateforme logicielle dédiée au stockage et à l’analyse de grandes quantités de données. Elle présente plusieurs caractéristiques clés :

1. **Système de fichiers distribué extensible** : Hadoop dispose d’un système de fichiers distribué très facilement scalable. Il gère automatiquement la distribution et le stockage des données sur ses différents nœuds. Pour augmenter sa capacité de stockage, il suffit d'ajouter de nouveaux nœuds à la plateforme.
2. **Exécution des traitements au plus près des données** : Les traitements sont envoyés directement vers les données, ce qui optimise la gestion de grandes volumétries de données sur des machines standard reliées par des réseaux classiques. Les nœuds de stockage agissent également comme nœuds de calcul pendant les traitements, ce qui permet d’accroître la capacité de traitement en augmentant simplement le nombre de nœuds de données.
3. **Tolérance aux pannes intégrée** : Hadoop est conçu pour fonctionner sur du matériel standard, souvent peu coûteux, ce qui rend les pannes inévitables. Pour y faire face, les données sont répliquées sur plusieurs nœuds pour garantir leur disponibilité. En cas de panne d'un nœud ou d’un réplicat, Hadoop réplique automatiquement les données restantes pour maintenir un bon taux de réplication. De plus, si une tâche de traitement échoue, elle est relancée sur un autre nœud, assurant ainsi une résilience sans intervention manuelle de l’utilisateur.
4. **Paradigme de programmation MapReduce** : Hadoop intègre le modèle de programmation MapReduce, particulièrement adapté pour extraire et filtrer des données réparties sur plusieurs nœuds, ainsi que pour effectuer des traitements simples sur ces données. Ce paradigme permet de chaîner des opérations MapReduce pour des tâches plus complexes. Cependant, MapReduce n’est pas idéal pour des calculs analytiques complexes ou itératifs. Dans ce cas, il est préférable d’utiliser un autre modèle de programmation et une autre plateforme.

# La plateforme Spark

## Définition

**Apache Spark** est un framework open-source de traitement distribué conçu pour gérer les charges de travail liées au **Big Data**. Il tire parti de la mémoire cache et d'une exécution optimisée des requêtes pour interroger rapidement des ensembles de données, quel que soit leur volume. Grâce à sa rapidité et à sa flexibilité, Spark se présente comme un moteur performant pour le traitement de données à grande échelle. Le noyau de Spark (Spark Core) repose sur les **RDD** (Resilient Distributed Datasets), qui représentent le type de données de base utilisé pour le calcul distribué.

Apache Spark permet d’effectuer des traitements sur d'immenses volumes de données de manière rapide et efficace. Il peut également répartir ces tâches sur plusieurs ordinateurs, que ce soit en mode autonome ou en combinaison avec d'autres outils distribués. Ces caractéristiques sont essentielles dans le contexte du **Big Data** et de l'**apprentissage automatique**, qui nécessitent une puissance de calcul massive pour explorer et analyser de vastes ensembles de données.

**Apache Spark** peut être utilisé avec plusieurs langages de programmation fonctionnels, tels que **Scala** (qui s'exécute sur la JVM Java), **Java**, **Python** et **R**. Depuis 2009, Apache Spark est devenu l'un des principaux frameworks de traitement distribué pour le **Big Data** à l'échelle mondiale. Il peut être déployé de diverses manières et est utilisé dans une grande variété de secteurs. En effet, des entreprises comme les **banques**, les **opérateurs de télécommunications**, les **sociétés de jeux**, ainsi que des **gouvernements** et des **grands géants technologiques** tels qu'**Apple**, **Facebook**, **IBM** et **Microsoft** s'en servent régulièrement.

## Les composants de spark

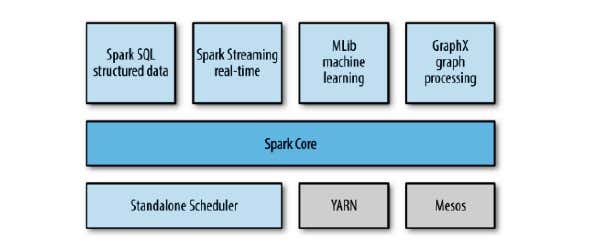


Figure 5 Les composants de Spark

### **Spark Core :**

Les fonctions de base de **Spark Core** incluent la planification des tâches, la gestion de la mémoire, la tolérance aux pannes, ainsi que l'interaction avec les systèmes de stockage. Le concept fondamental de Spark repose sur les **RDD** (Resilient Distributed Datasets), permettant de traiter efficacement des données via l'API RDD de Spark.

