

Big data Engineering

Yasser EL MADANI EL ALAMI

Année universitaire : 2025-2026

Plan

- I. Introduction au Bigdata
- II. Infrastructure et architecture Big data
- III. Analyse de flux de données en temps réel
- IV. Bases de données NoSQL
- V. Big Data Analytics

1

Readings

- Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale
4th Edition by Tom White
- Kafka The Definitive Guide Real-Time Data and Stream Processing at Scale
- NoSQL for Mere Mortals-Pearson. Dan Sullivan (2015)
- Spark: The Definitive Guide: Big Data Processing Made Simple 1st
Édition by Bill Chambers & Matei Zaharia

3

Organisation

- ❖ Composants du module
 - } Cours magistraux
 - } TD/TP
 - } Projet
- ❖ Volume Horaire
 - 2 séances / semaine
 - 7 semaines

2

4

Data never sleeps

Croissance **ex** de la quantité de données générée



<https://www.domo.com/learn/infographic/data-never-sleeps-9>

5

Data Volume Units

Unit	Value	Example
Kilobytes (KB)	1,000 bytes	a paragraph of a text document
Megabytes (MB)	1,000 Kilobytes	a small novel
Gigabytes (GB)	1,000 Megabytes	Beethoven's 5th Symphony
Terabytes (TB)	1,000 Gigabytes	all the X-rays in a large hospital
Petabytes (PB)	1,000 Terabytes	half the contents of all US academic research libraries
Exabytes (EB)	1,000 Petabytes	about one fifth of the words people have ever spoken
Zettabytes (ZB)	1,000 Exabytes	as much information as there are grains of sand on all the world's beaches
Yottabytes (YB)	1,000 Zettabytes	as much information as there are atoms in 7,000 human bodies

<https://mynasadata.larc.nasa.gov/basic-page/data-volume-units>

6

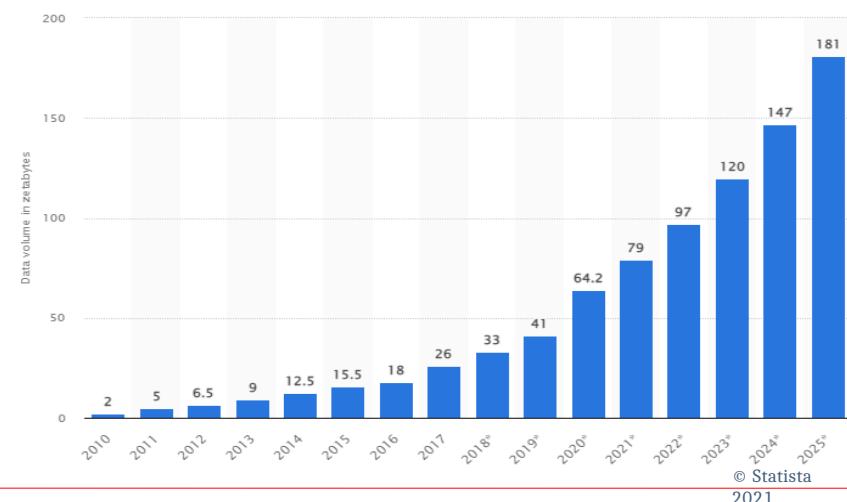
Préfixes multiplicatifs

Signe	Préfixe	Facteur	Exemple représentatif
k	kilo	10^3	une page de texte
M	méga	10^6	vitesse de transfert par seconde
G	giga	10^9	DVD, clé USB
T	téra	10^{12}	disque dur
P	péta	10^{15}	Ebay(2014)
E	exa	10^{18}	Google (2014)
Z	zetta	10^{21}	internet tout entier depuis 2010

<https://physics.nist.gov/cuu/Units/binary.html>

7

Volume de données : quelques statistiques



© Statista

2021

8

Volume de données : quelques statistiques

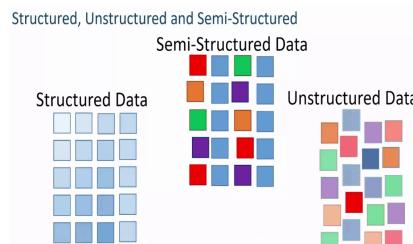
Compagnies	Données traitées (2014)	Données stockées (2014)
Google	100 Po	15 000 Po
Ebay	100 To	90 Po
Facebook	600 To	300 Po
Twitter	100 To	100 To
Baidu	10-100 Po	2 000 Po
NSA	29 Po	10 000 Po

9

Types de données

- ❖ Structurées (RDBMS)
- ❖ Non structurées (image, son , video, fichiers logs, etc.)
- ❖ Semi structurées (XML, Json, etc)

```
[ { "ID": "1",
  "Name": "Soufiane",
  "tel": "11111111"
}, { "ID": "2",
  "Name": "Youssra",
  "tel": ["2222","33333"]}]
```



Source : The Data Wiki

11

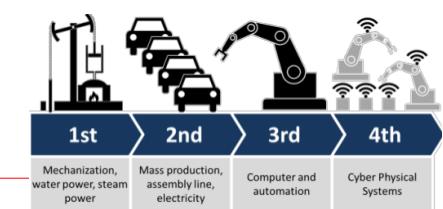
Sources big data

- ❖ **Capteurs mobiles** : GPS, accéléromètre, etc.
- ❖ **Réseaux sociaux** : 700 mises à jour Facebook/sec en 2012
- ❖ **Vidéosurveillance** : caméras de rue, magasins, etc.
- ❖ **Réseaux intelligents** : recueillir et agir sur l'information d'exploration géophysique – pétrole, gaz, etc.
- ❖ **Imagerie médicale** : révèle les structures internes du corps
- ❖ **Séquençage des gènes** : comprendre l'origine génomiques des maladies (prédition de maladies)

10

Big data: domaines

- ❖ **Healthcare**: remote monitoring
 - Wireless monitoring devices fournissent de grande quantité de données.
- ❖ **Financial**: detection de fraude (7 cents per 100 dollars)
 - Ex: visa 73milliards de transactions enregistrées dans 2 ans (36 terabytes)
- ❖ **Retailers**: predict trends, prepare for demand, optimize pricing & promotions, and monitor real-time analytics & results
- ❖ **Industry**: complexity of production and supplier networks
 - ❖ ...and many others



<http://hortonworks.com/big-data-insights/industries-benefit-big-data>

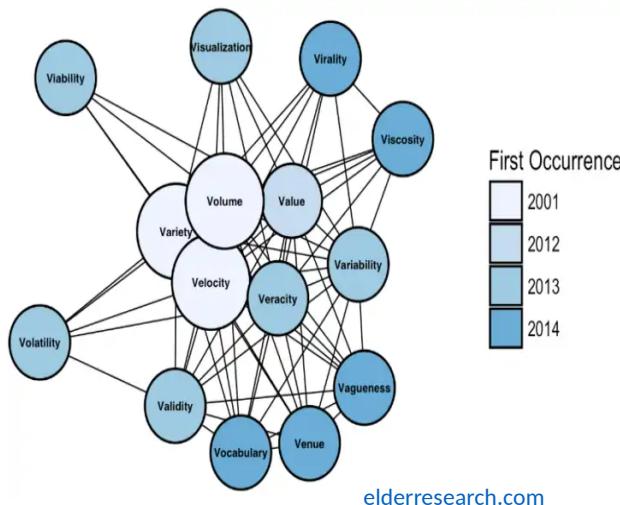
Définition

- ❖ Données massives ou mégadonnées
- ❖ **Def1:** Big data is a collection of large amounts of complex data that cannot be managed efficiently by the state-of-the-art data processing technologies (Philip Chen & Zhang, 2014)
- ❖ **Def2:** Techniques and tools for managing large amounts of data (collection, storage, processing, and restitution)

13

Big data: caractéristiques

- ❖ Gartner 2001: 3V
- ❖ IBM 2012: 4V
- ❖ 2015: 5V
- ❖ 2016: 7V
- ❖ 2017: 10V



15

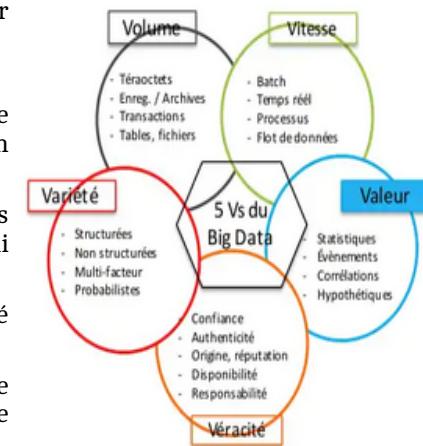
Définition

- ❖ Données massives ou mégadonnées
- ❖ **Def1:** Big data is a collection of large amounts of complex data that cannot be managed efficiently by the state-of-the-art data processing technologies (Philip Chen & Zhang, 2014)
- ❖ **Def2:** Techniques and tools for managing large amounts of data (collection, storage, processing, and restitution)

14

Caractéristiques du big data

- ❖ **Volume:** grande quantité de données à collecter et analyser
 - en augmentation constante
- ❖ **Vitesse (Velocity):** les données doivent être collectées et traitées en temps réduit (détection de fraudes, etc)
- ❖ **Variété:** traiter des données de différentes natures (structurée et non structurée) qui proviennent de sources différentes.
- ❖ **Vérité:** fait référence à la qualité, de la fiabilité et l'authenticité de l'information
- ❖ **Valeur:** capacité à faire sortir des données une réelle valeur ajoutée et de nouvelle connaissance pour le business



16

Problème de grande quantité de données

- ❖ Stockage des données:
 - Les moyens classiques sont limités
- ❖ Limitation de ressources de traitement
 - La configuration de la machine (CPU, RAM)
- ❖ Le coût pour l'extension de l'infrastructure est élevé
- ➔ Les méthodes classiques n'arrivent plus à manipuler cette grande quantité d'information
 - capture, stockage, traitement, analyse , visualisation etc.

17

Traitement parallèle

❖ Super calculateur

- Machine avec des performances de calculs extrêmement élevées (GFlops)
- Architecture MPP
- N'est pas toujours une bonne solution (cout, latence disque, etc.)

❖ Traitement distribuée :



[https://fr.wikipedia.org/wiki/Frontier_\(superordinateur\)](https://fr.wikipedia.org/wiki/Frontier_(superordinateur))

1 Sec - 122 MB
x Secs - 1048576 MB (1 TB)
 $x = 1048576 / 122 = 8595 \text{ secs}$
2 hr 22 mins

Temps de calcul ~ 1h

Temps d'accès (lecture) > 2h

Rank	System	Cores	Rmax [PFlop/s]	Rpeak [PFlop/s]	Power [kW]
1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/INNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	Frontier - HPE Cray EX255a, AMD Optimized 3rd Generation EPYC 4AC 20Hz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607

<https://top500.org>

Traitement parallèle



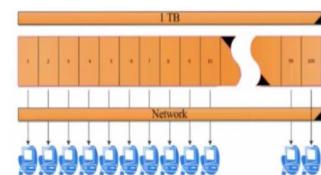
❖ Super calculateur

❖ Traitement distribuée :

- Plusieurs ordinateurs apparaissent comme un seul super ordinateur,
- communiquent entre eux par passage de message,
- opèrent ensemble pour atteindre un objectif commun
- Stockage et traitement des données

❖ Challenges

- Hétérogénéité, Ouverture, Sécurité, Évolutivité, concurrence, tolérance aux pannes, Transparence

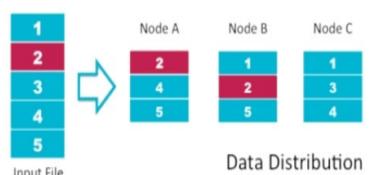


Temps de lecture = 150 min / 100 < 2min
Temps de calcul = 60 min / 100 < 1 min

18

Principe Big data

- ❖ La plupart des outils big data sont construits à base des concepts suivants:

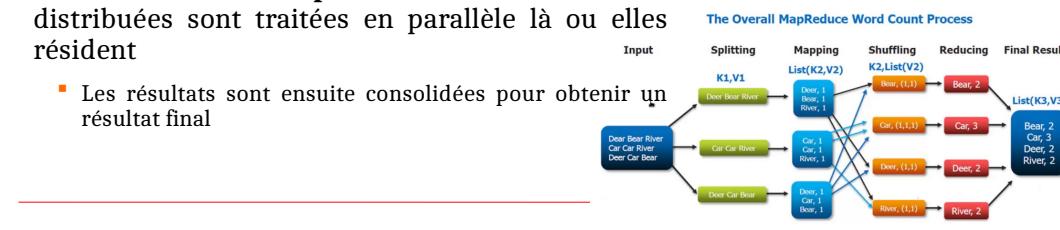


- ❖ La distribution des données: Répartir les données en blocs sur plusieurs machines

- système de fichiers transparent qui peut contenir des fichiers de grandes tailles(HDFS),
- bases de données spécifiques (HBase, Cassandra, ElasticSearch).

- ❖ Traitements en parallèle: Les données distribuées sont traitées en parallèle là où elles résident

- Les résultats sont ensuite consolidées pour obtenir un résultat final



Big data: Acteurs et solutions

- Les industriels du web (Google, Yahoo, Facebook, Twitter, etc.) ont été les premiers à confronter la grande quantité de données

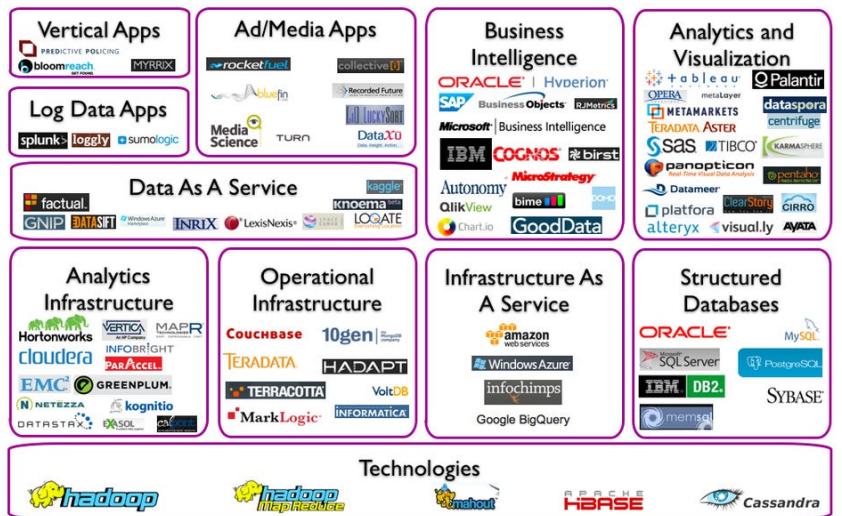
Société	Technologie développée	Type de technologie
Google	Big Table	Système de base de données distribuée propriétaire reposant sur GFS (Google File System). Technologie non Open Source, mais qui a inspiré Hbase qui est Open Source.
	MapReduce	Plate-forme de développement pour traitements distribués
Yahoo	Hadoop	Plateforme Java destinée aux applications distribuées et à la gestion intensive des données. Issue à l'origine de GFS et MapReduce.
	S4	Plateforme de développement dédiée aux applications de traitement continu de flux de données.
Facebook	Cassandra	Base de données de type NoSQL et distribuée.
	Hive	Logiciel d'analyse de données utilisant Hadoop.
Twitter	Storm	Plateforme de traitement de données massives.
	FlockDB	Base de données distribuée de type graphe.
LinkedIn	SenseiDB	Base de données temps réel distribuée et semi-structurée.
	Voldemort	Base de données distribuée destinée aux très grosses volumétries.

22

Big data: plateforme et outils



Big data landscape

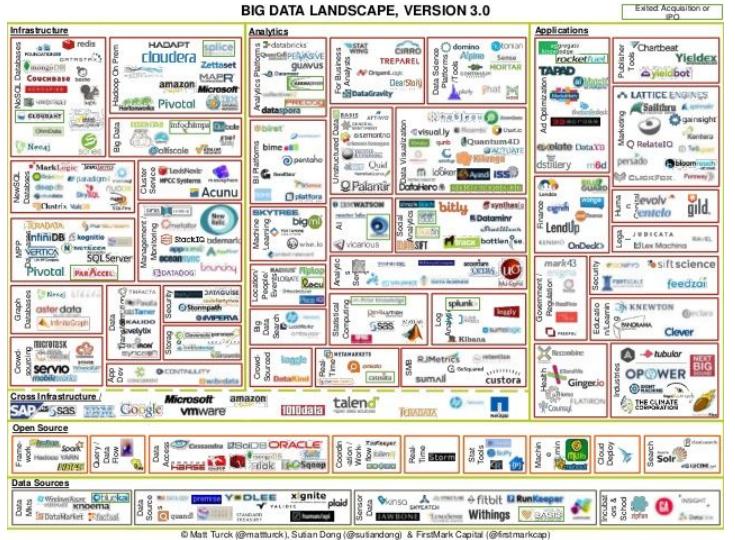


Copyright © 2012 Dave Feinleib

dave@vcdave.com

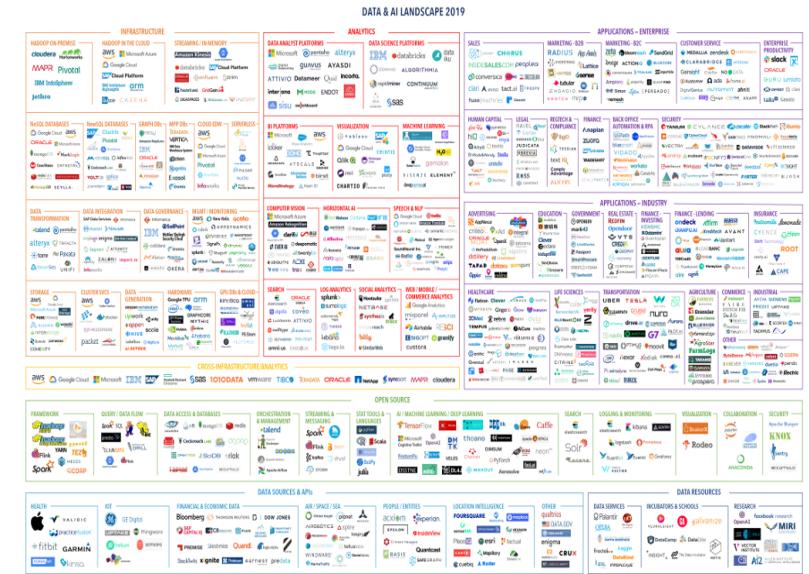
blogs.forbes.com/davefeinleib

Big data landscape



25

Big data landscape

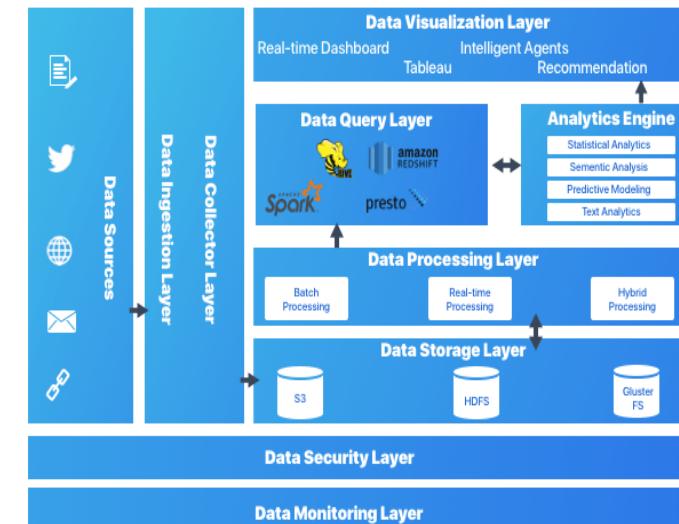


26

Big data: conclusion

- ❖ L'ère de la production massive des données
 - Apps, capteurs, fichiers logs, etc.
- ❖ Problème de stockage et de traitement s'imposent :
 - La capacité de stockage augmente et le Temps de lecture aussi
 - CPU d'une machine ne suffit plus
- ❖ Plusieurs solutions disponibles inspirées des travaux des industriels (Google)
- ❖ Hadoop est la solution la plus répondué

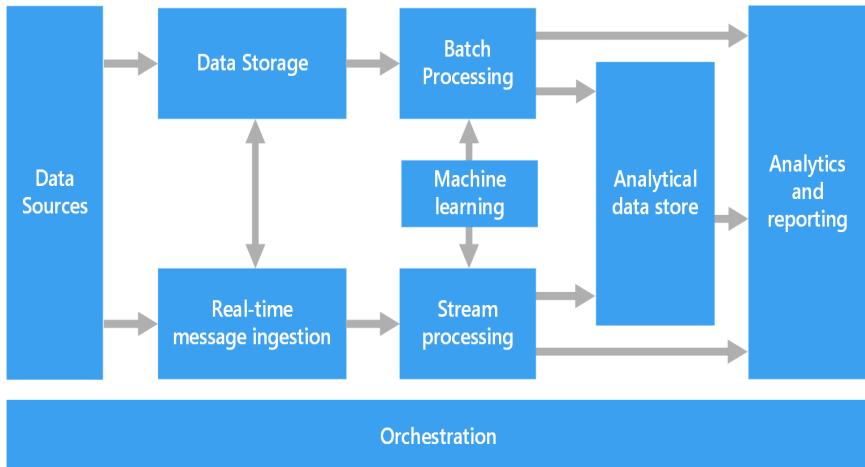
Architectural Pattern of Big Data



27

28

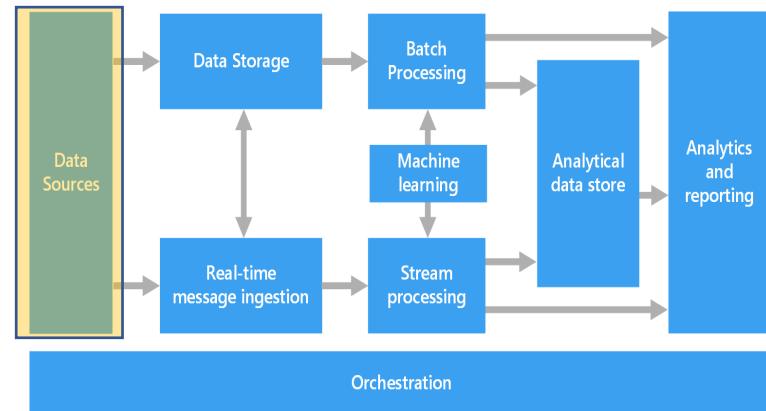
Composants d'une architecture Big Data



<https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/>

29

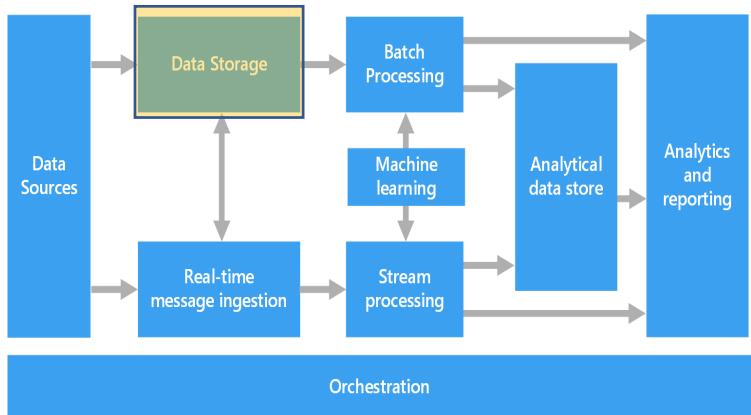
Composants d'une architecture Big Data



Sources de données peuvent être des BD, des fichiers, web logs, sources de données en temps réel (IoT devices)

<https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/> 30

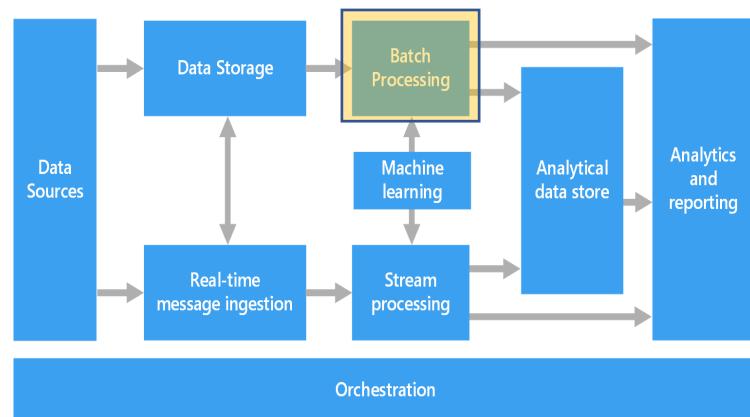
Composants d'une architecture Big Data



Stockage de données. Les données (traitement batch) sont généralement stockées dans un magasin de fichiers distribué qui peut contenir des volumes élevés de fichiers volumineux dans divers formats. data lake . (HDFS, MongoDB, etc.)

