

MASTER MIAGE 2ÈME ANNÉE  
UNIVERSITÉ PARIS NANTERRE

MÉMOIRE DE FIN D'ÉTUDES

---

**Méthodes d'analyse des processus  
métier pour le choix d'une  
architecture Big Data adaptée**

---



*Auteur :*  
LUDWIG SIMON

*Tuteur :*  
MCF. EMMANUEL HYON

Février 2019 — Juin 2019



## Remerciements

Remerciements



Résumé

Résumé



## Motivations

Le Big Data est un domaine très vaste, il y a une multitude d'outils pour effectuer les "mêmes" tâches. Il est donc très difficile de faire le bon choix lorsqu'on se lance dans ce domaine.

## Objectifs

Dans ce mémoire, nous allons dans un premier temps rappeler brièvement ce qu'est le Big Data et quand il est vraiment utile de l'utiliser. Dans un second temps nous allons devoir établir des critères permettant d'évaluer quelle catégorie d'outil convient le mieux et par la suite quel outil conviendra le mieux. Pour le choix de l'outil on utilisera en plus de critères de cas d'utilisation des benchmarks afin de récupérer des informations sur la consommation de ressources et de temps pour chaque outil. Ensuite, à l'aide de ces critères définis précédemment, nous allons définir des méthodes d'analyse afin de sélectionner les outils adéquats. Et pour finir nous allons appliquer nos méthodes sur un cas concret afin de vérifier l'efficacité de notre solution.





# Sommaire

<b>1</b>	<b>Les architectures Big Data</b>	<b>9</b>
1.1	Architecture Data Lake . . . . .	10
1.2	Architecture Lambda . . . . .	10
1.3	Architecture Kappa . . . . .	11
<b>2</b>	<b>Analyse des solutions logicielles existantes</b>	<b>13</b>
2.1	Message Broker . . . . .	13
2.2	Ingestion/Extraction de données . . . . .	14
2.3	Traitement des données . . . . .	14
2.4	Stockage des données . . . . .	15
2.5	Visualisation et Analyse des données . . . . .	16
<b>3</b>	<b>Critères d'analyse</b>	<b>17</b>
3.1	Type de traitement des données . . . . .	17
3.2	Format des données . . . . .	17
3.3	Perte de données admissible . . . . .	17
3.4	Volumétrie . . . . .	17
3.5	Performance . . . . .	17
<b>4</b>	<b>Implémentation : Exemple avec un processus métier</b>	<b>19</b>
4.1	Définition du processus métier . . . . .	19
4.2	Application des méthodes sur le processus métier . . . . .	19
4.3	Évaluation du résultat . . . . .	19
<b>Annexe A</b>	<b>Les différents concepts du Big Data</b>	<b>21</b>
A.1	Architecture Réactive . . . . .	21
A.2	Architecture Répartie . . . . .	23
A.3	Traitement des données . . . . .	23
	<b>Bibliographie</b>	<b>25</b>



# Introduction

## **Le Big Data**

## **Quand peut-on utiliser le Big Data ?**



# Les architectures Big Data

Une architecture Big Data est un regroupement d'outils permettant de gérer des données de leurs ingestion à leurs mise en valeur via des analyses. Il faut savoir qu'il existe plusieurs architectures dans le domaine du Big Data, et qu'elles répondent à des besoins différents. Nous allons nous intéresser aux trois architectures les plus importantes, étant donné que les autres sont des dérivées des trois architectures principales. Avant de voir en détail ces trois architectures, nous allons voir de manière plus générale les différents composants qui peuvent se retrouver dans des architectures Big Data [1]. Les différents composants pouvant se retrouver dans une architecture Big Data sont illustrés sur la figure 1.1.