### **Spark SQL :**

Spark SQL est un module de traitement de données qui permet d'utiliser des requêtes SQL pour manipuler les données. Il permet d'accéder à diverses sources de données telles que **JSON**, **Parquet**, les **tables Hive** et **JDBC**. Spark SQL offre plusieurs fonctionnalités : manipulation des données via SQL, HQL2 (Hive Query Language 2), ou une API de haut niveau personnalisée pour les ensembles de données, et permet de combiner SQL avec les opérations de manipulation de données fournies par l'API **Spark RDD**.

### **Spark Streaming :**

Spark Streaming est utilisé pour le traitement des données en temps réel. Bien que **Spark** soit initialement conçu pour le traitement par lots, il permet également le traitement des flux de données en continu grâce au concept de **micro-lots** proposé par Spark Streaming. Toutes ces bibliothèques fonctionnent sur les RDD comme abstraction des données, permettant ainsi une intégration fluide dans une seule application. Cela offre des avantages significatifs en termes de flexibilité et de performance pour les utilisateurs.

### **MLlib :**

MLlib est une bibliothèque d'apprentissage automatique distribuée et évolutive incluse dans **Apache Spark**. Elle contient un large éventail d'algorithmes d'apprentissage automatique, notamment pour la **classification**, la **régression**, le **clustering** et le **filtrage collaboratif**. MLlib offre également des outils pour l'évaluation des modèles et l'importation de données, facilitant ainsi le développement de solutions d'apprentissage automatique à grande échelle.

### **GraphX :**

GraphX est un composant de **Spark** dédié à la manipulation de graphes et aux calculs parallèles sur des structures de graphes. Il permet de traiter efficacement des données sous forme de graphes en exploitant les capacités de calcul parallèle de Spark.

## RDD (Resilient Distributed Data)

Un **RDD** (Resilient Distributed Dataset) est une collection de données en lecture seule, pouvant être partitionnée et distribuée sur un sous-ensemble de machines dans un cluster **Spark**. Il constitue l'un des composants essentiels du framework et est au cœur de son fonctionnement. L'ensemble des opérations de **Spark** repose sur les RDD, et l'API Spark est donc essentiellement un ensemble d'opérations permettant leur **création**, leur **transformation** et leur **exportation**.

Les données qui forment un RDD sont divisées en partitions, lesquelles sont ensuite distribuées à travers les différents nœuds du cluster Spark. Chaque nœud procède à des calculs en parallèle. Une série d'opérations parallèles peut être réalisée sur les RDD via l'API **Spark Core**, et ces opérations se divisent en deux catégories principales : les **transformations** et les **actions**.

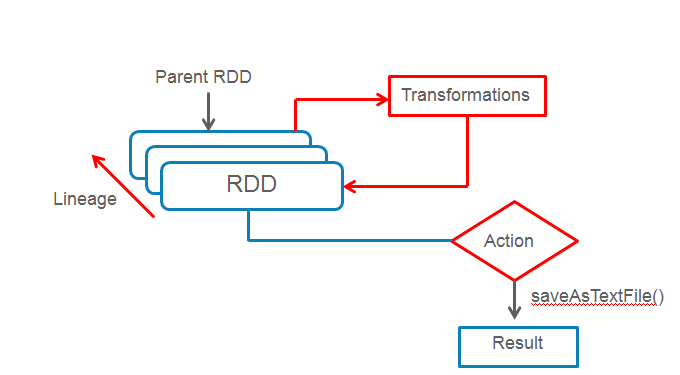


Figure 6 **Les transformations du RDD**

## Les transformations

Les **transformations** dans **Spark** génèrent un nouvel **RDD** en appliquant une opération sur un **RDD** existant. Par exemple, la transformation **filter(func)** utilise une fonction prédicat pour sélectionner les éléments qui correspondent à un critère donné, tandis que **map(func)** applique une fonction à chaque élément de l'ensemble de données, créant ainsi un nouvel ensemble de données à partir des résultats produits par cette fonction.

