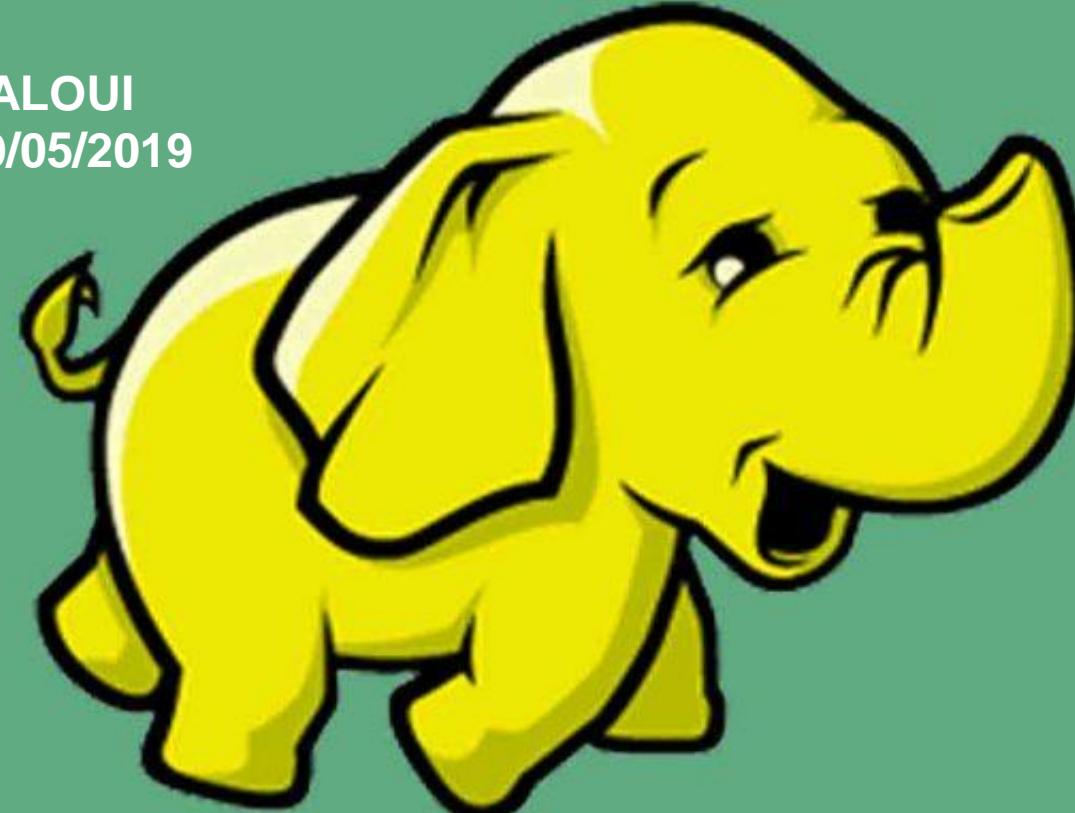
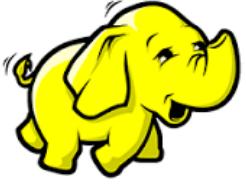


Formateur: Rabeh ALOUI  
Date de Formation: 09/05/2019

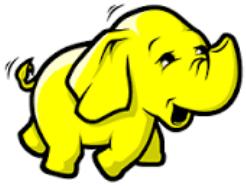


# hadoop



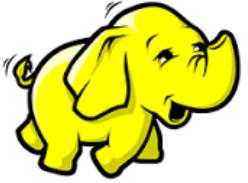
# Qui suis-je ?

- Technical Leader Big Data - Apache Spark
- Développeur Scala, Python
- Formateur Big Data



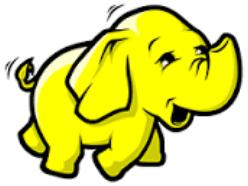
## Et vous ?

- Prénom ?
- Expérience ?
- Attentes ?



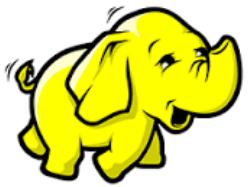
# Plan

1. Comprendre le BIG Data
2. Présentation d'Hadoop
3. HDFS
4. MapReduce
5. YARN
6. Hive
7. Sqoop
8. Oozie

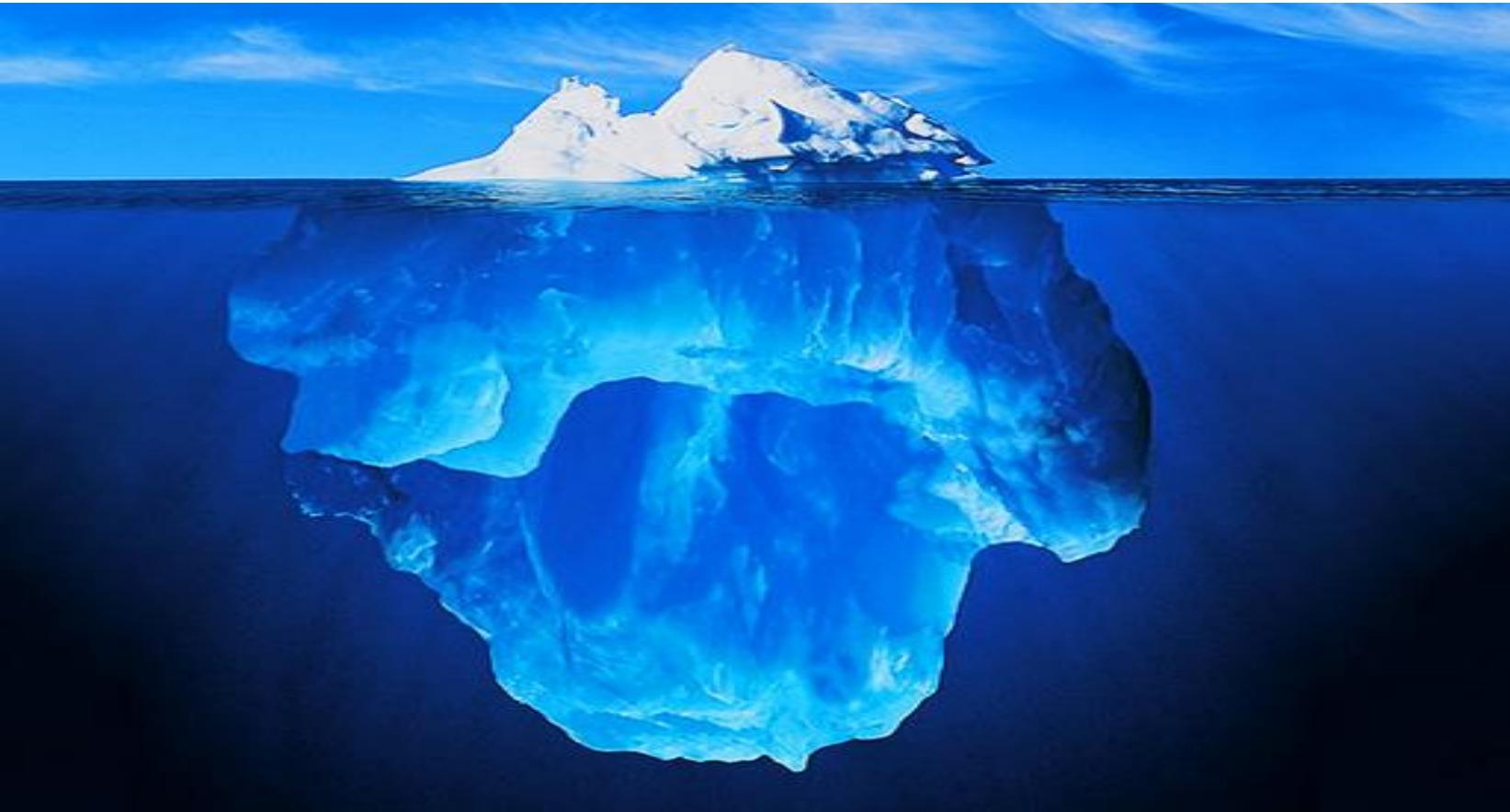


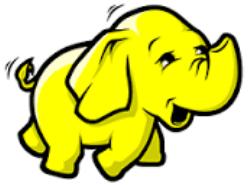
# Plan

## Comprendre le BIG Data



## Encore beaucoup de données non exploitées

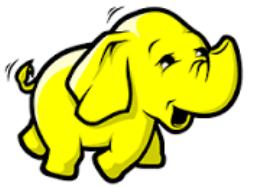




## La data est le nouveau pétrole

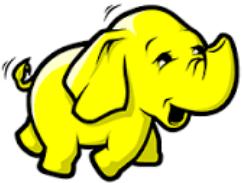


- Citation de Clive Humby (mathématicien et architecte anglais) reprise depuis par la commission européenne, le Gartner, le CEO Diggit, le CEO IBM ...



## Les outils ne sont plus adaptés

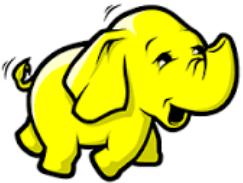




# De la donnée au Big Data

- **Octet:** Grain de riz
- **Kilo-octet:** Bol de riz
- **Mégaoctet:** 8 sacs de riz
- **Gigaoctet:** 3 semi-remorques
- **Téraoctet:** 2 porte-containers
- **Pétaoctet:** Manhattan couverte de riz
- **Exaoctet:** Côte ouest des USA couverte de riz
- **Zettaoctet:** couvrir l'océan pacifique de riz
- **Yottaoctet:** Remplir le volume de la terre de riz



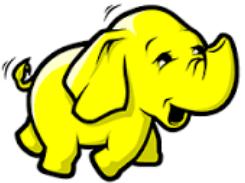


# De la donnée au Big Data

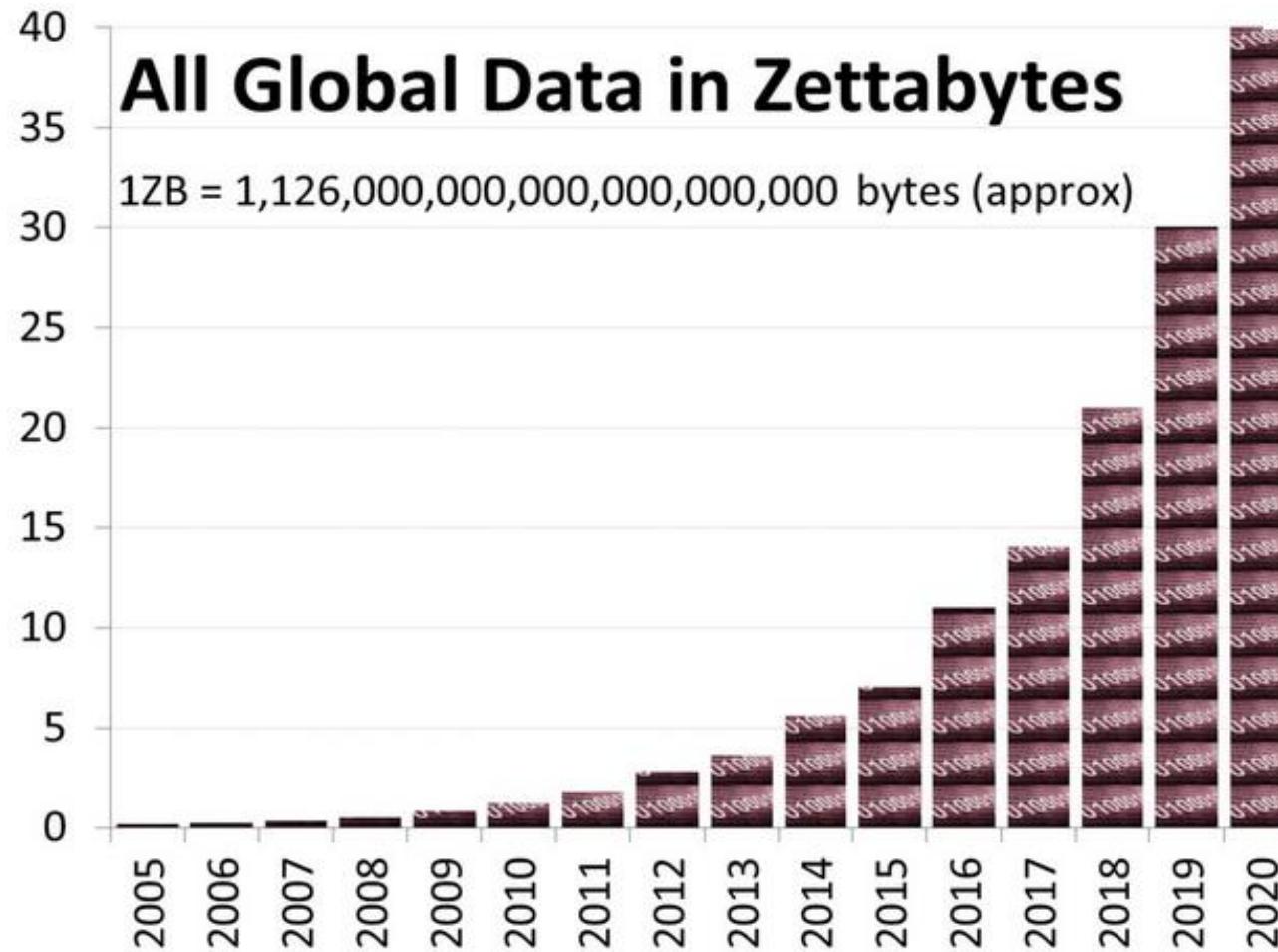
- **Octet**: Grain de riz
- **Kilo-octet**: Bol de riz
- **Mégaoctet**: 8 sacs de riz
- **Gigaoctet**: 3 semi-remorques
- **Téraoctet**: 2 porte-containers
- **Pétaoctet**: Manhattan couverte de riz
- **Exaoctet**: Côte ouest des USA couverte de riz
- **Zettaoctet**: couvrir l'océan pacifique de riz
- **Yottaoctet**: Remplir le volume de la terre de riz

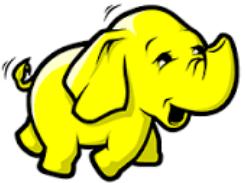


*Tous les 2 jours, nous générions 5 Exaoctets*



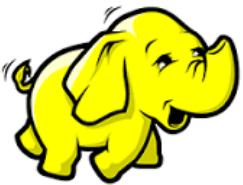
# De la donnée au Big Data





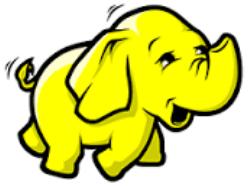
## De la donnée au Big Data

- 2004 – 2005 une année charnière(de transition): création d’Hadoop et en 2008 Hadoop devient Top Project Apache Open Source
- En 2013 nous avons généré autant de données que depuis la création de l’humanité.
- 90 % des données qui sont disponibles, à l’heure actuelle, ont été créées au cours des deux dernières années.
- En 2015, en effet, 7,9 ZB de données ont été générées (soit l’équivalent de ce que l’on peut stocker sur 250 milliards de DVD).
- Et si l’évolution se poursuit au même rythme, on devrait en être à 35 ZB à l’horizon 2020.