31

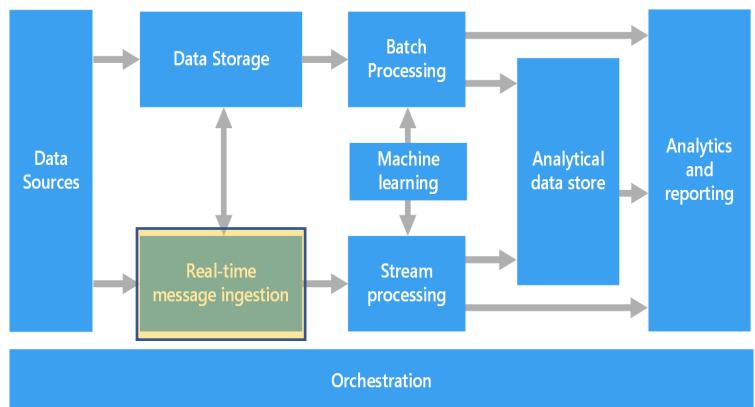
Composants d'une architecture Big Data



Traitement par batch: traiter des fichiers de données à l'aide de tâches (read/write/processing) par lots pour filtrer, agréger et préparer les données pour l'analyse. (Hive, Pig ou Map/Reduce, Spark.)

32

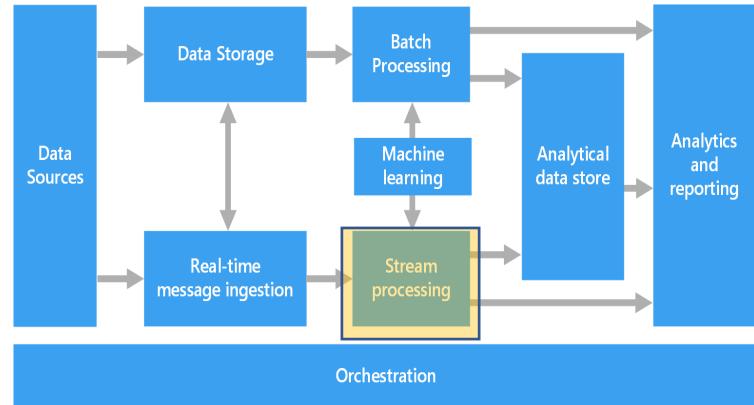
Composants d'une architecture Big Data



Ingestion de messages en temps réel: capturer et de stocker les messages en temps réel pour le traitement des flux. Il peut s'agir d'un simple magasin tampon de données de flux (Azure Event Hubs, Azure IoT Hub et Kafka).

33

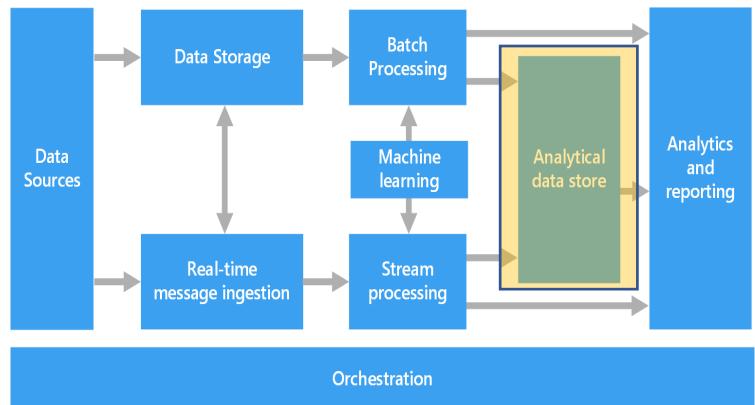
Composants d'une architecture Big Data



Traitement de Streaming: traiter les messages capturés en filtrant, en agrégant et en préparant les données pour l'analyse. Les données de flux traitées sont ensuite enregistrées. (Storm et Spark Streaming)

34

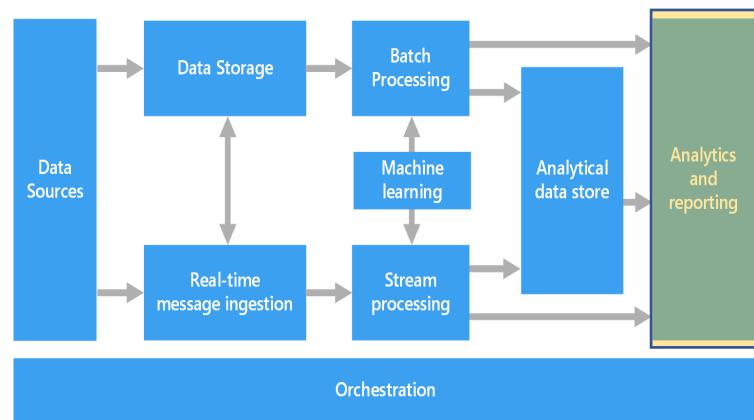
Composants d'une architecture Big Data



Magasin de données analytiques: préparer les données pour l'analyse dans un format structuré qui peut être interrogé à l'aide d'outils analytiques tel que les entrepôts de données relationnelles (solution BI traditionnelle) ou bien via une technologie NoSQL(HBase, Hive, Spark SQL).

35

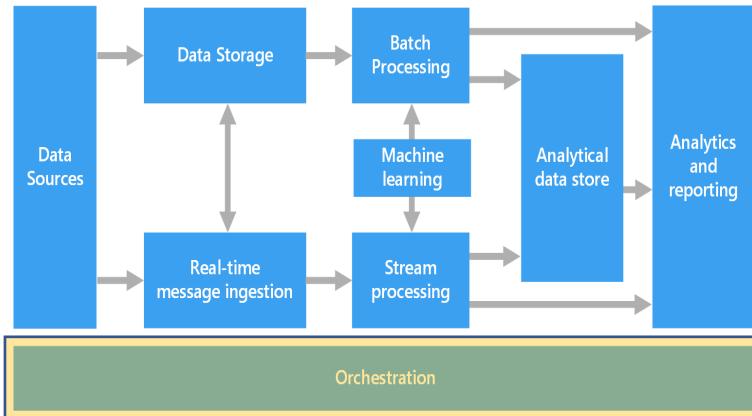
Composants d'une architecture Big Data



Analyse et rapport: fournir des analyses et de rapports. Ceci est possible à travers l'intégration d'une couche de modélisation des données, telle qu'un cube OLAP multidimensionnel ou un modèle de données tabulaire (Microsoft Power BI, Python ou R, Spark)

36

Composants d'une architecture Big Data



Orchestration: La plupart des solutions Big Data consistent en des opérations de traitement de données répétées, encapsulées dans des workflows.

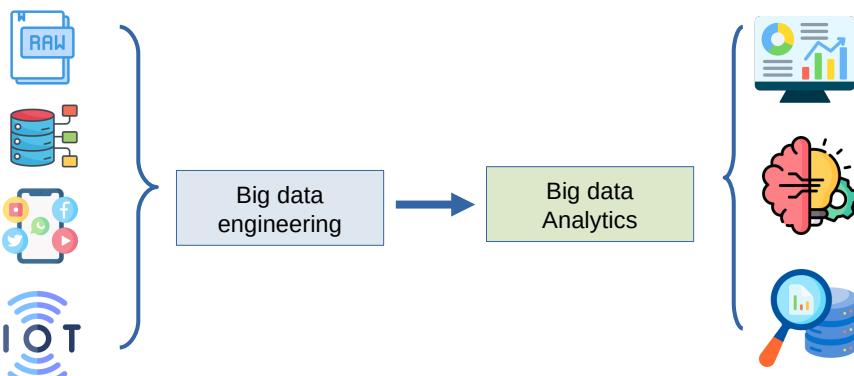
Transformer, déplacer les données source, charger les données traitées dans un magasin de données analytiques ou envoient les résultats directement vers un rapport ou un tableau de bord (Apache Oozie, Sqoop)

37

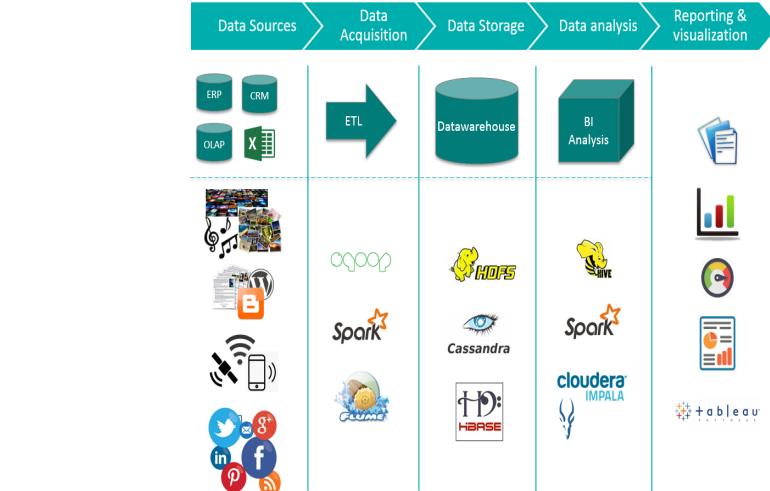
From raw Big Data to valuable insight

❖ The practice of **taking raw** data from a data sources and **processing it** so it's **stored and organized** for a downstream use case such as data analytics, business intelligence (BI) or machine learning (ML) model training. (**databricks**)

- **Big data Engineering**
- **Big data Analytics**



39



38

Big data engineering

❖ Se concentre sur la **conception**, la **construction** et la **maintenance** de l'infrastructure nécessaire pour gérer les **grandes quantités de données**.

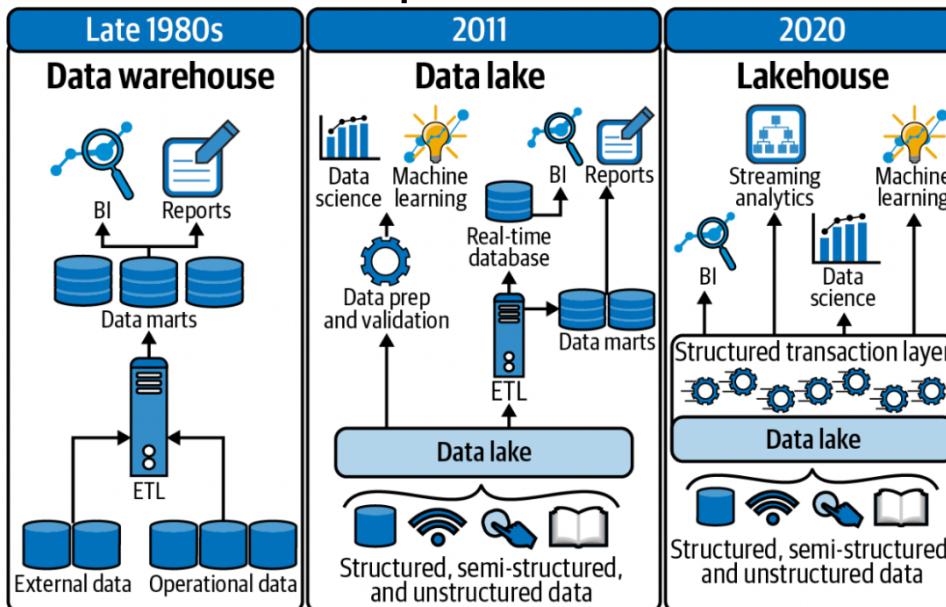
❖ **Création de pipelines de données qui permettent:**

- 1) **Ingestion** : Processus d'importation des données depuis une ou plusieurs sources (sur site, cloud, bases de données, flux de streaming) vers une **plateforme de données**.
- 2) **Transformation** : Traitement des données brutes pour les filtrer, standardiser, nettoyer et les agréger.
- 3) **Stockage** : Enregistrement des données dans un format structuré et optimisé, prêt pour l'analyse.

All about the movement, manipulation, and management of data. Lewis Gavin

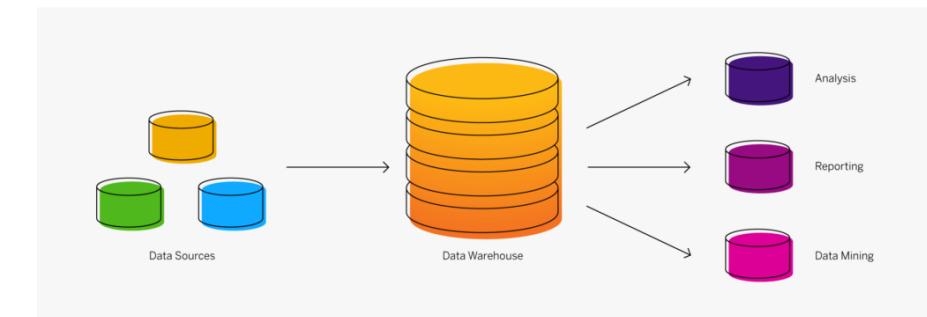
40

Data plateformes



Data plateformes : Datawarehouse

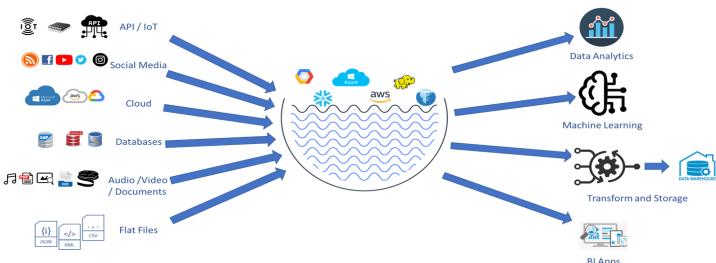
- ❖ **Définition:** BD centralisée optimisée pour les requêtes analytiques.
- ❖ **Types de Données :** Structurées uniquement.(schema on write)
- ❖ **Cas d'Utilisation :** Rapports, tableaux de bord, analyses BI traditionnelles.
- ❖ **Exemples :** SSAS, Amazon Redshift, Google BigQuery, Snowflake.
- ❖ **Limitations :** Rigidité, évolutivité & temps intégration élevé (ETL)



42

Data plateformes : Data lake

- ❖ **Définition :** Espace de stockage qui permet de conserver de grandes quantités de données **brutes** dans leur format natif.
- ❖ **Types de Données :** Structurées, semi-structurées, non structurées. (schema on read)
- ❖ **Cas d'Utilisation :** Stockage à long terme, analyses avancées, traitement Big Data, ML.
- ❖ **Exemples :** Amazon S3, Azure Data Lake, **Hadoop**
- ❖ **Limitations :** qualité des données, requêtes complexes lentes, manque de gouvernance (data swamp)



43

Data plateformes : LakeHouse

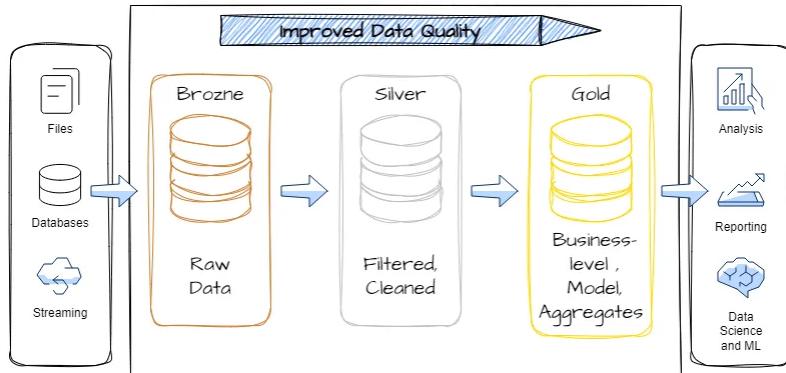
- ❖ **Définition :** Fusion des concepts de Data Lake et Data Warehouse, offrant des capacités de stockage brut et de traitement analytique avancé sur une même plateforme.
- ❖ **Types de Données :** Structurées, semi-structurées et non structurées.
- ❖ **Cas d'Utilisation :** Analytique en temps réel, IA/ML, gouvernance des données.
- ❖ **Exemples :** Databricks Lakehouse, Apache Iceberg, **Delta Lake**
- ❖ **Avantages :** ELT-ETL, Flexible schema with ACID transactions, gouvernance de données, stockage et calcul découpés



44

Medallion architecture

- Datalake : A popular pattern is the **medallion architecture**, qui définit trois étapes dans le processus — Bronze, Silver and Gold.



45

DataWarehouse vs Datalake vs Lakehouse

Factors	Data Lake	Data Lakehouse	Data Warehouse
o Types of Data	o Structured, Semi-structured and Unstructured Data	o Structured, Semi-structured and Unstructured Data	o Structured Data Only
o Cost	o \$	o \$	o \$\$\$
o Format	o Open Format	o Open Format	o Closed, Proprietary Format
o Scalability	o Scales to store any amount and any type of information at low cost	o Scales to store any amount and any type of information at low cost	o Due to vendor expenses, scaling up gets massively more expensive
o Reliability	o Low quality, Data swamp	o High quality, reliable information	o High quality, reliable information
o Performance	o Poor	o High	o High
o Intended Users	o Limited: Data Scientists	o Unified: Data analysts, data scientists, machine learning engineers	o Limited: Data Analysts

46

Big data Engineering

- Les ingénieurs en données sont responsables de la **création** de **pipelines** de **données** qui permettent la **collecte**, le **stockage** et le **traitement** efficace des données. Leur objectif principal est de garantir que les données sont accessibles, fiables et prêtes à être analysées
- Ingest** : the process of bringing data from one or more data sources into a data platform. (on premise, cloud, Dbs, Streams)
- Transform** : Takes raw ingested data and uses a series of steps ("transformations") to filter, standardize, clean and finally aggregate it so it's **stored** in a usable way.
- Orchestrate** : the way a data pipeline that performs ingestion and transformation is scheduled and monitored.

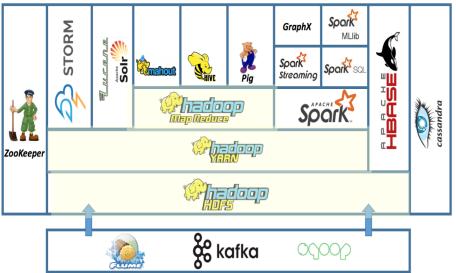
47

Big data Analytics

- Systematic and computational analysis of data
- find meaningful **patterns** in data, its **interpretation**, and its **presentation** or **communication** to relevant stakeholders.
- Understand the past to **predict** what will happen in the future.
- helps organizations to **make effective** and efficient decisions

48

Apache Hadoop



Infrastructure et architecture Big data



49

Apache Hadoop: caractéristiques



- ❖ **Tolèrent aux pannes:** la défaillance d'un nœud ne provoque pas l'échec de calcul
- ❖ **Evolutif (Scalable):** milliers de machines
- ❖ **Performant:** support du traitement de données massive (To, Po)
- ❖ **Parallélisme de données:** même traitement appliqué sur toutes les données
- ❖ **Economie:** matériel standard
- ❖ Dispose d'un vaste écosystème (HBase / ZooKeeper / Avro / etc.)

51

- ❖ Framework **open source** pour le traitement distribué de quantité massive de données sur des clusters de machines à l'aide de modèles de programmation simples
- ❖ Conçu par Doug cutting en 2004
- ❖ Inspiré par les publications **Mapreduce**, **GoogleFS** et **bigtable** de Google
- ❖ Ecrit en **java**

50

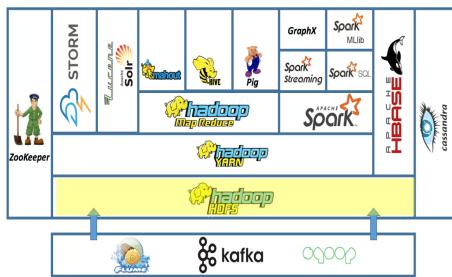
Apache Hadoop



- ❖ Hadoop se compose de deux principaux sous projets:
 - **MapReduce:** un mécanisme d'exécution parallèle de programmes
 - **Système de fichiers distribué Hadoop (aka. HDFS) :** répartit les données sur de nombreuses machines
- **Principe:**
 - Diviser les données sur une collection de machines (cluster)
 - Traiter les données directement là où elles sont stockées

52

Hadoop Distributed File System (HDFS)

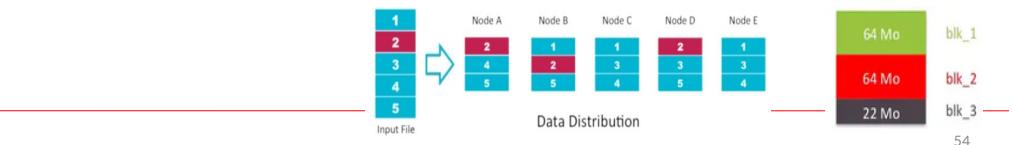


Hadoop Distributed File System



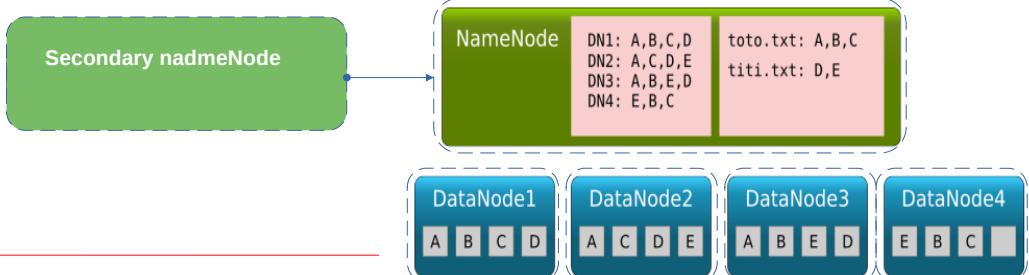
53

- ❖ HDFS est un système de fichiers distribué:
- ❖ les fichiers et dossiers sont organisés en arbre (comme Unix)
- ❖ les fichiers sont stockés sur un grand nombre de machines de manière transparente pour l'utilisateur. (un seul arbre)
- ❖ les fichiers sont décomposés en grands blocs (64 MO par défaut, 128 jusqu'à 1GO) dupliquée pour la fiabilité sur plusieurs machine
 - 3 machines (nombre configurable)



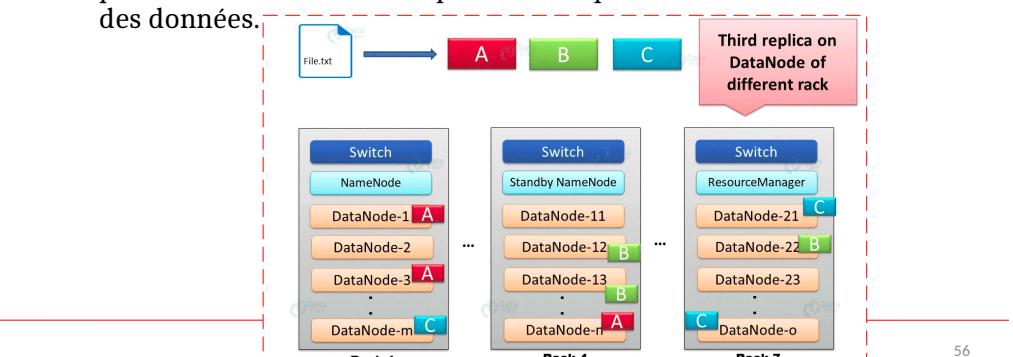
Organisation des machines pour HDFS

- ❖ Un cluster HDFS est constitué de machines jouant différents rôles exclusifs entre eux :
- ❖ **namenode (maître)**: Gérer l'état du système de fichiers.
 - ❖ contient tous les noms et blocs des fichiers, (annuaire)
- ❖ **secondary namenode**, : machine (secours) qui enregistre des sauvegardes de l'annuaire à intervalles réguliers.
- ❖ **Datanodes** : machines qui stockent les blocs des fichiers.