Les transformations dans Spark sont **évaluées paresseusement** (lazy evaluation), ce qui signifie qu'elles ne sont exécutées que lorsqu'elles sont nécessaires pour obtenir un résultat via une action. Jusqu'à ce moment-là, seule la séquence des transformations appliquées sur l'ensemble de données est conservée en mémoire.

## Les actions

Les actions sont des opérations qui retournent un résultat au programme principal après avoir effectué des calculs sur le RDD d'entrée. Par exemple, l'action **reduce(func)** réduit tous les éléments de l'ensemble de données en appliquant une fonction donnée, qui opère sur deux éléments à la fois pour produire un nouvel élément, jusqu'à ce que l'ensemble de données soit réduit à une seule valeur. De même,**count()** est une action qui renvoie le nombre d'éléments dans l'ensemble de données, tandis que **first()** renvoie simplement le premier élément. Ces actions peuvent également permettre d'écrire des données dans un stockage externe.

## Le DAG

Spark dispose d'un planificateur DAG avancé pour modéliser les dépendances entre les RDDs. À chaque exécution, une tâche Spark génère un DAG que le planificateur divise en différentes étapes de traitement. Ces étapes sont ensuite exécutées sur le cluster. Le DAG est formé à la fois lors des phases de map et de reduce pour refléter de manière complète les dépendances entre les opérations. Ce mécanisme permet à un travail simple de se terminer en une seule étape, tandis qu'une tâche plus complexe pourra être exécutée sur plusieurs étapes, sans être décomposée en plusieurs tâches. C'est l'une des raisons pour lesquelles Spark surpasse Hadoop en termes de performance.

# L’algorithme k-means

## Définition

Le clustering K-means est l'un des algorithmes d'apprentissage automatique non supervisé les plus simples à comprendre et à utiliser, et il est aujourd'hui largement répandu. En général, les algorithmes non supervisés analysent des ensembles de données en se basant uniquement sur les vecteurs d'entrée, sans utiliser de résultats étiquetés ou connus. K-means est une méthode de classification qui permet de réduire la complexité des grands ensembles de données en les divisant en groupes plus petits et plus simples. L'algorithme évalue différents modèles et regroupe les données similaires. La variable K représente le nombre de groupes dans lesquels les données sont divisées. Nous allons maintenant examiner les avantages et les inconvénients de l'algorithme de classification K-means afin d'évaluer les bénéfices de son utilisation.