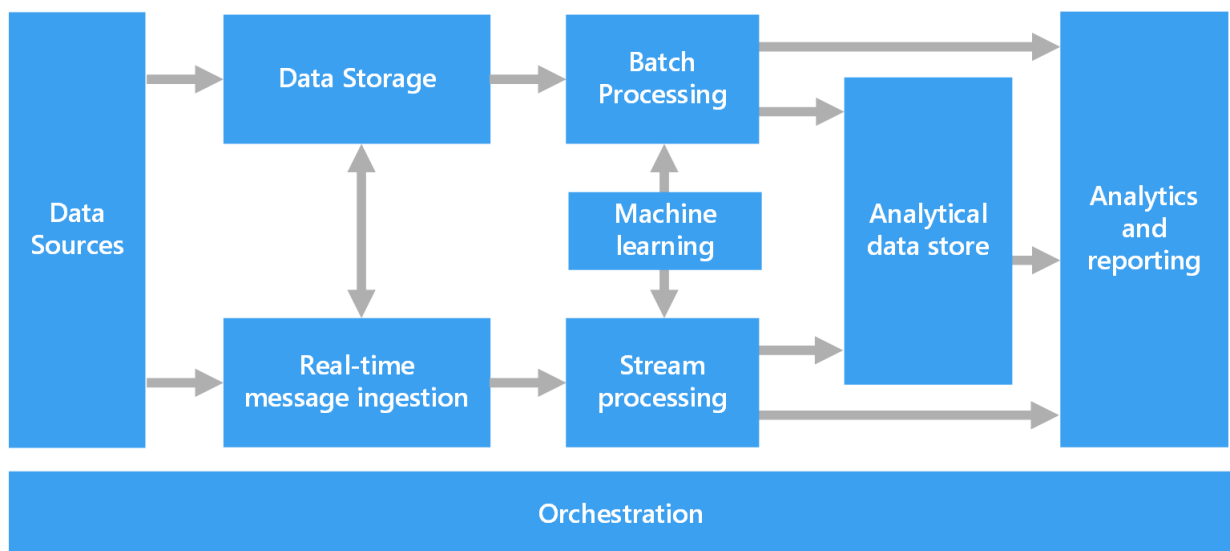


Figure 1.1 – Composants d'une architecture Big Data

Nous allons maintenant détailler le rôle de chaque composant présenté dans le diagramme ci dessus.

- **Source de données :** N'importe quelle solution Big Data a besoin d'une source de donnée en entrée. Voici quelques exemple de données qui peuvent être utilisée dans une architecture Big Data
  - Des données issues de bases de données relationnelles.
  - Des fichiers statiques produits par des applications, des fichiers de logs par exemple.
  - Des sources de données en temps réel, par exemple des données de capteurs récupérés via des appareils IoTs.

- **Stockage des données** : Une solution de stockage de données est indispensable dans le cas de traitement des données par lot (A.3.1), et peut s'avérer utile lors de traitements des données en temps réel (A.3.2) si l'on souhaite conserver les données reçus en plus de les traiter. Le deuxième cas où un stockage de données peut être utile pour le traitement en temps réel, est si l'on a besoin d'agréger des données statiques avec les données en temps réel. Ces solutions de stockage doivent être capables de gérer divers formats de données, et surtout ils doivent être distribués (A.2).
- **Traitement par lot (A.3.1)** : Les jeux de données étant trop lourds lors d'un traitement par lots, il est nécessaire de pouvoir exécuter une tâche de traitement de longue durée afin de filtrer, agréger et préparer les données en vue de leur analyse. En général, ce genre de traitement implique une lecture de fichier source et une écriture dans des nouveaux fichiers. Le traitement des données peut s'effectuer par des programmes Java, Scala ou Python ou encore via des outils spécialisés comme MapReduce ou Spark.
- **Ingestion de messages en temps réel** : Si la solution doit interagir avec des sources de données en temps réel, l'architecture doit impérativement implémenter un moyen de récupérer ces données et de les stocker temporairement dans une file d'attente afin de pouvoir les temporiser. Cela permet d'éviter la perte de données entrantes, et permet d'envoyer les données à la solution de traitement quand elle est disponible et de ne pas la surcharger. Généralement on utilise des messages brokers pour ce genre de tâches.
- **Traitement de flux (A.3.2)** : Une fois les données récupérées, elles doivent être filtrées, agrégées puis préparées pour l'analyse.
- **Magasin de données analytique** :
- **Analyse, rapports et visualisation des données** :
- **Orchestration** : La majorité des solutions Big Data consistent à effectuer des traitements de données répétitifs, ayant pour but de transformer les données sources, les déplacer entre plusieurs endroits et les envoyer dans un magasin de données analytique ou bien les envoyer directement à un outil d'analyse.

