**BARRY Alpha**

**COUDERETTE Allan**

**CLAUDE Benjamin**

**HOFFER Max**

**Le BIG DATA**

Sommaire:

I/ Présentation du big data

II/ Les logiciels de big data

1. Les logiciels liés au big data
2. L’environnement hadoop
3. Installation

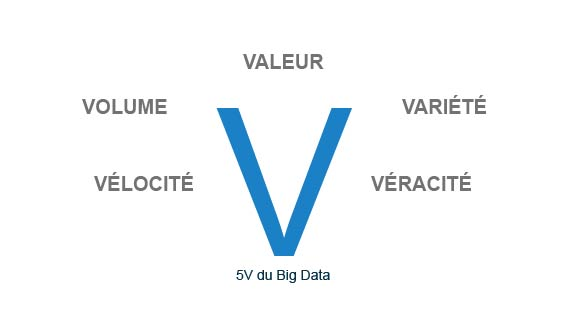
III/ La gestion de projet

I/ Présentation du big data

Qu’est-ce que le big data ?  
  
 Le big data c’est la gestion d’une énorme quantité de données. Ces données sont généralement tellement nombreuses qu’elles ne sont pas ou peu exploitable de manière optimisées avec des logiciels de gestion classique.

Les 5V du big data

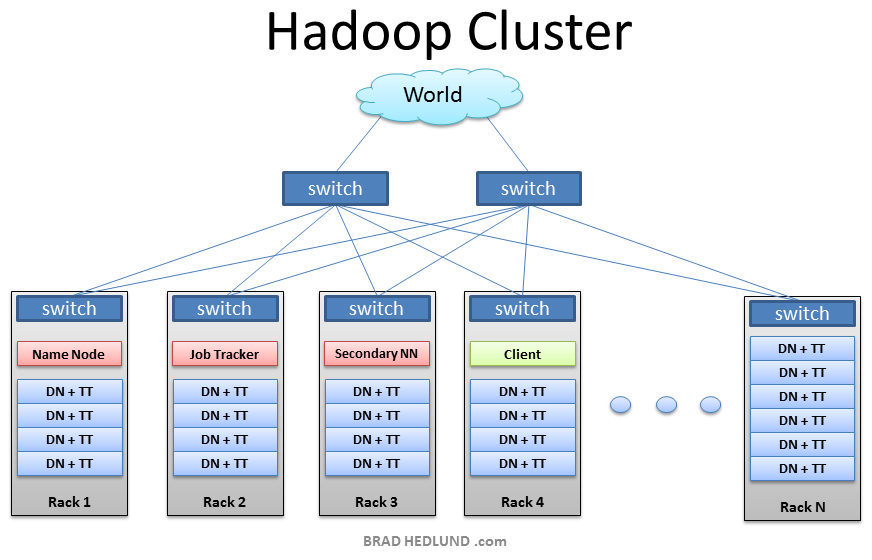
Les 5V du big data sont ce qui défini un système dit de big data. Ils signifient:



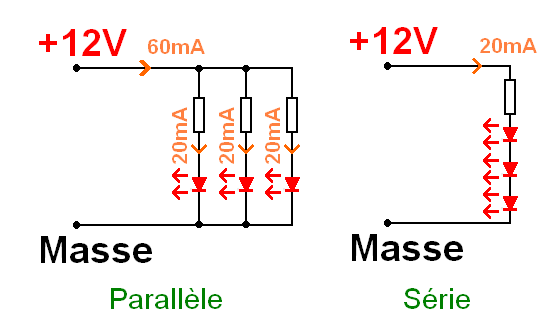
**Valeur** : C’est à dire ne pas stocker des données inutiles mais qui ont une valeurs. Cette valeur peut-être financière ou statistique.  
  
**Volume** : C’est d’avoir une quantité importantes de données.  
  
**Vélocité** : D’avoir un système qui répond rapidement face à la demande d’information ou à l’ajout de nouvelles données.  
  
**Variété** : Différent type de données. Les données peuvent être de tout type il peut même s’agir de fichiers binaire.   
  
**Véracité** : Les données soit exactes. (exemple: si notre source est un formulaire il se peut que des informations soit fausse).  
  
Ces 5V nous amène vers un autre point : les bases de données NoSQL.

Un cluster de serveurs, c’est quoi?

Un cluster de serveurs est simplement un ensemble de serveurs de données fonctionnant avec la même interface et étant lié entre eux. C’est l’une des caractéristique du BigData. Ils sont donc tous liées. Si un des serveurs à une surcharge de travaille elle va se répartir à d’autre et inversement.



Ils permettent une redondance des données et donc permet d’isoler les serveur en cas de pannes. On peut faire le parallèle avec des LED monté en série et en parallèle. En série si une led est défaillante toutes les autres s'éteignent alors que en parallèle juste une seul est éteinte.   
  
De même pour la répartition de charge. Si l’on augmente l’intensité dans le système elle va se répartir à part égal dans le nombre de LED et non surcharger une seule et même LED. Si on passe 63 mA dans le circuit, chaque branche recevras 21 mA.



Le Système HDFS

Hadoop Distributed File System (HDFS) est un système de fichier distribué permettant le stockage et la récupération de fichiers.  
  
Le NoSQL

Pour mettre en place une base de données répondant aux demandes du BIGDATA, L’utilisation des bases de données classique était impossible. Il a donc été développé un nouveau style de base de données dit NoSQL pour « Not only SQL » répondant à plusieurs caractéristiques :

- Le coût raisonnable et la facilité de mise en œuvre.

- Le partitionnement et la copie des fichiers de données sur plusieurs machines.

- La structure dynamique n’ayant pas de schéma de données fixe.

- L’évolutivité en ajoutant des colonnes, ce qui permet de traiter les données plus rapidement, surtout les plus volumineuses.

- La rapidité du transfert des données comparé aux bases de données classiques.

- L’évolutivité en ajoutant des nœuds supplémentaires dans le Cluster sans avoir besoin de faire une répartition.

Ces caractéristiques font que les bases de données NoSQL ne respectent pas les règles classiques des bases de données relationnels. En effet, les base de données NoSQL ne sont pas conformes au propriétés ACID et sont limités par le théorème CAP. Les réseaux sociaux appliquent fortement l’utilisation des bases de données NoSQL, vu leurs besoins compatibles avec CAP et contrairement aux banques nécessitant plus de rigidité.

Pour citer un exemple une banque ne peux pas utiliser de NoSQL car ceci présenterait un gros risque sur le plan économique. En effet si une transaction est effectuée il faut que l’opération de débit et de crédit se fasse sans encombre sans quoi il y a risque de perte. C’est pour cela que des mesures de sécurité sont appliqué avec l’utilisation du SQL qui fait que si des opérations ne se sont pas bien dérouler tout est annulé. Ces mesures de sécurité ralentissent donc l'accès aux données ce qui peut poser problème lors ce qu’il y a un très gros volume.

ACID vs BASE

Les propriété ACID pour les transactions d’une base de données relationnelle sont :

* **A**tomicité : Une transaction s’effectue entièrement ou pas du tout
* **C**ohérence : Le contenu d’une base doit être cohérent au début et à la fin d’une transaction
* **I**solation : Les modifications d’une transaction ne sont visibles/modifiables que quand celle-ci a été validée
* **D**urabilité : Une fois la transaction validée, l’état de la base est permanent (non affecté par les pannes ou autre)

Comme on l’a vu précédemment, Une base de données NoSQL ne respecte pas ces propriétés ACID. En effet, une base de données NoSQL peut être déployé sur plusieurs serveur ce qui nécessite une synchronisation de l’ensemble des serveurs lors d’une transaction. La transaction sera donc lente.