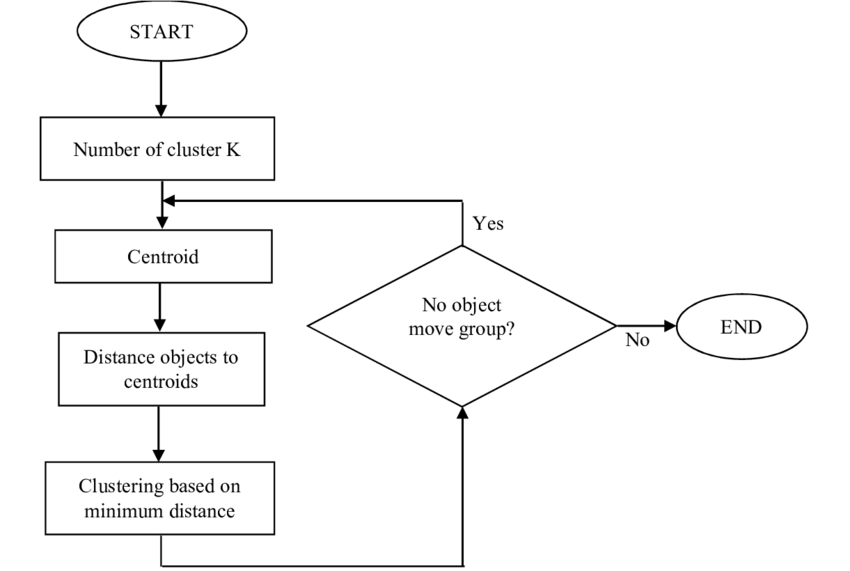


Figure 7 Fonctionnement de K-Means

## Avantage

* **Simplicité** : K-means est facile à implémenter et permet d'identifier rapidement des groupes dans des ensembles de données complexes.
* **Flexibilité** : L'algorithme s'adapte facilement aux changements dans les données, et des ajustements peuvent être réalisés rapidement.
* **Adapté aux grands ensembles de données** : K-means est efficace pour traiter de gros volumes de données et fonctionne plus rapidement que d'autres algorithmes pour les ensembles de données plus petits.
* **Efficacité** : L'algorithme est efficace pour partitionner de grands ensembles de données, bien qu'il soit optimal pour des clusters de forme sphérique.
* **Complexité temporelle** : Le temps de calcul est linéaire par rapport au nombre d'objets, ce qui peut augmenter l'exécution, mais il reste plus rapide que les méthodes hiérarchiques.
* **Clusters plus compacts** : Comparé aux algorithmes hiérarchiques, K-means produit des clusters plus serrés, particulièrement pour des clusters globulaires.
* **Facilité d'interprétation** : Les résultats sont simples à interpréter, avec des descriptions de clusters concises pour une meilleure compréhension.
* **Coût de calcul faible** : K-means est rapide et moins coûteux en termes de calcul que d'autres techniques de classification.
* **Précision** : L'algorithme améliore la précision de la classification et permet d’ajuster les clusters en fonction des données spécifiques, améliorant ainsi les résultats.

## Inconvénients

* **Ensemble de clusters non optimaux** : K-means ne permet pas de générer un ensemble de clusters optimal, et le nombre de clusters (K) doit être déterminé à l'avance avant d'obtenir les résultats.
* **Manque de cohérence** : Les résultats du clustering K-means peuvent varier d'une exécution à l'autre. Un choix aléatoire des clusters initiaux peut entraîner des résultats différents, créant ainsi une incohérence.
* **Effet uniforme** : L'algorithme génère des clusters de taille similaire, même lorsque les données d'entrée présentent des tailles variées.
* **Sensibilité à l'ordre des données** : L'ordre dans lequel les données sont traitées peut influencer les résultats du clustering.
* **Limitations de calcul** : Pour de très grands ensembles de données, l'utilisation d'une technique comme le dendrogramme peut entraîner des problèmes de mémoire et des plantages en raison de la charge computationnelle élevée.
* **Données numériques uniquement** : K-means ne peut être utilisé que sur des données numériques et n'est pas adapté aux types de données non numériques.
* **Prédiction du nombre de clusters** : Il est difficile de prédire le nombre optimal de clusters (K), et la qualité des clusters obtenus est également complexe à évaluer.

# Réalisation et implémentation

## Configuration de l'environnement Spark

L'implémentation de notre système de détection d'intrusion en réseau repose sur une architecture distribuée basée sur Apache Spark. La configuration de l'environnement Spark a été un défi majeur, mais nous avons réussi à le surmonter en utilisant Docker pour créer un cluster maître-travailleur (master-worker) sur un réseau privé. Cette approche nous a permis de simuler un environnement de production dans un cadre de développement contrôlé.

## Création du cluster Spark avec Docker

Pour configurer notre environnement Spark, nous avons choisi d’utiliser Docker pour containeriser les instances Spark. Docker nous a offert une solution flexible et portable pour déployer nos conteneurs sur différents hôtes. Nous avons créé un cluster Spark comprenant un maître et plusieurs travailleurs, chacun encapsulé dans son propre conteneur Docker.

La création de ce cluster a été rendue possible grâce à l'utilisation d'un réseau Docker privé. Ce réseau privé a permis aux conteneurs de communiquer entre eux de manière sécurisée et efficiente, sans interférer avec d'autres services réseau sur nos machines hôtes. La configuration du réseau Docker a été un élément crucial pour assurer la communication sans heurts entre le maître et les travailleurs.