## 1.1 Architecture Data Lake

## 1.2 Architecture Lambda

L'architecture Lambda a été créée dans le but de pouvoir à la fois traiter des données en batch et en streaming. Plus précisément, l'architecture Lambda est composée de deux couches afin de gérer à la fois les batch et le streaming. Une couche est dédiée pour traiter les données par batch, pour ensuite les rendre disponibles (ce qui est appelé une "view"). L'autre couche, s'occupe du streaming. Et à chaque donnée traitée, le résultat est mis à disposition. Grâce à ce fonctionnement lorsque l'on souhaite faire une requête, elle va pouvoir prendre en compte nos données récupérées par batch et par streaming.

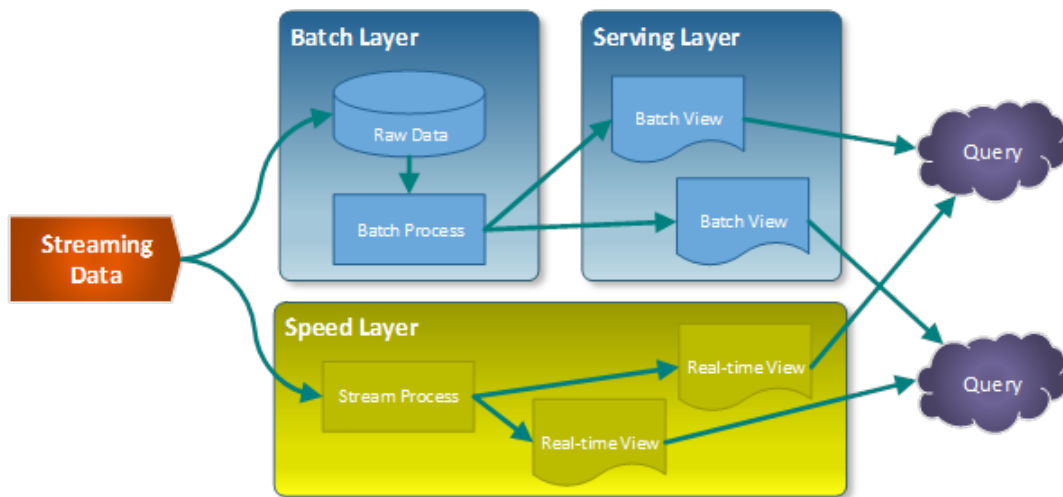


Figure 1.2 – Schéma de l'architecture Lambda

### 1.3 Architecture Kappa

L'architecture Kappa, se veut plus "simple". Le but étant de traiter toutes les données en streaming. Cela a pour conséquence de n'avoir qu'une seule couche contrairement aux deux nécessaires pour l'architecture Lambda. Mais cela ne permet pas de remplacer l'architecture Lambda, en effet il faut que les données que l'on récupère en Batch et la manière dont elles sont stockées, permettent de les traiter en streaming par la suite.