On peut néanmoins remarquer qu’une base de données NoSQL est caractérisé par des propriétés, on nomme cet ensemble de propriété « BASE » :

* **B**asically **A**vailable : quelle que soit la charge de la base de données (données/requêtes), le système garantie un taux de disponibilité de la donnée
* **S**oft-state : La base peut changer lors des mises à jour ou lors d'ajout/suppression de serveurs. La base NoSQL n'a pas à être cohérente à tout instant
* **E**ventually consistent : À terme, la base atteindra un état cohérent

Avec NoSQL, on remarque que la disparition de contrainte ACID permet d’optimiser le temps de réponse entre les transactions.

## Le théorème de Brewer

Le théorème de Brewer (dit théorème CAP) **caractérise une base de données en 3 points** :

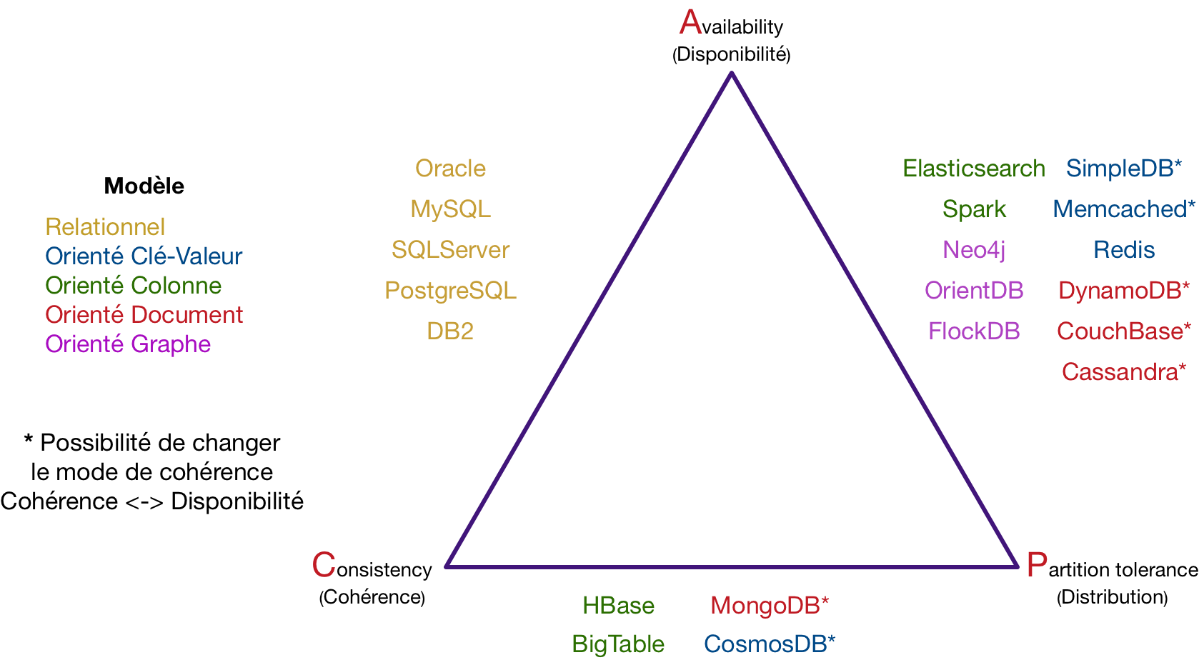
* **C**onsistency (Cohérence). Les données sont uniformisées, il n’y a pas deux manières de rentrer une même information.
* **A**vailability (Disponibilité). La donnée doit être disponible.
* **P**artition Tolerance (Distribution). Le résultat retourné doit être correcte peut importe le nombre de serveurs utilisés pour stocker les données.

Ce théorème nous dit aussi **qu’une base de données ne peut respecter au mieux que 2 points**. Une base de données peut donc être caractérisé par trois couples :

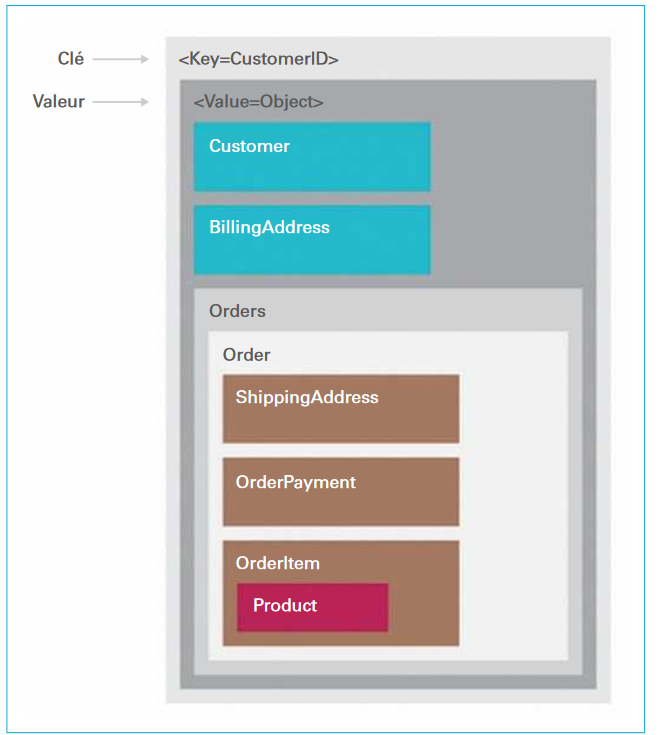
**CA** : Cohérence et Disponibilité, ceci n’est possible que dans le cadre des Base de données relationnel (donc les SQBDR)

**CP** : Cohérence et Distribution, on répartit notre information sur plusieurs serveur. La communication avec les différents serveurs prend du temps, on ne peut pas garantir la disponibilité des données. On retrouve cela dans la base NoSQL MongoDB.

**AP** : Disponibilité et Distribution, on privilégie le temp de réponse à la cohérence de l’information. La base de données NOSQL Cassandra a ces caractérisations.



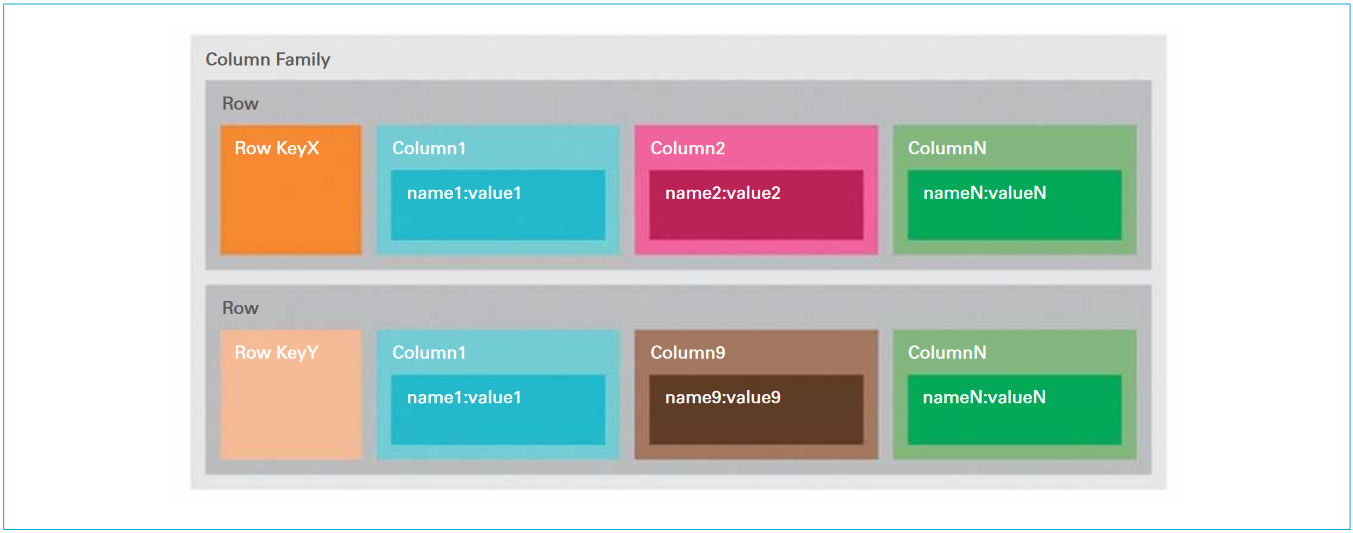
Certaines Base de données permettent de changer de caractéristique. On peut notamment passer du **AP** au **CP** avec MongoDB.