Intégration de travailleurs sur le même réseau

Un des avantages de notre configuration Docker était la possibilité d'ajouter des travailleurs supplémentaires à notre cluster Spark. Si plusieurs ordinateurs sont connectés au même réseau, ils peuvent facilement être configurés en tant que travailleurs additionnels. Cette flexibilité nous a permis d'escalader notre cluster en fonction des besoins en ressources computuelles.

Pour ce faire, nous avons configuré chaque ordinateur supplémentaire en tant que travailleur Spark en lançant un conteneur Docker avec la même image Spark que le maître. Les travailleurs ont été configurés pour se connecter au maître via l'adresse IP du réseau privé Docker. Cette configuration a été validée à travers plusieurs tests, où nous avons ajouté et supprimé des travailleurs à la volée, tout en maintenant la stabilité du cluster.

## Intégration de GitHub en tant que service cloud

Dans notre architecture, nous avons choisi d'utiliser GitHub comme service cloud pour partager l'API du maître Spark. Cette décision était motivée par la nécessité de disposer d'une solution simple et fiable pour partager des ressources et des configurations entre différents composants de notre système.

Nous avons utilisé GitHub Gist pour stocker et partager les configurations et les scripts nécessaires à la communication entre le maître et les travailleurs. GitHub Gist nous a offert une plateforme conviviale pour stocker des fichiers de configuration, des scripts de démarrage et d'autres artefacts essentiels. Cette approche a simplifié la gestion des configurations et a permis à notre équipe de collaborer efficacement, même lorsqu'elle travaillait à distance.

Configuration de Kafka

La configuration de Kafka a été un élément clé de notre architecture, car elle nous a permis de gérer le flux de données en temps réel provenant du réseau de trafic. Le volume des données était considérable, avec des millions de lignes de données par seconde, ce qui rendait Kafka idéal pour cette tâche.

Utilisation de Kafka pour le streaming de données

Kafka a été choisi comme solution de streaming de données en raison de sa capacité à gérer des flux de données massifs avec une faible latence et une haute fiabilité. Notre système de détection d'intrusion en réseau nécessitait de traiter des données en temps réel, ce qui a rendu Kafka indispensable.

Nous avons configuré Kafka pour capturer les données de trafic réseau à partir de différentes sources. Les données étaient ingérées dans des topics Kafka, qui servaient de points de jonction pour les consommateurs Spark. La configuration de Kafka a été optimisée pour gérer le volume élevé de données, en ajustant les paramètres de partitionnement, de réplication et de rétention des messages.

Intégration de FastAPI en tant que producteur

Pour alimenter Kafka avec les données de trafic réseau, nous avons développé une plateforme productrice basée sur FastAPI. FastAPI a été choisi en raison de sa simplicité d'utilisation et de sa performance élevée. Notre plateforme productrice était responsable de collecter les données de trafic réseau et de les envoyer aux topics Kafka appropriés.

La plateforme FastAPI a été conçue pour être scalable et résiliente. Elle a été testée sous différentes charges de travail pour s'assurer qu'elle pouvait gérer le volume de données attendu sans perte de performance. Les tests ont montré que FastAPI était capable de gérer des milliers de requêtes par seconde, ce qui était parfait pour notre cas d'utilisation.

Intégration de Spark en tant que consommateur

Spark a été configuré en tant que consommateur Kafka, responsable de traiter les données en temps réel. La configuration de Spark comme consommateur Kafka a été une étape cruciale pour assurer la continuité du traitement des données.

Nous avons utilisé la bibliothèque Kafka de Spark pour lire les données des topics Kafka. La configuration de Spark a été optimisée pour traiter les données en temps réel, en ajustant les paramètres de partitionnement, de mémoire et de parallélisme. Les données consommées par Spark étaient traitées en utilisant des RDD (Resilient Distributed Datasets), qui offrent une abstraction de haut niveau pour le traitement parallèle des données.

# Outils et bibliothèques utilisés

Notre projet a fait usage d'une variété d'outils et de bibliothèques open-source, chacun jouant un rôle crucial dans l'architecture globale de notre système de détection d'intrusion en réseau.