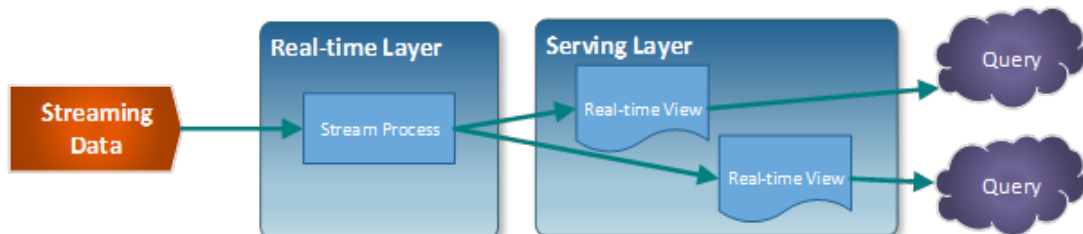


Figure 1.3 – Schéma de l'architecture Kappa





# Analyse des solutions logicielles existantes

Une fois nos critères définit, il faut décomposer le Big Data qui est un domaine très vaste, en plusieurs catégories [2]. Une fois cette décomposition effectuée, on va pouvoir s'intéresser aux outils existant permettant de réaliser chacune des tâches. On va noter leurs points faibles et leurs points forts ainsi que la manière dont ils sont censés être utilisés.

## 2.1 Message Broker

Un Message broker [3], Agent de message en français, est un moyen de communication utilisant des messages entre deux applications (Ex : Communication entre un serveur et un client). Un message broker permet une communication asynchrone entre applications. L'utilisation de cette solution permet de pouvoir facilement filtrer les messages que l'on reçoit et de stocker temporairement les messages reçus afin d'éviter les pertes de données. Ce dernier cas, s'avère très utile dans le cas où l'application chargée de la réception des données n'est pas en fonctionnement pendant un certains temps. Il existe deux types de communications avec un message broker :

### **Publisher / Subscriber**

Dans ce mécanisme, l'entité envoyant les données est nommé "Publisher" et l'entité les récupérant est nommé "Subscriber". Le publisher va envoyer des données dans des topics<sup>1</sup> afin que les Subscribers de ce topic puissent les récupérer. Un publisher peut envoyer des données dans un ou plusieurs topics, et les subscribers peuvent être abonné à un ou plusieurs topics (Voir figure 2.1).

### **Point-to-point communication**

La communication point à point est la forme la plus simple de Producteur/Consommateur. Le producteur envoie ses données dans une queue et le consommateur va lire les messages dans la queue. Tout comme le modèle précédent, il peut y avoir plusieurs

---

1. Un topic est une catégorie dans laquelle les messages produits sont stockés.

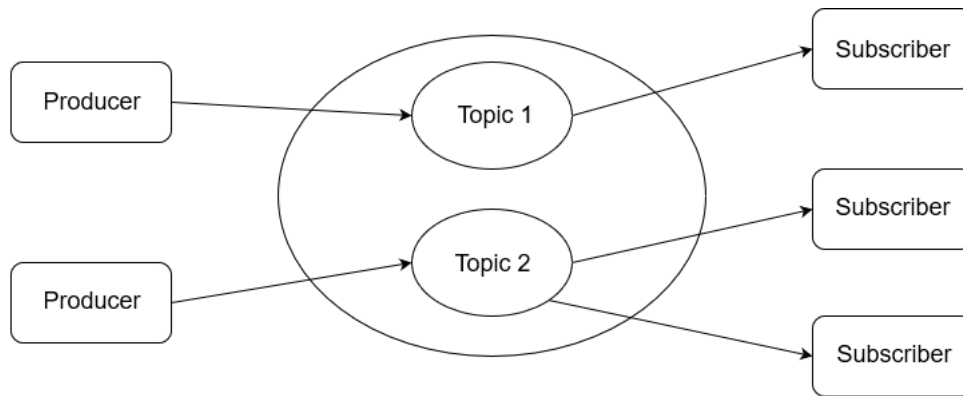


Figure 2.1 – Schéma du principe Producer/Subscriber

producteur et consommateurs sur la même queue, mais si plusieurs consommateurs sont présents, ils ne recevront des portions différentes des messages afin de favoriser le traitement concurrentiel.

### 2.1.1 Kafka

### 2.1.2 ActiveMQ

### 2.1.3 RabbitMQ

## 2.2 Ingestion/Extraction de données

La première catégorie, qui est aussi la première étape d'une architecture Big Data, c'est le récupération de données. Plus précisément comment nous allons récupérer des données, soit via des requêtes sur des sources externes, soit des sources externes nous envoient directement des données.