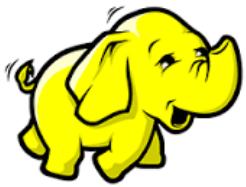
# De la donnée au Big Data





# Plan

**Qu'est ce que le Big Data?**



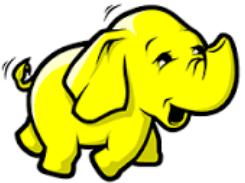
# Définition

## Wikipédia

- Le Big Data désigne la problématique d'avoir un ensemble de données à traiter tellement volumineux qu'il devient très difficile, voire impossible, de le faire avec les outils existants.

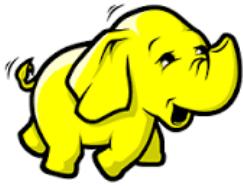
## Apparition du mot Big Data

- Article présenté au congrès de l'Econometric Society en 2000



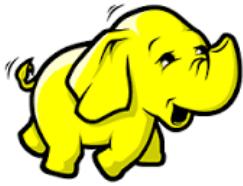
## Définition

- Les données ne sont plus seulement utilisées pour tester la validité d'une hypothèse mais peuvent permettre de découvrir des structures causales cachées, que l'on n'avait pas envisagées au départ.
- Le Big Data permet grâce à l'analyse prédictive de réaliser des prédictions en répondant à trois questions clés : Que s'est-il passé ? Que se passe-t-il ? Que va-t-il se passer ?
- Le Big data oblige à repenser l'organisation de l'entreprise car les informations doivent circuler au-delà des silos Existants.
- Ceci oblige à repenser complètement la capture, le stockage, l'analyse et la visualisation.



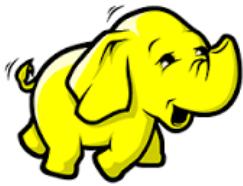
# Les concepts sous-jacents

- NoSQL
- Analyse temps réel
- Cloud
- Non structuré
- Machine Learning
- Stockage low cost
- Open Data



## Pourquoi maintenant ?

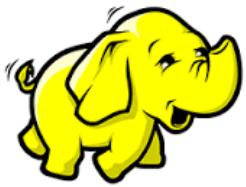
- La puissance des micro-processeurs
- Les coûts de stockage: Pour 1Go : de 1 000 000 \$ à moins de 1\$
- La vitesse des réseaux: De 9 600 bites/s à 10 Gb/s



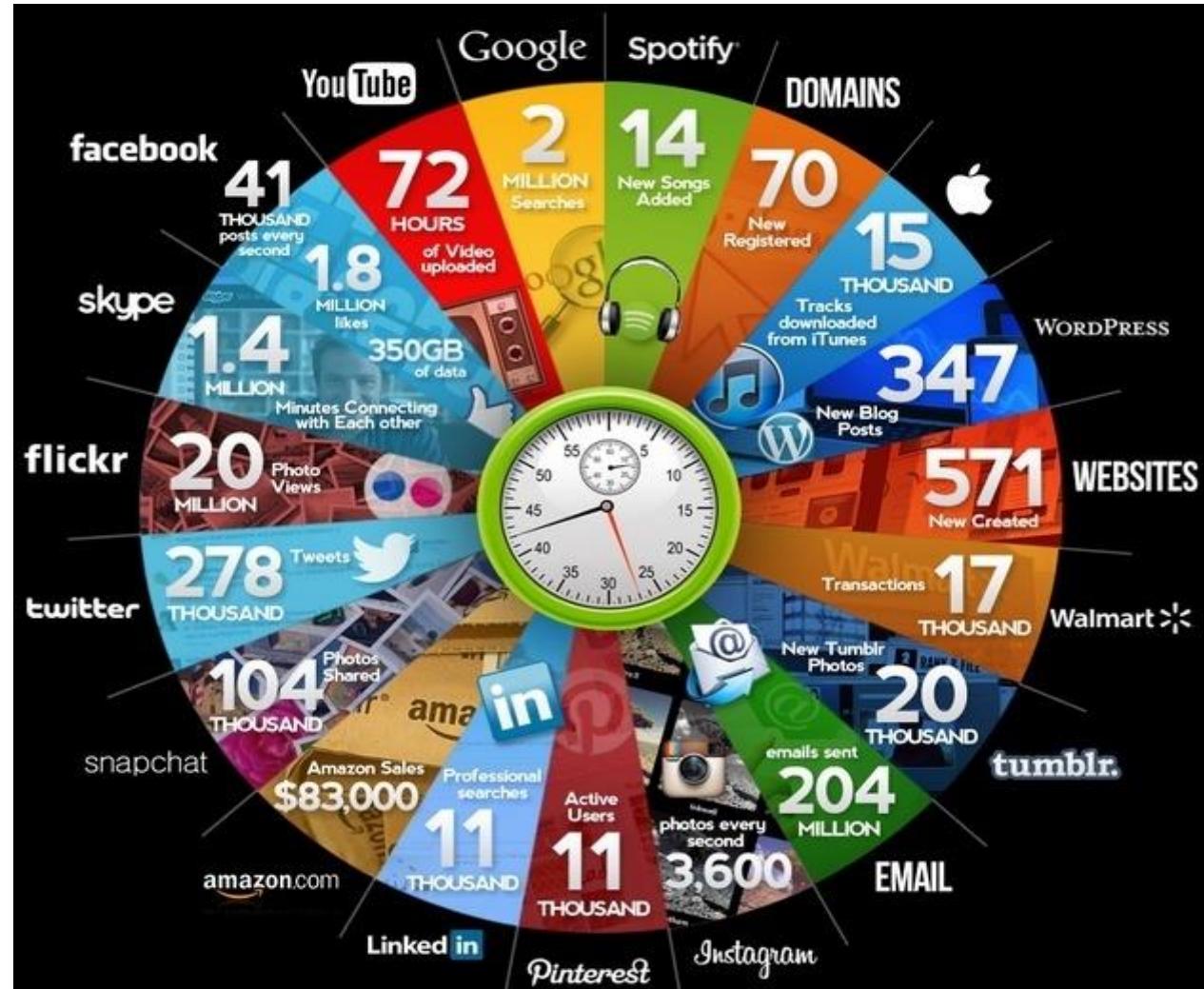
## Les 3 V du BIG DATA

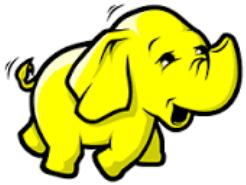
### 1- Volume de données

- Parmi ceux qui génèrent cette volumétrie de données: ceux qu'on appelle les GAFA (Google, Apple, Facebook, Amazon) + Twitter, Microsoft, etc.



# Les 3 V du BIG DATA

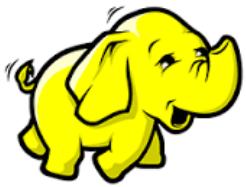




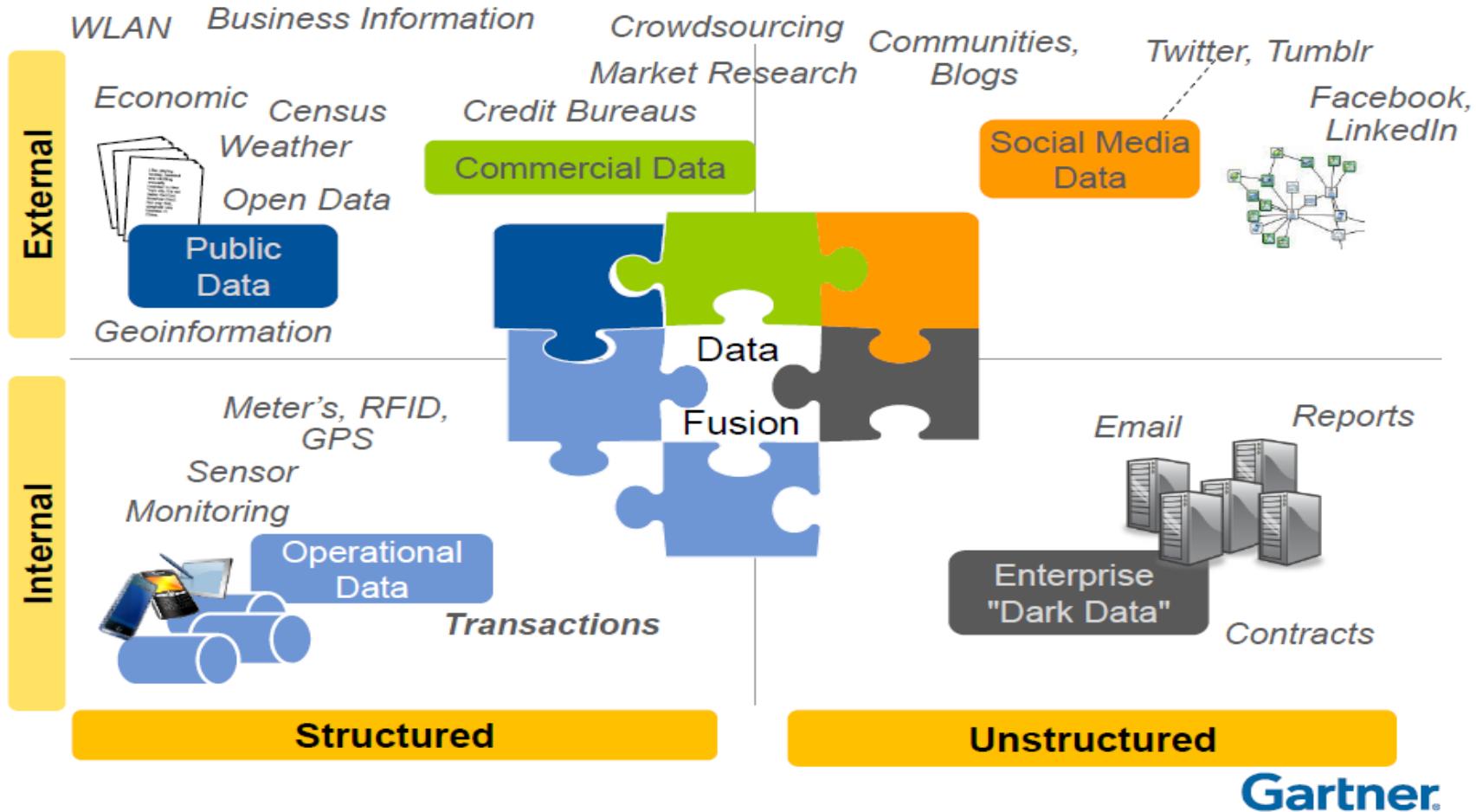
## Les 3 V du BIG DATA

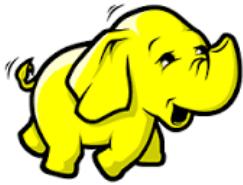
### 2- Variété de données

- Données structurées: systèmes opérationnels, CRM , table SQL, etc.
- Données non structurées: vidéo, images, etc.
- Données semi structurées: Twitter via API, fichier XML, JSON, Email, etc.
- Certaines de ces données pouvaient être consommées via des ETL comme *Talend*, mais toujours il y a le souci de la volumétrie, et de la vitesse d'intégration