HDFS rack awareness

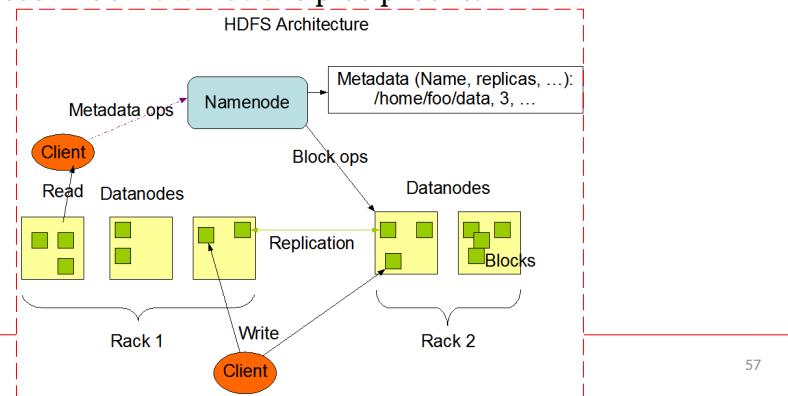
- ❖ **Rack** : ensemble de DataNodes connectés à l'aide du même switch réseau.
- ❖ **Rack awareness** : mécanisme de tolérance aux pannes consiste à placer les répliques sur des racks différents.
 - ➔ Éviter la perte de données en cas de panne d'un rack entier
 - ➔ permet d'utiliser la bande passante de plusieurs racks lors de la lecture des données.



56

High availability

- ❖ **DataNodes** envoie en continu des messages de « heartbeat » au **namenode** après une durée déterminée (3 secondes par défaut).
- ❖ Si le **namenode** ne reçoit pas de retour, le **datanode** est considéré comme étant « dead »
- ❖ Quand un utilisateur demande à accéder à ses données, le **NameNode** fournit l'adresse IP du **DataNode** le plus proche.



57

59

Configuration hadoop

- ❖ Plusieurs fichiers de configuration

- } **core-site.xml** : indique l'host et le port du Namenode.

```
<?xml version="1.0"?>
<xmstylesheet type="text/xsl" href="configuration.xsl">
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master.local:9000</value>
    <description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation.</description>
  </property>
</configuration>
```

- } **hdfs-site.xml** : Configurations du NameNode/datanode(taille de block, le chemin du namespace et fichiers logs)

```
<?xml version="1.0"?>
<xmstylesheet type="text/xsl" href="configuration.xsl">
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
    <description>The actual number of replications can be specified when the file is created.</description>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/srv/hadoop/datanode</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/srv/hadoop/namenode</value>
  </property>
</configuration>
```

<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html>

58

Commandes HDFS dfs

- ❖ La commande hdfs dfs et ses options permet de gérer les fichiers et dossiers :
hdfs dfs -help
 - hdfs dfs -ls [noms...] (pas d'option -l)
 - hdfs dfs -cat nom
 - hdfs dfs -mv ancien nouveau
 - hdfs dfs -cp ancien nouveau
 - hdfs dfs -mkdir dossier
 - hdfs dfs -rm -f -r dossier

Échanges entre HDFS et le monde

- ❖ Pour placer un fichier dans HDFS:

- hdfs dfs -copyFromLocal fichiersrc fichierdst
 - hdfs dfs -put fichiersrc [fichierdst]

- ❖ Pour extraire un fichier de HDFS:

- hdfs dfs -copyToLocal fichiersrc dst
 - hdfs dfs -get fichiersrc [fichierdst]

- ❖ Exemple

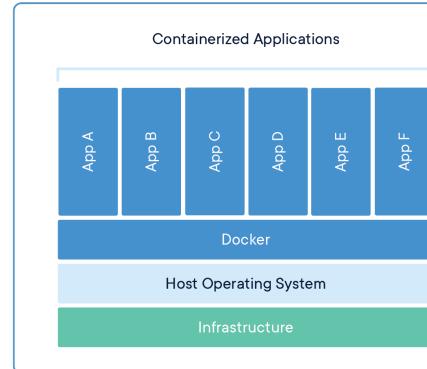
```
hdfs dfs -mkdir -p livres
wget http://www.textfiles.com/etext/FICTION/alice.txt
hdfs dfs -put alice.txt livres
hdfs dfs -ls livres
hdfs dfs -get livres/center_earth
```

60

Docker et conteneurs

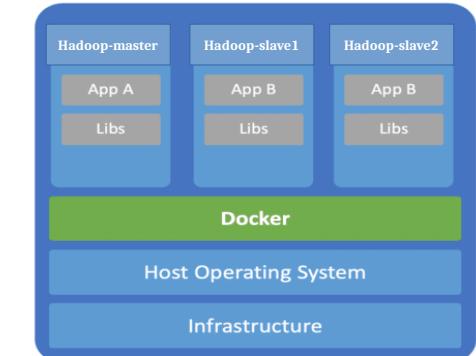
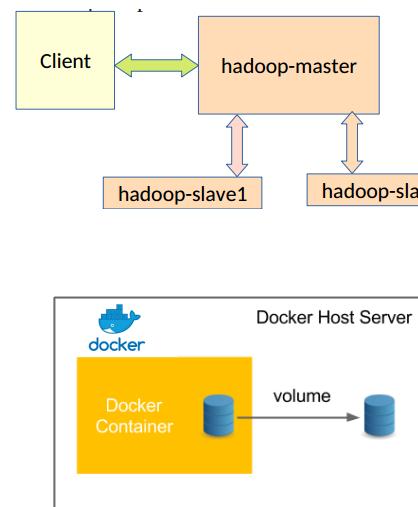


- ❖ Une plate-forme de conteneurs logicielle conçue pour développer, expédier et exécuter des applications utilisant la technologie des conteneurs.
- ❖ Une **image Docker** est un package logiciel léger, **autonome** et **exécutable** qui comprend tout ce qui est nécessaire pour exécuter une application : **code, environnement d'exécution, outils système, bibliothèques** système et paramètres.
- ❖ les images deviennent des **conteneurs** au moment de l'**exécution** sur Docker Engine



61

Cluster Hadoop avec Docker



yassern1/hadoop-spark-jupyter:1.0.3

API Java pour HDFS

- ❖ Hadoop propose une API Java pour accéder aux fichiers de HDFS.
- ❖ Deux classes principales :
 - **FileSystem** représente l'arbre des fichiers (*file system*). Elle permet de réaliser les ops de base (copier des fichiers local HDFS, renommer, créer, etc).
 - **FileStatus** gère les informations d'un fichier ou dossier :
 - ✓ taille avec `getLen()`,
 - ✓ nature avec `isDirectory()` et `isFile()`,
- ❖ la classe **Configuration**: connaître la configuration du cluster HDFS,
- ❖ la classe **Path** représente les noms complets des fichiers.

63

Exemple

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

public class FileStatus {
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        FileSystem fs;
        fs = FileSystem.get(conf);
        Path nomcomplet = new Path("/user/root/input", "purchases.txt");
        FileStatus infos = fs.getFileStatus(nomcomplet);
        System.out.println(Long.toString(infos.getLen())+" octets");
        fs.rename(nomcomplet, new Path("/user/root/input", "achats.txt"));
    }
}
```

64

Informations sur les fichiers

```
public class HDFSInfo {  
    public static void main(String[] args) throws IOException {  
        // obtenir un object représentant HDFS  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
        // nom du fichier à lire  
        Path nomcomplet = new Path("./achats.txt");  
        if (! fs.exists(nomcomplet)) {  
            System.out.println("Le fichier n'existe pas");  
        } else {  
            // afficher taille du fichier  
            FileStatus infos = fs.getFileStatus(nomcomplet);  
            System.out.println(Long.toString(infos.getLen()) + " octets");  
            // liste des blocs du fichier  
            BlockLocation[] blocks = fs.getFileBlockLocations(infos, 0, infos.getLen());  
            System.out.println("le nombre de blocks est:" + blocks.length + " blocs :");  
            for (BlockLocation blocloc: blocks) {  
                System.out.println("\t" + blocloc.toString() + " length:" + blocloc.getLength());  
            }  
        }  
        // fermer HDFS  
        fs.close();  
    }  
}
```

65

Création d'un fichier HDFS

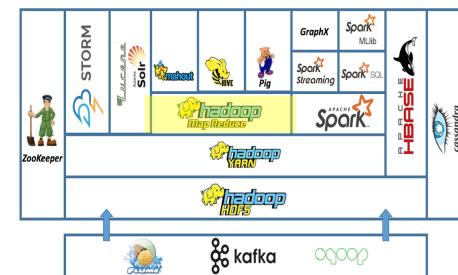
```
public class WriteHDFS {  
  
    public static void main(String[] args) throws IOException {  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
        Path nomcomplet = new Path("bonjour.txt");  
        if (! fs.exists(nomcomplet)) {  
            FSDataOutputStream outStream = fs.create(nomcomplet);  
            outStream.writeUTF("Bonjour tout le monde !");  
            outStream.close();  
        }  
        fs.close();  
    }  
}
```

67

Lecture d'un fichier HDFS

```
public class ReadHDFS {  
  
    public static void main(String[] args) throws IOException{  
        Configuration conf = new Configuration();  
        FileSystem fs = FileSystem.get(conf);  
        Path nomcomplet = new Path("achat.txt");  
        FSDataInputStream inStream = fs.open(nomcomplet);  
        InputStreamReader isr = new InputStreamReader(inStream);  
        BufferedReader br = new BufferedReader(isr);  
        String line = br.readLine();  
        System.out.println(line);  
        inStream.close();  
        fs.close();  
    }  
}
```

66



Map Reduce



68

❖ Hadoop se compose de deux principaux sous projets:

- **MapReduce:** un mécanisme d'exécution parallèle de programmes
- **Système de fichiers distribué Hadoop (aka. HDFS) :** répartit les données sur de nombreuses machines

Principe:

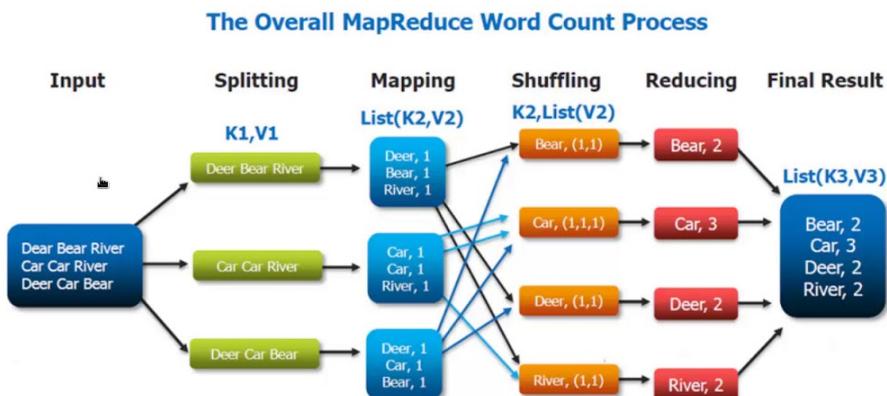
- Diviser les données sur une collection de machines (cluster)
- Traiter les données directement là où elles sont stockées
- Possibilité d'ajouter des machines

69

MapReduce

❖ Repose sur deux opérations:

- ▶ **Map:** transformer les données issues des blocs en une série de **clé/valeur**.
- ▶ **Reduce:** appliquer un traitement (Fonctions d'agrégation)à toutes les valeurs de chacune des clé valeurs pour produire un ensemble plus petit de tuples ou de paires clé-valeur.



'1

MapReduce

❖ Modèle de programmation pour le traitement des **données massives**

❖ Paradigme visant à généraliser l'approche « **diviser pour régner** » pour qu'elle soit applicable à tous problèmes

❖ **Framework for parallel computing**

❖ La présentation sous forme rigoureuse et généralisable apparaît avec le whitepaper R&D Google department

« **mapreduce: Simplified data Processing on large clusters** »

70

Map et reduce

❖ Opérations fonctionnelles d'ordre supérieur

❖ **Map:** ($f, [a, b, c, \dots]$) $\rightarrow [f(a), f(b), f(c), \dots]$

- ▶ Applique une fonction à tous les éléments d'une liste
- ▶ ex.: $\text{map}((f: x \rightarrow x + 1), [1, 2, 3]) = [2, 3, 4]$
- ▶ Intrinsèquement parallèle

❖ **Reduce:** ($f, [a, b, c, \dots]$) $\rightarrow f(a, f(b, f(c, \dots)))$

- ▶ Applique une fonction récursivement à une liste
- ▶ Ex: $\text{reduce}((f: x, y \rightarrow x + y), [1, 2, 3, 4]) = (1 + (2 + (3 + 4))) = 10$

❖ Purement fonctionnel

72

Modèle de données $\langle k, v \rangle$

❖ Tableau associatif (i.e. ~hashtable)

- ▶ Modèle simple de base de données
 - data store

❖ MapReduce opère sur des couples $\langle k, v \rangle$

- ▶ **En entrée, en sortie, en interne**
- ▶ Pas nécessairement dans le même espace de nommage à chaque étape

Map reduce

❖ Map $\langle k, v \rangle \rightarrow \text{List} \langle k_2, v_2 \rangle$

- ▶ Application d'une fonction sur chaque donnée de départ
- ▶ k et k_2 : pas nécessairement dans le même domaine de définition

❖ Shuffle $\text{List} \langle k_2, v_2 \rangle \rightarrow \text{List} \langle k_2, \text{list}(v_2) \rangle$

- ▶ Tri des résultats par cléf
- ▶ on regroupe les v_2 qui ont la même k_2
- ▶ Similaire à *group by* k_2 en SQL

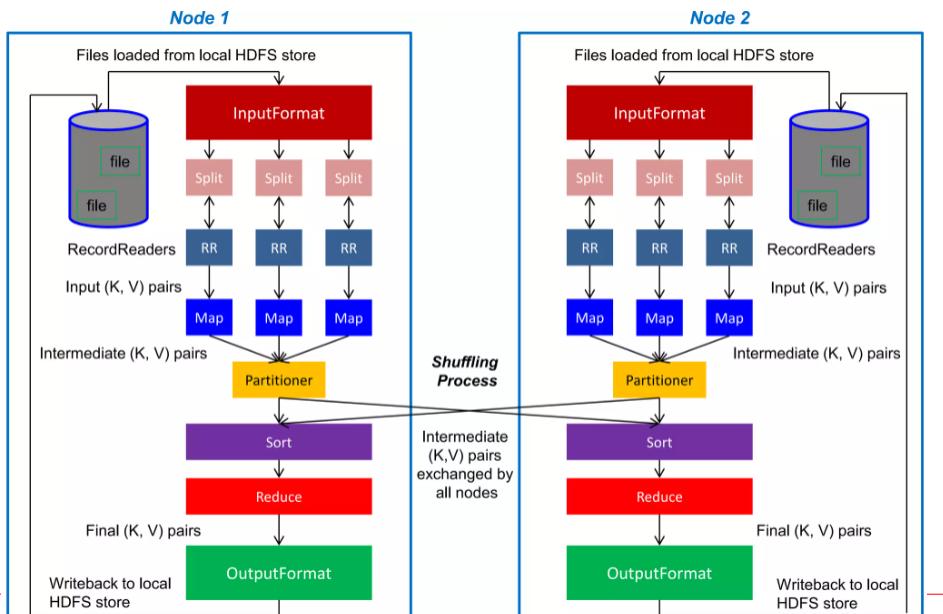
❖ Reduce $\text{list} \langle k_2, \text{list}(v_2) \rangle \rightarrow \text{list} \langle k_3, v_3 \rangle$

- ▶ Application d'une fonction de réduction sur tous les groupes $\langle k_2, v_2 \rangle$
- ▶ Fonction d'agrégation (sum, avg, min, max, etc.)

73

74

MapReduce : Exemple d'exécution



MapReduce

❖ On distingue 4 étapes

- **Découper** (split) les données en plusieurs fragments
- **Mapper** chacun de ces fragments pour obtenir des **clés/valeurs**
- **Grouper** (shuffle) ces clés/valeurs **Par clé**
- **Réduire** (reduce): les groupes des clés/valeurs en une forme finale avec une valeur (fct agrégation) pour chacune des clés distinctes

❖ Le problème devient alors parallélisable



76

Mapreduce exemple

- ❖ Compter les occurrences des mots d'une collection de documents
- ❖ Entrée : < nom, contenu >
- ❖ Map : pour chaque mot du contenu → <mot, 1>
 - ▶ Big data vs data → <Big,1>, < data,1>, < vs,1>, <data,1>
- ❖ Shuffle : regroupe les couple <mot, 1> par mot
 - ▶ <Big,1>, < data,1>, < vs,1>, <data,1> → <Big,1>, < data,(1,1)>, <vs,1>
- ❖ Reduce : <mot, nombre> -> <mot, total>
 - ▶ < data,(1,1)> → <data,2>
- ❖ Sortie : collection de <mot, occurrences>
 - ▶ <Big,1>, < data,2>, <vs,1>

77

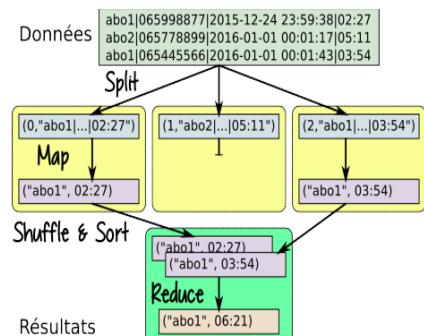
Méthodologie MapReduce

- 1) Choisir une manière de découper les données de telle sorte que le Map soit parallélisable
- 2) Définir quelle clé utiliser pour le problème
- 3) Ecrire le prog pour l'opération du Map
- 4) Ecrire le prog pour l'opération du reduce

78

Exercice

- ❖ Une entreprise de téléphonie veut calculer la durée totale des appels téléphoniques d'un abonné à partir d'un fichier contenant tous les appels de tous les abonnés (n° d'abonné, n° appelé, date, durée d'appel)

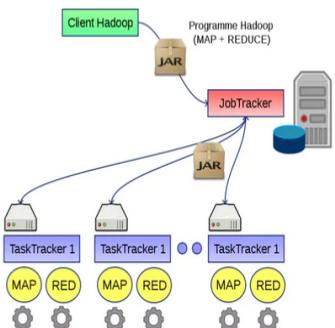


79

MapReduce Hadoop

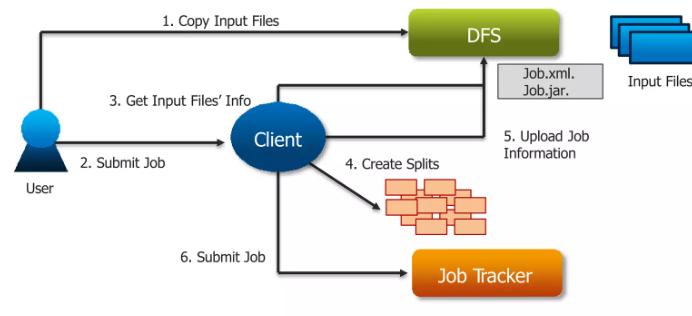
- ❖ Comme pour HDFS, la gestion des tâches se base sur deux composants:

- **JobTracker**: recevoir la tâche à exécuter (jar), les données d'entrée et le rep où stocker les résultats
 - ✓ Un job tracker par cluster
- **TaskTracker**: Communique avec le JT, va recevoir les opérations map/reduce à effectuer et les blocs correspondants
 - ✓ Un taskTracker par machine cluster



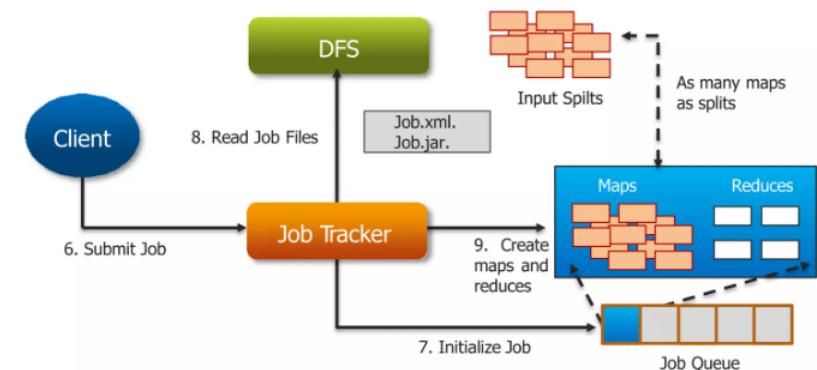
80

JobTracker



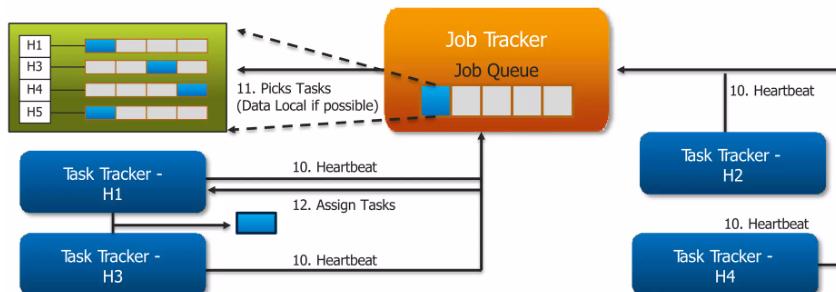
81

JobTracker



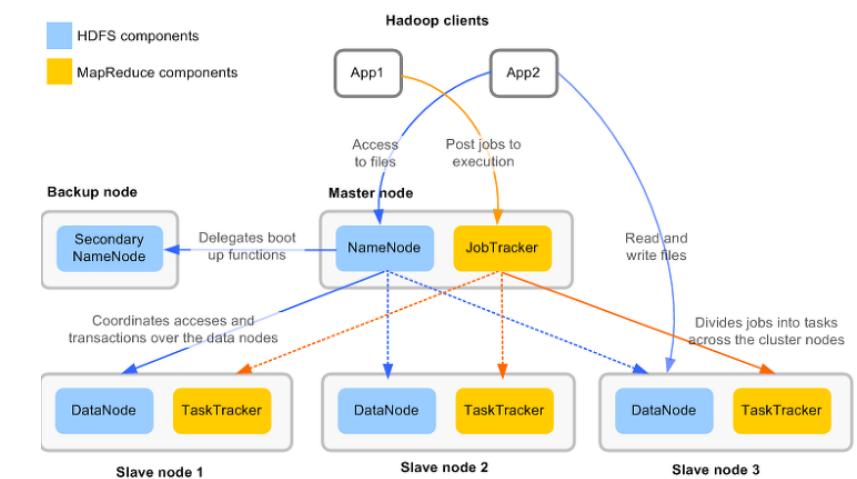
82

JobTracker/TaskTracker



83

Architecture générale de Hadoop



84

API Java pour MapReduce

- ❖ la programmation d'un job MapReduce en Java nécessite définir trois classes :
 - Une classe de **Mapper**: contient une seule méthode, appelée map qui reçoit une paire clé-valeur en paramètre et génère un nombre quelconque de paires.
 - Une classe de **Reducer**: contient également une seule méthode, appelée reduce qui reçoit une liste de paires en paramètre et génère une seule paire.
 - Une classe générale qui crée un **Job** faisant référence aux deux précédentes classes.
- ❖ Les deux premières sont des classes génériques (*templates*) paramétrées par les types des clés et des valeurs.

85

Squelette de Mapper