### 2.2.1 Apache Nifi

### 2.2.2 Talend

## 2.3 Traitement des données

Une fois les données reçues, des traitements sont nécessaires afin de pouvoir stocker les données au format souhaité ou bien pour faire un tri des données utiles.

## **2.3.1 Batch**

### **2.3.1.1 Spark**

### **2.3.1.2 Hadoop MapReduce**

## **2.3.2 Streaming**

### **2.3.2.1 Spark Streaming**

### **2.3.2.2 Apache Storm**

## **2.4 Stockage des données**

Une partie très importante du Big Data est le stockage des nombreuses données que l'ont reçoit. Il existe énormément de manières différentes de stocker des données selon la manière dont nous voulons les utiliser par la suite.

## **2.4.1 Time Series**

### **2.4.1.1 OpenTSDB**

### **2.4.1.2 InfluxDB**

## **2.4.2 Graph**

### **2.4.2.1 Neo4j**

### **2.4.2.2 JanusGraph**

## **2.4.3 Clés/Valeurs**

### **2.4.3.1 Redis**

### **2.4.3.2 RoxDB**

## **2.4.4 Documents**

### **2.4.4.1 CouchDB**

### **2.4.4.2 CouchBase**

### **2.4.4.3 MongoDB**

## **2.4.5 Wide Column**

### **2.4.5.1 HBase**

### **2.4.5.2 Cassandra**

## **2.4.6 Système de fichiers**

### **2.4.6.1 Hadoop HDFS**

## **2.5 Visualisation et Analyse des données**

### **2.5.1 Kibana**

### **2.5.2 Banana**

### **2.5.3 Grafana**

### **2.5.4 Tableau**

### **2.5.5 Click**

## Critères d'analyse

### **3.1** Type de traitement des données

Streaming - Micro Batch - Batch

### **3.2** Format des données

### **3.3** Perte de données admissible

### **3.4** Volumétrie

### **3.5** Performance



## Implémentation : Exemple avec un processus métier

### **4.1** Définition du processus métier

### **4.2** Application des méthodes sur le processus métier

### **4.3** Évaluation du résultat





# Les différents concepts du Big Data

## A.1 Architecture Réactive

Le nombre de données augmentant très rapidement et les clients demandant un service le plus rapide possible et disponible à tout moment, il est de ce fait important d'avoir une infrastructure facilement adaptable à ce flux. C'est le but de l'architecture réactive [4]. Elle repose sur quatre principes :

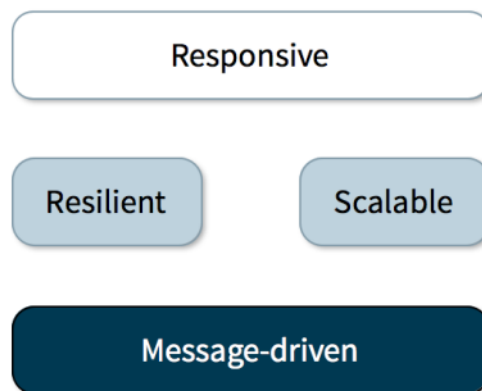


Figure A.1 – Schéma de l'architecture réactive

- Responsive : C'est l'objectif à atteindre.
- Scalable et Resilient : L'objectif ne peut pas être atteint sans remplir ces deux conditions.
- Message Driven : C'est la base qui permettra d'accueillir tous les composants afin d'obtenir une architecture réactive

**Responsive** : Un système responsive doit être en mesure de réagir rapidement à toutes les requêtes peu importe les circonstances, dans le but de toujours fournir une expérience utilisateur positive. La clé afin de proposer ce service, est d'avoir un système élastique et résilient, les architecture "Message Driven" fournissent les bases pour un système réactif. L'architecture Message Driven est aussi important pour avoir un système responsive, parce que le monde fonctionne de manière asynchrone tout comme elle. Voici un exemple : Vous voulez vous faire du café, mais vous vous rendez compte que vous n'avez plus de crème et de sucre.