# Les 3 V du BIG DATA

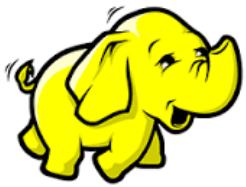




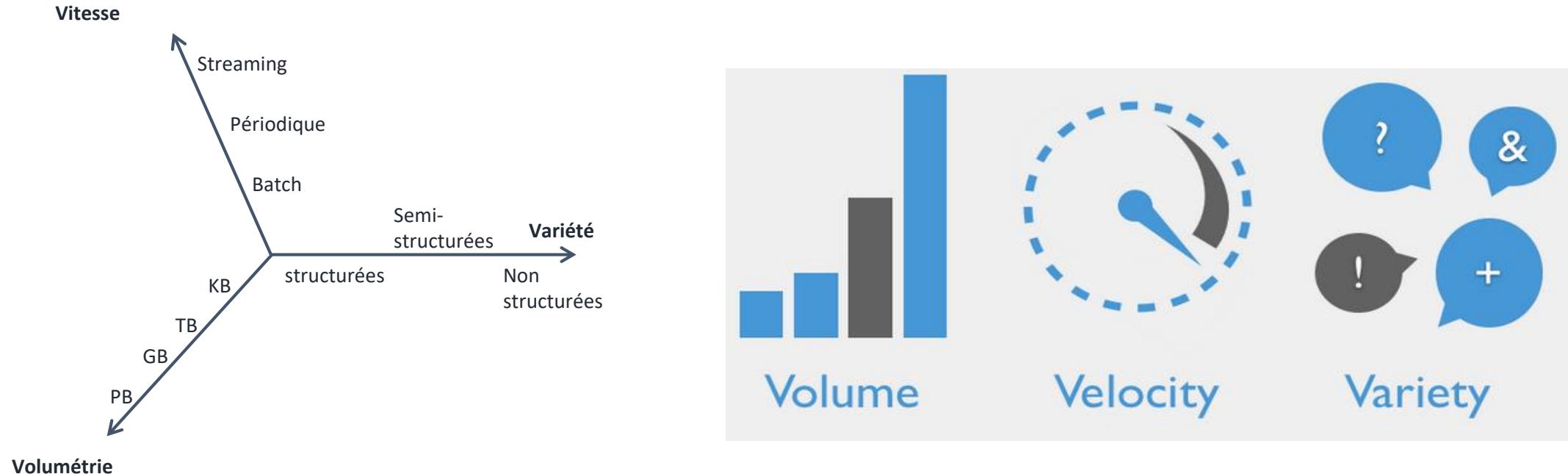
## Les 3 V du BIG DATA

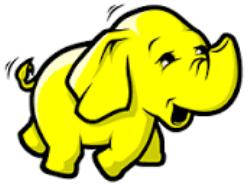
### 3- Vélocité de données = rapidité

- Production et collecte en temps réel
- De nos jours, avec l'émergence des réseaux sociaux, on souhaite avoir une information la plus pertinente et la plus proche de la réalité en quasi temps réel



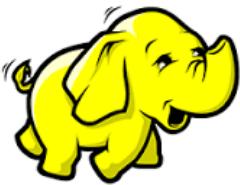
# Les 3 V du BIG DATA



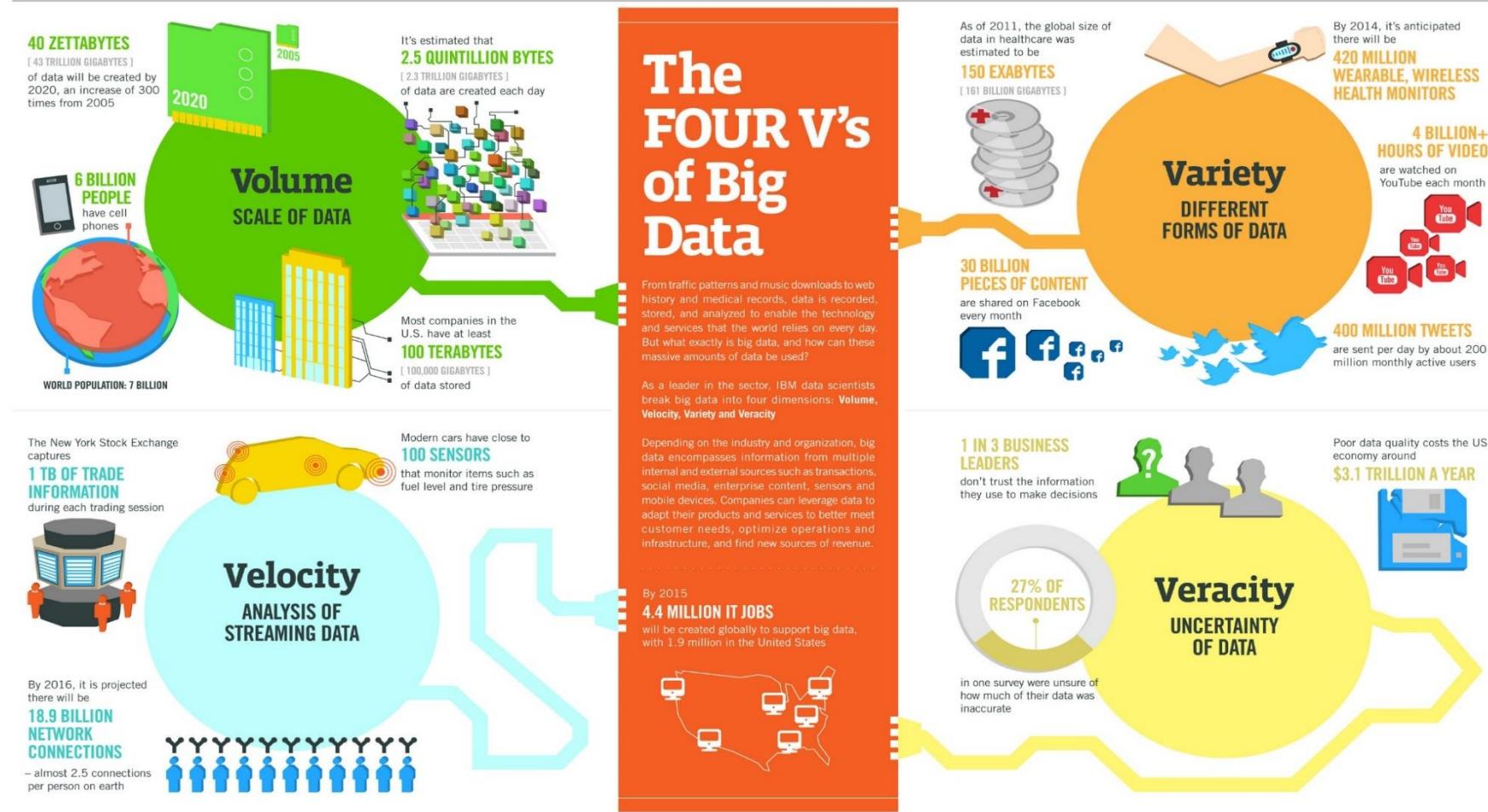


# Les 3 V du BIG DATA

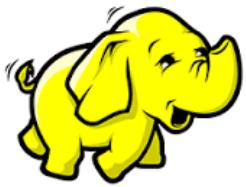
- Batch: Lancement d'un ensemble de scripts
- Périodique: Exemple chez EDF pour le marché d'affaire, on interroge nos sources toutes les 20 minutes, et on propage nos modifications de façon hebdomadaire (mise à jour du miroir)
- Streaming: Click Stream : Parcours client
- Structurés: CRM
- Semi Structuré: API Twitter
- Ajout d'un 4è V: Véracité (Informations diffusées par la concurrence)



# Les 4 V du BIG DATA



IBM



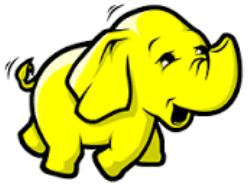
# Les 5 V du BIG DATA

## Véracité

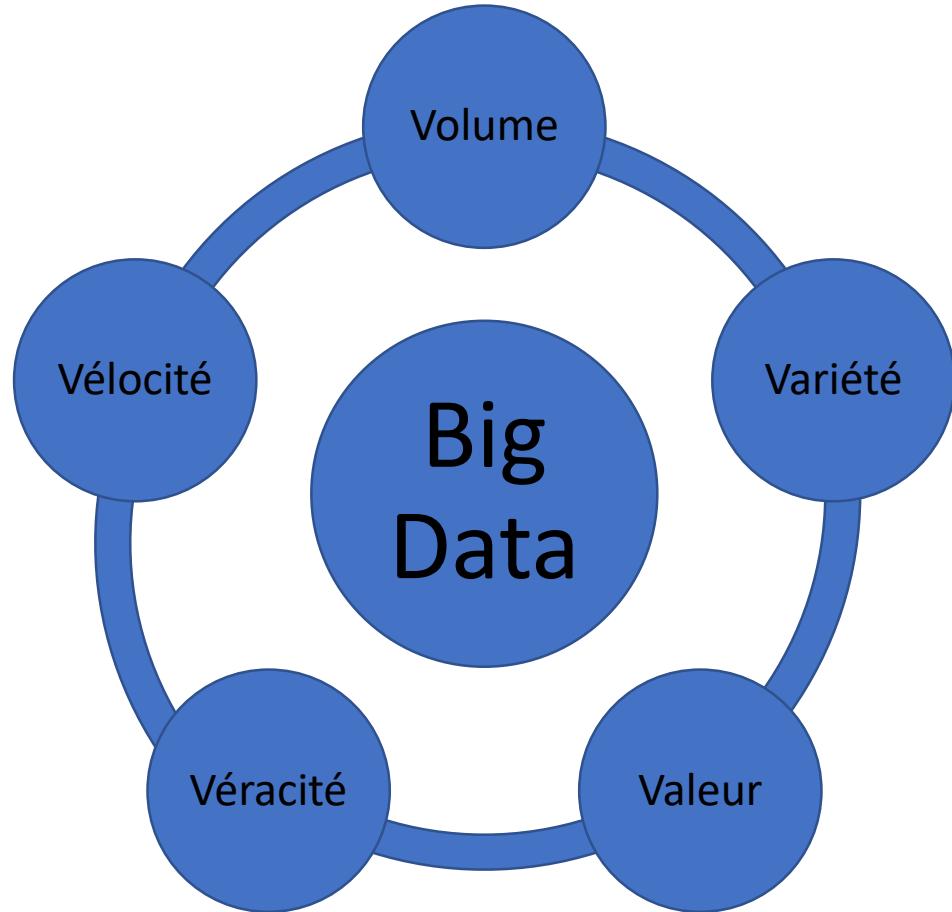
- La véracité ou fiabilité des données est notamment menacée par les comportements déclaratifs (sur formulaires), par les diversités des points de collecte, par la multiplication des formats de données et par l'activité des robots et faux profils innombrables sévissant sur Internet

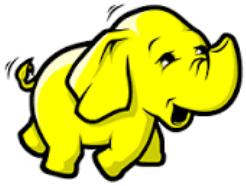
## Valeur

- Dans un contexte d'infobésité, il s'agit d'être capable de se concentrer sur les données ayant une réelle valeur et étant actionnables.
- Avec ces 5 V on ne peut pas réaliser la capture, stockage, analyse et visualisation avec les outils traditionnels



# Les 5 V du BIG DATA





# Les changements

## Transformation numérique



## Changement de comportement



La recherche d'information

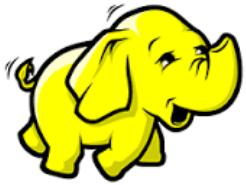
L'intervention des croyances déterminantes

La comparaison des offres

La formation d'une attitude

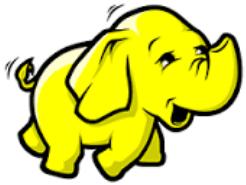
Le choix final

Modification du processus



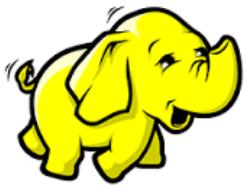
## Les changements

- La transformation numérique: digitalisation des entreprises et de la relation client et des industries avec les capteurs et les applications mobiles
- Avec nos smartphone ou sur internet nous pouvons commander un taxi, un appartement pour la nuit, faire nos courses,... toutes ces activités laissent des traces et génèrent de la données.
- Maintenant c'est l'information, la data qui est valorisée: Uber n'est propriétaire d'aucune automobile, Airbnb ne dispose d'aucune chambre et Alibaba n'a aucun stock. Mais ils savent qui a besoin de quoi et quand



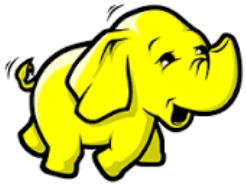
# Les changements

- IoT : les objets connectés, génère énormément de données à fréquence réduite, camion connectés, optimisation des trajets et de la consommation du fuel.
- Les réseaux : les comportements et les sentiments. Utilisé beaucoup dans la finance et les actions.
- Les données météo a un impact direct sur la consommation et la production (sur le chiffre d'affaire d'un film au cinéma ...).
- Open data: des données de collectivités (démographie, statistiques sur les impôts,...) à consommer par plein d'applications et de nouveau cas d'usage, permet par exemple cibler l'ouverture d'un magasin ou pas.



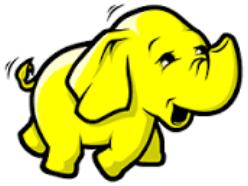
# Les objectifs principaux des projets Big Data

Gains	Entreprises ayant constaté un gain
Meilleure prise de décision (data science)	69%
Amélioration des processus opérationnels	54%
Amélioration de la connaissance client (ex: e-commerce)	52%
Réduction des coûts	47%



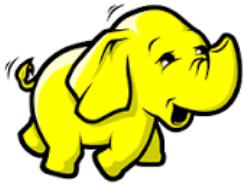
# Les domaines

- Santé
- Transport
- Energie
- Culture et Archive
- Industrie
- ....