```
public class TraitementMapper extends Mapper<TypCleE, TypValE, TypCleI, TypValI>
{
    @Override
    public void map(TypCleE cleE, TypValE valE, Context context) throws
Exception
    {
        /** traitement: cleI = ..., valI = ... */
        TypCleI cleI = new TypCleI(...);
        TypValI valI = new TypValI(...);
        context.write(cleI, valI);
    }
}
```

Types de données et Interface Writable

- ❖ La classe Mapper est paramétrée par 4 types (spéciaux)
- ❖ Les types Text, IntWritable sont des implémentations d'une interface appelée Writable qui comprend:
 - un constructeur: IntWritable val = new IntWritable(34);
 - un modificateur : void set(nouvelle valeur); val.set(35);
 - un accesseur : type get() int v = val.get();
- ❖ L'interface Writable permet la *sérialisation des objets* pour échanger les objets entre machines.

type	description
Text	chaîne UTF8 quelconque
BooleanWritable	représente un booléen
IntWritable	entier 32 bits
LongWritable	entier 64 bits
FloatWritable	réel IEEE 32 bits
DoubleWritable	réel IEEE 64 bits

87

Squelette de Reducer

```
public class TraitementReducer extends Reducer<TypCleI, TypValI, TypCleS, TypValS>
{
    @Override
    public void reduce(TypCleI cleI, Iterable<TypValI> listeI, Context
context) throws Exception
    {
        TypCleS cleS = new TypCleS();
        TypValS valS = new TypValS();
        for (TypValI val: listeI) {
            /** traitement: cleS.set(...), valS.set(...) */
        }
        context.write(cleS, valS);
    }
}
```

88

Squelette de Reducer

❖ La méthode reduce reçoit une collection de valeurs venant du Mapper.

❖ CleI et ValeursI sont les clés et valeurs intermédiaires.

❖ Il faut itérer sur chacune pour produire la valeur de sortie du réducteur.

❖ Comme pour map, la classe est paramétrée par les types des clés et des valeurs à manipuler. Ce sont des Writable : Text, IntWritable, etc.

N.B: les **types des clés et valeurs d'entrée du reducer** doivent être exactement **les mêmes** que les types des clés et valeurs de sortie du **mapper** (sinon erreur d'exécution)

89

Squelette de TraitementDriver

La classe principale qui crée et lance le job MapReduce

```
public class TraitementDriver extends Configured implements Tool
{
    public static void main(String[] args) throws Exception{
        configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "titre_traitement");
        job.setJarByClass(TraitementDriver.class);
        job.setMapperClass(TraitementMapper.class);
        job.setReducerClass(TraitementReducer.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        boolean success = job.waitForCompletion(true);
        return success ? 0 : 1;
    }
}
```

91

Context object

❖ permet au Mapper/Reducer d'interagir avec le reste du système Hadoop

- } **Accès aux données** : permet au Mapper/reducer d'accéder aux paramètres de configuration et aux métadonnées de tâche (chemins d'E/S, param supplémentaires spécifiés)
- } **Méthodes output** : fournit des méthodes pour émettre des paires clé-valeur.
 - Dans un mapper/reducer, on utilise généralement context.write(key, value) pour générer les résultats de la fonction map/reduce.
- } **État de la tâche** : récupérer l'ID de tentative de la tâche, savoir si la tâche est en cours d'exécution ou terminée.
- } **Accès à la configuration** : accéder aux paramètres de configuration personnalisés qui ont été définis pour le job.

90

Squelette de TraitementDriver

❖ La méthode **run** est chargée de créer et lancer un Job.

❖ L'enchaînement général est comme suit:

1. Obtenir une instance de Configuration. Elle contient les options telles que les formats des fichiers, leur nom HDFS complet, etc.
2. Créer un Job, lui indiquer les classes concernées : *mapper* et *reducer*.
3. Fournir les noms complets des fichiers à traiter et à produire.
4. Indiquer les types des clés et valeurs. Par défaut, ce sont des Text.
5. Attendre la fin du job et retourner un code d'erreur.

92

Compilation et lancement de traitement

1. Compilation

```
hadoop com.sun.tools.javac.Main Traitement*.java
```

2. Emballage dans un fichier jar.

```
jar cfe Traitement.jar TraitementDriver Traitement*.class
```

3. Préparation : mettre en place les fichiers à traiter, supprimer le dossier de sortie

```
hdfs dfs -rm -r -f sortie
```

4. Lancement:

```
yarn jar Traitement.jar input output
```

5. Résultats dans le dossier sortie

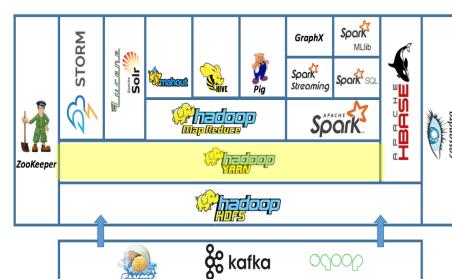
```
hdfs dfs -cat sortie/part-r-00000
```

Compilation et lancement de traitement

- ❖ Hadoop peut exécuter des programmes MapReduce écrits dans différents langages :

- ◆ Java
- ◆ Python
- ◆ Ruby

93



Gestionnaire de ressources

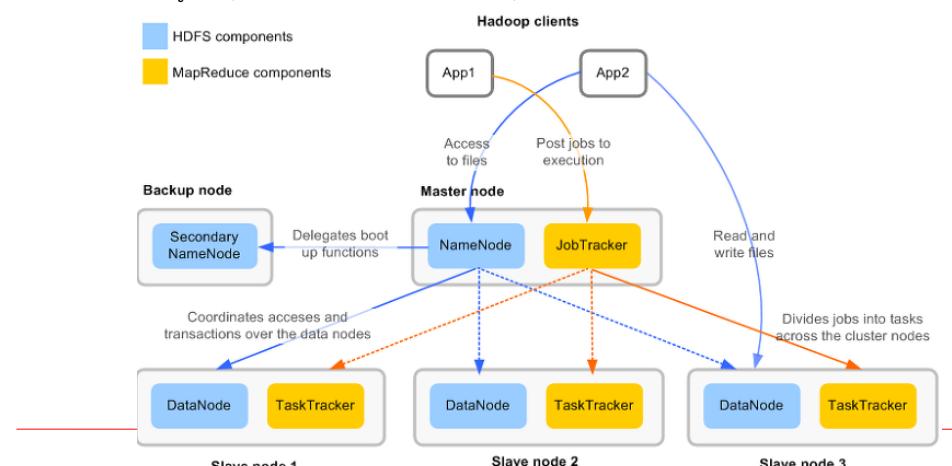


95

Yarn

- ❖ Le **jobTracker** à plusieurs responsabilités:

- Gérer les ressources du cluster
- Gérer les jobs (ordonnancer, surveiller, etc)



94

96

Yarn

Solution Yarn :

- Yet Another Ressource Negotiator
- mécanisme permettant de gérer des travaux (*jobs*) sur un cluster de machines.
- Répond au problème de **scalabilité**: séparation de la **gestion de l'état** du cluster et la **gestion (allocation) des ressources**
- permet aux utilisateurs de lancer des *jobs* MapReduce sur des données présentes dans HDFS, et de suivre (*monitor*) leur avancement,
- Éventuellement YARN peut déplacer (transparent) un processus d'une machine à l'autre en cas de défaillance ou d'avancement jugé trop lent.

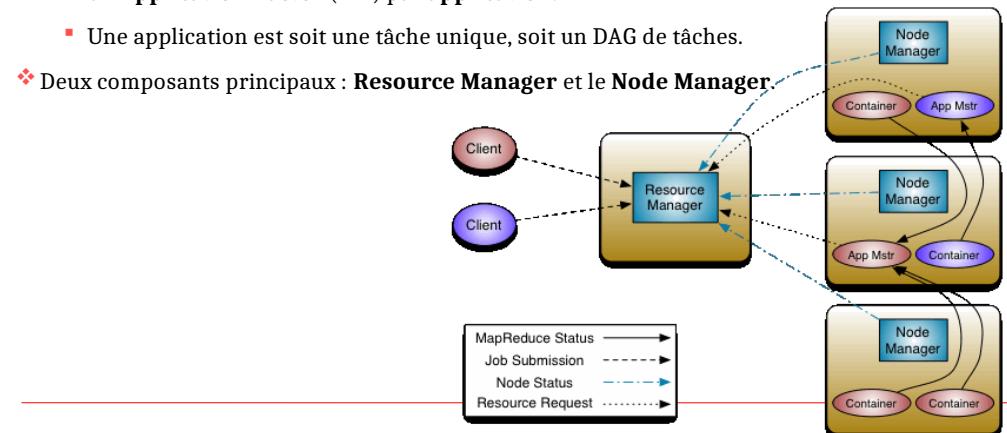
97

Yarn

❖ Yarn : séparer les fonctionnalités de **gestion des ressources** et de **planification/surveillance** des tâches dans des démons distincts.

- un **ResourceManager (RM) global**
- un **ApplicationMaster (AM) par application**.
- Une application est soit une tâche unique, soit un DAG de tâches.

❖ Deux composants principaux : **Resource Manager** et le **Node Manager**



Yarn

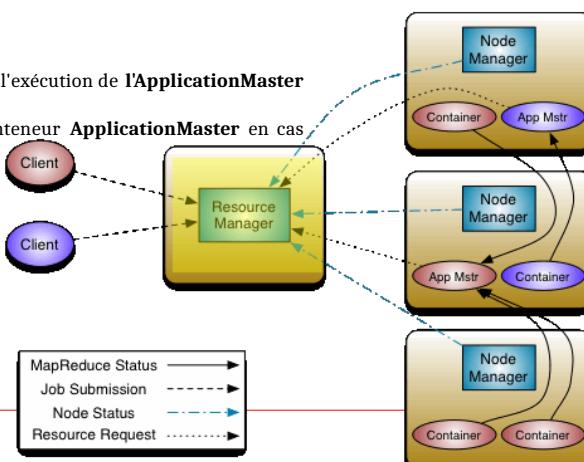
❖ **Resource Manager** : arbitre les ressources entre toutes les applications du système.

❖ Le **Scheduler** :

- allouer des ressources aux différentes applications en cours d'exécution soumises à des contraintes familières de capacités, de files d'attente, etc.
- exécuter sa fonction sur la base du container.

❖ L'**ApplicationsManager** :

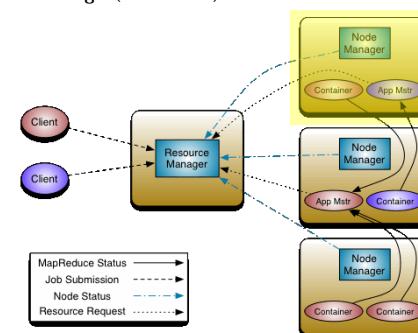
- l'acceptation des soumissions de job,
- la négociation du premier conteneur pour l'exécution de l'**ApplicationMaster** spécifique à l'application
- fournit le service de redémarrage du conteneur **ApplicationMaster** en cas d'échec.



Yarn

❖ **NodeManager** : agent par machine responsable :

- des **containers**: conteneur de ressource est une **notion abstraite** qui intègre des éléments tels que la mémoire, le processeur, le disque, le réseau, etc
- **Surveillance** de l'utilisation des ressources (processeur, mémoire, disque, réseau)
- **Rapporter au ResourceManager(Scheduler)**



100

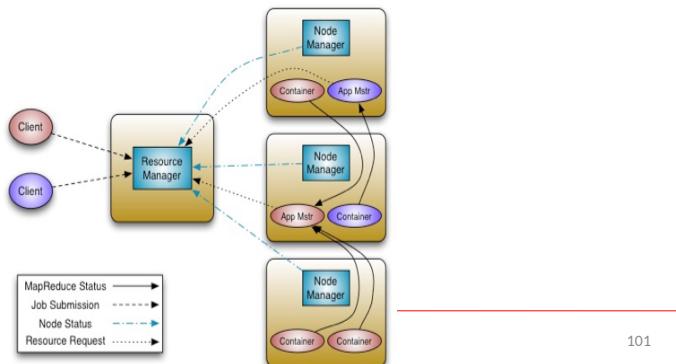
Yarn (déroulement)

ResourceManager

- RM accepter et programmer les taches et gère les ressources du clusters

Node Manager (remplace le taskTracker)

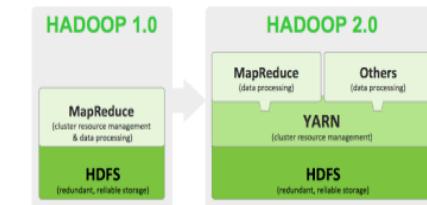
- AM est alloué par app pour gérer les taches (déployées sur les nœuds esclaves)
- Container s'exécutant sur un Node Manager: c'est une allocation de ressource sur un NM.
- AM prend le Container et le présente au NM gérant la machine sur lequel le Conteneur a été alloué



101

Déroulement de l'exécution sous Yarn

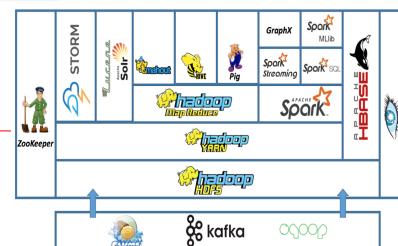
- Le client soumet le travail (jar) au **resourceManager**
- Le **RM** alloue un container, un **applicationMaster** sur le cluster et lance la classe driver du prog.
- L'applicationMaster** va lancer le prog (map/reduce) sur le container alloué
- Le client peut tjs contacter le prog via le **AM** sans passer par le **resourceManager**



102

Ecosystème Hadoop

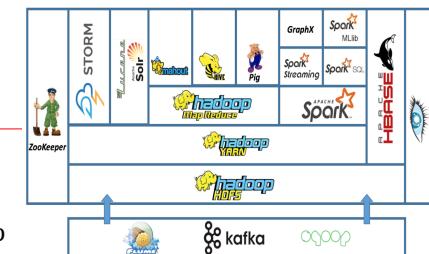
- HDFS: SF distribué d'Apache Hadoop
- YARN: gestionnaire de ressources et des tâches dans un cluster Hadoop
- MapReduce : framework pour le traitement de grands ensembles de données en parallèle
- Spark : un moteur d'analyse unifié pour le traitement de données à grande échelle
- PIG, HIVE : Services de traitement de données à l'aide de requêtes du type SQL
- HBase : système de gestion de base de données non-relationnelles distribué
- Mahout, Spark MLlib : des services pour faire du Machine Learning



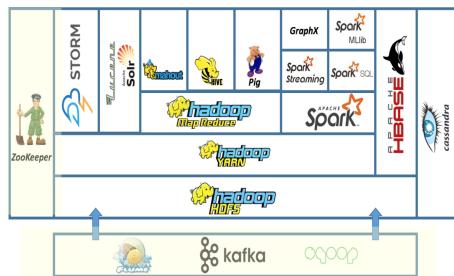
103

Ecosystème Hadoop

- Apache Drill : Moteur de requête SQL sur Hadoop
- Zookeeper : Pour la gestion de configuration des SD pour assurer la coordination entre les nœuds du Cluster
- Oozie : Système de planification de workflow pour gérer les jobs Hadoop
- Flume : service pour collecter, agréger et déplacer des fichiers logs
- Sqoop : Transfert de données entre Hadoop et des BDR.
- Solr & Lucene : Pour la recherche et l'indexation
- Ambari : Mise à disposition, monitoring et maintenance de cluster
- Apache Zeppelin : Web based notebook qui permet des analyses de données interactives basées sur les données et des documents collaboratifs avec SQL, Scala, Python, R et plus



104

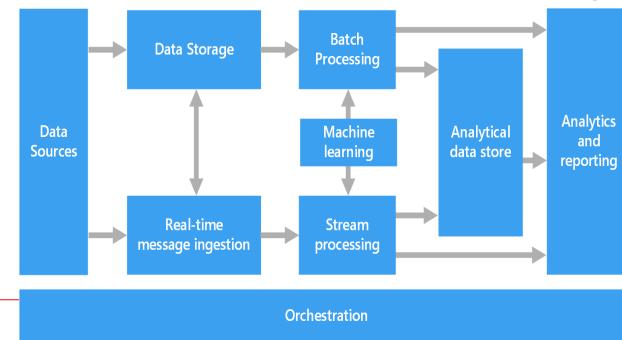


Analyse de flux en temps réel



105

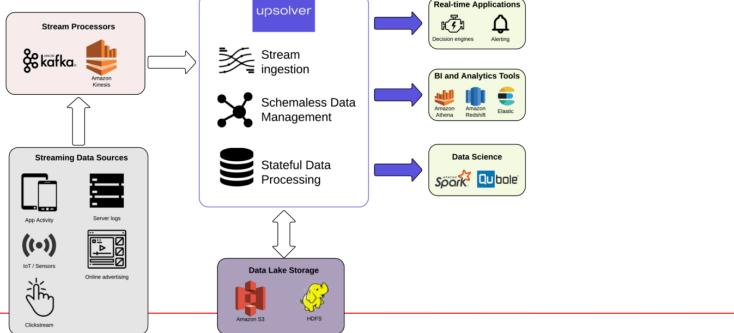
- ❖ **Big data** : Volume croissant de données générées en continu (réseaux sociaux, capteurs IoT, transactions financières, etc.).
- ❖ **Besoin en temps réel** : Prise de décisions instantanées grâce au traitement de flux de données.
- ❖ Le traitement des données en temps réel se définit comme la capacité à **ingérer**, **traiter** et **analyser** des données au fur et à mesure qu'elles sont générées, sans délai significatif



106

Architectures basées sur les flux

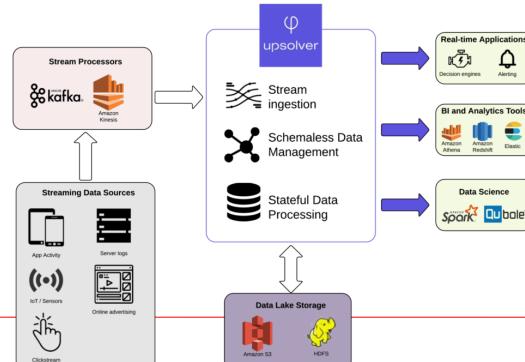
- ❖ **Principe** : Se concentrent sur le traitement des données en temps réel en tant que flux continus.
 - } Offre une grande flexibilité et permet de réagir rapidement aux changements dans les données.
- ❖ **Avantages** : Flexibilité, scalabilité et adaptabilité aux nouveaux types de données.
- ❖ **Limitations** : Complexité de la mise en œuvre et nécessité d'une expertise en streaming de données.



107

Architectures basées sur les flux

- ❖ **Cas d'utilisation** : Surveillance des performances du réseau informatique en temps réel.
 - } Apache Kafka ingère les données des capteurs réseau, Apache Storm les traite pour détecter les anomalies et les visualiser en temps réel.
 - } **Exemple** : Amazon utilise des architectures basées sur les flux de données pour surveiller ses infrastructures en temps réel.



108

Architectures temps réel

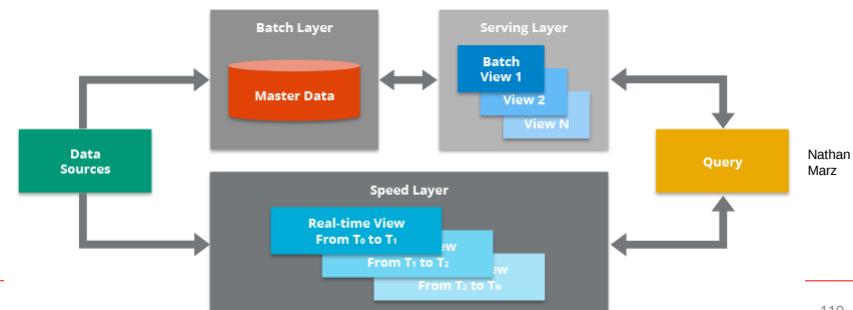
❖ Deux principales architectures :

- } Lambda Architecture
- } Kappa Architecture

109

Lambda Architecture

- ❖ **Principe** : Deux pipelines distinctes traitent les données en temps réel et en batch.
- } **Pipeline temps réel** offre une faible latence pour les analyses critiques,
- } **Pipeline batch** assure la cohérence et la complétude des données pour des analyses plus approfondies.
- ❖ **Avantages** : Flexibilité, scalabilité et capacité à gérer des volumes de données importants.
- ❖ **Limitations** : Complexité de la mise en œuvre et coûts de maintenance élevés.



110

Lambda Architecture

❖ **Cas d'utilisation** : Détection de fraude en temps réel dans les transactions financières.

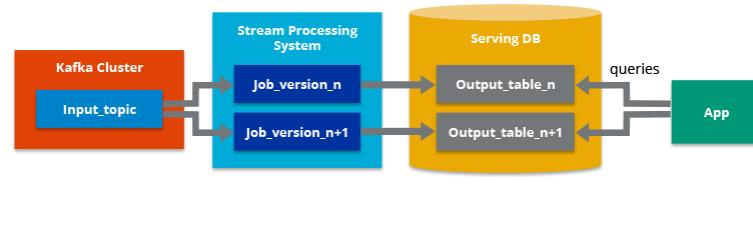
- } **Pipeline temps réel** : Apache Kafka ingère les transactions, Apache Spark les analyse pour détecter les anomalies.
- } **Pipeline batch** : Apache Hadoop Hive stocke et analyse les données historiques pour identifier les patterns de fraude.
- } **Exemple** : PayPal utilise une architecture Lambda pour détecter les fraudes en temps réel.



111

Kappa Architecture

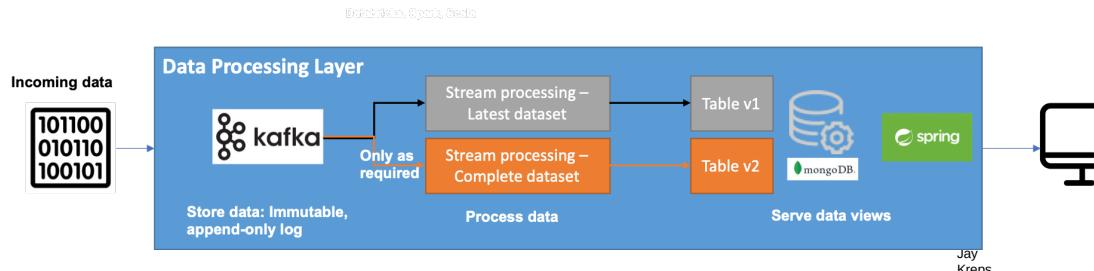
- ❖ **Principe** : Unification du traitement des données **en temps réel** et **en batch** en un seul **pipeline**.
- } Cette approche simplifie l'architecture et réduit les coûts de maintenance.
- ❖ **Avantages** : Simplicité, évolutivité et coûts réduits.
- ❖ **Limitations** : Latence plus élevée pour les analyses critiques et complexité du traitement des données historiques.



112

Kappa Architecture

- ❖ **Cas d'utilisation :** Analyse des clics en temps réel sur un site web e-commerce..
- ❖ **Pipeline unifiée :** Apache Flink ingère et traite les flux de clics en temps réel, Apache Pinot permet des analyses ad-hoc et des tableaux de bord.
- ❖ **Exemple :** Netflix utilise une architecture Kappa pour analyser les clics et les interactions des utilisateurs en temps réel.



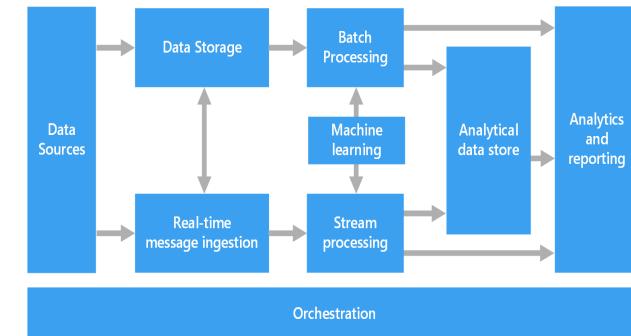
113

Système de messagerie (MOM)

- ❖ Responsable du transfert de données d'une application à une autre,
- ❖ basé sur le concept de file d'attente de messages fiable.
 - ▶ **messages asynchrone** entre les app clientes et le système de messagerie.
- ❖ Deux types de modèles de messagerie sont disponibles
 - ▶ **système point à point**
 - ▶ **Système publication-abonnement (pub-sub).**

MOM

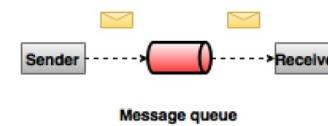
- ❖ Les architectures Kappa et Lambda reposent sur des **Message-oriented middleware**
- ❖ **Gérer des flux de données** à haut débit et en temps réel,
- ❖ Assurer un **découplage** entre les composants de l'architecture qui favorisent l'agilité, l'autonomie et l'évolutivité.



114

Système de messagerie point à point

- ❖ les messages sont conservés dans une file d'attente.
- ❖ Un ou plusieurs consommateurs peuvent consommer les messages de la file d'attente,
- ❖ un message particulier ne peut être consommé que par un seul consommateur au maximum.
- ❖ **Exemple :** un système de traitement des commandes,
 - ▶ chaque commande sera traitée par un processeur de commandes,
 - ▶ plusieurs processeurs de commandes peuvent également fonctionner en même temps.

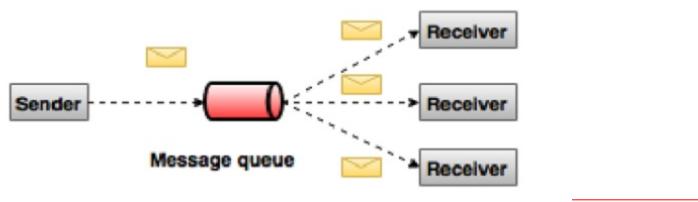


115

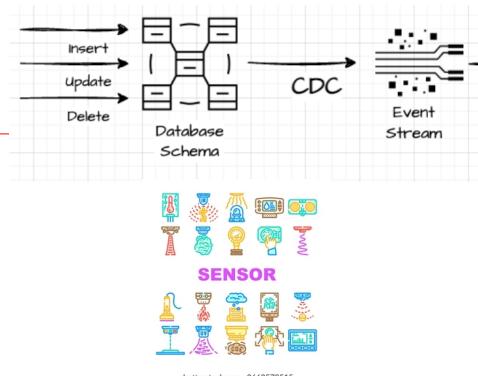
116

Système de messagerie pub/sub

- ❖ les messages sont conservés dans un sujet (topic).
- ❖ les consommateurs peuvent s'abonner à un ou plusieurs sujets et consommer tous les messages de ce sujet.
- ❖ **Exemple :** Dish TV,
 - ▶ publie différentes chaînes (sport, les films, la musique, etc.),
 - ▶ s'abonner à son propre ensemble de chaînes et les obtenir chaque fois que ses chaînes abonnées sont disponibles.



117



shutterstock.com - 266057815

Événement

- ❖ Tout type d'action, d'incident ou de changement identifié ou enregistré par une app.

■ Exemples :

- **E-commerce** : un paiement,
- **Social network app** : un clique, like,
- **IoT app** : un relevé de température.
- **DB**: insert, update, ...

- ❖ Une combinaison de notifications qui peut être utilisé pour déclencher une autre activité/état



9

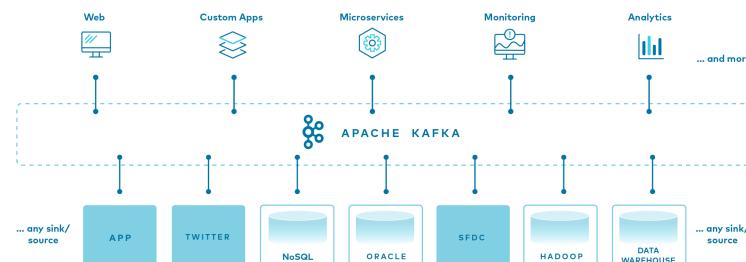
Apache kafka

- ❖ Système de messagerie tolérant aux pannes basé sur le pattern **publish/subscribe**
- ❖ Initié par LinkedIn
- ❖ Projet Apache open source en 2011.
- ❖ Kafka est écrit en Scala et Java..
- ❖ Présente plusieurs avantages :
 - **Fiable** : distribué, partitionné, répliqué et tolérant aux pannes.
 - **Évolutif** : évolue facilement sans temps d'arrêt.
 - **Performances** : maintient des performances stables même si de nombreux To de messages sont stockés.
- ❖ **Event based Architecture**

118

Qu'est-ce que la diffusion d'événements ?

- ❖ Concept du « toujours actif ».
- **Capturer** des données en temps réel ; sous la forme de **flux d'événements** ; à partir de sources d'événements (BDs, capteurs, appareils mobiles, services cloud etc) ;
- **Stocker** durablement les flux d'événements pour une récupération ultérieure ;
- **Manipuler, traiter** et réagir aux flux d'événements en temps réel ainsi que rétrospectivement ;
- Acheminer les flux d'événements vers différentes technologies de destination selon les besoins.

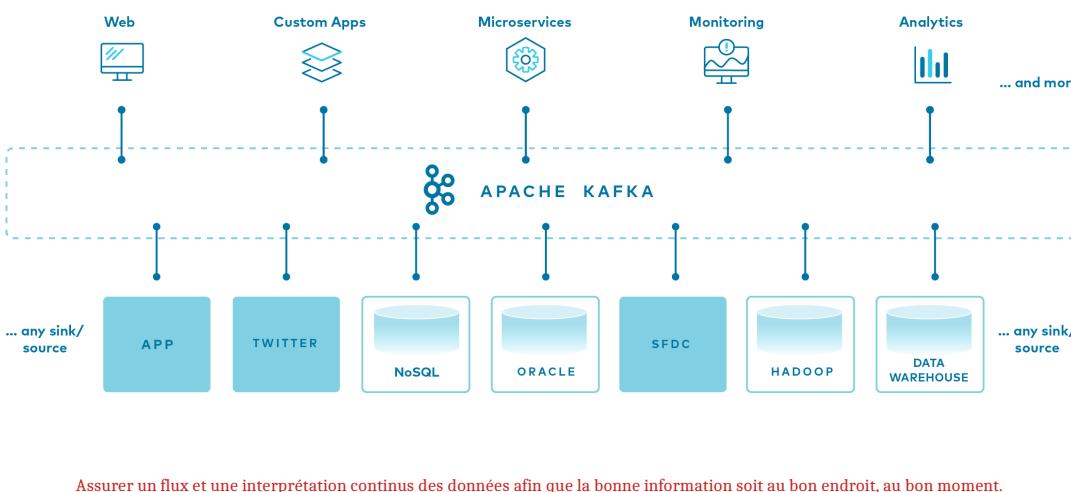


Assurer un flux et une interprétation continués des données afin que la bonne information soit au bon endroit, au bon moment.

120

Qu'est-ce que la diffusion d'événements ?

- ❖ Concept du « toujours actif ».



Assurer un flux et une interprétation continués des données afin que la bonne information soit au bon endroit, au bon moment.

121

KAFKA : APIs

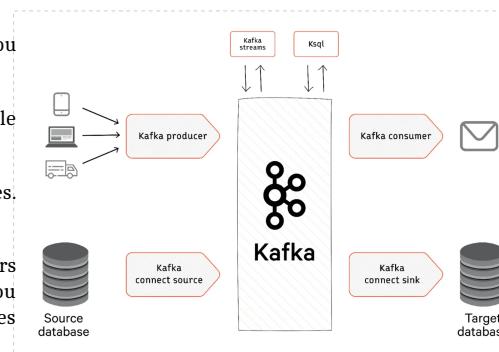
- ❖ API d'administration : gérer et inspecter les topics, les brokers et d'autres objets Kafka.

- ❖ API Producer : publier un flux d'événements dans un ou plusieurs sujets Kafka.

- ❖ API Consumer : s'abonner à un ou plusieurs sujets et traiter le flux d'événements qui leur sont produits.

- ❖ API Kafka Streams : traitement de flux et des microservices. (fenêtrages, agrégations, jointures, etc.).

- ❖ API Kafka Connect : créer et exécuter des connecteurs d'importation/exportation de données qui consomment ou produisent des flux d'événements depuis et vers des applications externes.



- Exemple un connecteur vers une BD telle que PostgreSQL peut capturer chaque modification apportée à un ensemble de tables.
- la communauté Kafka fournit des centaines de connecteurs prêts à l'emploi.

123

Pourquoi utiliser le streaming d'événements ?

- ❖ Grande variété de cas d'utilisation dans une pléthore d'industries et d'organisations :
- ❖ **Finance** : traiter les paiements et les transactions financières en temps réel.
- ❖ **logistique et industrie automobile** : suivre et surveiller les véhicules les flottes et les expéditions en temps réel.
- ❖ **Industrie**: capturer et analyser en continu les données des capteurs des appareils IoT ou d'autres équipements.
- ❖ **Hôtellerie et les voyages, et les applications mobiles** : collecter et réagir immédiatement aux interactions et aux commandes des clients.
- ❖ **Médecine** : Surveiller les patients hospitalisés et prévoir les changements d'état pour assurer un traitement rapide en cas d'urgence.

122

Kafka : Architecture -concepts fondamentaux

- ❖ Event record :

- ❖ Tout type d'action, d'incident ou de changement identifié ou enregistré par une app. (message, enregistrement)

- ❖ une clé/valeur, un horodatage et des en-têtes de métadonnées facultatifs.

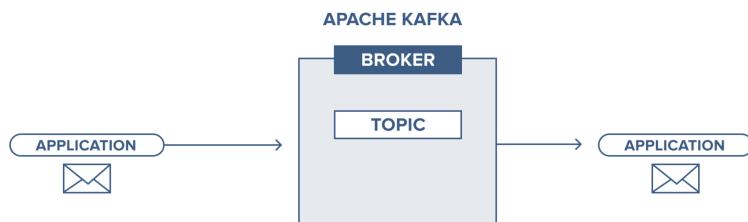
- ❖ Tableau d'octet qui peut contenir tout type d'information /format

- ❖ **Exemple** : `{"key" :"Alice",
"value":"A effectué un paiement de 200 $ à Bob",
timestamp : "lundi 7 novembre 2022 11:50:04 GMT+01:00"
}`

124

Kafka : Architecture -concepts fondamentaux

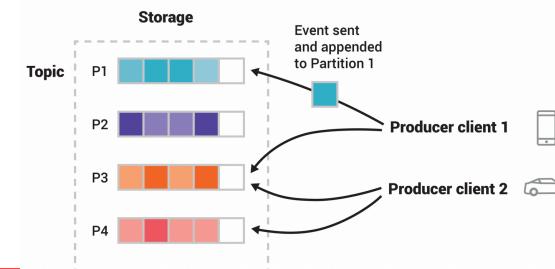
- ❖ **TOPIC:**
 - ❖ Unité d'organisation et de stockage des événements.
 - ❖ catégorie dans laquelle les enregistrements sont stockés et publiés.
 - ❖ filtrage des événements selon une sémantique commune.
 - ❖ Fichier log d'événements : les événements sont immuables



125

Kafka : Architecture -concepts fondamentaux

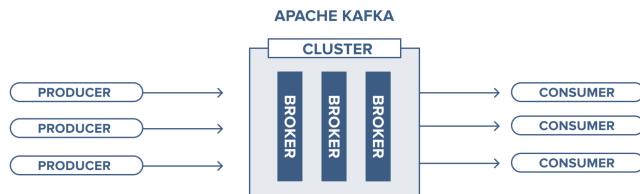
- ❖ Les topics Kafka sont divisés en un certain nombre de **partitions**
 - ❖ Lorsqu'un nouvel événement est publié dans un topic, il est ajouté à l'une de ses partitions.
- ❖ **Offset:** identifiant séquentiel unique d'un enregistrement dans une partition
 - ❖ Les événements avec la même clé sont écrits dans la même partition
- ❖ Paralléliser l'accès au enregistrements (R/W) d'un topic.



126

Kafka : Architecture -concepts fondamentaux

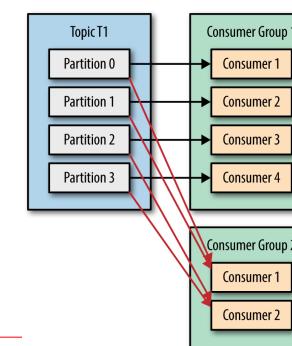
- ❖ **Broker :** serveur exécutant Kafka
- ❖ Un **cluster** Kafka se compose de plusieurs brokers
- ❖ **Producer:** envoi des enregistrements au broker (leader de partition).
- ❖ Le broker rattache tout nouveau enregistrement produit à la queue d'une partition



127

Kafka : Architecture -concepts fondamentaux

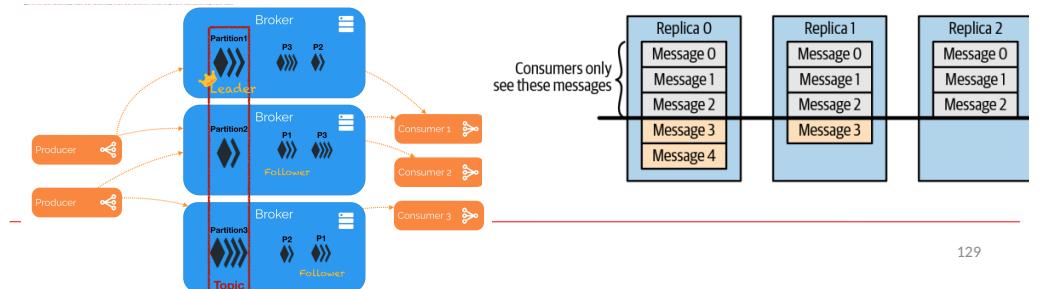
- ❖ **Consumer:** consomme des lots (**batches**) d'enregistrements du broker
- ❖ **Consumers** peuvent lire à partir de n'importe quel offset de leur choix.
- ❖ **Group consumer :** regrouper plusieurs consumers
 - ❖ dans un topic, chaque partition est consommée par un seul consommateur du groupe.,



128

Kafka : Architecture -concepts fondamentaux

- ❖ La **réPLICATION** : implémenter au niveau partition
- ❖ **Leader** : broker responsable des R/W sur une partition
- ❖ **Followers** : brokers contenant les répliques(**follower** replica) de la partition principale (**leader** replica)
- ❖ les demandes **W** sont acheminées au **leader** alors que les demandes **R** peuvent être envoyées au leader/followers
- ❖ Les partitions **followers** consomment les enregistrements du leader comme de simples consommateurs kafka



129

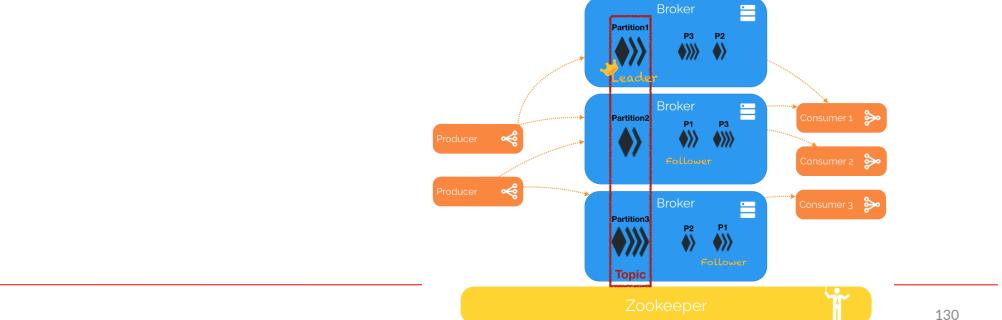
API Kafka Producer

- ❖ La classe **KafkaProducer** fournit une option pour connecter un broker:
- ❖ fournit une méthode d'envoi pour envoyer des enregistrements de manière asynchrone à un topic :
- ❖ `producer.send(new ProducerRecord<byte[],byte[]>(topic, partition, key1, value1) , callback);`
- ❖ **ProducerRecord** : gérer un tampon d'enregistrements en attente d'envoi.
- ❖ **Callback** : Un rappel fourni par l'utilisateur à exécuter lorsque l'enregistrement a été reconnu par le serveur (null indique aucun rappel).
- ❖ **public void flush()** : s'assurer que tout les messages envoyés précédemment sont complétés
- ❖ **public Map metrics()** : récupérer les métadatas des partitions

131

Kafka et zookeeper

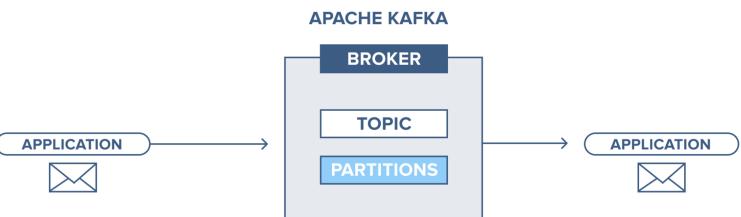
- ❖ ZooKeeper est un service centralisé
- ❖ maintenir les informations de configuration et de nom
- ❖ fournir une synchronisation distribuée et des services de groupe.
- ❖ Zookeeper est utilisé pour gérer et coordonner les brokers Kafka.
- ❖ Il permet de notifier les producteurs/consummateurs de la présence/échec d'un broker dans le cluster.



130

API ProducerRecord

- ❖ paire clé/valeur qui sera envoyée au cluster Kafka.
- ❖ Créer un enregistrement avec des paires partition, clé et valeur
- ❖ `public ProducerRecord<string, string>(string topic, int partition, k key, v value)`
- ❖ `public ProducerRecord<string, string>(string topic, k key, v value)`
- ❖ `public ProducerRecord<string, v value>(string topic, v value)`



132

EventProducer

```
import java.util.Properties;
import org.apache.kafka.clients.producer.*;

public class EventProducer {

    public static void main(String[] args) throws Exception{
        if(args.length == 0){
            System.out.println("Enter topic name");
            return;
        }

        String topicName = args[0].toString();
        // create instance for properties to access producer configs
        Properties props = new Properties();

        props.put("bootstrap.servers", "localhost:9092"); //Assign localhost id
        props.put("acks", "all"); //Set acknowledgements for producer requests.
        props.put("retries", 0); //If the request fails, the producer can automatically retry,
        props.put("batch.size", 16384); //Specify buffer size in config
        props.put("linger.ms", 1); //Reduce the no of requests less than 0
        props.put("buffer.memory", 33554432); //amount of memory available to the producer for buf.
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

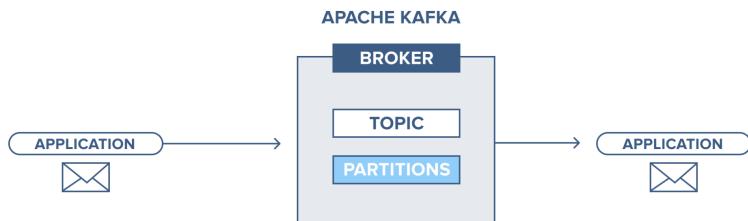
        Producer<String, String> producer = new KafkaProducer <String, String>(props);

        for(int i = 0; i < 10; i++)
            producer.send(new ProducerRecord<String, String>(topicName, Integer.toString(i),
                Integer.toString(i)));
        System.out.println("Message sent successfully");
        producer.close();
    }
}
```

133

API ConsumerRecord

- ❖ paire clé/valeur qui sera reçue au cluster Kafka.
- ❖ Créer un enregistrement consumer avec des paires partition, clé et valeur
- ❖ Agit comme un conteneur pour ConsumerRecord.
- ❖ **public ConsumerRecord(string topic,int partition, long offset,K key, V value)**
- ❖ **public ConsumerRecords(java.util.Map<TopicPartition,java.util.List <ConsumerRecord<K,V>>> records)**
- ❖ Permet de conserver la liste des ConsumerRecord par partition pour un sujet particulier



135

API Kafka Consumer

- ❖ La classe **KafkaConsumer** fournit une option pour connecter un broker:
- ❖ fournit des méthodes suivantes :
- ❖ **public java.util.Set<TopicPartition> assignment()** : obtenir l'ensemble des partitions actuellement attribuées par le consommateur.
- ❖ **public void subscribe(jList<String> topics, ConsumerRebalanceListener listener)** : s'abonner à la liste de sujets donnée pour obtenir des partitions signées dynamiquement.
- ❖ **public void unsubscribe()** : se Désabonner des sujets de la liste de partitions donnée.
- ❖ **public void seek (TopicPartition partition, long offset)** : Récupère la valeur de décalage actuelle que le consommateur utilisera lors de la prochaine méthode poll().

134

EventConsumer

```
public class EventConsumer {
    public static void main(String[] args) throws Exception {
        //Kafka consumer configuration settings
        String topicName = args[0].toString();
        Properties props = new Properties();

        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "test");
        props.put("enable.auto.commit", "true");
        props.put("auto.commit.interval.ms", "1000");
        props.put("session.timeout.ms", "30000");
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);

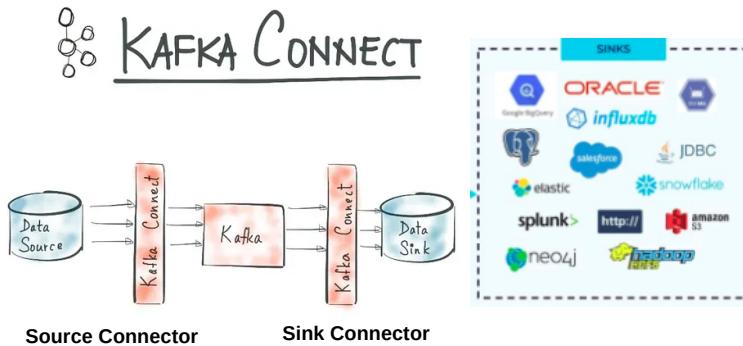
        //Kafka Consumer subscribes list of topics here.
        consumer.subscribe(Arrays.asList(topicName))