Il existe plusieurs type de base de données NoSQL.  


### Modèle orienté « clé-valeur »

Les données sont simplement représentées par un couple clé-valeur. La valeur peut être une simple chaîne de caractères, un objet sérialisé... Cette absence de structure et de typage a un impact important sur le requêtage. Ainsi la complexité d’une requête qui sera dans un système relationnel prise en charge par le langage SQL, sera dans ce modèle NoSQL prise en charge par l’application interrogeant la base de données, la communication avec la base se limitant généralement à des ordres PUT, GET et DELETE. Les systèmes NoSQL orientés clé-valeur les plus connus sont Memcached, Amazon’s Dynamo, Redis, Riak et Voldemort créé par LinkedIn.

### Modèle orienté « colonnes »

Ce modèle (figure 3) ressemble à première vue à une table du modèle relationnel, du fait que les attributs sont regroupés en famille de colonnes. Ainsi, deux attributs qui sont fréquemment interrogés ensemble seront stockés au sein d’une même famille de colonnes. Cependant la différence est que, dans cette base NoSQL orientée colonnes, le nombre de colonnes est dynamique, alors que dans une table relationnelle, le nombre de colonnes est fixé dès la création du schéma de la table. De plus, dans ce modèle, contrairement au modèle relationnel, le nombre de colonnes peut varier d’un enregistrement à un autre, ce qui évite de retrouver des colonnes ayant des valeurs inconnues (Null Value). Les systèmes NoSQL orientés colonnes les plus connus sont principalement HBase, implémentation Open Source du modèle BigTable développé par Google, et Cassandra, projet Apache qui respecte l’architecture distribuée de Dynamo d’Amazon, et le modèle BigTable de Google.

### Modèle orienté « documents »

Ce modèle (figure 2) se base sur le paradigme clé-valeur précédent ; cependant dans ce nouveau modèle la valeur est un document de type JSON ou XML. Ainsi, dans ce modèle, les données sont stockées à l’intérieur de documents. Un document peut être vu comme un n-uplet d’une table dans le monde relationnel, à la différence toutefois que les documents peuvent avoir une structure complètement différente les uns des autres. L’avantage est de pouvoir récupérer, via une seule clé, un ensemble d’informations structurées de manière hiérarchique.   
La même opération dans le monde relationnel impliquerait plusieurs jointures. Les systèmes NoSQL orientés documents les plus connus sont CouchDB d’Apache, RavenDB (destiné aux plateformes .NET/Windows avec la possibilité d’interrogation via LINQ) et MongoDB.

### Modèle orienté « graphe »

Ce modèle (figure 4) qui repose sur la théorie des graphes, permet de représenter les données sous la forme de graphes. Le modèle s’appuie sur la notion de nœuds, de relations et de propriétés qui leur sont rattachées. Les entités sont alors les nœuds du graphe et les relations que partagent les entités sont alors des arcs que ces entités. Ce modèle est notamment adapté au traitement des données des réseaux sociaux.   
Notons que les systèmes NoSQL orientés graphe trouvent un certain intérêt pour des applications dans le domaine du Web Sémantique, dans la gestion de bases de données de triplets RDF (triple-stores), permettant de stocker des connaissances ou ontologies, un triplet étant une arête d’un graphe. Les systèmes NoSQL orientés graphe les plus connus sont Neo4J, Infinite Graph, OrientDB.

Ces différents type de base nous amène à devoir étudier plusieur type d’architecture.  
  
Les différents type d’architecture

a) L'architecture DataLake

Cette architecture est implémentée via Hadoop qui est un logiciel libre.

La version libre de HADOOP est arrivée en 2008 et supporte :

- HDFS

- MapReduce

- HBASE.

Hadoop permet de traiter beaucoup de donné en peu de temps exemple avec 1300 nœuds Yahoo a traité 500GB en 59s.

MapReduce n’est pas utile pour des tâche analytique et itérative.

Il utilise l’algorithme mapreduce afin de réduire son temps de calcul. C’est un algorithme développé par Google.

HBASE est un système de base de donnée non relationnel (c'est-à-dire une Base de données NoSQL)

HIVE est une infrastructure d’entrepôt de données. Il nous permet principalement de faire des requêtes SQL au sein de HBASE.

AVANTAGE :

L’avantage d’un datalake est surtout sa facilité de stockage de données ainsi que sa rapidité de traitement. En effet le fait de pouvoir stocker des données semi structurés voir pas du tout structuré et surtout sans type particulier permet de les stocker beaucoup plus facilement.

Un accès à plus de données. Effectivement l'avantage d’avoir la possibilité de stocker plus facilement les données nous permet d’en avoir et d’en gérer plus. Sur un but commercial.

Pour le projet : Plus de docs sur les outils…

INCONVÉNIENT :

Un désordre rapide possible étant donnés que les données n’ont pas de structures et donc l’exploitation de ces données peut s'avérer rapidement compliqué.

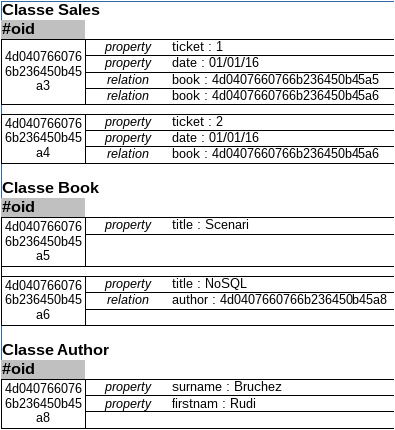
Ce qui fait leurs force (la rapidité) leurs fait aussi défaut effectivement l'absence de sécurité avec les règles peut poser des problèmes sur certaines bases. Par exemple, le fait de ne pas avoir de règles de transaction pour une base de paiement est impensable.

Un stockage de plus de données ce qui pose un soucis d’ordre physique peut amener une entreprise à délocaliser ses serveurs et donc s’exposer à un plus grande latence.

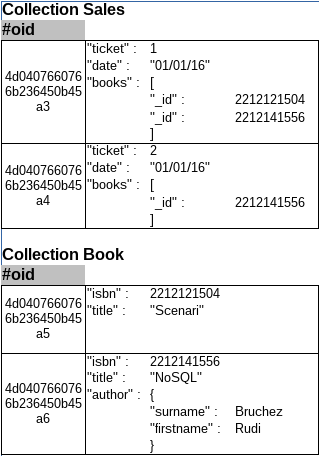
Exemple de NoSQL:

JSON/XML donc soit en format graphe/colonne/document

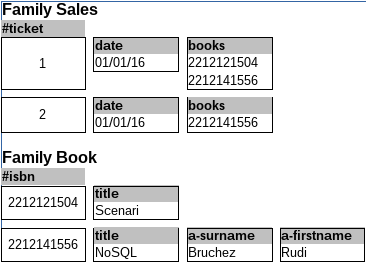
Graphe :



Document:



Colonne:



# b) L'architecture Lambda

Le but de l'architecture Lambda est de fournir un modèle de traitement presque temps réel sur des volumes importants de données, en proposant un nouveau modèle de calcul. Ce modèle essaie de trouver l'équilibre entre la tolérance aux pannes, les contraintes de latence (latence très faible pour les lectures/écritures) et le débit des disques durs en se basant à la fois sur les traitements batch qui fournissent des vues batch et les traitements temps réel qui fournissent des vues, puis les joint avant leur présentation.

Hadoop n'est pas capable de traiter un grand volume de données qui doit satisfaire une faible latence, même en ajoutant d'autres serveurs de calcul, d'où la naissance de cette architecture qui ne remet pas en question le paradigme MapReduce, mais propose une amélioration, afin de contourner les contraintes de latence de Hadoop.