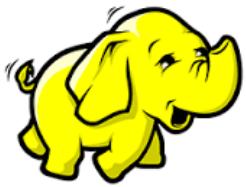
# Plan

## Présentation d'Hadoop



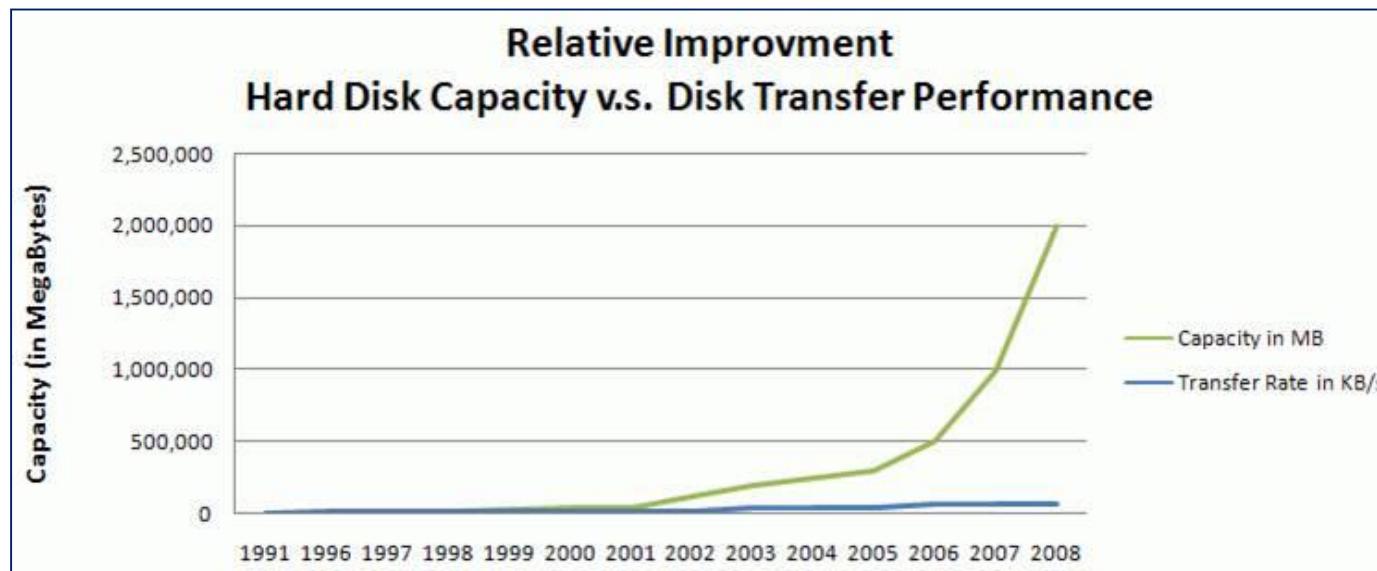
## Quel est le talon d'Achille d'un ordinateur ?

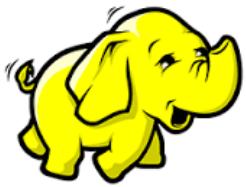
- Un talon d'Achille est une faiblesse fatale en dépit d'une grande force générale,
- Réponse : le disque dur !



# Stockage et transferts de données

- Le problème est simple : les capacités de stockage (et de production de données) ont évolué plus vite que les performances de transfert.
- Un ordinateur est un système comparable à une chaîne : le composant le plus lent décide de la vitesse de l'ensemble.





# La solution

## Solution

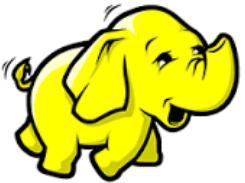
- Paralléliser l'écriture et la lecture des données sur plusieurs disques durs.

## Problèmes induits

- Le risque de panne est plus important: réPLICATION
- Répartir au mieux les données pour en faciliter la consultation

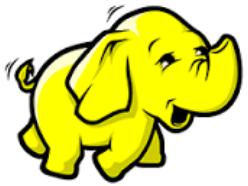
## Hadoop

- solution fiable, scalable, open source
- Fonctionnant sur du commodity hardware pour stocker et analyser des données.



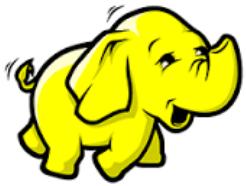
## La solution

- Le matériel de base: un dispositif ou un composant de dispositif qui est relativement peu coûteux, largement disponible et plus ou moins interchangeable avec d'autres matériels de ce type.
- Un système est dit *fiable* lorsque la probabilité de remplir sa mission sur une durée donnée correspond à celle spécifiée dans le cahier de charge.
- Scalable: La *scalabilité* est la capacité d'un dispositif informatique à s'adapter au rythme de la demande et étendre le matériel



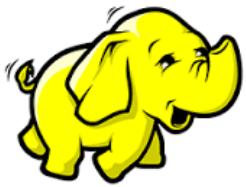
# Apache Hadoop

- Hadoop est un écosystème et non une application en soi même.
- Hadoop est la combinaison entre le stockage *hdfs*, le traitement *mapreduce* et la gestion de ressources *yarn*.
- MapReduce est un système de traitement orienté batch, non orienté analyse temps réel.
- Hadoop est souvent utilisé pour référer à un écosystème de projets plus larges, dont certains vont plus loin que les traitements batch.
  - Base de données NoSQL: Hbase, Cassandra, ...
  - Interactive SQL : Impala, Hive on Tez, ...
  - Itérative processing : Spark, Flink, ...
  - Stream processing : Storm, Spark Streaming, ...



## Hadoop vs les systèmes de BDD relationnelles

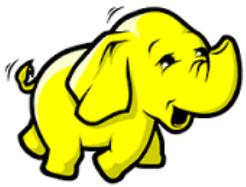
- Les données sont généralement normalisées dans les BDD pour éviter les erreurs d'intégrité et de redondance.
- La normalisation pose des problèmes à Hadoop car la lecture des enregistrements n'est plus une opération locale (recherche = latence + répartition sur les nœuds).



# Hadoop vs les systèmes de BDD relationnelles

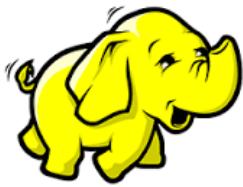
- MapReduce est complémentaire aux BDD relationnelles
- Hadoop et les BDD évoluent et les frontières deviennent floues

	<b>Hadoop</b>	<b>BDD relationnelles</b>
Lecture	Parcours de l'intégralité des données (batch)	Requête temps réel ou mise à jour sur une portion des données sur des données indexées
Ecritures	Données écrites une fois et parcourues en lecture plusieurs fois	Données mises à jour continuellement
Volumétrie	Pétabytes (variable)	Gigabytes
Accès	Batch	Temps réel (réponse) et batch
Transaction	Aucune	Propriétés ACID
Structure	Schema-on-read	Schema-on-write
Intégrité	Faible	Forte
Scalabilité	Linéaire	Non linéaire



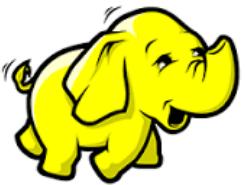
# Hadoop vs les systèmes de BDD relationnelles

- MapReduce ne respecte pas: atomicité, cohérence, isolation et durabilité
- Schema on read: au moment de la lecture il n'y a pas de restriction au moment de l'écriture.
- Schema on write: une restriction au moment de l'écriture
- L'intégrité des données: la cohérence et non redondance et la qualité n'est pas garantie naturellement par hadoop (point faible des systèmes NoSQL )
- Scalabilité: il peut monter en charge, passage à l'échelle, on peut rajouter des nœuds.



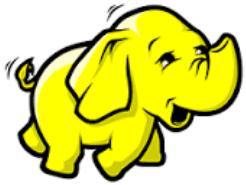
# Use Cases Commun

- Extract, Transform, and Load (ETL)
- Data Analysis
- Text mining
- Graph Creation and Analysis
- Data Storage
- Collaborative Filtering
- Prediction Models
- Sentiment Analysis
- Risk Assessment

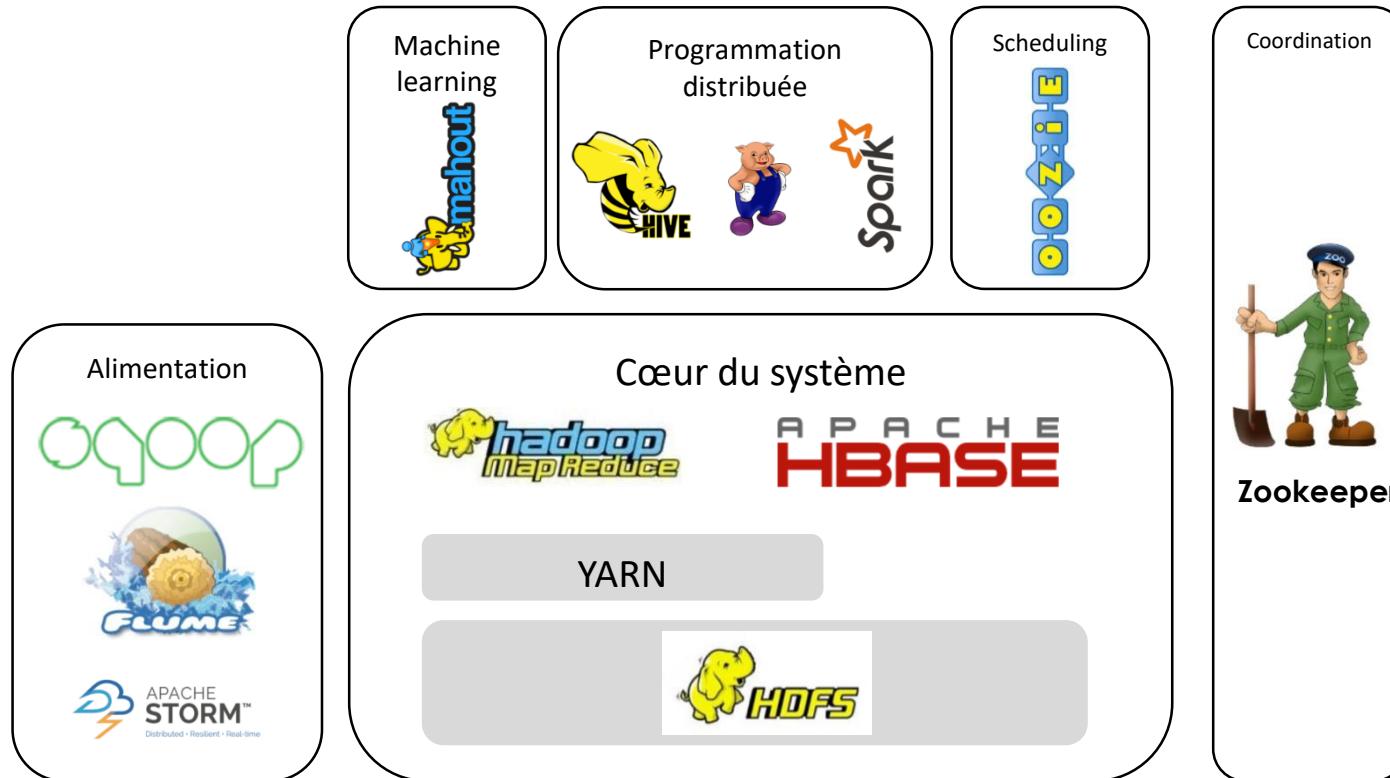


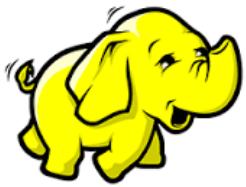
## Hadoop: un peu d'histoire

- Créé par Doug Cutting, le logo d'Hadoop est inspiré par la peluche du fils de Doug Cutting.
- 2002: Apache Nutch: moteur de recherche web open source, l'architecture ne pourrait pas tenir des milliards d'enregistrements.
- 2003: Google publie un papier sur GFS (Google File System), système de fichier distribués résolvant la problématique stockage.
- 2004: Nutch Distributed Filesystem (NDFS).
- 2004: Google publie un papier introduisant MapReduce.
- 2005: implémentation de MapReduce dans Nutch avec NDFS.
- 2006: création d'Hadoop. Doug Cutting rejoint Yahoo.
- 2008: Yahoo annonce que son moteur d'indexation est un cluster Hadoop de 10 000 cœurs.