        //print the topic name
        System.out.println("Subscribed to topic " + topicName);
        int i = 0;

        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(100);
            for (ConsumerRecord<String, String> record : records)
                // print the offset,key and value for the consumer records.
                System.out.printf("offset = %d, key = %s, value = %s\n", record.offset(), record.key(),
                    record.value());
        }
    }
}
```

Kafka connect

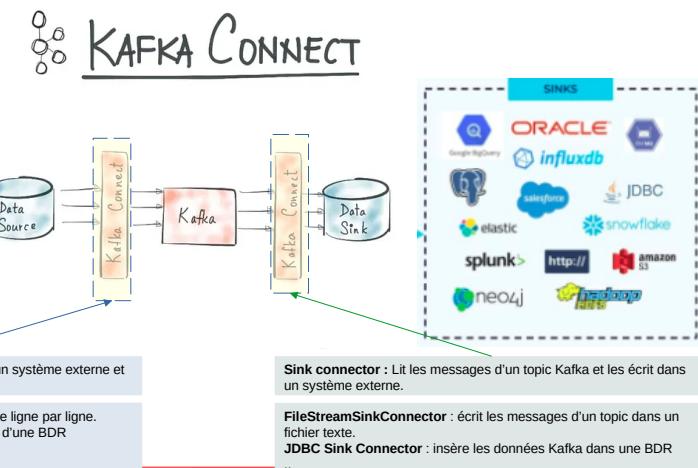
- Composant Kafka pour l'intégration en streaming entre Kafka et d'autres systèmes externes (BDs, services cloud, FS, KV stores).
- Exécute des **connecteurs**, qui implémentent la logique personnalisée pour interagir avec un système externe.



137

Kafka connect

- Deux types de **connecteurs** : source & sink

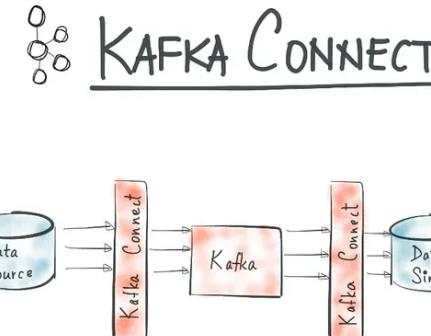


https://docs.confluent.io/platform/current/connect/kafka_connectors.html

139

Kafka connect (fonctionnalités)

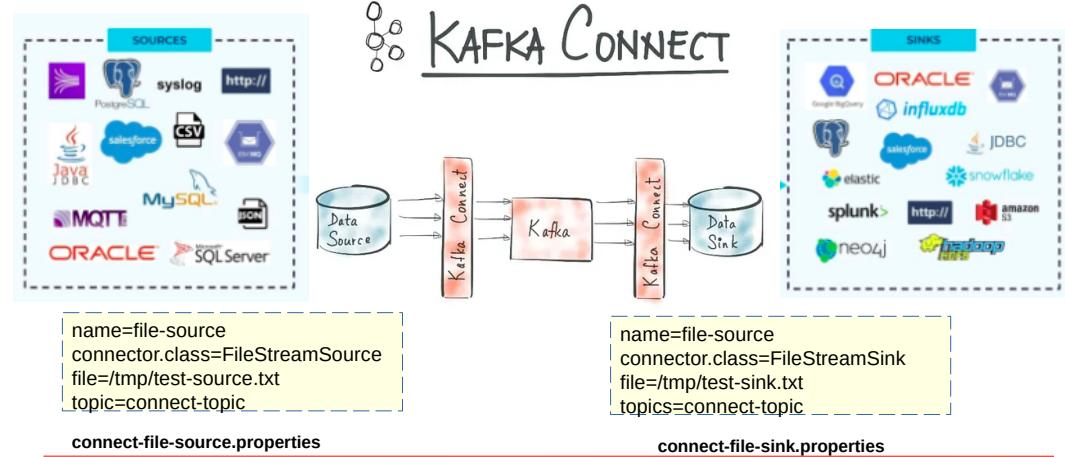
- Framework commun pour les connecteurs kafka :
 - **API (java)**: simplifier le développement, le déploiement et la gestion des connecteurs.
 - **Moteur d'exécution** : distributed / standalone
 - **Interface REST** : soumettre et gérer les connecteurs.



138

Kafka connect

- Les connecteurs sont plug-and-play : il suffit de configurer un fichier **.properties**

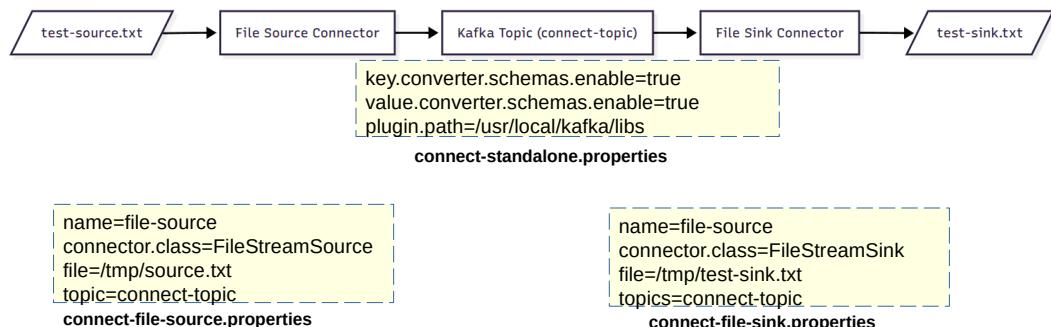


140

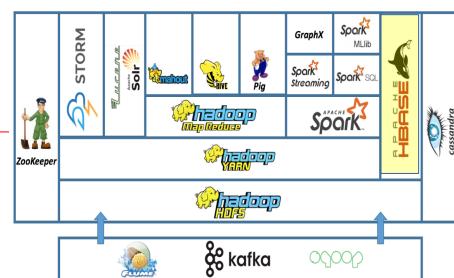
Kafka connect (exemple)

```
connect-standalone.sh connect-standalone.properties connect-file-source.properties
```

```
connect-standalone.sh connect-standalone.properties connect-file-sink.properties
```



141



BASES DE DONNÉES NoSQL

Orientées colonnes



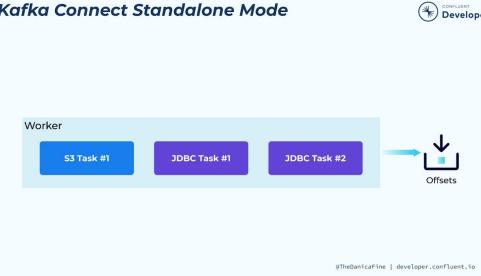
143

Kafka connect (Mode)

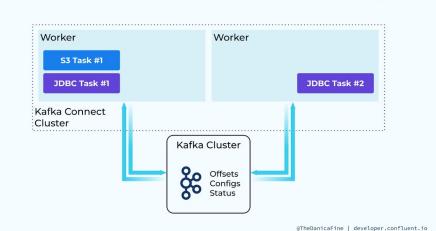
• Mode d'exécution

- **standalone**: utile pour le développement et les tests de Kafka Connect sur une machine locale.
- **Distributed** : exécute les nœuds Connect sur plusieurs machines (nœuds), formant ainsi un cluster Connect.

Kafka Connect Standalone Mode



Kafka Connect Scalability



142

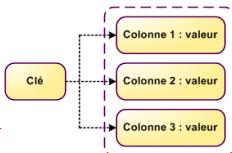
HBase



- ❖ SGBD distribué, non-relationnel et orienté colonnes,
- ❖ Open source
- ❖ développé au-dessus du système de fichier HDFS.
- ❖ permet un accès aléatoire en écriture/lecture en temps réel à un très grand ensemble de données.

144

BDD orientée colonnes



« Orientée ligne »

Id	Nom	Prénom
1	Brico	Juda
2	Diote	Kelly

« Orientée Colonne »

Ligne « row » Colonne « column » Valeur « value »

1	Nom	Brico
1	Prénom	Juda
2	Nom	Diote
2	Prénom	Kelly

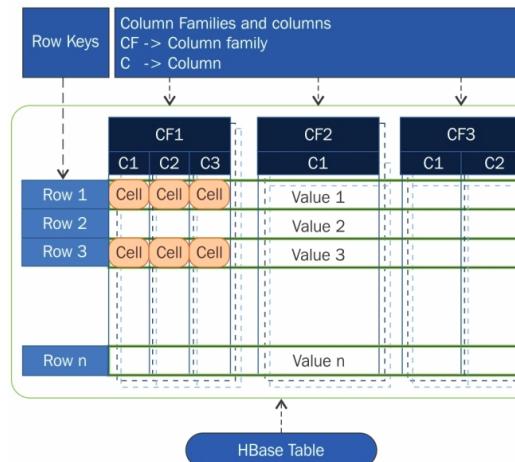
Exemple : Hbase (Hadoop), Cassandra (Facebook, Twitter), BigTable (Google)

145

HBase : Modèle de données

Le modèle se base sur six concepts :

- ❖ **Table** : les données sont organisées dans des tables.
- ❖ **Row** : dans chaque table, les données sont organisées dans des lignes identifiées par une clé unique (traitée comme un tableau d'octets).
- ❖ **Column Family** : Les données au sein d'une ligne sont regroupées par column family.
 - ↳ Chaque ligne de la table a les mêmes column families, qui peuvent être peuplées ou pas.
 - ↳ Les column families sont définies à la création de la table dans HBase.

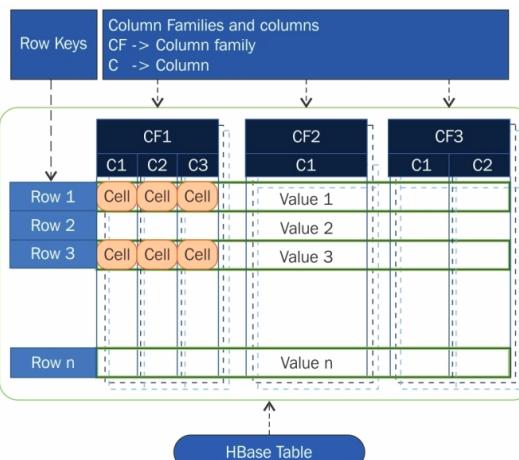


146

HBase : Modèle de données (suite)

Le modèle se base sur six concepts, qui sont :

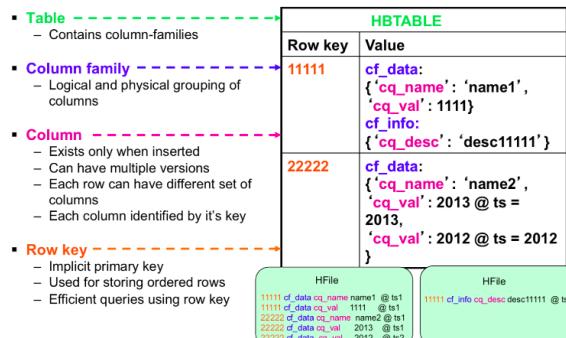
- ❖ **Column qualifier** : L'accès aux données au sein d'une column family se fait via le column qualifier (column).
 - ↳ n'est pas spécifié à la création de la table mais plutôt à l'insertion de la donnée.
 - ↳ Comme les rowkeys, le column qualifier n'est pas typé, il est traité comme un tableau d'octets.
- ❖ **Cell** : La combinaison du RowKey, de la Column Family ainsi que la Column qualifier identifie d'une manière unique une cellule.
 - ↳ Les données stockées dans une cellule sont appelées les valeurs de cette cellule.
 - ↳ Les valeurs n'ont pas de type, ils sont toujours considérés comme tableaux d'octets.
- ❖ **Version** : Les valeurs au sein d'une cellule sont versionnées.
 - ↳ Les versions sont identifiées par leur timestamp (de type long).
 - ↳ Le nombre de versions est configuré via la Column Family.
 - ↳ Par défaut, ce nombre est égal à trois.



147

HBase : caractéristiques

- ❖ Les données dans HBase sont stockées sous forme de HFiles, par colonnes, dans HDFS.
- ❖ Chaque HFile se charge de stocker des données correspondantes à une column family particulière.



148

HBase : caractéristiques

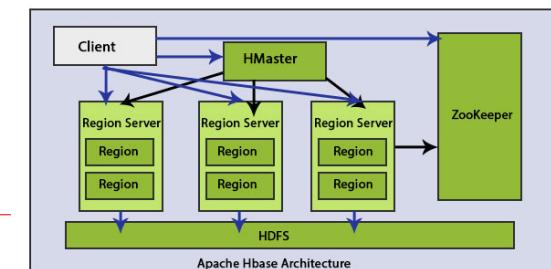
- ❖ HBase n'a pas de schéma prédéfini,
- ❖ Il faut **définir les familles de colonnes** à la création des tables, car elles représentent l'organisation physique des données
- ❖ HBase est décrite comme étant un magasin de données clef/valeur, où la clef est la combinaison (row-column family-column-timestamp) représente la clef, et la cell représente la valeur.

Row Key	customer		sales	
ROW_ID	name	city	product	amount
101	John White	Los Angeles, CA	Chairs	\$400.00
102	Jane Brown	Atlanta, GA	Lamps	\$200.00
103	Bill Green	Pittsburgh, PA	Desk	\$500.00
104	Jack Black	St. Louis, MO	Bed	\$1,600.00

149

HBase : Architecture

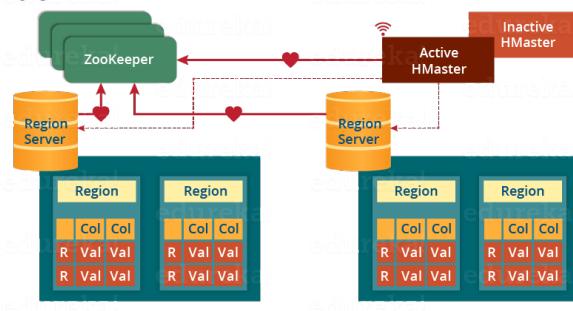
- ❖ Physiquement, HBase est composé de trois types de serveurs de type Master/Slave.
 - } **Region Servers:** permettent de fournir les données pour lectures et écritures. Pour accéder aux données, les clients communiquent avec les RegionServers directement.
 - } **HBase HMaster :** gère l'affectation des régions, les opérations de création et suppression de tables.
 - } **Zookeeper:** permet de maintenir le cluster en état



150

HBase : Architecture

- ❖ Le **DataNode** de Hadoop permet de stocker les données que le **Region Server** gère.
- ❖ Toutes les données de HBase sont stockées dans des fichiers HDFS. Les **RegionServers** sont **colocalisés** avec les **DataNodes**.
- ❖ Le **NameNode** permet de maintenir les métadonnées sur tous les blocs physiques qui forment les fichiers.

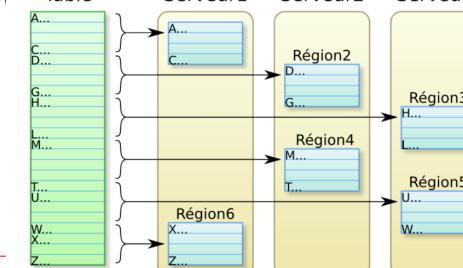


151

HBase : Region Server

- ❖ Les tables HBase sont divisées horizontalement, par **row** en plusieurs **Regions**.
- ❖ Une **region** contient toutes les lignes de la table comprises entre deux clefs données (contigües).
- ❖ Les **regions** sont affectées à des noeuds dans le cluster, appelés **Region Servers**,

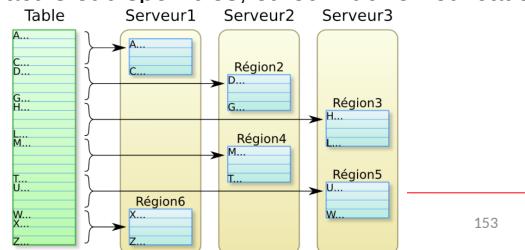
- } permettent de servir les données pour la lecture et l'écriture.
- } Un **region server** peut contenir plusieurs **regions**.



152

HBase : Master

- ❖ Le **HBase Master** est responsable de :
 - ↳ coordonner les region servers en assignant les régions au démarrage,
 - ↳ les réassignant en cas de récupération ou d'équilibrage de charge,
 - ↳ monitorer des instances des region servers dans le cluster.
 - ↳ fournir une interface pour la création, la suppression et la modification des tables.
- ❖ HBase utilise **Zookeeper** comme service de coordination pour maintenir l'état du serveur dans le cluster.
- ❖ **Zookeeper** sait quels serveurs sont actifs et disponibles, et fournit une notification en cas d'échec d'un serveur.



Shell de HBase

- ❖ HBase offre plusieurs mécanismes pour travailler, dont :
 - ↳ Un **shell** où on tape des commandes,
 - ↳ Une **API** à utiliser dans des programmes **Java**
 - ↳ Une **API Python** appelée **HappyBase**.
- ❖ Il y a aussi une page Web dynamique qui affiche l'état du service et permet de voir les tables.
- ❖ On va d'abord voir le shell HBase. On le lance en tapant :

hbase shell

- ❖ Il faut savoir que c'est le langage Ruby qui sert de shell.
- ❖ Les commandes sont écrites dans la syntaxe de ce langage.

Exemple

Enregistrer les coordonnées et les achats de clients.

- ❖ construire une table contenant trois familles :
 - ↳ La **famille personne** contiendra les informations de base :
 - colonnes **personne:nom** et **personne:prenom**
 - ↳ La **famille coords** contiendra l'adresse :
 - colonnes **coords:rue**, **coords:ville**, **coords:cp**, **coords:pays**
 - ↳ La **famille achats** contiendra les achats effectués :
 - colonnes **achats:date**, **achats:montant**, **achats:idfacture**
- ❖ HBase autorise à dé-normaliser un schéma (redondance dans les informations) afin d'accéder aux données plus rapidement.

154

Commandes HBase de base

- ❖ Voici les premières commandes :
 - ↳ **status** affiche l'état du service HBase
 - ↳ **version** affiche la version de HBase
 - ↳ **list** affiche les noms des tables existantes
 - ↳ **describe 'table'** décrit la table dont on donne le nom.
- ❖ Attention à bien mettre les noms de tables, de familles et de colonnes entre '...'
- ❖ Les commandes suivantes sont les opérations CRUD : créer, lire, modifier, supprimer.

156

Création d'une table

❖ Deux syntaxes :

```
{ create 'NOMTABLE', 'FAMILLE1', 'FAMILLE2'...
{ create 'NOMTABLE', {NAME=>'FAMILLE1'}, {NAME=>'FAMILLE2'}...
```

❖ La seconde syntaxe est celle de **Ruby**.

```
{ On spécifie les familles par un dictionnaire {propriété=>valeur}.
{ D'autres propriétés sont possibles :
  • VERSIONS : indiquer le nombre de versions à garder.
```

❖ Remarques :

```
{ Les familles doivent être définies lors de la création. (coûteux de créer une famille ultérieurement).
{ On ne définit que les noms des familles, pas les colonnes.
{ Les colonnes sont créées dynamiquement.
```

157

Ajout/suppression

❖ Ajout de cellules

```
❖ Un n-uplet est composé de plusieurs colonnes.  
❖ L'insertion d'un n-uplet se fait colonne par colonne.  
❖ On indique la famille de la colonne.  
❖ Les colonnes peuvent être créées à volonté.
```

```
put 'NOMTABLE', 'CLE', 'FAM:COLONNE', 'VALEUR'
```

❖ Suppression de cellules

```
{ Il y a plusieurs variantes selon ce qu'on veut supprimer, (une cellule, ou tout un n-uplet) :
```

```
deleteall 'NOMTABLE', 'CLE', 'FAM:COLONNE', TIMESTAMP
deleteall 'NOMTABLE', 'CLE', 'FAM:COLONNE'
deleteall 'NOMTABLE', 'CLE'
```

159

Destruction d'une table

❖ Opération en deux temps :

```
{ désactiver la table d'abord, puis la supprimer :
```

1. **disable** 'NOMTABLE'

2. **drop** 'NOMTABLE'

❖ Désactiver la table permet de bloquer toutes les requêtes.

158

Affichage de n-uplets

❖ Commande **get** affiche les valeurs désignées par une seule clé.

❖ Possibilité de spécifier le nom de la colonne avec sa famille (éventuellement le timestamp)

```
get 'NOMTABLE', 'CLE'
get 'NOMTABLE', 'CLE', 'FAM:COLONNE'
get 'NOMTABLE', 'CLE', 'FAM:COLONNE', TIMESTAMP
```

❖ La première variante affiche toutes les colonnes ayant cette clé.

❖ La deuxième affiche toutes les valeurs avec leur timestamp.

160

Recherche de n-uplets

- ❖ Commande **scan** affiche les n-uplets sélectionnés par les conditions.

 } Ecrire les conditions en Ruby.

```
scan 'NOMTABLE', {CONDITIONS}
```

- ❖ Parmi les conditions possibles :

 } pour sélectionner certaines colonnes.