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** :

Mode de fonctionnement classique des applications big data type Hadoop, cette couche est responsable de deux choses : récupérer les données et les stocker en format brut (AS-IS : pour pouvoir répondre à de nouveaux besoins métier sans impacter les données initiales) dans des puits de données (Data Lake en anglais), et lancer périodiquement des traitements sur les données, pour précalculer les résultats sous forme de vue logique. Le résultat est ensuite stocké typiquement dans des bases en lecture seule et les mises à jour remplacent les vues logiques précalculées.

Cette couche peut être implémentée à l'aide d’Apache Hadoop, MapReduce, Spark

**Couche de vitesse ou temps réel (Speeding) :**

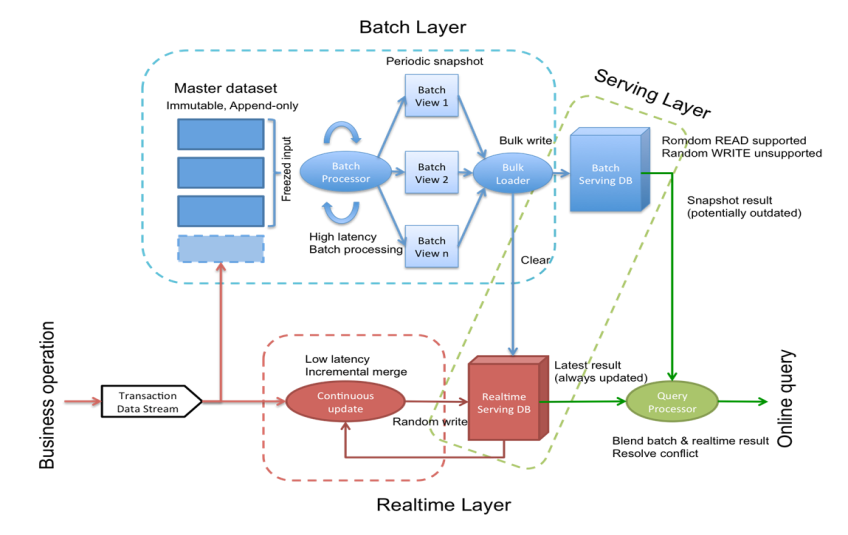
Traiter les nouveaux flux de données en temps réel, sans aucun prétraitement (correction des jeux des données). Cette couche minimise la latence et fournit en temps réel des vues avec les données les plus récentes. C'est un fonctionnement en mode continu unitairement pour chaque nouvelle donnée. Les résultats (les vues temps réel) fournis par cette couche ne sont pas aussi fiables, que ceux de la couche batch.

Cette couche peut être implémentée à l'aide d’Apache Storm ou Spark Streaming.

**Couche de service (Serving) :**

Rend exploitables les résultats précalculés par la couche batch et la couche temps réel, pour effectuer des requêtes à la volée (ad hoc).

Cette couche peut être implémentée à l'aide des technologies NoSQL Apache HBase, Cassandra, et ElasticSearch qui permettent de merger les vues batch et les vues temps réel.



*Figure 1: Architecture Lambda*

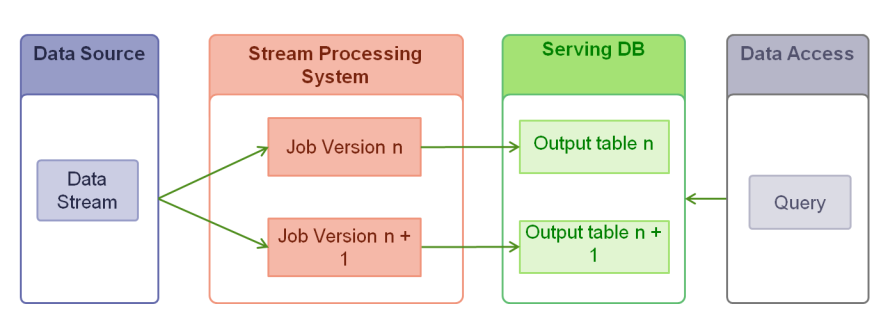
Suite à ces recherche nous avons donc choisie de nous spécialiser dans l’utilisation de l’architecture DataLake. Les raisons de ce choix sont principalement le fait que Hadoop soit un environnement libre et avec une communauté grandissante et relativement importante.

c) Architecture Kappa :

Partant d'un constat que la plupart des solutions sont capables de faire à la fois des traitements temps réel (streaming) et traitements batch, l'architecture Kappa permet de simplifier l'architecture Lambda, en fusionnant la couche batch et la couche Speeding.

Elle apporte aussi une modification sur les SGBD qui doivent être un système de fichiers de log immuable.

L'architecture Kappa n'est pas destinée au stockage des données, mais uniquement à leur traitement, comme le montre le schéma suivant :



Pour plus d'informations, voir : <http://milinda.pathirage.org/kappa-architecture.com/>

Kappa n’étant également pas liée à une seule technologie, on peut y associer différents outils, comme le montre le schéma ci-dessous :

* Stockage/temps réel : Kafka permet la sauvegarde des messages pour pouvoir ensuite les retraiter
* Traitements : Storm, Spark
* Couche de service : Cassandra, Hive, HBase, Outil maison, etc…

II/ Les logiciels du big data

1. Les logiciels liés au big data

Dans cette partie, nous allons tout d’abord présenter différents logiciels de big data présent sur le marché, ainsi que ceux que nous avons choisi pour ce projet.  
  
DynamoDB

Il s’agit du logiciel de gestion de données créer par Amazon. Il se base donc sur le noSQL en particulier sur les structure document et par clefs. Les langages permettant de l’utiliser sont:

Java, Node.js, Go, C# .NET, Perl, PHP, Python, Ruby, Haskell et Erlang.

MongoDB

Logiciel de gestion de données structuré en document. Il est coder en C++ et à la particularité de pouvoir être répartie sur plusieurs ordinateurs différents.

Il est sous licence apache.Les langages d’utilisations sont:

C,C++,Dart,Erlang,Go,Haskell,Java,JavaScript,.NET (C# F#, PowerShell, etc),Perl,PHP,Python,Ruby,Scala

Hadoop

Apache hadoop est un outil permettant de paralléliser les traitements en les stockant sur plusieurs unités de durs. D'une part, il propose un système de stockage distribué via son système de fichier HDFS (Hadoop Distributed File System) et ce dernier offre la possibilité de stocker la donnée en la dupliquant, une grappe de serveurs Hadoop n'a donc pas besoin d'être configuré avec un système RAID qui devient inutile.

Le RAID est un ensemble de techniques de virtualisation du stockage permettant de répartir des données sur plusieurs disques durs afin d'améliorer soit les performances, soit la sécurité ou la tolérance aux pannes de l'ensemble du ou des systèmes.

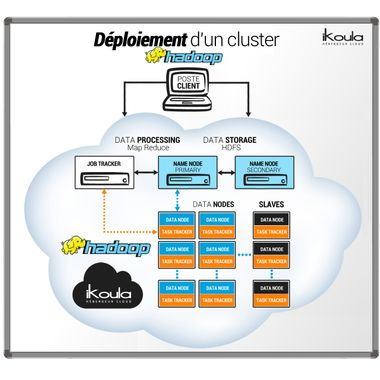
|  |  |
| --- | --- |
|  | *Le RAID est un ensemble de techniques de virtualisation du stockage permettant de répartir des données sur plusieurs disques durs afin d'améliorer soit les performances, soit la sécurité ou la tolérance aux pannes de l'ensemble du ou des systèmes.*  *Cf. wikipedia* |

D'autre part, Hadoop fournit un système d'analyse des données appelé MapReduce. Ce dernier officie sur le système de fichiers HDFS pour réaliser des traitements sur des gros volumes de données.

Hadoop est beaucoup plus utile dans un environnement multi-machine mais il n'a de sens dans un environnement mono-machine que pour tester la configuration de l'installation ou fournir un environnement de développement MapReduce (prochain article). Comme nous allons le faire avec l’api **SPARKS** dans un prochain tutoriel.