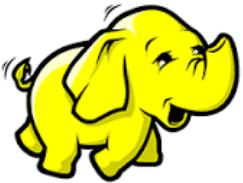
# Hadoop



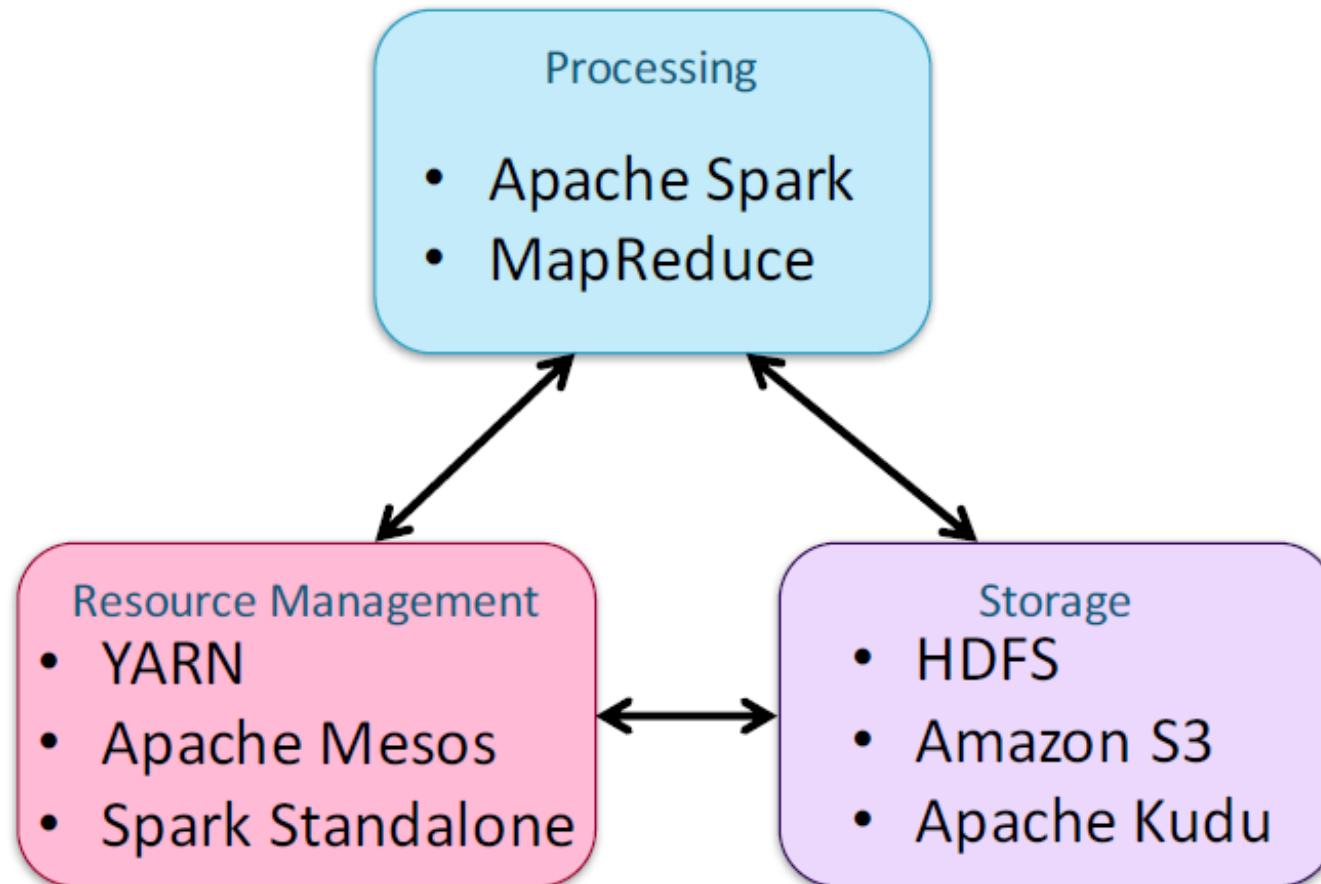


# Hadoop

- ZooKeeper est un logiciel de gestion de configuration pour systèmes distribués, basé sur le logiciel Chubby développé par Google.
- Oozie est un Workflow scheduler permettant de gérer les job Hadoop
- Storm : Event processor : système de calcul distribué en temps réel. Storm facilite le traitement fiable de flux de données sans limite, en temps réel

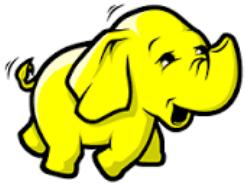


# Hadoop



*A Hadoop Cluster*

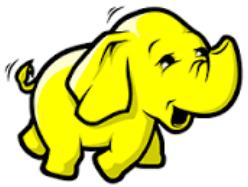




# Hadoop

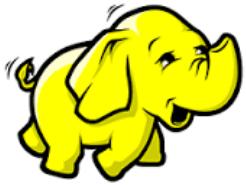
## Les composants Hadoop

- Ingesting data (Flume, Sqoop, Kafka)
- Storing data (HDFS, HBase, Kudu)
- Processing data (Spark, Hadoop MapReduce, Pig)
- Modeling data as tables for SQL access (Impala, Hive)
- Exploring data (Hue, Search)
- Protecting Data (Sentry)

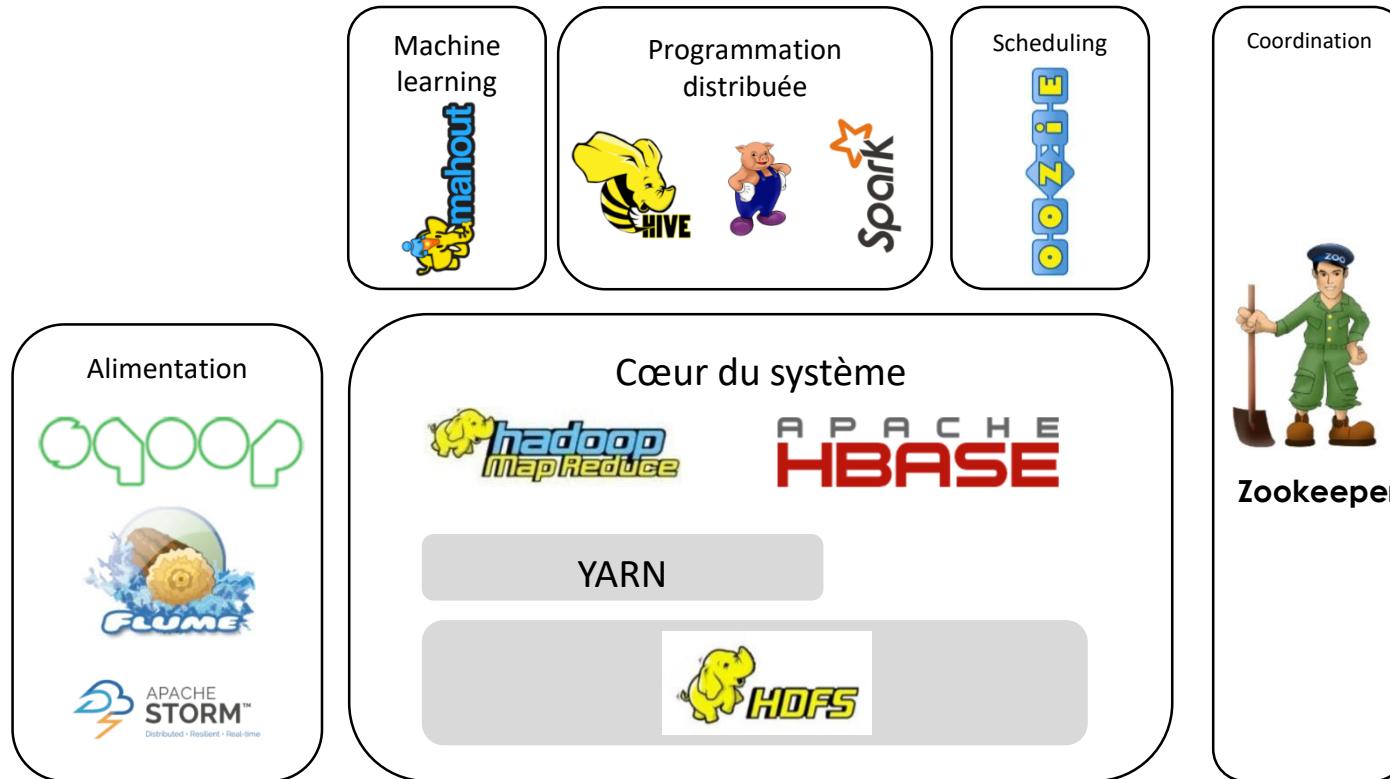


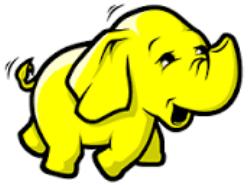
# Plan

## HDFS



# Ecosystème Hadoop





# Cluster

## Cluster

- Est un groupe de computers qui fonctionnent ensemble

## Node

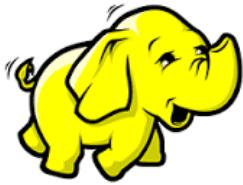
- Un computer individuel dans le cluster

## Daemon

- Est un program qui s'exécute dans le node

## Rack

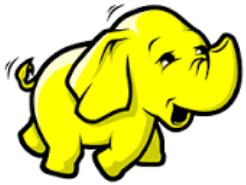
- Est un ensemble de node dans le cluster



# Qu'est-ce qu'HDFS

HDFS : Hadoop Distributed File System

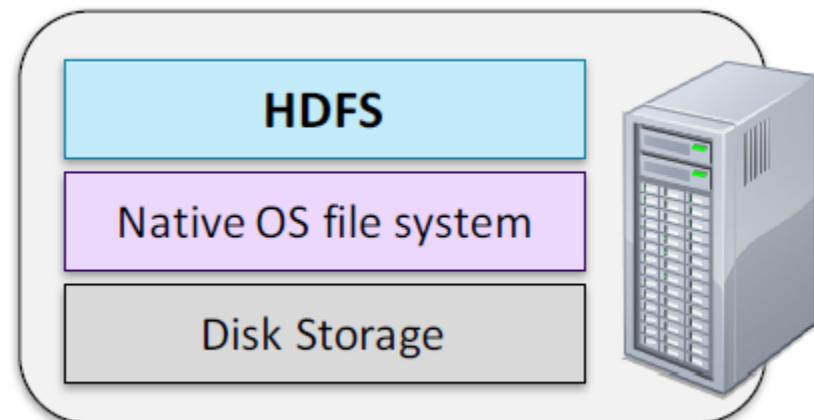
- Système de fichiers distribué sur lequel repose le cluster Hadoop
- Stockage primaire des applications Hadoop
- Distribue la donnée sur plusieurs machines
- Abstraction de la couche physique d'un cluster

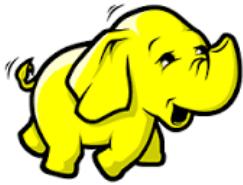


# Qu'est-ce qu'HDFS

HDFS : Hadoop Distributed File System

- HDFS est système de fichier écrit en Java
- Basé sur Google File System
- Situé en top d'un système de fichier native, comme ext3, ext4, or xfs
- Stockage répliqué d'une quantité de données volumineuse

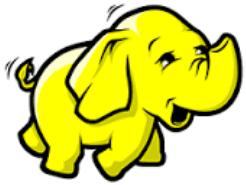




# Qu'est-ce qu'HDFS

Points communs avec des systèmes de fichiers classique

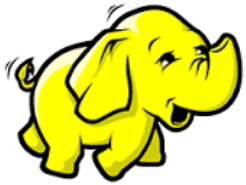
- Notion de blocs (plus petite unité de stockage exploitable)
- Métadonnées (permettant de retrouver les blocs)
- Droits/Privilèges
- Arborescence de répertoires



# Qu'est-ce qu'HDFS

Différences avec des systèmes de fichiers classique

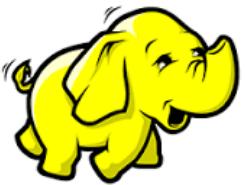
- Séparation avec le noyau du système d'exploitation
  - Portabilité et déploiement possible sur différents systèmes d'exploitation.
- Système distribué:
  - Chaque nœud d'un cluster correspond à un sous-ensemble du volume global de données du cluster.
  - Le simple ajout d'un nouveau nœud augmente la capacité de stockage globale du cluster.



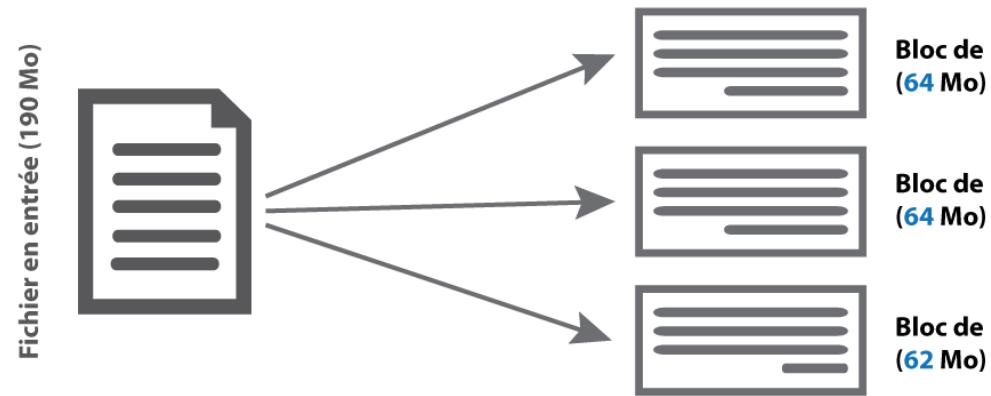
# Qu'est-ce qu'HDFS

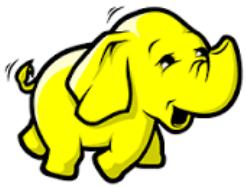
Différences avec des systèmes de fichiers classique

- Tailles des blocs:
  - Systèmes classiques : 4 Ko
  - HDFS : 64 Mo auparavant recommandé (128 Mo depuis 2013)
  - Possibilité de monter à 256 Mo, 512 Mo ou même 1 Go.
  - La raison principale de la grande taille des block: réduire le coût de SEEK TIME
    - Temps de recherche: 10ms
    - Taux de transfert : 100MB/s
    - Pour un ratio Temps de recherche/Taux de transfert = 1%
    - → Taille des blocks = 100MB



# Qu'est-ce qu'HDFS

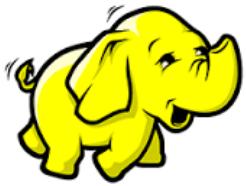




# Qu'est-ce qu'HDFS

## Taille des blocs

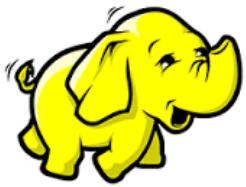
- HDFS utilise des tailles de blocs largement supérieures à ceux des systèmes classiques.
- Par défaut, la taille est à 64 Mo. Il est possible de monter à 128, 256, 512 Mo voire 1 Go.
- Alors que sur des systèmes classiques, la taille est généralement de 4 Ko.
- L'intérêt de fournir des tailles plus grandes permet de réduire le temps d'accès à un bloc.
- Augmentation de la taille des blocs
  - En fonction de l'augmentation de la rapidité de transfert des disques
  - En fonction de l'opération Map de Map/Reduce : map travaille bloc par bloc, ce qui réduit le nombre d'opérations map en parallèle et donc l'efficacité du cluster.