```
COLUMNS=>[FAM:COLONNE,...]
```

 } pour sélectionner les n-uplets de [CLE1, CLE2]

```
STARTROW=>'CLE1', STOPROW=>'CLE2' .
```

- ❖ Ou alors (exclusif), une condition basée sur un filtre :

```
FILTER=>"PrefixFilter('binary:client')"
```

- ❖ Il existe de nombreux filtres,

161

Filtres

- ❖ L'ensemble des filtres d'un scan doit être placé entre "...".

- ❖ Plusieurs filtres peuvent être combinés avec AND, OR et les parenthèses.

Exemple

```
{ FILTER =>  
"PrefixFilter('client') AND ColumnPrefixFilter('achat')" }
```

- ❖ PrefixFilter('chaîne') : accepte les valeurs dont la clé commence par la chaîne

- ❖ ColumnPrefixFilter('chaîne') : accepte les valeurs dont la colonne commence par la chaîne.

162

Filtres

- ❖ plusieurs filtres qui comparent quelque chose à une valeur constante

Syntaxe

```
XXXXFilter(operator,value)
```

- ❖ **XXXXFILTER** : RowFilter, FamilyFilter, QualifierFilter

- ❖ **Operator** <, <=, =, !=, > ou >= (sans quote)

- ❖ **Value** :

 } 'binary:chaine' pour une chaîne telle quelle

 } 'substring:chaine' pour une sous-chaîne

 } 'regexpstring:motif' pour un motif egrep

```
{ FILTER => "ValueFilter(=,'substring:bigdata')"
```

163

HBase

- ❖ **SingleColumnValueFilter(fam','col',operator,Value)** garde les n-uplets dont la colonne 'fam:col' correspond à la constante

- ❖ Compter les n-uplets d'une table, en configurant le cache pour en prendre 1000 à la fois

```
count 'NOMTABLE', CACHE => 1000
```

- ❖ HBase n'est qu'un stockage de mégadonnées.

 } pas de dispositif d'interrogations sophistiqué (pas de requêtes imbriquées, d'agrégation, etc.)

- ❖ Pour des requêtes SQL sophistiquées, il faut faire appel à Hive.

 } Hive est un SGBD qui offre un langage ressemblant à SQL et qui s'appuie sur HBase.

164

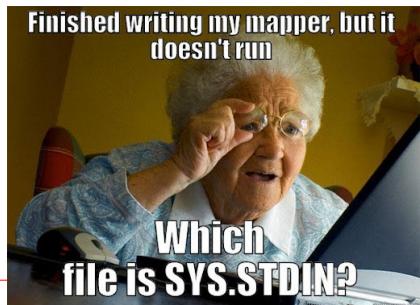
HBase : API Java

- ❖ Ecrire des programmes Java qui accèdent aux tables HBase.
- ❖ Classes et de méthodes de bases pour :
 - } créer une table
 - } ajouter des cellules
 - } récupérer une cellule
 - } parcourir les cellules
- ❖ NB: spécifications en évolution

165

Programmation MapReduce : Un Défi Complexe !

- **Faible Abstraction** : Écrire des jobs MapReduce nécessite une compréhension approfondie des concepts bas niveau.
- **Code Verbeux** : Une simple opération peut nécessiter plusieurs lignes de code Java.
- Perte de temps et risque d'erreurs élevé



167

Big Data Query Processing

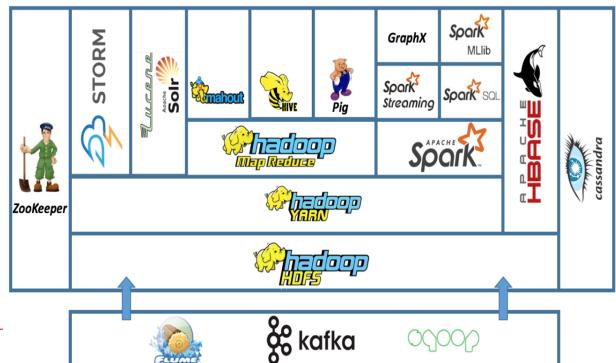
Apache
Pig

166



Apache Pig

- Plateforme simplifiée pour l'analyse de données massifs sur des cluster Hadoop.
- Utilise un langage de script « Pig Latin ».
- Fournit un haut niveau d'abstraction pour le traitement via MapReduce.



168

Philosophie d'apache Pig



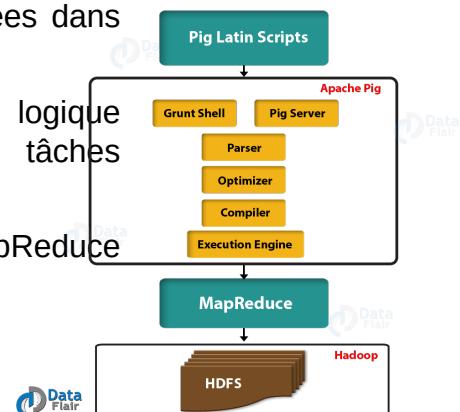
- **Pigs Eat Anything**
 - ✓ Traiter tous types de données (structuré, semi-structuré et non structuré).
 - ✓ Lire les données issues de multiples sources comme HBase, Cassandra, Hive.
 - ✓ Support plusieurs format : texte, sequence, ORC et parquet.
- **Pigs live Everywhere**
 - ✓ Implémenté en premier sur Hadoop.
 - ✓ Traite aussi les données du système de fichiers local.
- **Pigs Are Domestic Animals**
 - ✓ Simple et facile à utiliser
- **Pigs fly**
 - ✓ Traiter les données de façon rapide via un optimiseur interne.

169

Architecture Apache Pig



- **Parser** : vérifier la syntaxe du script.
- **Optimiser** : garantit que les données dans le pipeline doivent être minimales
- **Compiler**: compile ensuite le plan logique optimisé en une série de tâches MapReduce.
- **Execution engine** : les tâches MapReduce sont soumises à Hadoop.



171

Apache Pig : caractéristiques



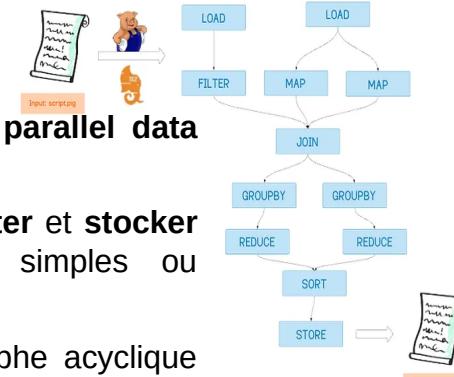
- **Facile à utiliser :**
 - ✓ Pig latin comprend des opérateurs pour de nombreuses opérations de données traditionnelles (join, sort, filter, etc.)
 - ✓ Les scripts Pig sont convertis en interne en tâches Map Reduce et sont exécutés sur les données stockées dans HDFS
- **Optimisé :**
 - ✓ optimiser automatiquement le plan d'exécution, permettant ainsi à l'utilisateur de se concentrer sur la sémantique plutôt que sur l'efficacité.
- **Extensible :**
 - ✓ Ecrire des fonctions UDF propres aux utilisateurs en plusieurs langages de programmation (java, python, etc.).

170

Apache Pig



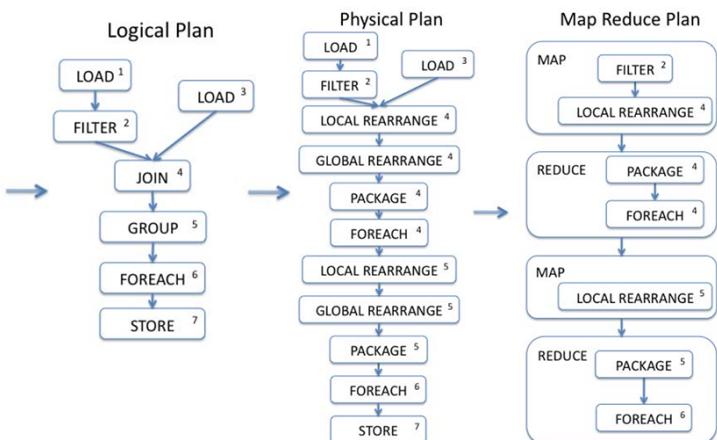
- Le langage pig latin est qualifié de « **parallel data flow language** ».
- Il permet de décrire comment **lire, traiter et stocker** des données à travers des flux simples ou complexes.
- Un script Pig Latin représente un graphe acyclique dirigé (**DAG**), où les flux de données sont les arêtes et les opérateurs de traitement sont les nœuds.



172



```
Pig Latin
A = LOAD 'file1' AS (x, y, z);
B = LOAD 'file2' AS (t, u, v);
C = FILTER A by y > 0;
D = JOIN C BY x, B BY u;
E = GROUP D BY z;
F = FOREACH E GENERATE
    group, COUNT(D);
STORE F INTO 'output';
```

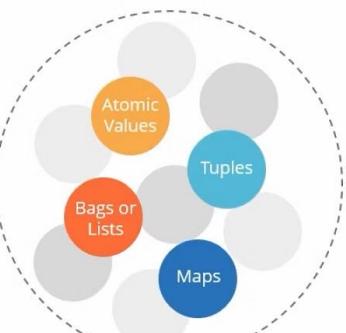


173

Modèle de données



- Atom:** donnée élémentaire qui peut être de plusieurs types (un champ)
 - ✓ Ex : Ahmadi
- Tuple :** Ensemble ordonné de champs (similaire à une ligne dans une table dans une RDB)
 - ✓ Ex : (ahmadi,ahmed,20-09-1995)
- Bag:** Collection de tuples qui peuvent avoir n'importe quel nombre de champs (schéma flexible).
 - ✓ Ex : '{ (ahmadi,ahmed,20-09-1995),(khaldi,khalid,31-03-1980)}'
- Map:** série de paires clé-valeur.
 - ✓ La clé doit être unique et de type chararray. La valeur peut être de n'importe quel type.
 - ✓ Un Map est représenté par '[]' où la clé et la valeur sont séparées par '#'
 - ✓ Ex : [nom#Ali, âge#10]
 - ✓ Ex de tuple complexe: (3 , [nom#Faridi, age#23] , { (math, 12.5), (info, 15) })



ahmadi	ahmed	20-09-1995
Khaldi	Khalid	31-03-1980

175

Type de données

- Pig latin fournit un grand nombre de type de données simples (**ATOM**)

Type	Description	Example
int	4-byte	100
long	8-byte	100L or 100l
float	4-byte	100.1f or 100.1F
double	8-byte	100.1e2
biginteger	Java biginteger	100000000000
decimal	Java decimal	100.0000000001
boolean	True/false	true/false
chararray	UTF-8 string	Big data is everywhere
bytearray	Binary data	Binary data
datetime	Date and time	2015:12:07T10:10:20.001+00.00

174

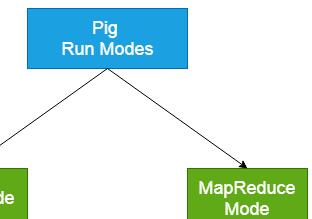
Modes d'exécution



Mode local

- ✓ s'exécute dans une seule JVM
- ✓ utilisé pour les expérimentations de développement et le prototypage.
- ✓ fonctionne sur un système de fichiers local, (input et output)

```
root@hadoop-master:~# pig -x local
```



Mode MapReduce (default mode)

- ✓ Pig convertit le script en tâches MapReduce et les exécute sur le cluster hadoop.

```
root@hadoop-master:~# pig -x mapreduce
```

176

Mécanismes d'exécution



▪ Les scripts pig peuvent s'exécuter :

- ✓ **Mode interactif (shell Grunt)** – saisir les instructions latines Pig et obtenir la sortie (en utilisant l'opérateur Dump).
- ✓ **Mode par batch (script)** – écrire le script latin Pig dans un seul fichier avec l'extension **.pig**.
- ✓ **Mode intégré (UDF)** – définir des fonctions propres à l'utilisateur dans des langages de programmation tels que Java, et de les utiliser dans notre script.

177

Pig Latin : Chargement



■ alias = LOAD 'Source' [USING function] [AS schema]

- **Source:** nom de fichier ou répertoire
- **function:** nom d'une fonction (classe Java) de chargement des données. Le résultat est un type Pig (tuples / bag / map).
 - **Exemples** PigStorage("Séparateur"), TextLoader , JsonLoader, ...
- **schema:** utilisé pour donner des noms et des types explicites aux différents membres du bag des données d'entrée chargées.
 - **Syntaxe (Champ1 : Type1, Champ2 : Type2 , ...)**
 - **Exemple :** artists = LOAD '/user/root/artists-pig.json' USING JsonLoader('id: chararray, firstName:chararray, lastName:chararray, birth:chararray')

Pour accéder aux membres du bag: artists.\$0 et artists.\$3 désignent respectivement artists.id et artists.birth

179

Structure d'un script Pig Latin



■ Généralement, un script pig latin est constitué de 3 phases :

- 1) **Load** – pour charger des données à partir de HDFS en leur donnant un schéma
- 2) **Transformation** Opérateurs relationnels: FILTER, ORDER, DISTINCT, JOIN, GROUP, FOREACH, UNION, etc
- 3) **DUMP ou STORE** Afficher le résultat sur l'écran ou le stocker dans un fichier.

■ Commentaires:

- ✓ /* commentaire sur plusieurs lignes */
- ✓ -- commentaire en une ligne

- Le résultat de chaque instruction Pig est une collection de n-uplets : **relation (bag)**.

178

Pig Latin : Transformation



- Chaque instruction Pig prend une relation en entrée et produit une nouvelle relation en sortie.

✓ **alias = INSTRUCTION entree PARAMETRES ...;**

- Pig permet soit d'enchaîner les instructions par le mécanisme des alias, soit par un appel imbriqué.
alias4=LIMIT(ORDER(FILTER(LOAD...)) ...)....;

■ alias1=LOAD...;

■ alias2=FILTER alias1 ...;

■ alias3=FILTER alias1 ...;

■ alias4=JOIN alias1 ..., alias3;

180



Pig Latin : Affichage/ Sauvegarde



■ DESCRIBE relation ;

- ✓ Affiche la description d'une relation,
- ✓ **Exemple :** artists = LOAD '/user/root/artists-pig.json' USING JsonLoader('id: chararray, firstName:chararray,lastName:chararray, birth:chararray');
artists: {id: chararray,firstName: chararray,lastName: chararray,birth: chararray}

■ Dump (sample) relation ;

- ✓ Lance le calcul MapReduce de la relation et affiche les résultats à l'écran. C'est seulement à ce stade que la relation est calculée
- ✓ **Exemple :** DUMP artists ;

```
(artist:2313238,Nicole,Sougou,None)
(artist:2313239,Amadou,Mbow,1993)
(artist:2326723,Oona,Roche,None)
(artist:2400303,Jean-Michel,Lengali,None)
```

■ STORE STORE alias INTO 'repertoire' [USING fonction];

- ✓ sauvegarde les données sur le système de fichier (HDFS ou local selon le mode d'utilisation de Pig):
- ✓ **Exemple :** STORE artists INTO '/user/root/pigout/artists' USING PigStorage(',') ;

181

ORDER et LIMIT



- L'opérateur **ORDER** Classer les n-uplets selon un ou plusieurs champs indiqués

Syntaxe : ORDER relation BY champ [ASC|DESC], ...

Exemple: F = ORDER artists BY birth ;

- Retourner une relation ayant un premier champ supplémentaire, le rang des n-uplets par rapport au critère indiqué.

Syntaxe : RANK relation BY champ [ASC|DESC], ...

Exemple: F = RANK artists BY birth ;

- Conserver les N premiers n-uplets de la relations. Utiliser généralement avec ORDER

Syntaxe : LIMIT relation N

Exemple : order_prices=ORDER purchases BY sale DESC;

```
top_ten=LIMIT order_prices 10;
DUMP top_ten;
```

- On peut écrire le script de manière imbriquée

Exemple: top_ten=LIMIT (ORDER purchases BY sale DESC) 10;

183

FILTER

- L'opérateur **FILTER** permet de filtrer les éléments d'un bag selon une condition:

Syntaxe : DEST = FILTER source BY condition;

Exemple: F = FILTER artists BY birth > 15;

- La **condition** peut être:

- ✓ **Comparaison** : même opérateurs qu'en java.
- ✓ **nullité(vide)** d'un champ: IS NULL,IS NOT NULL
- ✓ **connecteurs logiques** : (mêmes opérateurs qu'en SQL) AND,OR et NOT

182

DISTINCT



- L'opérateur **DISTINCT** sert à supprimer les n-uplets en double.

Syntaxe : DISTINCT relation;

Exemple: countries = DISTINCT pays ;

184

Opérateur GROUP BY



- Permet de grouper les tuples d'un bag selon un **champ**:

Syntaxe : destination = **GROUP** source **BY** champ;

- **l'instruction** génère un bag (destination) dont chaque tuple est constitué de deux champs:

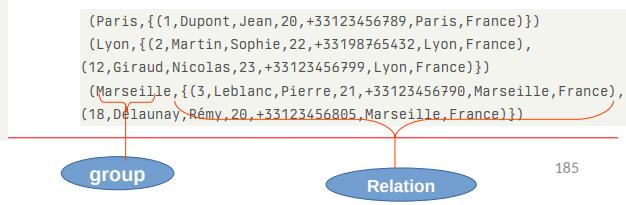
- ✓ Le 1er (**group**), est un champ simple dont la valeur est celle du champ de groupement.
- ✓ Le 2nd est un bag portant le nom de bag **source** sur lequel on a appliqué GROUP. Il contient tous les tuples de source ayant la même valeur du champ de groupement.

Exemple: G = GROUP students BY city;

DESCRIBE G;

Le résultat G: {group: chararray , students:{(id:chararray,name:chararray,age:int,tel :chararray,city :chararray, Country :chararray)}}

```
1,Dupont,Jean,20,+33123456789,Paris,France  
2,Martin,Sophie,22,+33198765432,Lyon,France  
3,Leblanc,Pierre,21,+33123456790,Marseille,France  
4,Durand,Alice,19,+33123456791,Toulouse,France  
5,Lemoine,Lucie,23,+33123456792,Bordeaux,France  
6,Muller,Hugo,20,+33123456793,Nice,France  
7,Dauv,Camille,22,+33123456794,Lyon,France
```



FOREACH imbriqué

- Pour effectuer plusieurs traitements avant le **GENERATE**.

Syntaxe : FOREACH relation{

```
Traitements1 ;  
Traitement2 ...;  
GENERATE...}
```

Exemple

```
R =LOAD ....;  
G = GROUP R BY gender;  
F = FOREACH G {  
    LC1 = FOREACH R GENERATE dep;  
    LC2 = DISTINCT LC1;  
    GENERATE group, LC2;  
};  
DUMP F
```

FOREACH GENERATE



- L'opérateur **FOREACH ... GENERATE** permet de parcourir les tuples d'un bag et de générer d'autres tuples.

Syntaxe : destination = FOREACH source GENERATE expr1 AS champ1,...;

Champ1 peut être un champ de la relation fournie ou bien un valeur calculée

Exemple: Déterminer l'achat maximal par client

GR = GROUP purchases BY client_id;

```
Max_p= FOREACH GR GENERATE group AS client_id,  
MAX(purchases.sale) AS max_sale;
```

- L'opérateur **FOREACH** sert à faire une projection

FOREACH purchases GENERATE client_id

- Réorganiser les champs d'un bag :

FOREACH purchases GENERATE Id_sale AS id, (sale, date) AS details ;

Opérateur JOIN



- Permet de faire des jointure externes entre deux bags ou plus:

Syntaxe : Bag_résultat = **JOIN** left_bag **BY** champ1 [**LEFT|RIGHT|FULL**] [**OUTER**], right_bag **BY** champ2;

- Dans la nouvelle relation, les champs sont nommés relation1::champ1a, relation1::champ1b, ...

Exemple:

```
R1 = LOAD 'employees.txt' Using PigStorage(',') As (id:chararray, name : chararray, salary: int,  
depno: chararray);
```

```
R2 = LOAD 'dep.txt' Using PigStorage(',') As (depno:chararray, depname: chararray);
```

```
R3= JOIN R1 BY depno LEFT, R2 BY depno ;
```

```
(R1 ::id:chararray, R1 ::name : chararray, R1 ::salary: int, R1 ::depno: chararray, R2 ::depno:chararray,  
R2 ::depname: chararray) ;
```

Opérateurs Pig



- Pig propose plusieurs opérateurs :
- Opérateurs d'agrégation :SUM, AVG, MAX, MIN, COUNT, COUNT_STAR
- Opérateur de Cast: (type) champ
- Opérateurs arithmétiques: +, -, *, /, %
- Il y a d'autres opérateurs :
 - CONCAT(v1, v2, ...) concatène les valeurs fournies.
 - DIFF(sac1, sac2) compare les deux bags et retourne un bag contenant les éléments qui ne sont pas en commun.
 - SIZE retourne le nombre d'éléments du champ fourni.

189

Exécution et paramétrage d'un script Pig



- Pour lancer un script pig:

```
exec [[-param param_name = param_value] [...] ] file  
run [[-param param_name = param_value] [...] ] file
```

N.B: Avec run les bags sont accessibles dans le shell Grunt.

- Exemple:

```
R= LOAD 'user/root/employees.txt' Using PigStorage(',') As (id:chararray, name : chararray, salary: int, department: chararray) ;
```