Avant de rentrer dans la vive du sujet, certaines choses ont besoin d’être défini :

* Un namenode : Un namenode est un service qui s’occupera de gérer l’état du système de fichiers. Il stocke l'emplacement de tous les fichiers dans le système de fichiers local.
* Un datanode : Les datanode sont des workers. C’est-à-dire qu’il effectue les opérations demandées par le namenode. Ils vont se charger de transmettre au client les blocs correspondant au fichier à transmettre et en écriture vont retourner l’emplacement des blocs créés.
* HDFS : HDFS (Hadoop Distributed File System) est un système de fichier distribué permettant de stocker et de récupérer des fichiers rapidement. Il s’agit de l’un des composants basiques du framework Hadoop Apache, et plus précisément de son système de stockage.

Schéma de fonctionnement de HADOOP:  
  
NameNode: C’est ce qui pendant longtemps à étais un des seuls point de cassure du système. Il n’en ai plus un depuis l’apparition du backup Node. C’est lui qui gere l’ensemble de l’arbre de données. Si il ne fonctionne pas l’ensemble du système est HS. Il permet donc la communication entre l’utilisateur et la base.  
  
DataNode: C’est ce qui permet la connection entre les données et le Namenode.  
  
JobTracker: Il est essentiel pour l’utilisation de MapReduce. Il communique directement avec le NameNode pour l'accès aux données et ne passe pas par les dataNode.  
  
  
SPARK

Apache Spark est un framework de traitements Big Data open source pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d’utilisation.

Spark présente plusieurs avantages par rapport aux autres technologies big data et MapReduce comme Hadoop et Storm. D’abord, Spark propose un framework complet et unifié pour répondre aux besoins de traitements Big Data pour divers jeux de données, divers par leur nature (texte, graphe, etc.). Spark permet également à des applications sur une grappe de serveurs d’Hadoop d’être exécutées beaucoup plus rapidement.

Spark est à défaut de différents outils utilisé par Hadoop tel que MapReduce beaucoup plus facile à mettre en place tout en permettant d’effectuer des traitement de données complexes.

Apache Spark contient une bibliothèque « core ». Sur ce framework Core de Spark s'appuient un certain nombre de bibliothèques plus haut niveau, en l'occurrence :

* Spark SQL, un module de manipulation des données sous une forme structurée (comme peut l'être SQL dans le monde des SGBDR)
* Spark Streaming, répondant aux problématiques de flux de données « temps réel »
* MLib, une bibliothèque de Machine Learning
* GraphX pour la manipulation de données orientées graphe.

MapReduce

MapReduce est un **modèle de programmation créé par Google pour le traitement et la génération de larges ensembles de données sur des clusters d’ordinateurs**. Il s’agit d’un composant central du Framework logiciel Apache Hadoop, qui permet le traitement résilient et distribué d’ensembles de données non structurées massifs sur des clusters d’ordinateurs, au sein desquels chaque nœud possède son propre espace de stockage. Concrètement, le framework propose deux fonctionnalités principales. Il répartit le labeur sur les différents nœuds du cluster (map), puis les organise et réduit les résultats fournis par chaque nœud en une seule réponse cohérente à une requête. Cela est rendu possible grâce à son système de fichiers distribués HDFS.  
  
**La différence entre spark et mapReduce.**

La différence notable entre spark et mapReduce est leurs façon de fonctionner en effet mapReduce va travailler sur le disque ce qui va rendre l'opération plus longue. Alors que Spark va travailler en mémoir et donc tout iras plus vite.

MapReduce sera donc plus utile si le budget de l’entreprise n’est pas “illimité” car il serait plus performant que spark si les données à traiter sont plus grande que la mémoire disponible.   
  
Spark sera quand à lui meilleurs sur les point suivants:  
  
Vitesse de gestion des données (si mémoir suffisante)  
Pour les application itérative sur les données.  
Lors d’un besoin immédiat d’information et de traitement de la donnée autrement dit en calcul en temps réel.  
Les calcule de graf ce qui est directement lié aux calculs itératif.   
Sur le machine learning  
Sur la jointure de données.

Pour résumer MapReduce est une solution plus économique que Spark mais moins efficace.

**Hive**:

Hive à étais développé par Facebook et est aujourd’hui utilisé par exemple par le service de Streaming Netflix.  
Hive permet de faire des requêtes de type SQL directement sur la base HDFS. Ce langage est le HiveQL.   
Les requêtes sont donc convertie en map reduce afin de pouvoir fonctionner sur le HDFS.  
  
La différence entre le SQL HiveQL est que toutes les fonctions ne sont pas assurées. Par exemple les transactions ainsi que les vues matérialisées ne sont pas disponibles.

Les différence avec Spark:

-Spark est disponible en plusieurs langage. Là où Hive n’est disponible que en Java.  
-Spark ne duplique pas lmes données là où Hive le fait.  
-Hive permet d’avoir des droits d’accés par utilisateur se que Spark ne permet pas.  
  
  
Les faiblaisses de Hive:

-N’offre pas de traitement en temps réel.  
-Peut prendre du temps  
-Pas de support pour les transactions.

b) Installation:

Nous détaillons l’installation de Hive car c’est un outil souvent utiliser donc nous l’avons installer mais pas utiliser.

HIVE:  
Telecharger Hive depuis le site https://hive.apache.org/downloads

Une fois le fichier télécharger, l'extraire avec le commande tar

tar zxvf apache-hive-version-bin.tar.gz

Une fois l'extraction réussi, copier le fichier dans le repertoire /usr/local/hive.

Cette commande est à utiliser en étant en Administrateur (su)

mv apache-hive-version-bin /usr/local/hive

Maintenant, il faut préparer l'environnement en lui indiquant les différentes variables d'environnement,

à rajouter dans le fichier bashrc

export HIVE\_HOME=/usr/local/hive

export PATH=$PATH:$HIVE\_HOME/bin

export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/\*:.

export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/\*:.

Il ne reste plus qu'à lancer le fichier bashrc

source ~/.bashrc

IV/Configurer Hive

Pour configurer hive avec hadoop, il faut editer le fichier hive-env.sh, situé dans le dossier conf

Il faut rajouter dans ledit fichier la ligne suivante :

export HADOOP\_HOME=/usr/local/hadoop

On a maintenant besoin d'une base de donnée externe pour configurer Metastore.

--> Dl et installer le logiciel (tjrs dans /usr/local/nomdulogiciel)

V/Configurer Hive MetaStore

Configurer Hive metastore revient à indiquer où est stocké la base de donnée.

Il faut éditer le fichier /usr/local/hive/conf/hive-site.xml

Dans un premier temps, il faut copier le fichier de template grâce à la commande suivante :

cd $HIVE\_HOME/conf

cp hive-default.xml.template hive-site.xml

Il faut maintenant insèrer les lignes suivantes dans le fichier hive-site.xml:

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:derby://localhost:1527/metastore\_db;create=true </value>

<description>JDBC connect string for a JDBC metastore </description>

</property>

Enfin, on crée un fichier nommé jpox.properties, qui va définir les propriétés de notre métastore :

Ce fichier va contenir les lignes suivantes :

javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl

org.jpox.autoCreateSchema = false

org.jpox.validateTables = false

org.jpox.validateColumns = false

org.jpox.validateConstraints = false

org.jpox.storeManagerType = rdbms

org.jpox.autoCreateSchema = true

org.jpox.autoStartMechanismMode = checked

org.jpox.transactionIsolation = read\_committed

javax.jdo.option.DetachAllOnCommit = true

javax.jdo.option.NontransactionalRead = true

javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver

javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore\_db;create = true

javax.jdo.option.ConnectionUserName = APP

javax.jdo.option.ConnectionPassword = mine

Pour vérifier que l'installation de Hive s'est bien déroulé, il suffit d'executer la commande suivante :

cd $HIVE\_HOME

bin/hive

Un message apparait indiquant que hive est bien lancé, on peux désormais utiliser des commandes spécifiques à Hive.