Première approche :

- Mettre du café dans votre machine.
- Aller faire vos courses pendant que la machine fait couler le café.

- Acheter de la crème et du sucre.
- Rentrer chez vous.
- Boire votre tasse de café.
- Profiter de la vie !

Seconde approche :

- Aller faire vos courses.
- Acheter de la crème et du sucre.
- Rentrer chez vous.
- Mettre du café dans votre machine.
- Regarder impatiemment la cafetière se remplir.
- Expérimenter le manque de caféine.
- Et enfin, boire votre tasse de café.

Grâce à la première approche, on peut clairement voir la différence de gestion de l'espace et du temps. C'est grâce à cela que cette architecture est un atout primordiale pour assurer un système responsive.

### **Resilient :**

La plupart des applications sont conçus par rapport à un fonctionnement idéal. C'est à dire qu'elle ne prévoient pas de gérer correctement les erreurs qui peuvent intervenir plus souvent que ce qu'on pourrait croire. Cela à pour effet de ne pas garantir une disponibilité continu et peut provoquer la perte de crédibilité d'un service. La résilience est là pour pallier à ce problème, elle à pour but de récupérer les erreurs émises et de les traiter correctement afin de ne pas provoquer d'interruption de service et de ne pas impacter l'expérience utilisateur.

### **Scalable :**

La résilience et l'élasticité vont de pair dans la construction d'une architecture réactive. L'élasticité permet d'adapter facilement le système à la demande et d'assurer sa réactivité peu importe la charge qu'il subit. L'élasticité est un point très important, lorsque votre plateforme reçoit énormément de connexion cela veut dire que votre service connaît un succès important, c'est donc le pire moment pour avoir un service non disponible. Cela entrainerait une perte de crédibilité et de clients.

Il existe deux moyens différents de rendre élastique une application :

- La mise à l'échelle verticale
- La mise à l'échelle horizontale (Architecture Répartie)

La mise à l'échelle verticale consiste à maximiser l'utilisation des ressources de notre machine. Cela peut se faire par l'utilisation de programmes fonctionnant en asynchrone sur plusieurs threads. Tandis que la mise à l'échelle horizontale consiste à augmenter le nombre de machine afin d'avoir plus de ressources à notre disposition. Pour le moment on ne vas pas trop s'attarder sur la division horizontale car une partie lui est dédiée.

La mise à l'échelle verticale est certes la moins couteuse, mais ce n'est pas la plus simple à mettre en place. En effet le multithreading permet de tirer la maximum des performances de la machine, mais cela ajoute une certaine complexité au programme. Par exemple le fait de travailler avec des variables mutables entre différents threads n'est pas une tâche aisée, cela provoque donc des limitations à la mise à l'échelle

verticale. De plus, une fois avoir exploiter au maximum une machine en ayant mis en place la mise à l'échelle verticale, si on a encore besoin de ressources supplémentaire la mise à l'échelle horizontale devient indispensable.

### Message-Driven :

Le Message-Driven Architecture

## A.2 Architecture Répartie

## A.3 Traitement des données

Après avoir récupérer des données, nous devons passer à l'étape du traitements des données. Celui à plusieurs rôles, en effet il peut servir à formater les données, leurs apportés de la cohérence en les combinant à des données déjà présentent. Et pour finir les rediriger vers le stockage souhaité. Le traitement des données peut se faire de deux manière différentes. La première solution est le traitement par Batch, et la seconde est le traitement en temps réel [5]. Chacune possède ses avantages et inconvénients, nous allons voir ça plus en détails.

### A.3.1 Batch

Le traitement par Batch, consiste à traiter un important volume de données à un instant T. Le traitement par batch est surtout utilisé dans les cas ou nous avons des données stockés de manière journalière, et que nous avons besoin de tout traités en fin de journée. Il n'est pas rare de voir des tâches de traitements par Batch s'exécuter dans la nuit, étant donnée que l'on traite une masse de donnée importante, on sollicite la machine pendant une longue période. Réaliser ce traitement durant des périodes creuses permet de largement diminuer l'impact sur l'utilisation de la plateforme.