```
F= FILTER R BY department==$dep ;
```

```
G= GROUP F BY department ;
```

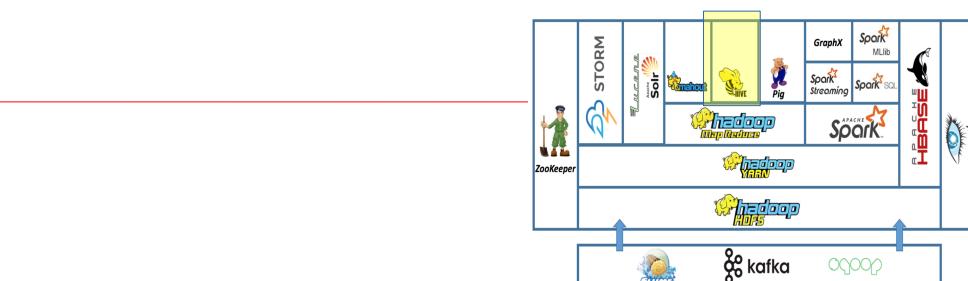
```
H= FOREACH G GENERATE group As department, AVG(salary) As salaire_moy ;
```

191

FLATTEN

- Permet de mettre à plat un champ qui est d'un type complexe (tuple, bag, map)
 - Pour un champ de type tuple, flatten le remplace par ses champs
 - **Exemple:** Pour un bag R: {a:int, t: (b:int, c:int)}
- F= Foreach R Generate a, flatten (t);
F:{a: int, t:{b:int, c:int}}
res = FOREACH F GENERATE a, t:b AS b, t:c AS c;
res:{a: int, b:int, c:int}

192



Hadoop Datawarehouse



192

Système de gestion vs. Système de décision.

	Système de gestion (opérationnel)	Système de décision (analyse)
Objectifs	dédié au métier et à la production ex: facturation, stock, personnel	dédié au management de l'entreprise (pilotage et prise de décision)
Pérennité	données volatiles ex: le prix d'un produit évolue dans le temps	données historisées ex: garder la trace des évolutions des prix, introduction d'une information datée
Optimisation	pour les opérations associées ex: passage en caisse (lecture de code barre)	pour l'analyse et la récapitulation ex: quels les produits achetés Ensembles
Granularité de données	Totale et atomique, on accède directement aux informations atomiques	agrégats, niveau de synthèse selon les besoins de l'analyse

SGBD: Objectifs et traitements

- BD-OLTP représentent les données sous forme aplatie: relation, données normalisées

produit	région	vente	date	vendeur
écrou	Est	50	01012004	X
écrou	Ouest	60	12122003	X
écrou	Centre	110	01112003	Y
vis	Est	70	01042004	Y
vis	Ouest	80	10022004	Z
vis	Centre	90	29032004	Y
boulon	Est	120	05052004	X
boulon	Ouest	10	24042004	Z
boulon	Centre	20	11022004	Y
joint	Est	50	01032004	X
joint	Ouest	40	01102003	Y
joint	Centre	70	01012003	Z

produit	prix	fournisseur
écrou	44	CC
vis	2	DD
boulon	3	VV
joint	1	BB

fournisseur	ville
...	...

OLTP: Objectifs et traitements

- Les systèmes opérationnels « OLTP » On-Line Transaction Processing).
- Dédiés aux tâches quotidienne de l'entreprise (facturation, achat, production, finance, etc.)
- le mode de travail est transactionnel
 - insérer, modifier, interroger (CRUD) des informations **rapidement**, efficacement, en sécurité.

30/10/2025

194

OLTP: Objectifs et traitements

- Les systèmes OLAP sont des systèmes conçus pour l'aide à la prise de décision. (Mode de travail: **OLAP On-Line Analytical Processing**)
- La plupart du temps sont utilisés en lecture (utilisateurs)
- **Données** multi-dimensionnelles , historisées et agrégées.(i.e. ventes par vendeur, par date, par ville...)
- Les **objectifs** principaux sont
 - regrouper, organiser des informations provenant de sources diverses,
 - les **intégrer** et les stocker pour donner à l'utilisateur une vue orientée métier,
 - retrouver et analyser l'information **facilement** et rapidement.

30/10/2025

196

30/10/2025

195

DW-OLAP

- Offre une vue multidimensionnelle des données pour faciliter l'analyse
- Fournir des données compréhensibles pour les décideurs
- Quelle est la quantité de Ventes du produit X dans la région Y ?

produit	région	vente
écrou	Est	50
écrou	Ouest	60
écrou	Centre	110
vis	Est	70
vis	Ouest	80
vis	Centre	90
boulon	Est	120
boulon	Ouest	10
boulon	Centre	20
joint	Est	50
joint	Ouest	40
joint	Centre	70

30/10/20



	Est	Ouest	Centre
écrous	50	60	110
vis	70	80	90
boulons	120	10	20
joints	50	40	70

Tableau croisé

197

Apache Hive : principe

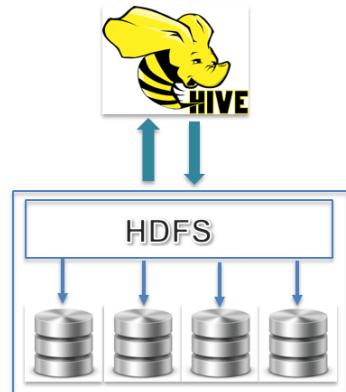


- ❖ **Lecture, écriture et gestion** de PO de données à l'aide d'un langage de requêtage proche de **SQL (HiveQL)**.
- ❖ Les requêtes sont transformées en jobs MapReduce.
- ❖ Hive peut être intégré à d'autres NoSQL DB comme Cassandra et HBase

199

Apache Hive

- ❖ Plateforme de data warehousing pour hadoop:
 - ✓ Distribué
 - ✓ tolérant aux pannes
- ❖ Développé chez Facebook (Meta),
- ❖ Projet open source de l'Apache Software Foundation en 2010
- ❖ Repose sur le système HDFS de hadoop
- ❖ Permet des **analyses** de données à grande échelle (**OLAP**)

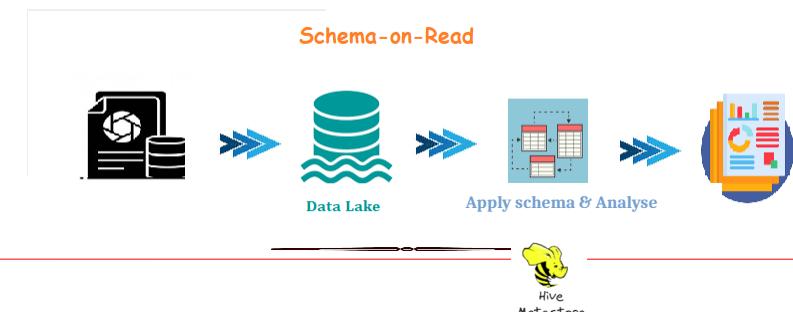


198

Apache Hive : Schema on read



- ❖ Hive permet de manipuler les données (sur HDFS) comme si elles étaient stockées dans des **tables** à l'aide d'un **schéma**.
- ❖ Le schéma (**métadonnées**) comporte les **noms** et **types** des colonnes, et structure les informations en tables exploitables par HiveQL.
- ❖ Le **schéma** est enregistré dans une BDR appelée **metastore** (derby par défaut)



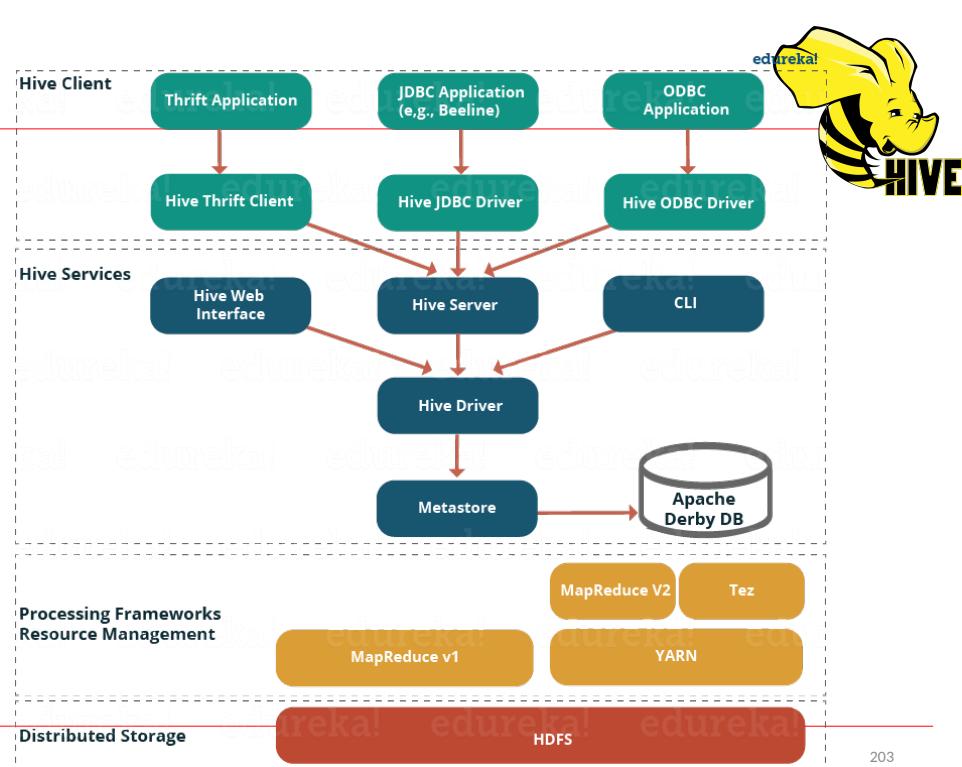
200

Apache Hive : fonctionnalités



- ❖ Hive fournit plusieurs fonctionnalités :
 - ◆ Modèle de requête simple et optimisé (moins de code que MR)
 - ◆ **HQL** a une syntaxe **similaire à SQL**
 - ◆ **UDF** ou User Defined Functions (java/scala)
 - ◆ supporte plusieurs formats de fichiers tels que ORC, SEQUENCEFILE, Record Columnar File et TEXTFILE, etc
 - ◆ Pilotes JDBC et ODBC permettant à de nombreuses applications d'extraire des données Hive (HWC, etc.)

201

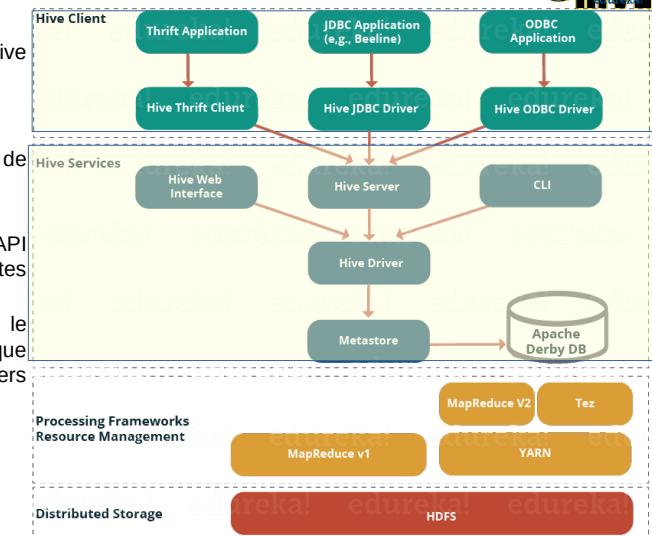


203

Apache Hive : Architecture



- ❖ **Hive client :**
 - ◆ API pour communiquer avec hive (thrift, JDBC, ODBC)
- ❖ **Hive Services**
 - ◆ CLI : interface de ligne de commande(hive)
 - ◆ **Web interface,**
 - ◆ **Hive server :** expose une API simple pour exécuter des requêtes HQL
 - ◆ **Metastore:** BDR pour stocker le schéma des BD, tables ainsi que leurs mappage sur des fichiers HDFS

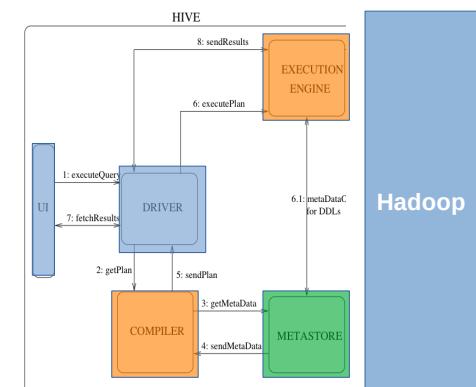


202

Apache Hive : Architecture



- **Hive Driver :** Gérer le cycle de vie d'une requête HQL :
- **Compiler :** analyse la requête en utilisant les métadonnées stockées dans le metastore et génère un plan d'exécution (DAG).
- **Optimizer :** effectue les opérations de transformation sur le plan d'exécution et divise la tâche pour améliorer l'efficacité et l'évolutivité.
- **Execution engine :** exécute le plan d'exécution créé par le compilateur dans l'ordre de leurs dépendances à l'aide de Hadoop



204

Apache Hive : Data Types

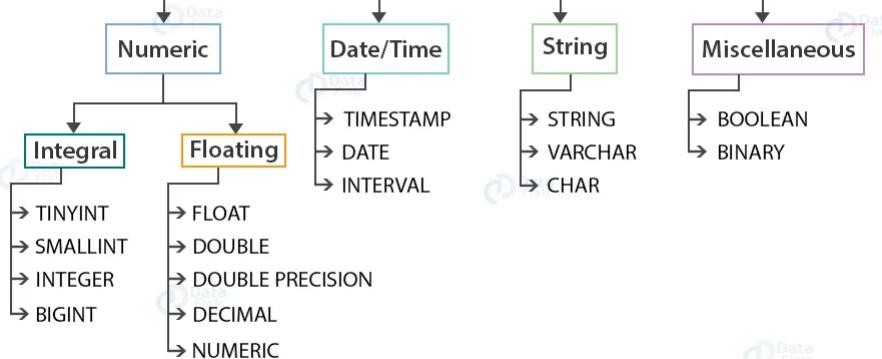


Data Flair

Apache Hive : Data Type

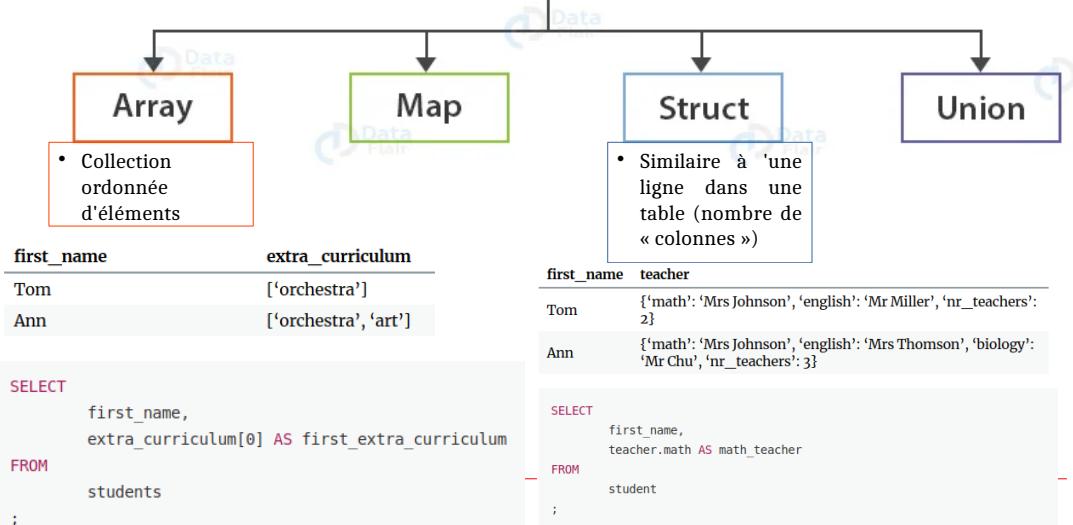


Primitive Data Types



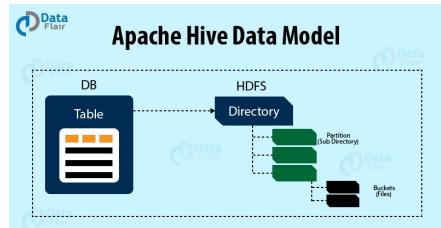
205

Complex Data Types



Apache hive : Data Model

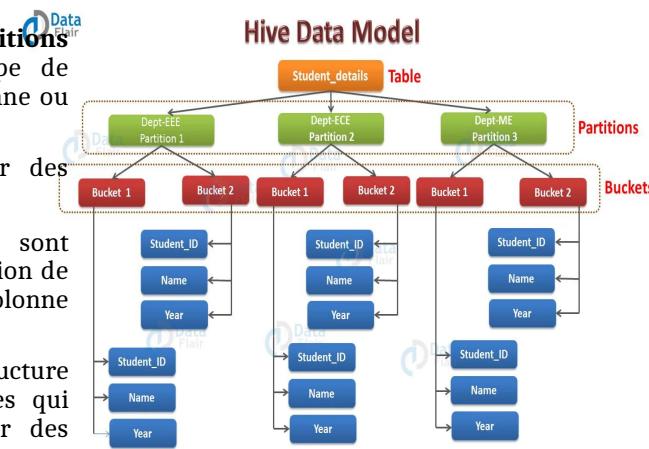
- Table : similaire au table des BDR. Permet de stocker des données
- Hive dispose de deux types de tables :
 - Managed table** (Par défaut) : les données sont déplacées dans un emplacement pré-configuré (data warehouse) de Hive :
 - Lors de la suppression, les données et les metadata sont supprimés.
 - External table** : L'emplacement des données externes est spécifié au moment de la création de la table



209

Apache hive : Data Model

- Hive organise les tables en **partitions** pour **regrouper** le même type de données en fonction d'une colonne ou d'une clé de partition.
 - Accélérer** les requêtes sur des slices de données.
- Les tables ou partitions sont subdivisées en **buckets** en fonction de la fonction de hachage d'une colonne de la table.
 - Donner une structure supplémentaire aux données qui peuvent être utilisées pour des **requêtes plus efficaces**.



210

Base de données

- Une Base de données Hive décrit une **collection de tables**.

Créer une BD

- Syntaxe : `CREATE DATABASE [If NOT EXISTS] <db_name>` ;
- Exemple :

```
CREATE DATABASE nom_database;
SHOW DATABASES;
USE nom_database;
```

Supprimer une BD

- Syntaxe : `DROP DATABASE <db_name> [CASCADE]` ;
- Exemple :

```
DROP DATABASE nom_database CASCADE;
```

Base de données

- Une Base de données Hive décrit une **collection de tables**.
- Afficher les BDs : `SHOW DATABASES` ;
- Donner la **description** d'une BD : `DESCRIBE NOM_DB` ;
- Choisir une BD : `USE NOM_DB` ;
- Connaitre le nom de la BD en cours d'utilisation: `SELECT current_database()` ;
- Afficher la liste des **tables** de la BD : `show TABLES` ;

211

212

Création d'une table



- Pour **créer** une table :

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [nom_db.] nom(schéma) (col_name data_type [COMMENT  
'col_comment'], ...) ROW FORMAT DELIMITED DescFormat
```

- Les directives situées après le schéma indiquent la manière dont les données sont stockées dans le fichier CSV.

Exemples :

- FIELDS TERMINATED BY '': « ; pour séparer les champs »
- COLLECTION ITEMS TERMINATED BY '': il y a un « , entre les éléments d'un **ARRAY** »
- MAP KEYS TERMINATED BY '': « : entre les clés et les valeurs d'un **MAP** »
- LINES TERMINATED BY '\n': « \n enfin de ligne »
- STORED AS TEXTFILE (plusieurs formats possibles : TEXTFILE, SEQUENCEFILE, PARQUET, AVRO, RCFILE, ORC)
- PARTITIONED BY (col_name data_type)

213

Manipulation de tables



- Pour **afficher la structure** d'une table :

```
DESCRIBE [NOM_DB.]NOM_TABLE;
```

- Vider** une table: **TRUNCATE TABLE** nom_table ; --applicable pour une table interne

- Supprimer** une table : **DROP TABLE** nom_table ;

215

Chargement des données



- Pour **charger** un **fichier** (ex. csv)qui se trouve sur HDFS, dans la table:

```
LOAD DATA INPATH '/input/data' [OVERWRITE] INTO TABLE purchases;
```

NB : Hive déplace le fichier CSV dans ses propres répertoires

- Pour **empêcher** Hive de **déplacer** le fichier, On peut utiliser:

```
CREATE EXTERNAL TABLE ....
```

- On peut aussi **charger** un fichier **local** (pas HDFS) :

```
LOAD DATA LOCAL '/shared_volume/purchases.csv' OVERWRITE INTO TABLE  
purchases;
```

} Le fichier est copié alors dans HDFS

- Insérer** des données dans une table :

```
INSERT INTO TABLE nom_table VALUES ('valeur1', 'valeur2');
```

214

Requête de base



Sélection

```
SELECT * FROM nom_table;  
SELECT colonne1, colonne2 FROM nom_table WHERE condition;
```

Agrégation

```
SELECT colonne1, COUNT(*) FROM nom_table GROUP BY colonne1;
```

Jointure

```
SELECT a.colonne1, b.colonne2  
FROM table1 a  
JOIN table2 b  
ON a.id = b.id;
```

216

Optimisation DES REQUETES

- Le **partitionnement** est utilisé pour organiser logiquement les données dans des répertoires en fonction des valeurs des colonnes (comme l'année ou la région)

```
CREATE TABLE nom_table_partition (
    colonne1 TYPE,
    colonne2 TYPE
)
PARTITIONED BY (colonne_partition TYPE);
```

- Le **bucketing** divise les données de manière uniforme en buckets de taille fixe en fonction du hachage d'une colonne et est souvent utilisé pour améliorer les performances d'opérations telles que les jointures. (hdfs files)

```
CLUSTERED BY (colonne) INTO n BUCKETS;
```

217

Format de fichiers

- Formats de stockage en lignes (enregistrements):**

- Fichiers plats / fichiers texte



- CSV et fichiers délimités

- JSON

- SequenceFile

- Avro

- Autres formats: XML, YAML

- Formats de stockage orientés colonnes:**

- RC (Record Columnar File)/ ORC (Optimized Row Columnar File)

- Parquet

219

Introduction aux formats de fichiers

- Rôle des formats de fichiers :

- Stockage efficace des données.
- Optimisation des performances pour le traitement analytique.

- Critères de choix :

- Taille des fichiers (compression).
- Compatibilité avec les outils.
- Vitesse de lecture/écriture.
- Support des schémas.



218

Formats de fichiers : data model

Logical

	Col A	Col B	Col C
Row 0	A0	B0	C0
Row 1	A1	B1	C1
Row 2	A2	B2	C2
Row 3	A3	B3	C3
Row 4	A4	B4	C4
Row 5	A5	B5	C5

Physical

A0	B0	C0	A1	B1	C1	A2	B2	C2
A3	B3	C3	A4	B4	C4	A5	B5	C5

Row-wise

A0	A1	A2	B0	B1	B2	C0	C1	C2
A3	A4	A5	B3	B4	B5	C3	C4	C5

Columnar

A0	A1	A2	B0	B1	B2	C0	C1	C2
A3	A4	A5	B3	B4	B5	C3	C4	C5

Hybrid

220



- Format de sérialisation binaire.
- Compact et rapide.
- Utilise un schéma JSON pour définir la structure des données.

Avantages :

- Adapté aux systèmes distribués.
- Idéal pour les données évolutives (schéma auto-descriptif).
- Supporté par Hadoop, Kafka, Spark.

```
{
  "type": "record",
  "name": "Utilisateur",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "nom", "type": "string"}
  ]
}
```

```
CREATE TABLE table_avro
STORED AS AVRO
TBLPROPERTIES ('avro.schema.url'='chemin_du_schema_avro');
```

221

Format parquet

- Format de fichier (binaire) orienté colonne open source

- Très efficace pour des requêtes **analytiques** rapides.

- Optimisé pour de grands volumes de données

Avantages

- Compression de données
- Compatible avec plusieurs outils big data (spark, hive, impala, etc.).
- Prise en charge de types de données complexes
- Parquet is a columnar format that stores the data in row groups!*

	Column 1	Column 2	Column 3	Column 4	Column 5
	Product	Customer	Country	Date	Sales Amount
Row Group 1	Ball	John Doe	USA	2023-01-01	100
	T-Shirt	John Doe	USA	2023-01-02	200
Row Group 2	Socks	Maria Adams	UK	2023-01-01	300
	Socks	Antonio Grant	USA	2023-01-03	100
Row Group 3	T-Shirt	Maria Adams	UK	2023-01-02	500
	Socks	John Doe	USA	2023-01-05	200

223

Optimized Row Columnar (ORC)

- Format binaire optimisé pour Hive.
- Stockage en colonnes pour améliorer les requêtes analytiques.
- Supporte l'indexation, la compression, et les statistiques.

Avantages :

- Performances élevées pour les lectures séquentielles.
- Compact grâce à la compression intégrée.
- Optimisé pour les grandes tables partitionnées.

```
CREATE TABLE table_orc (
  colonne1 INT,
  colonne2 STRING
)
STORED AS ORC;
```



222

Sequence file

- un format binaire qui stocke des paires clé-valeur dans un format sérialisé.

- Utilisé principalement avec Hadoop et MapReduce

Avantages :

- Optimisé pour des applications de traitement par lots avec Hadoop.
- stocker de grandes quantités de données structurées
- Échange de données entre des programmes MapReduce.

SequenceFile File Layout

Data	Key	Value	Key	Value	Key	Value	Key	Value
------	-----	-------	-----	-------	-----	-------	-----	-------

224

File formats : summary

File Format	File Extension	Binary or Text	Schema	Schema Evolution	Compression	Splittable	Column Filtering
Parquet	.parquet	Binary	Yes	Yes	RLE, Dict & Snappy	Yes	Yes
Orc	.orc	Binary	Yes	Yes	RLE, Dict & ZLib	Yes	Yes
Avro	.avro	Binary	Yes	Yes	Snappy Deflate	Yes	No
Petastorm	.petastorm	Binary	Yes	Yes	RLE, Dict & Snappy	Yes	Yes

<https://www.hopsworks.ai/>

225

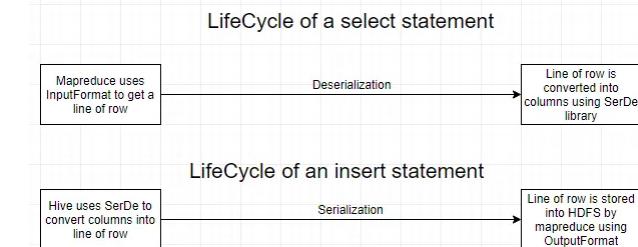
SerDe :Pourquoi ?

- Formats diversifiés** : Hive doit gérer un large éventail de formats de fichiers (texte, Avro, ORC, Parquet, etc.).
- Personnalisation** : Les utilisateurs peuvent créer leurs propres SerDe pour traiter des formats spécifiques non pris en charge nativement.
- Compatibilité** avec les anciens systèmes : Les SerDe assurent une continuité pour les formats de données existants.

227

SerDe

- Composant utilisé par Hive pour :
 - Sérialiser** : Écrire des données dans un format spécifique.
 - Désérialiser** : Lire des données dans Hive à partir d'un format spécifique.
 - Traduire les données brutes (par exemple, stockées dans HDFS) en colonnes exploitables par Hive.



226

SerDe Natif

- Hive** inclut plusieurs SerDe pré-intégrés pour gérer les formats courants :
 - LazySimpleSerDe** : Par défaut pour les fichiers texte délimités.
 - AvroSerDe** : Pour lire et écrire des fichiers Avro.
 - OrcSerde** : Pour les fichiers ORC.
 - ParquetHiveSerDe** : Pour les fichiers Parquet.

```
CREATE TABLE example_table (
    id INT,
    name STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS TEXTFILE;
```

Hive prend en charge directement des formats comme ORC et Parquet sans nécessiter de configuration SerDe explicite

228