Par exemple, la commande "show tables;" permet d'afficher toutes les tables

Pour l’installation de hadoop et de spark :

Hadoop :

Installation windows

L’installation d’HADOOP s’effectuera sur le système d’exploitation windows parce que celui-ci permet une installation simple et sans trop de configuration par rapport à linux. Pour pouvoir installer HADOOP, nous allons avoir besoin installer les outils suivants :

* Le JDK (de préférence le 1.8)
* Apache hadoop (dans notre cas la version 2.7.1 pesant 200 MO)
* hadooponwindows-master

Tout cela est disponible via le lien suivant :

https://github.com/sardetushar/hadooponwindows

Une fois les différents outils installés puis extrait, nous allons dans un premier temps déclarer nos variables d’environnements :

* JAVA\_HOME (pointant sur le dossier bin du JDK)
* HADOOP\_HOME (pointant vers le bin du dossier hadoop2.7.1 extrait)
* Puis dans le PATH déclarer le chemin vers c’est 2 composants.

Après cela, nous allons devoir remplacer les fichiers contenus dans le dossier /etc et /bin du dossier d’hadoop 2.7.1 par ceux contenu dans le dossier hadooponwindows-master.

Une fois cela réalisé, nous allons procéder à la configuration de 4 fichiers :

* *core-site.xml*
* *hdfs-site.xml*
* *mapred-site.xml*
* *hadoop-env.cmd*

Ainsi, nous allons inscrire les lignes suivants dans les fichiers correspondants :

**core-site.xml :**

<!-- C'est à cette adresse que nous pourrons accéder à notre BDD depuis l’API spark avec JAVA -->

<configuration>

*<property>*

*<name>fs.defaultFS</name>*

*<value>hdfs://localhost:9000</value>*

*</property>*

*</configuration>*

**hdfs-site.xml :**

<!-- Création du namenode et datanode contenu dans un dossier data que nous créerons au préalable -->

*<configuration>*

*<property>*

*<name>dfs.replication</name>*

*<value>1</value>*

*</property>*

*<property>*

*<name>dfs.namenode.name.dir</name>*

*<value>D:\hadoop\hadoop-2.7.1\data\namenode</value>*

*</property>*

*<property>*

*<name>dfs.datanode.data.dir</name>*

*<value>D:\hadoop\hadoop-2.7.1\data\datanode</value>*

*</property>*

*</configuration>*

**mapred-site.xml :**

*<configuration>*

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*

*</configuration>*

**hadoop-env.cmd:**

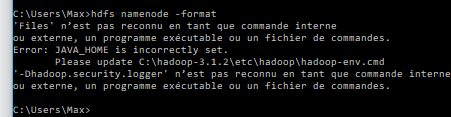
<!-- On déclare notre JAVA\_HOME -->

*set JAVA\_HOME=CheminVersLeDossierJDKSansLeBin*

Une fois que la configuration des différents fichiers est effectuée, nous allons, depuis le terminal de commande, lancer la commande suivante :

***hdfs namenode –format***

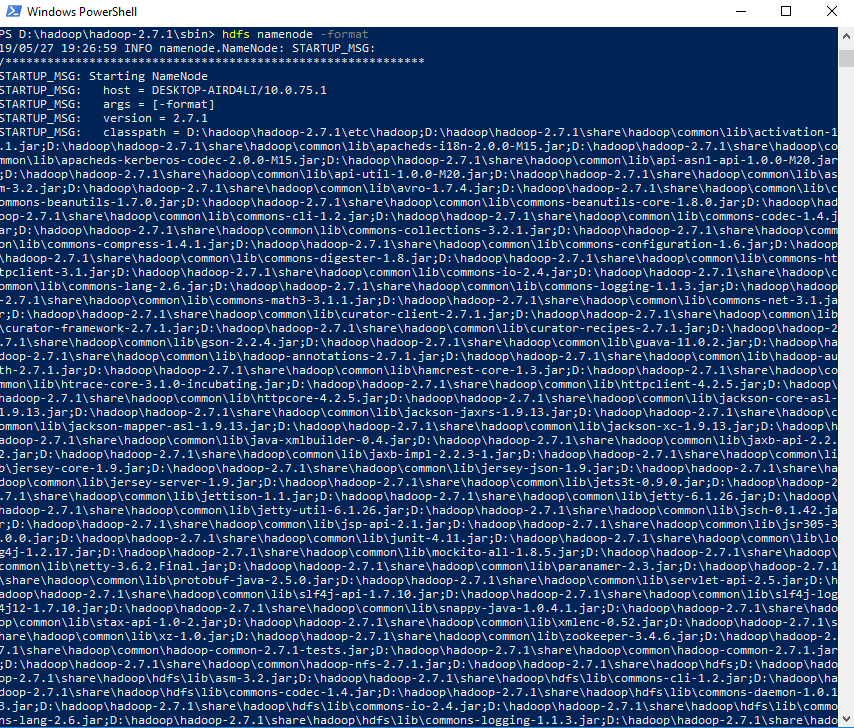
Si l’erreur suivante se présente, vérifié que le **JAVA\_HOME et HADOOP\_HOME** pointe bien vers leur bin respective et que l’on ne pointe pas vers le dossier bin mais le dossier racine du jdk dans hadoop-env.cmd.



**Annexe 1: Erreur possible suite à l'exécution de la commande *hdfs namenode –format***

|  |  |
| --- | --- |
|  | *A noter que sur windows, lorsque l’on veut spécifier dans le chemin d’accès le dossier “Program Files”, nous écrirons* ***PROGRA~1*** *à la place.* |

Si tout va bien, une suite d’instructions est censée se lancer.

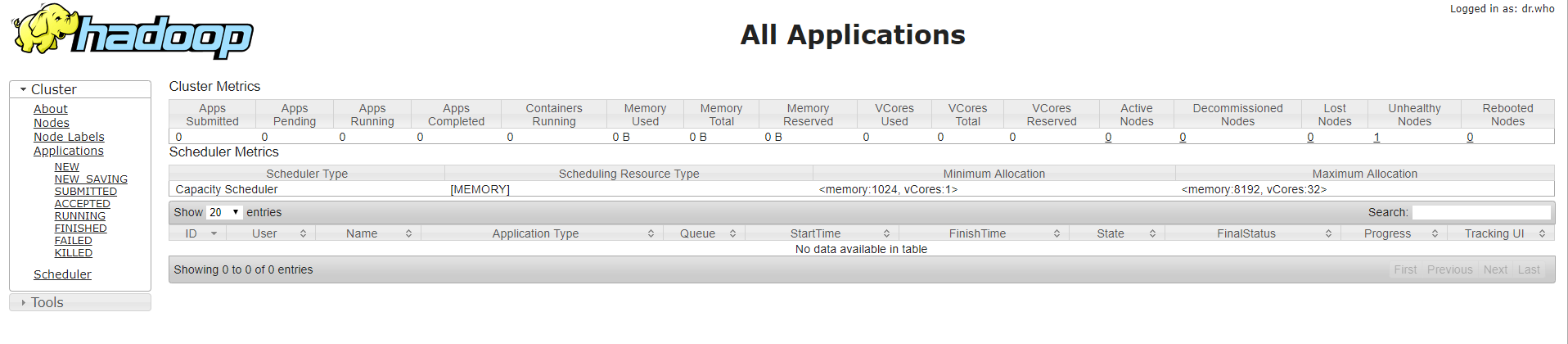


**Annexe 2: Résultat obtenu suite à l'exécution de la commande hdfs namenode –format**

Il nous restera plus qu’à nous rendre dans le dossier **sbin** de hadoop puis de lancer l’exécutable **start-all.cmd**.