# Qu'est-ce qu'HDFS

## RéPLICATION

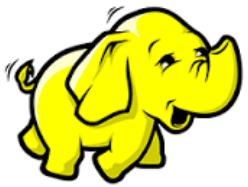
- HDFS fournit un système de réPLICATION des blocs dont le nombre de réPLICATIONS est CONFIGURABLE. Par défaut 3
- Pendant la phase d'écRITURE, chaque bloc dont le fichier est constitué est réPLIQUÉ sur plusieurs nœuds.
- Pour la phase de lecture, si un bloc est indisponible sur un nœud, les copies de ce bloc seront disponibles sur d'autres nœuds.



# Qu'est-ce qu'HDFS

## Avantage du système de bloc HDFS

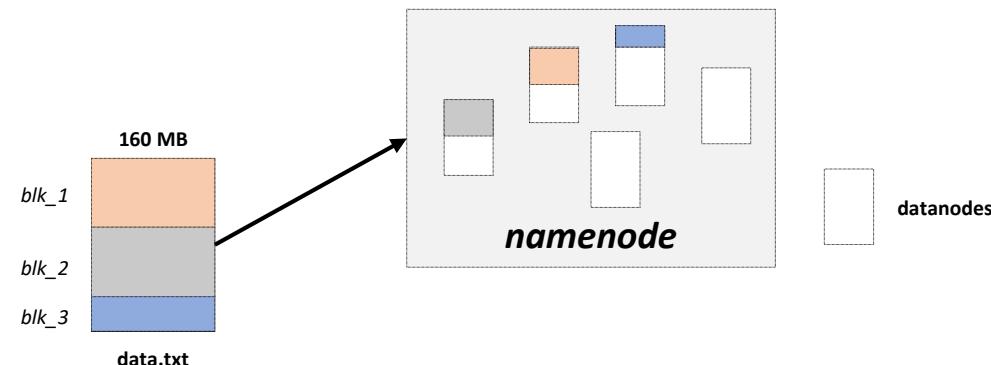
- Blocs de taille fixe
  - Très simple de calculer la capacité en nombre de blocs pouvant être stockés
- Simplification du stockage dans les datanodes
  - La gestion des métadonnées, des droits, etc., est le rôle des du namenode
- Si Taille du fichier < Taille d'un bloc, alors le fichier n'occupe pas tout le bloc
- Le fichier est stocké sur un ensemble de blocs et de nœuds du cluster
  - Simple de stocker un fichier dont la taille est plus importante que la taille d'un disque
- Facilité de réPLICATION d'un bloc d'un datanode à un autre

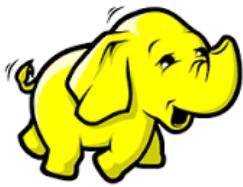


# Qu'est-ce qu'HDFS

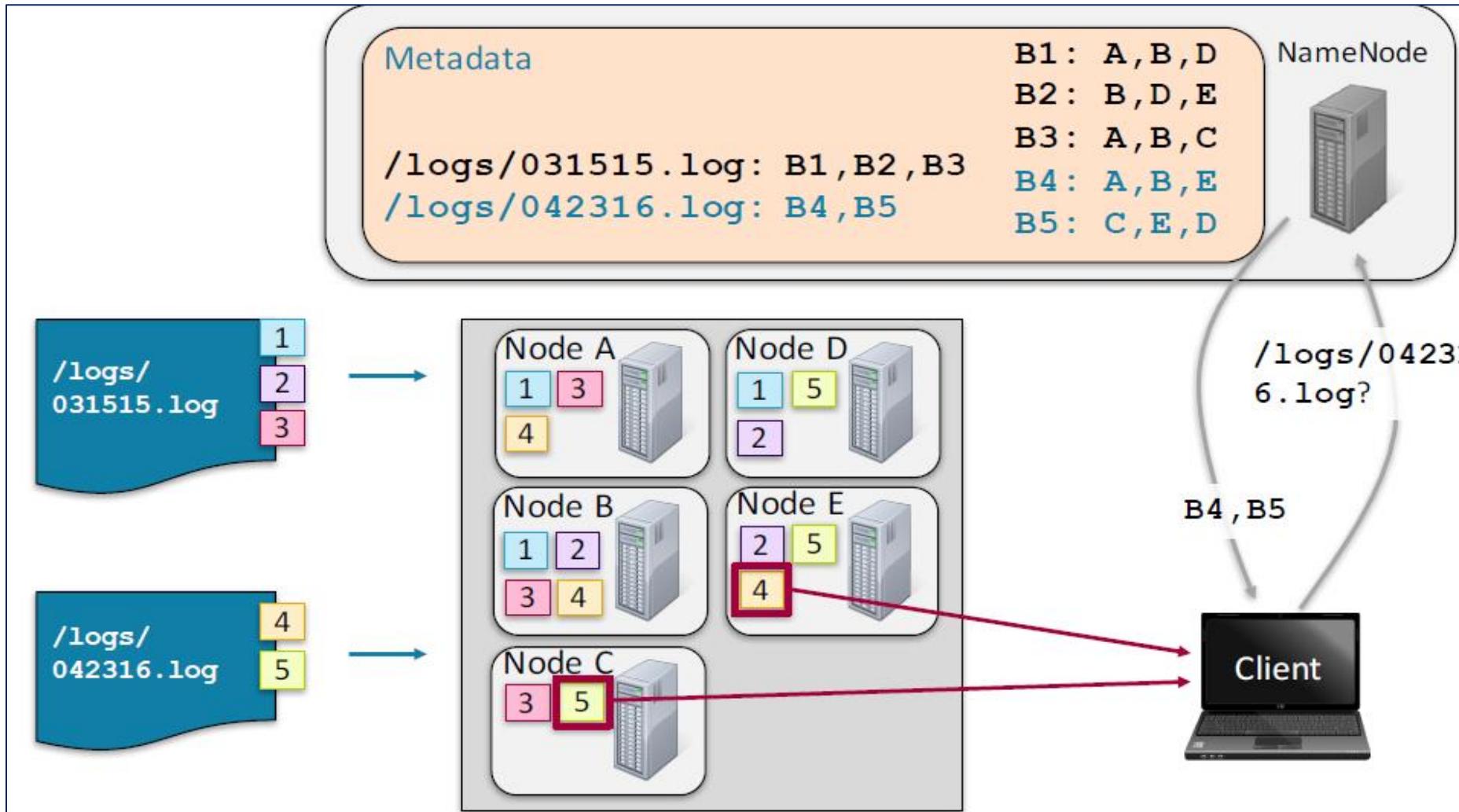
Avantage du système de bloc HDFS

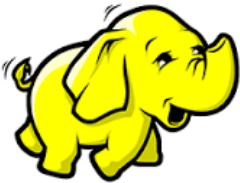
- RéPLICATION sur différents nœuds du cluster
  - Haute disponibilité
  - Tolérance aux pannes





# Qu'est-ce qu'HDFS

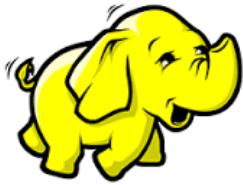




# Pourquoi HDFS ?

HDFS est conçu pour

- Stocker de très gros volumes de données (Peta Octet) sur un grand nombre de machines
- Fonctionner sur des commodity hardware
- Prévenir des pannes grâce à la réPLICATION
- DéTECTER des échecs et reprendre à partir de points d'arrêt = battement de cœur
- RéDUIRE la taille de la bande passante
- ConNAÎTRE l'emplacement de la donnée (possibilité de rechercher les emplacements de tous les blocs d'un fichier)
- AccÉDER directement à la donnée depuis un datanode
- GarANTIR la cohérence de la donnée
- Write-once-read-many : on écrit qu'une seule fois pour de nombreux accès à la donnée



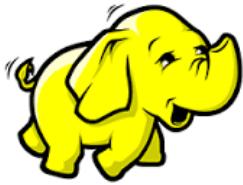
# Pourquoi HDFS ?

## Infrastructure commode

- Infrastructure sur des niveaux différents
- Plusieurs nœuds sur plusieurs racks
- Machines low-cost
- On priviléie beaucoup de machines moins performantes que peu de machines très performantes

## Write-once-read-many :

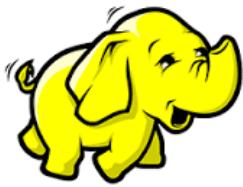
- Un fichier, une fois créé, écrit sur disque et fermé, ne sera plus modifié.
- Cela simplifie la gestion des problèmes de cohérence et d'accès concurrents.
- Les applications ont simplement la possibilité d'ajouter de la données, pas de la modifier.



## Pourquoi HDFS ?

HDFS N'EST PAS conçu pour

- Traiter une multitude de petits fichiers.
- Le namenode garde en mémoire les métadonnées.
- Le nombre de fichiers dans le système de fichiers est donc contraint par la mémoire disponible sur le namenode.
- Des modifications multiples des fichiers.
- Dans HDFS, les fichiers doivent être écrits par un unique utilisateur.



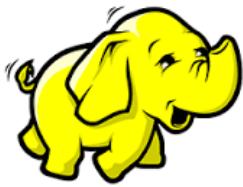
# Architecture HDFS

Qu'est ce qu'un namenode ?

- Service central/maître permettant la gestion du système de fichier HDFS

Fonctionnement d'un namenode :

- Le namenode gère :
  - L'espace de noms
  - L'arborescence du file system
  - La localisation des données
  - Les métadonnées des fichiers
  - La réPLICATION des blocs



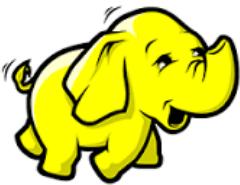
# Architecture HDFS

2 types de fichiers utilisés

- fsimage : fichier image de toutes les métadonnées du système à un instant  $t$
- edits\_xxx : fichiers de logs contenant toutes les modifications du file system

Au démarrage d'un Namenode

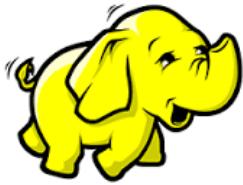
- Lecture de l'état du file system HDFS depuis le fsimage
- Application des modifications à partir des fichiers edits\_xxx
- Réécriture du nouvel état du fichier dans le fsimage



# Architecture HDFS

## Le Namenode

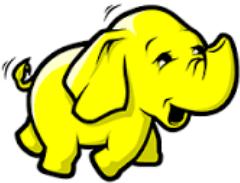
- Un *Namenode* est un service maître qui s'occupe de gérer l'état du système de fichiers.
- Il maintient l'arborescence du système de fichiers et les métadonnées de l'ensemble des fichiers et répertoires d'un système Hadoop.
- Le Namenode a une connaissance des *Datanodes* dans lesquels les blocs sont stockés.
- Ainsi, quand un client sollicite Hadoop pour récupérer un fichier, c'est via le Namenode que l'information est extraite.
- Ce Namenode va indiquer au client quels sont les Datanodes qui contiennent les blocs.
- Il ne reste plus au client qu'à récupérer les blocs souhaités



# Architecture HDFS

## Métadonnées

- Liste des fichiers stockés sur le file system
- Association entre les fichiers et leurs blocs
- Association entre les blocs et les Datanodes
- Localisation des blocs dans les Datanodes
- Reconstruite à chaque démarrage du Namenode (safe mode)



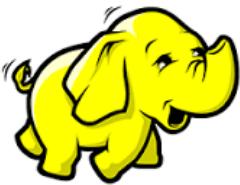
# Architecture HDFS

## Attributs des fichiers

- Date de création du fichier
- Date de suppression du fichier
- Date d'accès
- Facteur de réPLICATION
- etc...

## Gestion des Métadonnées

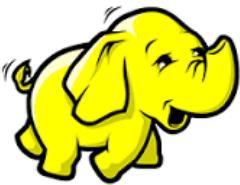
- Les métadonnées sont montées en mémoire
- Vigilance sur la taille de la mémoire du Namenode



# Architecture HDFS

## Métadonnées

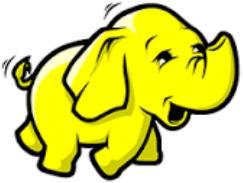
- Toutes les métadonnées sont stockées physiquement sur le disque système dans deux types de fichiers spécifiques edits\_xxx et fsimage.
- La connaissance de la position des blocs dans les Datanodes est reconstruite à chaque démarrage du Namenode dans un mode appelé safe mode.
- Pendant le safe mode, l'écriture sur HDFS est impossible, le Namenode charge les fichiers edits\_xxx et fsimage\_xxx et attend le retour des Datanodes sur la position des blocs.
- Une fois toutes les opérations réalisées, le safe mode est relâché et l'accès en écriture est de nouveau autorisé.
- La durée du safe mode peut être très long si on a beaucoup de fichiers à traiter.



# Architecture HDFS

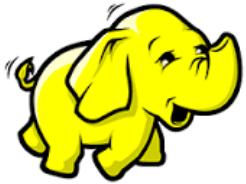
## Gestion de la mémoire

- Le Namenode charge tout en mémoire.
- Cela devient donc problématique si vous avez énormément de petits fichiers à gérer. D'après la documentation officielle de Cloudera, chaque fichier, répertoire et bloc dans HDFS est représenté comme un objet dans la mémoire et occupe 150 octets.
- Exemple : S'il y a 10 millions de fichiers à gérer, le Namenode devra disposer d'un minimum de 1,5 Go de mémoire.
- C'est donc un point important à prendre en compte lors du dimensionnement un cluster.