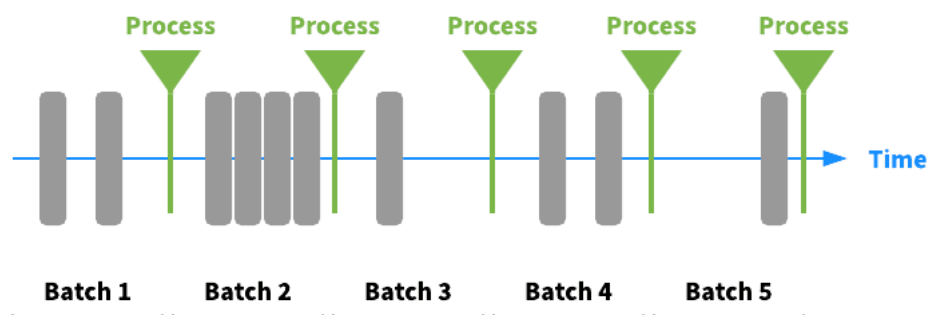


Figure A.2 – Schéma du traitement par batch

### A.3.2 Temps Réel

Un traitement de données est considéré comme étant en temps réel si il s'effectue en une seconde ou moins après la réception de la donnée. Il peut être de deux types, soit des micro batchs soit en streaming.

#### A.3.2.1 Micro-Batch

Le traitement par micro batch est basé sur le même principe que le traitement par batch, à l'exception qu'il s'exécute beaucoup plus régulièrement (Toutes les secondes ou moins) et que le nombre de données à traiter est donc significativement plus faible. Le micro batch est surtout utilisé dans les cas où notre système ne peut pas directement réagir lorsqu'une donnée arrive, on va donc récupérer les données très régulièrement afin de garantir un traitement en temps réel ou du moins dans le délai le plus bref possible.

#### A.3.2.2 Streaming

Le traitement en streaming s'appuie sur l'architecture réactive. En effet, contrairement aux traitements par batch et micro batch, ici on ne va pas récupérer des données de temps en temps. Dès qu'une donnée arrive on va la récupérer et la traiter immédiatement. De par son fonctionnement le traitement en streaming ne nécessite pas de stockage en amont contrairement aux autres type de traitement.

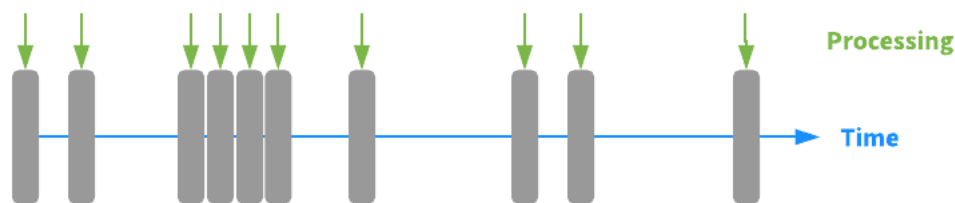


Figure A.3 – Schéma du traitement en streaming

De manière générale, on peut pas forcément se permettre de n'utiliser que l'un de ces deux type de traitement de données, nous avons régulièrement besoin de les utiliser en même temps.

# Bibliographie

- [1] Zoiner Tejada et olprod. *Architectures Big Data*. url : <https://docs.microsoft.com/fr-fr/azure/architecture/data-guide/big-data>.
- [2] Christophe Parageaud. *Big Data, panorama des solutions*. url : <https://blog.ippon.fr/2016/03/31/big-data-panorama-des-solutions-2016/>.
- [3] <https://www.tibco.com>. *What is a Message Broker?* url : <https://www.tibco.com/reference-center/what-is-a-message-broker>.
- [4] Kevin Webber. *What is Reactive Programming?* url : <https://blog.redelastic.com/what-is-reactive-programming-bc9fa7f4a7fc>.
- [5] Laura Shiff. *Real Time vs Batch Processing vs Stream Processing : What's The Difference?* url : <https://www.bmc.com/blogs/batch-processing-stream-processing-real-time/>.