4 terminaux sont censés s’ouvrir confirmant le bon fonctionnement de l’outil. Ainsi nous pourront accéder à diverse composant via l’URL de notre navigateur. Si la configuration/port et semblable à celle effectué dans ce document :

* Localhost:8088 :



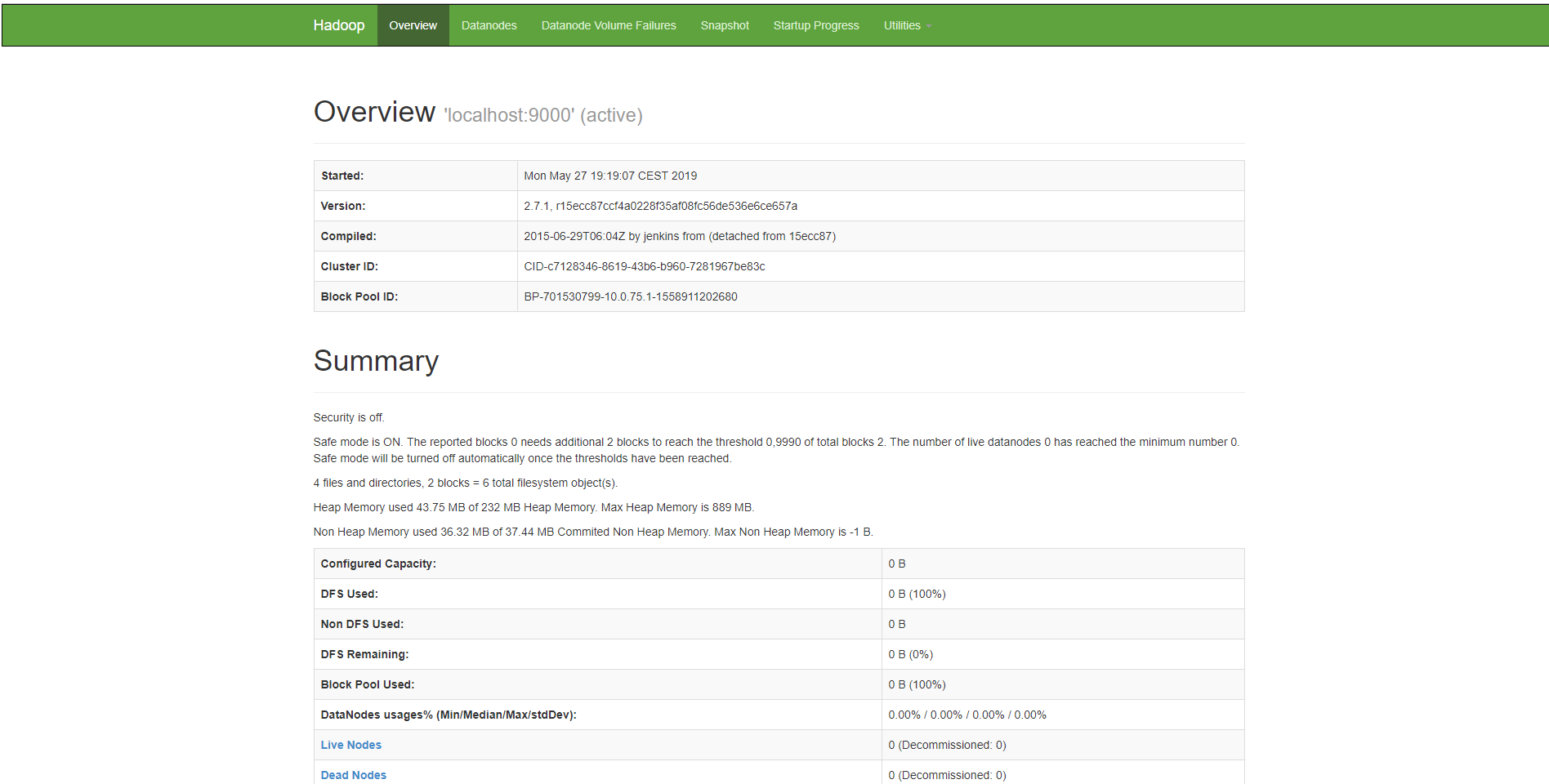
**Annexe 3: Erreur possible suite à l'exécution de la commande *hdfs namenode –format***

* Localhost:9000 : le port permettant d’accéder à notre BDD, soit l’HDFS

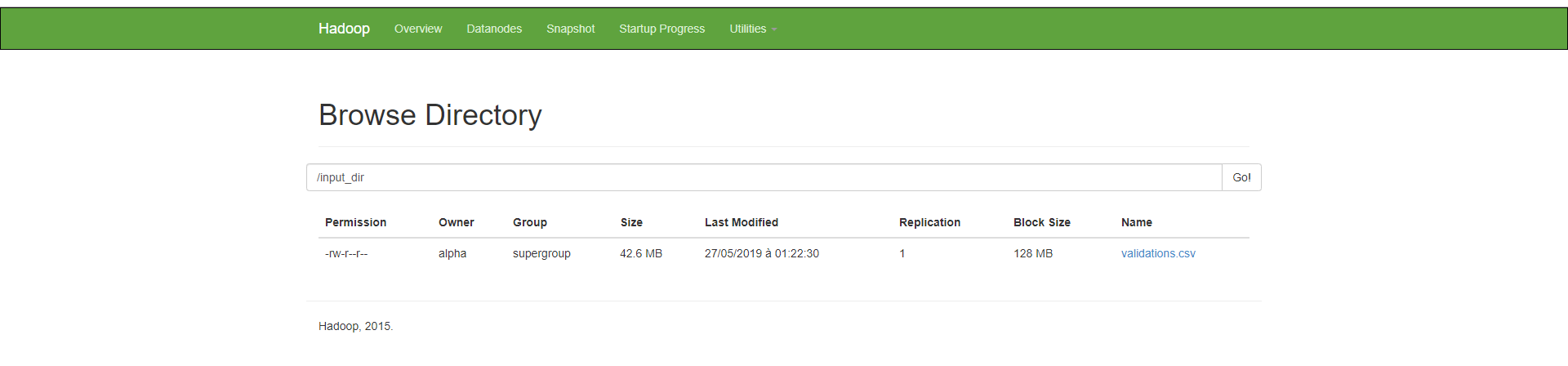


**Annexe 4: Résultat de l’URL**

* Localhost:50070 : l’environnement on l’on stocke notre BDD.



**Annexe 5: Environnement de stockage de la BDD**



**Annexe 6: Fichier .csv contenu dans hadoop**

Voici différentes commande utile quant à la bonne utilisation de hadoop:

* hadoop fs -mkdir /dossierDestinataire : création de dossier.
* hadoop fs -put cheminVersMonFichier /dossierDestinataire : Ajout d’un fichier (txt, csv etc..) dans notre base.

# Spark

## Installation windows

Télécharger Spark (le binary)

<https://www.apache.org/dyn/closer.lua/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz>

Dézipper le de tel sorte à avoir C:\spark-2.4.3

Ajouter le dossier en variable d’environnement :

Variable : SPARK\_HOME

Value : C:\spark-2.4.3

Puis ajouter le jdk dans le path :

C:\spark-2.4.3\bin

Maintenant on teste :

Sur le cm den mode admin, faire spark-shell

Il existe aussi un autre moyen d’installer spark Hadoop et Hbase d’un coup d’une manière bien plus simple. Pour celà nous allons utiliser docker en effet il existe un container ayant Hadoop,HBase et spark.  
  
L’installation du container

Les instructions sont faites pour un OS linux mais elles peuvent s’adapter à Windows étant donné que docker est disponible sur celui-ci.