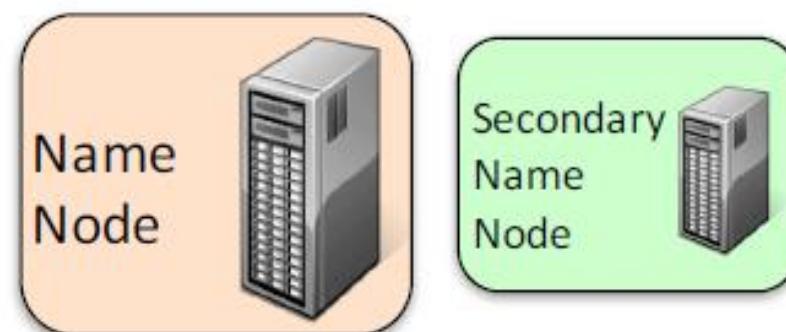


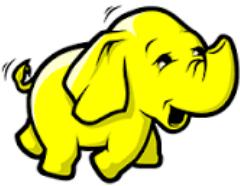
# Architecture HDFS

- Le Namenode reste en exécution tout le temps, s'il s'arrête, le cluster sera inaccessible: Single-Point-of-Failure(SPOF)
- Deux Namenode: Active et Standby et un troisième Secondary Namenode
- Le nœud maître appelé Namenode contient et stocke tous les noms et blocs des fichiers ainsi que leur localisation dans le cluster. On peut donc le voir comme un gros annuaire.
- Une autre machine, appelée Secondary Namenode sert de Namenode de secours en cas de défaillance du nœud maître et il a donc pour rôle de faire des sauvegardes régulières de l'annuaire et la fusion des fichiers fsimage et edits
- En cas de défaillance, le Namenode est remplacé manuellement par le Secondary
- Standby Namenode : introduit avec Hadoop 2, il fait le même travail d'un Secondary Namenode, et remplace automatiquement le Namenode en cas de panne.



# Architecture HDFS

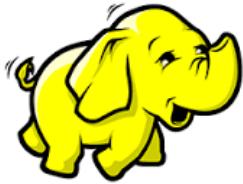




# Architecture HDFS

## Namenode

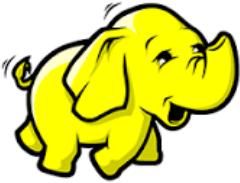
- Namenode gère l'espace de noms, l'arborescence du système de fichiers et les métadonnées des fichiers et des répertoires
- Il centralise la localisation des blocs de données répartis dans le cluster.
- Quand un client sollicite Hadoop pour récupérer un fichier, c'est via le Namenode que l'information est extraite. Ce Namenode va indiquer au client quels sont les Datanodes qui contiennent les blocs.
- Le Namenode reçoit régulièrement un *battement de cœur* et un *Blockreport* de tous les Datanodes dans le cluster afin de s'assurer que les Datanodes fonctionnent correctement.
- Un *Blockreport* (ou rapport de bloc) contient une liste de tous les blocs d'un Datanode.



# Architecture HDFS

## Namenode

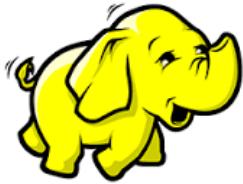
- En cas de défaillance du Datanode, le Namenode choisit de nouveaux Datanodes pour de nouvelles réPLICATIONS de blocs de données, équilibre la charge d'utilisation des disques et gère également le trafic de communication des Datanodes.
- Les métadonnées sont stockées sur disque dur (fichier fsimage) et chargées dans la mémoire vive du Namenode lors du démarrage du cluster



# Architecture HDFS

## Secondary Namenode

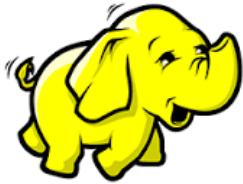
- Secondary Namenode donne l'impression que c'est un substitut du Namenode.
- Mais ce n'est pas le cas, le Secondary Namenode effectue des tâches de maintenance pour le compte du Namenode.
- Plus précisément, le Secondary Namenode met à jour le fichier `fimage` à intervalle régulier en y intégrant le contenu des fichiers `edits`.
- Le Secondary Namenode intervient soit : lorsque le fichier `edits` atteint une taille prédéfinie ou à intervalle régulier (par exemple une fois par heure).
- Représente un Backup du Namenode
- Prend le relais lors de problèmes sur le Namenode principal
- Vérifie périodiquement l'état du Namenode



# Architecture HDFS

## Inconvénients

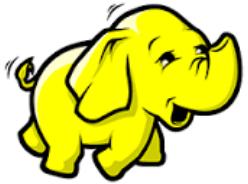
- Point unique de défaillance, si le Namenode est arrêté, il n'y a plus moyen d'accéder aux données.
- Accès impossible aux données si le Namenode est arrêté



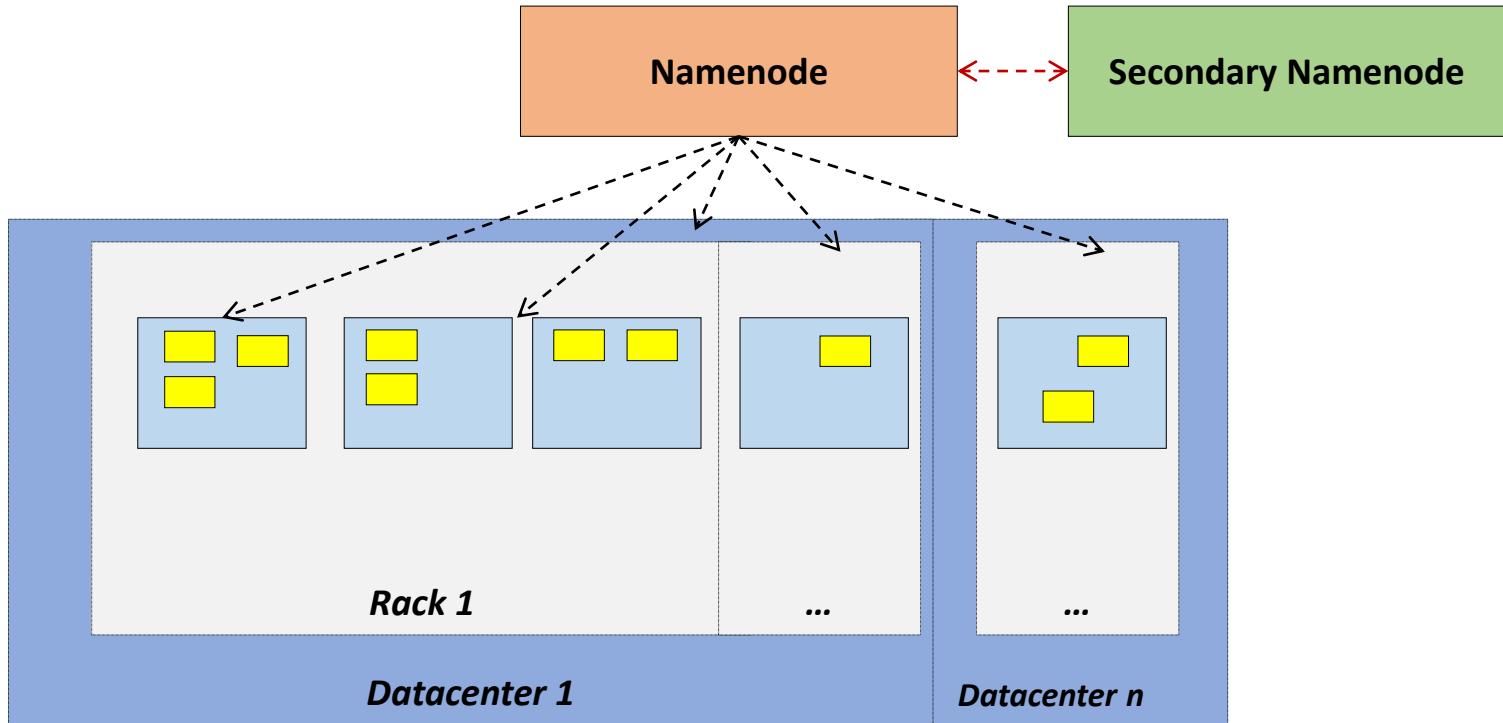
# Architecture HDFS

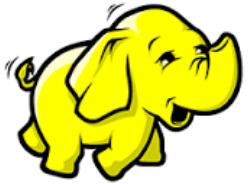
## Datanode

- Serveur de bloc / serveur de données
  - Stockage des blocs de données sur le file system local
- Sous les ordres du Namenode
- Opérations de lectures et d'écritures
  - Lecture : Transmission au client des blocs du fichier demandé
  - Ecriture : Retourne au Namenode l'emplacement des blocs créés
- Initialisation du Namenode (safe mode)
  - Retourne au Namenode l'emplacement des blocs qui sont stockés
- Retourne de manière périodique la liste des blocs stockés (Block Report)