## PySpark pour le traitement distribué

PySpark a été le choix naturel pour le traitement distribué des données. PySpark nous a permis d'écrire du code Spark en Python, ce qui a facilité la collaboration au sein de l'équipe. PySpark a été utilisé pour lire les données Kafka, les transformer et appliquer les algorithmes de clustering pour détecter les intrusions.



## Kafka pour le streaming de données

Kafka a été utilisé pour gérer le flux de données en temps réel. Kafka a été choisi pour sa capacité à gérer des flux de données massifs avec une faible latence. Notre configuration Kafka a été optimisée pour gérer le volume de données attendu, en ajustant les paramètres de partitionnement et de réplication.



## MLlib pour l'analyse de clustering

MLlib, la bibliothèque de machine learning de Spark, a été utilisée pour appliquer des algorithmes de clustering sur les données de trafic réseau. MLlib nous a offert une gamme d’algorithm, y compris k-means, qui ont été utilisés pour détecter les comportements anormaux dans le trafic réseau.

FastAPI pour la production de données

FastAPI a été utilisé pour développer la plateforme productrice de données. FastAPI a été choisi pour sa performance élevée et sa simplicité d'utilisation. Notre plateforme productrice était responsable de collecter les données de trafic réseau et de les envoyer aux topics Kafka.



Docker pour la virtualisation

Docker a été utilisé pour containeriser les instances Spark et Kafka. Docker nous a permis de créer un environnement de développement isolé et répétable. Nos conteneurs Docker ont été configurés pour communiquer sur un réseau privé, ce qui a simplifié la gestion du cluster Spark.



GitHub pour la gestion des configurations

GitHub a été utilisé comme service cloud pour partager les configurations et les scripts nécessaires à notre système. GitHub Gist a été utilisé pour stocker et partager les fichiers de configuration et les scripts de démarrage. Cette approche a simplifié la gestion des configurations et a permis à notre équipe de collaborer efficacement.

# Prétraitement des Données

Le prétraitement des données est une étape essentielle dans tout projet d'analyse de données ou d'apprentissage automatique, et notre projet de détection d'intrusion en réseau n'échappe pas à cette règle. Dans le contexte de notre projet, nous travaillons avec des données de trafic réseau qui sont susceptibles d'être brutes, inconsistantes et contenant des informations superflues. Le prétraitement vise à rendre ces données aptes à l'analyse, en les nettoyant, en les standardisant et en les mettant en forme pour une utilisation optimale dans nos modèles de détection.

## 1. Nettoyage des Noms de Colonnes :

- Le premier pas consiste à standardiser les noms de colonnes pour garantir une accessibilité et une consistency uniformes. Dans les données de trafic réseau, il peut y avoir des noms de colonnes non standardisés ou contenant des erreurs, ce qui pourrait perturber les analyses ultérieures.

## 2. Suppression des Colonnes Irrelevantes :

- Il est crucial d'identifier et de supprimer les colonnes qui ne contribuent pas à l'analyse, telles que les doubles ou les caractéristiques irrelevantes. Par exemple, des champs tels que "Fwd Header Length" et "Fwd Header Length.1" pourraient être des doublons ou ne pas être utiles pour la détection des intrusions.

## 3. Gestion des Valeurs Infinites et Null :

- Les données de trafic réseau peuvent contenir des valeurs manquantes ou infinies dues à des erreurs de mesure ou de collecte. Il est essentiel de décider des stratégies pour gérer ces valeurs, que ce soit en les remplissant avec des valeurs par défaut, en les interpolant ou en supprimant les enregistrements concernés, pour maintenir l'intégrité des données.

## 4. Conversion des Étiquettes :

- Les étiquettes catégorielles, telles que "normal" et "intrusion", doivent être transformées en un format approprié pour les modèles d'apprentissage automatique. Cela peut impliquer une codification one-hot ou label encoding, selon la nature des données et des algorithmes utilisés.

5. Préparation des Caractéristiques :

- La sélection et la création de caractéristiques pertinentes pour la détection des intrusions sont essentielles. Cela peut inclure l'échelle des caractéristiques numériques, la codification des variables catégorielles et la création de termes d'interaction ou de caractéristiques dérivées pour améliorer les performances du modèle.