# Prérequis :

* Avoir télécharger et dézipper le fichier
* Avoir installer Docker
* Avoir démarrer Docker

# Compiler L’image Docker:

cd hadoop-cluster-docker

./build-image.sh

# Démarrer L’image:

./start-container.sh

# Lancer hadoop :

./start-hadoop.sh

# Lancer HBase:

./start-hbase.sh

# Lancer Spark :

spark-shell

# Tutoriel Hadoop :

Après avoir démarrer Hadoop, on va créer un répertoire input :

**hadoop fs –mkdir -p input**

On va charger le fichier purshase.txt :

**hadoop fs –put purchases.txt input**

On affiche le contenu du répertoire input pour vérifier :

**hadoop fs –ls input**

On va afficher les dernières lignes du fichier avec :

**hadoop fs -tail input/purchases.txt**

Notons que sur <http://localhost:50070> affiche les informations de notre namenode et que <http://localhost:8088> permet d'afficher les informations du ressource manager de Yarn et visualiser le comportement des différents jobs

Ouvrir le fichier pom.xml et ajouter les dépendances :

**<dependencies>**

**<dependency>**

**<groupId>org.apache.hadoop</groupId>**

**<artifactId>hadoop-common</artifactId>**

**<version>2.7.2</version>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core -->**

**<dependency>**

**<groupId>org.apache.hadoop</groupId>**

**<artifactId>hadoop-mapreduce-client-core</artifactId>**

**<version>2.7.2</version>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->**

**<dependency>**

**<groupId>org.apache.hadoop</groupId>**

**<artifactId>hadoop-hdfs</artifactId>**

**<version>2.7.2</version>**

**</dependency>**

**<dependency>**

**<groupId>org.apache.hadoop</groupId>**

**<artifactId>hadoop-mapreduce-client-common</artifactId>**

**<version>2.7.2</version>**

**</dependency>**

**</dependencies>**

Tutoriel Spark:

Une fois avoir lancer les containers et Hadoop,

On crée un fichier file.txt contenant un texte :

**touch file.txt**

**echo hello > file.txt**

On va ensuite charger le fichier dans le HDFS :

**hadoop fs -put file.txt**

On lance le shell spark:

**spark-shell**

Ensuite, pour s'entraîner à spark, on va :

Séparer les mots selon les caractères d'espacement

**val lines = sc.textFile("file1.txt")**

Appliquer un *map* sur les mots obtenus qui produit le couple (*<mot>*, 1)

**val words = lines.flatMap(\_.split("\\s+"))**

Puis un *reduce* qui permet de faire la somme des 1 des mots identiques

**val wc = words.map(w => (w, 1)).reduceByKey(\_ + \_)**

Puis, on sauvegarde le résultat dans un fichier file1.count

**wc.saveAsTextFile("file.count")**

On vas télécharger le fichier file.count dans le HDFS:

**hadoop fs -get file1.count**

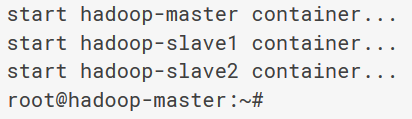
Puis on va regarder le contenu du fichier « part-00000 » crée par le chargement :

**cat part-00000**

# 

# Tutoriel HBase shell:

Une fois avoir Démarrer l’image avec ./start-container.sh :



On lance Hadoop avec la commande **./start-hadoop.sh**

On lance ensuite HBase avec la commande **start-hbase.sh**

On démarre le shell de HBase en tapant la commande **hbase shell**



On vas créer une table en tapant la commande :

**create ‘table1 ’,’colonne1’,’colonne2’**

En tapant la commande **list** , on vas lister les tables créent.

Pour insérer une ligne dans la table, on tape la commande :

**put ‘table1 ’,’1’,’colonne1:souscolonne1’,’valeur1’**

**put ‘table1 ’,’1’,’colonne1:souscolonne2’,’valeur2’**

**put ‘table1 ’,’1’,’colonne2:souscolonne3’,’valeur3’**

**put ‘table1 ’,’1’,’colonne2:souscolonne4’,’valeur4’**

Cette série de ligne à permis de créer un objet table1 ayant une colonne 1 et une colonne 2 comprenant respectivement une souscolonne1 et une souscolonne2 pour la première colonne et une souscolonne3 et une souscolonne4 pour la deuxième.  
  
table1 ⇒ nom de la table  
1 ⇒ Id de la colonne  
colonne1:souscolonne1 ⇒ nous permet de définir le degret du tableau (1 sous colonne 3D, 2 sous colonne 4D, etc…)

valeur1 ⇒ valeur mis dans la colonne

Pour afficher la table « table1 », on peut exécuter la commande :

**scan ‘table1 ’**

Pour afficher une ligne en particulier, on peut faire la commande :

**get ‘table1’,'1',{COLUMN => 'colonne1:souscolonne1'}**

On récupèrera donc l’information « valeur1 ».

Pour pouvoir insérer un ensemble de données, par exemple le contenu d’un fichier text.txt, on l’ajoute au HDFS avec la commande :

**hadoop fs -put text.txt input**

On crée la table correspondante :

**hbase shell**

**create 'Table\_name','famille\_colonne'**

**exit**

Puis on importe nos données en appelant une entité selon leur format (ici TSV):

**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \**

**-Dimporttsv.separator=',' \**

**-Dimporttsv.columns=HBASE\_ROW\_KEY,famille\_colonne:colonne1,famille\_colonne:colonne2 \**

**Table\_name input**

**III/ La gestion de projet**

(mettre en évidence les CR hebdo)

Afin de nous organiser à 4 pour effectuer les différentes tâches du projet nous avons mis en place diverses choses.   
  
Un “chef de la communication” a été nommé. Afin de centraliser les informations de chacun pour les communiquer au mieux à notre tuteur de projet.

Gestion des ressources et communications :  
  
La première et la plus utile est la mise en place d’un google drive afin d’avoir un système de fichier partager.   
En parallèle de ça nous utilisions un serveur de discussion via tchat mais également vocal lors ce que nous avions des questions sur les partie des autres.  
  
 Toutes les semaines nous avions également une réunion avec notre superviseur de projet a qui nous faisions un point sur l’avancement. Ce point permettais également d’adapter les tâches futur par rapport à nos difficultées.  
  
Gestion du temps :

Tout d’abord un diagramme de GANTT à était fait afin d’avoir une vue global des différentes tâches du projet à faire et quand est-ce elles doivent être faites.  
  
Suite à ça nous l’avons adapté de semaines en semaine afin d’être le plus en adéquation avec notre avancement réel.  
  
Résumé par semaine:

Semaine 1:

La première semaine du projet était principalement consacré au renseignement. En effet, le Bigdata étant un sujet vaste et très récent, les documentations ne sont pas donc pas disponible facilement.   
  
A la fin de celle-ci nous avions donc de bonne informations sur ce qu'était le big DATA et ce qu’il implique.   
  
Semaine 2:

Etude des différentes architecture et choix des outils.   
Sur cette semaine nous avons chacun fait des recherche sur un type d’architecture. Elle nous a permis donc de découvrir 4 types d’archi différentes. C’est donc à la fin de cette semaine que notre choix c’est porté sur l’architecture dataLake. Notre choix c’est porté sur l’environnement HADOOP en utilisant Hbase et Hive.

Semaine 3:  
  
L’installation des outils. Durant cette semaine nous voulions installer tous les outils sur une distribution Linux. Nous avons durant cette semaine eux quelques souci d’installations. en effet l’installation de Hadoop sous linux nécessite beaucoup de dépendance.   
Par contre nous avons changer de choix de logiciels. Nous avons choisie à ce moment là de partir sur HADOOP + SPARK.  
  
Semaine 4:

Suite à la semaine précédente nous avons choisie d’installer les outils sous windows ( moins de possibilité de configuration mais plus de facilité d’installation. On a donc fait l’installation sur windows et commencer à essayer d'utiliser les outils. Le squelette du PP à étais commencé.  
  
Semaine 5:

Nous avons trouvé un conteneur docker qui contient hadoop/Hbase et SPARK. Nous avons donc pus l’installer sur linux et windows. On a réussi à importer des données sur hadoop et à les extrader avec Spark et mapreduce. Nous avons également commencé le Compte rendu.  
  
Semaine 6:  
  
Nous avons fait fonctionner l’ensemble des outils présent sur le docker et continuer le CR.  
  
Semaine 7:

Installation de Hive ainsi que finalisation des slides.  
  
Semaine 8:

Fin du Rapport. Dernière version des slides faites. Création de la vidéo de présentation.