# Table des matières

<b>1</b>	<b>Les architectures Big Data</b>	<b>9</b>
1.1	Architecture Data Lake . . . . .	10
1.2	Architecture Lambda . . . . .	10
1.3	Architecture Kappa . . . . .	11
<b>2</b>	<b>Analyse des solutions logicielles existantes</b>	<b>13</b>
2.1	Message Broker . . . . .	13
2.1.1	Kafka . . . . .	14
2.1.2	ActiveMQ . . . . .	14
2.1.3	RabbitMQ . . . . .	14
2.2	Ingestion/Extraction de données . . . . .	14
2.2.1	Apache Nifi . . . . .	14
2.2.2	Talend . . . . .	14
2.3	Traitement des données . . . . .	14
2.3.1	Batch . . . . .	15
2.3.1.1	Spark . . . . .	15
2.3.1.2	Hadoop MapReduce . . . . .	15
2.3.2	Streaming . . . . .	15
2.3.2.1	Spark Streaming . . . . .	15
2.3.2.2	Apache Storm . . . . .	15
2.4	Stockage des données . . . . .	15
2.4.1	Time Series . . . . .	16
2.4.1.1	OpenTSDB . . . . .	16
2.4.1.2	InfluxDB . . . . .	16
2.4.2	Graph . . . . .	16
2.4.2.1	Neo4j . . . . .	16
2.4.2.2	JanusGraph . . . . .	16
2.4.3	Clés/Valeurs . . . . .	16
2.4.3.1	Redis . . . . .	16
2.4.3.2	RoxDB . . . . .	16

2.4.4	Documents . . . . .	16
2.4.4.1	CouchDB . . . . .	16
2.4.4.2	CouchBase . . . . .	16
2.4.4.3	MongoDB . . . . .	16
2.4.5	Wide Column . . . . .	16
2.4.5.1	HBase . . . . .	16
2.4.5.2	Cassandra . . . . .	16
2.4.6	Système de fichiers . . . . .	16
2.4.6.1	Hadoop HDFS . . . . .	16
2.5	Visualisation et Analyse des données . . . . .	16
2.5.1	Kibana . . . . .	16
2.5.2	Banana . . . . .	16
2.5.3	Grafana . . . . .	16
2.5.4	Tableau . . . . .	16
2.5.5	Click . . . . .	16
<b>3</b>	<b>Critères d'analyse</b>	<b>17</b>
3.1	Type de traitement des données . . . . .	17
3.2	Format des données . . . . .	17
3.3	Perte de données admissible . . . . .	17
3.4	Volumétrie . . . . .	17
3.5	Performance . . . . .	17
<b>4</b>	<b>Implémentation : Exemple avec un processus métier</b>	<b>19</b>
4.1	Définition du processus métier . . . . .	19
4.2	Application des méthodes sur le processus métier . . . . .	19
4.3	Évaluation du résultat . . . . .	19
<b>Annexe A</b>	<b>Les différents concepts du Big Data</b>	<b>21</b>
A.1	Architecture Réactive . . . . .	21
A.2	Architecture Répartie . . . . .	23
A.3	Traitement des données . . . . .	23
A.3.1	Batch . . . . .	23
A.3.2	Temps Réel . . . . .	24
A.3.2.1	Micro-Batch . . . . .	24
A.3.2.2	Streaming . . . . .	24







# Table des figures

1.1	Composants d'une architecture Big Data . . . . .	9
1.2	Schéma de l'architecture Lambda . . . . .	11
1.3	Schéma de l'architecture Kappa . . . . .	11
2.1	Schéma du principe Producer/Subscriber . . . . .	14
A.1	Schéma de l'architecture réactive . . . . .	21
A.2	Schéma du traitement par batch . . . . .	23
A.3	Schéma du traitement en streaming . . . . .	24