Le prétraitement dans un environnement Big Data, comme celui basé sur Apache Spark, nécessite une approche optimisée pour la performance dans un contexte distribué. Chaque étape de prétraitement doit être conçue pour s'intégrer efficacement dans la pipeline de traitement distribué.

Le prétraitement est également un processus itératif, qui peut nécessiter plusieurs passes sur les données, raffinant les étapes en fonction des analyses initiales. Il est important de garantir la validation et la reproductibilité des étapes de prétraitement, même sans code spécifique, en documentant clairement chaque étape pour assurer une transparence et une reproductibilité de la procédure.

Enfin, le prétraitement joue un rôle crucial dans la réussite de notre système de détection d'intrusion en facilitant l'application d'algorithmes de clustering et en assurant une efficacité et une précision optimales dans la détection des intrusions.

# Implémentation du système de détection

## Mise en place du streaming avec Kafka

**Introduction à Kafka dans le contexte du projet**

Kafka a été choisi pour sa capacité à gérer des flux de données massifs en temps réel, ce qui est essentiel pour notre système de détection d'intrusion en réseau. Kafka permet une ingestion de données.

**FastAPI comme producteur de données**

Nous avons développé une plateforme FastAPI qui lit les données des sources et les envoie sous forme de lignes JSON à Kafka. FastAPI assure une communication efficiente avec les sources de données, garantissant ainsi un flux continu et fiable de données vers Kafka.

**Configuration des topics Kafka et ingestion de données**

Les topics Kafka sont configurés pour gérer l'ingestion de données avec un nombre optimal de partitions. Cette configuration assure une distribution uniforme des données et évite les goulets d'étranglement.

**Avantages du streaming en temps réel avec Kafka**

Kafka permet de traiter les données en temps réel, ce qui est crucial pour la détection d'intrusions. Sa capacité à gérer de grandes quantités de données garantit une analyse proactive et une réponse rapide aux menaces potentielles.

Implémentation de l'apprentissage distribué

**Vue d'ensemble de l'apprentissage distribué avec Spark**

L'apprentissage distribué avec Apache Spark permet de traiter de vastes quantités de données en parallèle, ce qui est essentiel pour notre système de détection d'intrusion.

**Approche initiale avec la forêt aléatoire**

Nous avons initialement utilisé un modèle de forêt aléatoire en mode non streaming, où le modèle est entraîné une fois que toutes les données sont disponibles. Cependant, cette approche ne convient pas à un traitement en temps réel.

**Introduction de Streaming KMeans**

Streaming KMeans, issu du package MLlib, est un modèle d'apprentissage incrémental conçu pour le learning en ligne. Il met à jour le modèle à mesure que de nouvelles données arrivent, ce qui est idéal pour notre cas d'usage.

**Détails d'implémentation et raisonnement**

L'implémentation de Streaming KMeans permet d'adapter le modèle en temps réel, offrant une détection d'intrusion plus précise et réactive. Cette approche souligne l'importance de l'apprentissage incrémental pour les systèmes de détection d'intrusion.

## Optimisation des performances

**Stratégies d'optimisation des tâches Spark**

Nous avons ajusté les paramètres de Spark pour optimiser la parallélisation et la gestion de la mémoire, ce qui améliore les performances globales du système.

**Mesure et ajustement des performances**

Des tests de charge ont été effectués pour mesurer les performances et identifier les points de saturation. Des ajustements ont été apportés pour garantir une performance constante sous différents types de charge.

**Optimisations spécifiques pour Kafka et les algorithmes d'apprentissage**

Nous avons optimisé la configuration Kafka pour une ingestion de données plus rapide et avons choisi des formats de données sérialisés efficaces pour minimiser le temps de traitement.

**Résumé des optimisations**

Les optimisations apportées assurent que le système est capable de traiter de grandes quantités de données en temps réel, offrant une détection d'intrusion précise et efficiente.

Ce chapitre décrit en détail l'implémentation du système de détection d'intrusion en réseau, mettant en avant les choix techniques et les raisons qui les sous-tendent. Chaque section contribue à la création d'un système robuste et efficient, aligné avec les objectifs du projet.