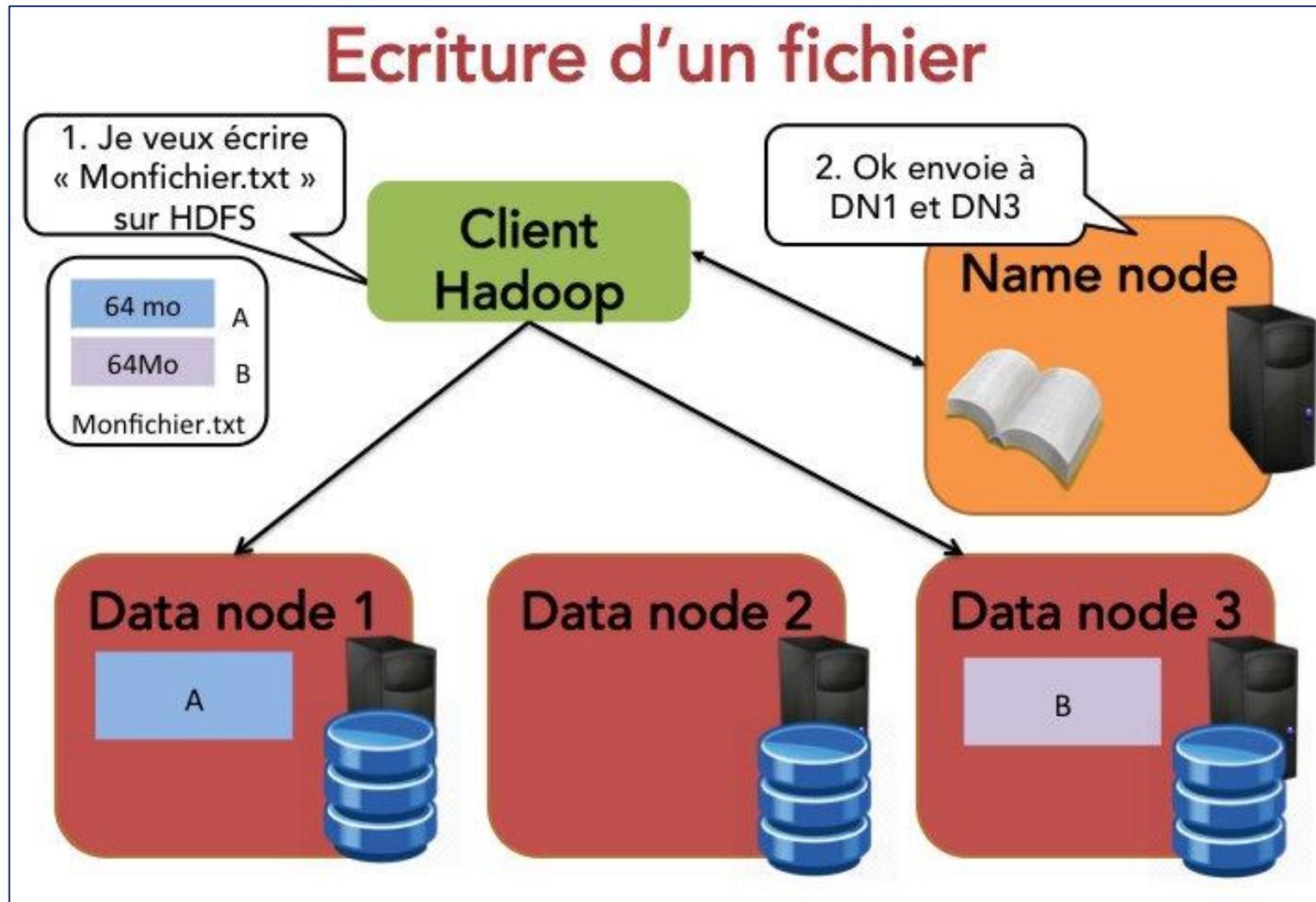


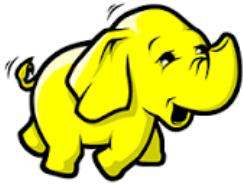
# Architecture HDFS





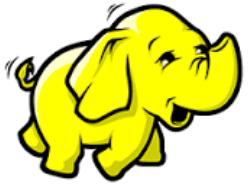
# Mode d'écriture HDFS





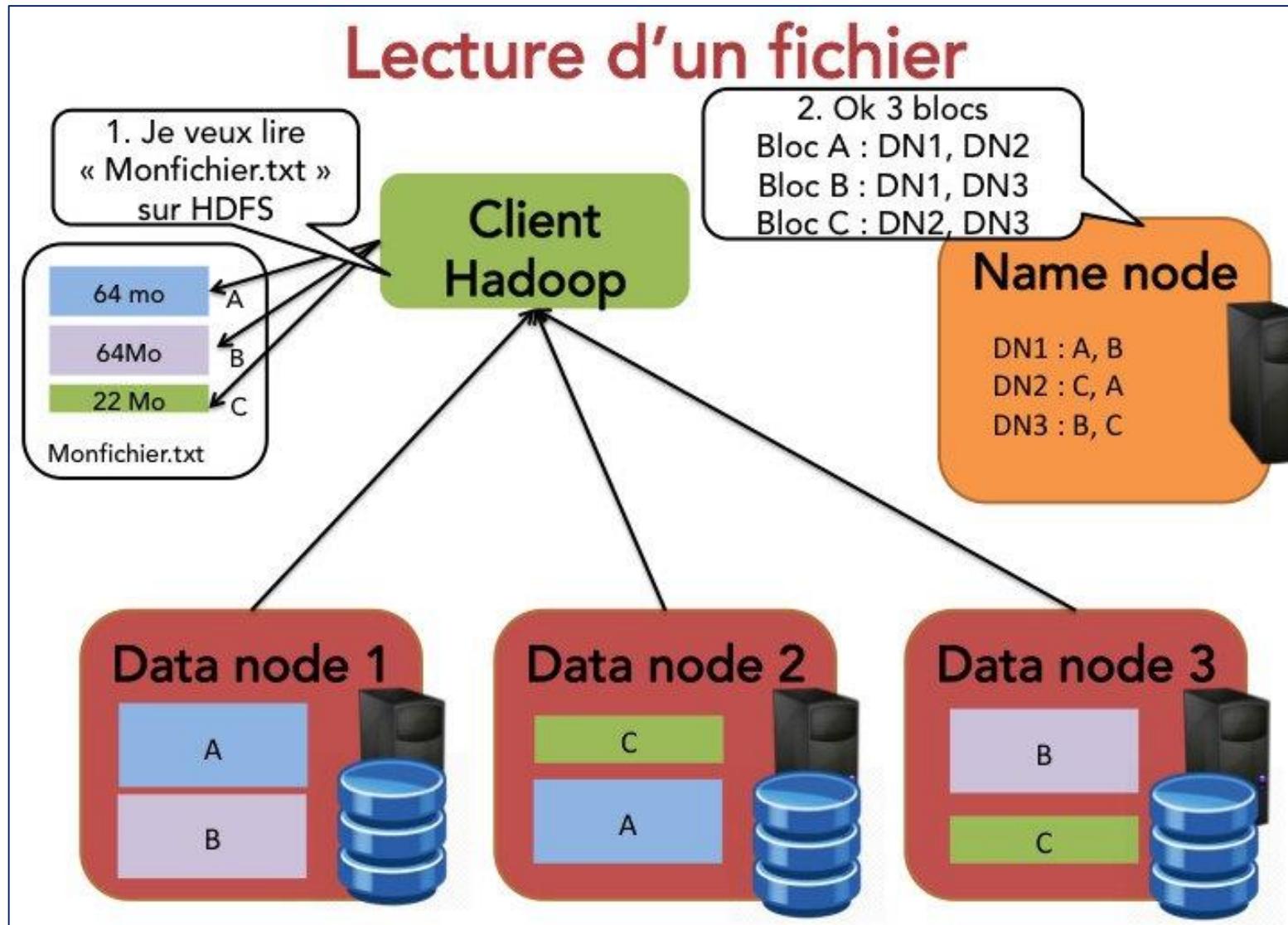
## Mode d'écriture HDFS

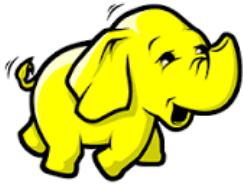
- 1- Le client indique au Namenode qu'il souhaite écrire un bloc.
- 2- Le Namenode indique le Datanode à contacter.
- 3- Le client envoie le bloc au Datanode.
- 4- Les Datanodes répliquent les blocs entre eux.
- 5- Le cycle se répète pour le bloc suivant.



# Mode de lecture HDFS

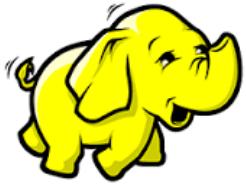
## Lecture d'un fichier



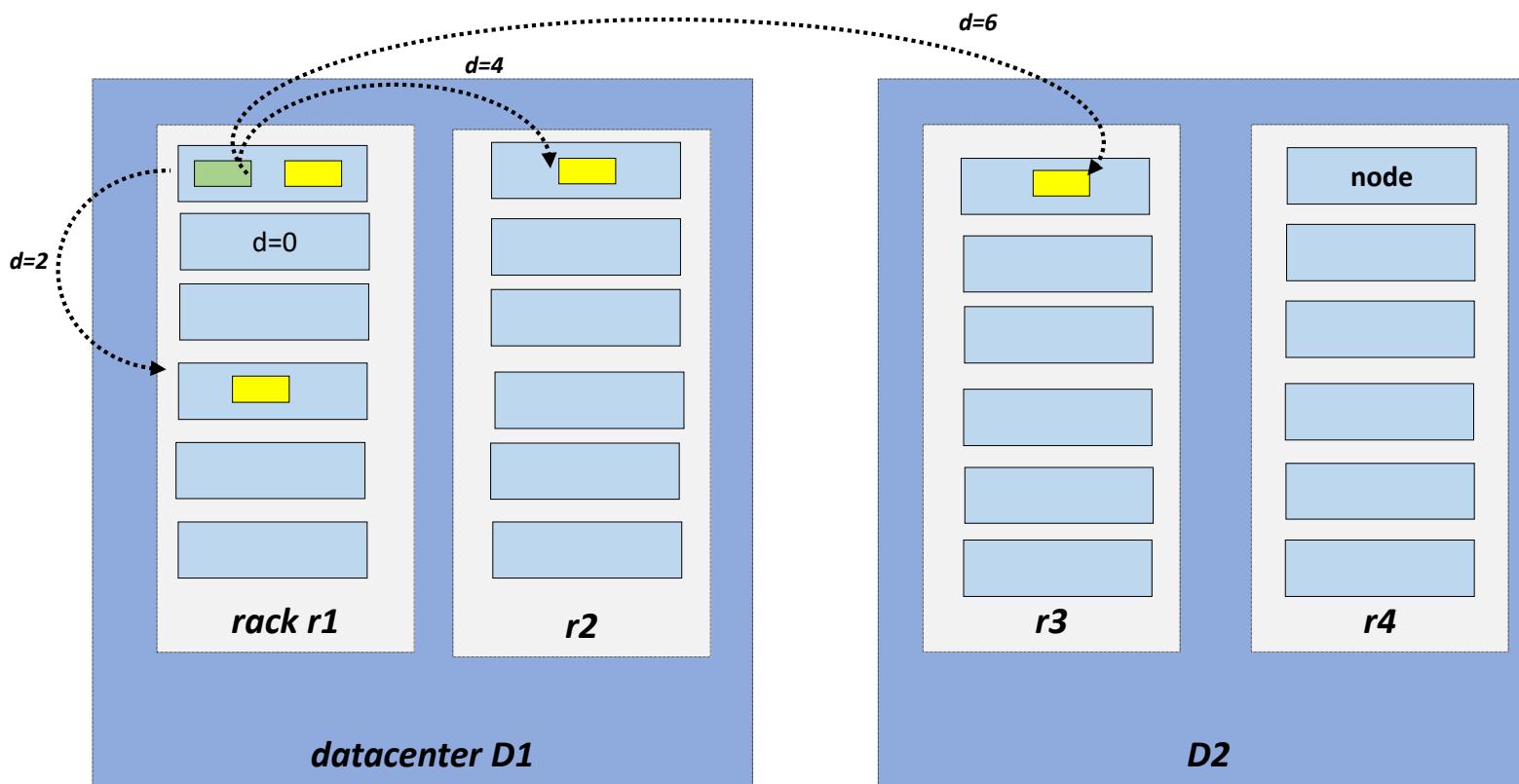


## Mode de lecture HDFS

- 1- Le client indique au Namenode qu'il souhaite lire un fichier.
- 2- Le Namenode indique sa taille ainsi que les différents Datanodes contenant les blocs.
- 3- Le client récupère chacun des blocs sur l'un des Datanodes.
- 4- Si le Datanode est indisponible, le client en contacte un autre.

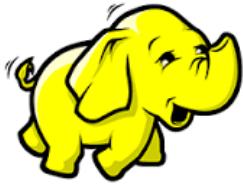


# Stratégie de réPLICATION HDFS



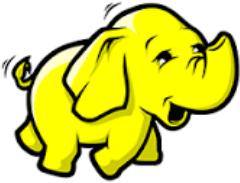
## *Distances d :*

- $(/d1/r1/n1,/d1/r1/n1) = 0$
- $(/d1/r1/n1,/d1/r1/n2) = 2$
- $(/d1/r1/n1,/d1/r2/n3) = 4$
- $(/d1/r1/n1,/d1/r3/n4) = 6$



## Stratégie de réPLICATION HDFS

- D=0 : processus sur le même nœud
- D=2 : différents nœuds sur le même rack
- D=4 : différents nœuds sur différents racks
- D=6 : différents nœuds sur différents Datacenter
- Actuellement Hadoop n'est pas encore prévu pour fonctionner entre plusieurs data center



# Stratégie de réPLICATION HDFS

## Principe de la réPLICATION

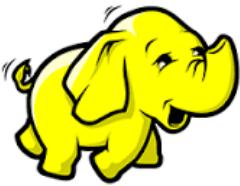
- Copie de tous les blocs de données sur différents nœuds.
- Permet d'assurer la cohérence des données lorsqu'un nœud tombe.
- Version originale des blocs + 3 copies (réplicas) par défaut qui est paramétrable

## Disponibilité des réplicas

- L'emplacement des réplicas est paramétrable
- Les clients accèdent à la version de la donnée la plus proche.

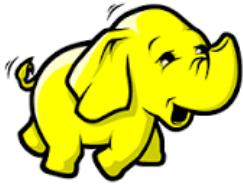
## Rôle du Namenode dans la réPLICATION

- Moteur de la réPLICATION.
- Déetecte les Datanodes défectueux et les Datanodes sur lesquels seront stockés les réplicas
- Gestion de la charge



## Stratégie de réPLICATION HDFS

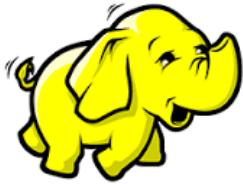
- Trade-off entre fiabilité/bande-passante en écriture et en lecture
- Ex: replica sur le même nœud, impact minimal sur la bande passante mais pas de redondance
- Ex: replica sur 2 Datacenter différents, redondance garantie si perte d'un Datacenter mais impact maximal sur la bande passante (dépendance de la connexion entre les 2 Datacenter)
- Stratégie Hadoop par défaut:
  - Premier réplica sur le nœud du client (un client extérieur a un nœud choisi aléatoirement en évitant les nœuds trop chargés ou trop occupés)
  - Deuxième réplica sur un nœud aléatoire « off-rack »
  - Troisième réplica sur un nœud aléatoire du même rack que le deuxième
  - D'autres réplicas sont créés aléatoirement sur le cluster



# Problématique des petits fichiers

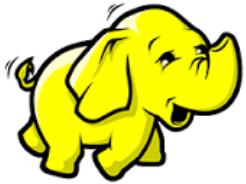
## Inconvénients des petits fichiers

- Petit fichier = fichier plus petit qu'un block (par défaut 64Mb)
- Chaque fichier représente un objet pour le Namenode
  - 150 bytes par objet
  - 10 millions de petits fichiers → 3Gb de mémoire utilisés dans le Namenode
- HDFS est optimisé pour de gros fichiers
  - La lecture d'une multitude de petits fichiers implique beaucoup de temps de recherche et de saut d'un Datanode à un autre



## Quelques commandes HDFS

- ***hdfs dfs -ls*** : Lister les fichiers et sous-répertoires d'un répertoire
- ***hdfs dfs -cat*** : afficher les contenu d'un fichier
- ***hdfs dfs -text*** : afficher les contenu d'un fichier même compressé
- ***hdfs dfs -chmod, chgrp, -chown*** : modifier les droits
- ***hdfs dfs -put, -get, -copyFromLocal, -copyToLocal*** : Copie de fichiers entre file system local et HDFS
- ***hdfs dfs -mv*** : Déplacement de fichiers entre file system local et HDFS
- ***hdfs dfs -stat*** : statistiques sur les ressources
- ***hdfs dfs -rm*** : supprimer un fichier

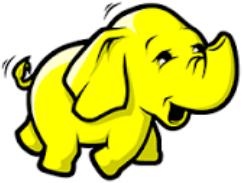


# Hadoop Data Storage Formats

Les formats les plus courants

- Text
- SequenceFiles : Stockage key/value sous format binaire
- Apache Avro : encodage binaire, schéma Json embarqué dans le fichier
- Apache Parquet : schéma embarqué dans le fichier, stockage en colonne

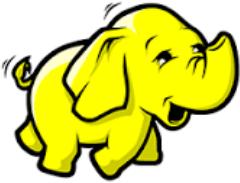
Feature	Text	Sequence Files	Avro	Parquet
Supported by many tools	✓		✓	✓
Good performance at scale		✓	✓	✓
Binary format		✓	✓	✓
Embedded schema			✓	✓
Columnar organization				✓



# Hadoop Data Storage Formats

## Text

- le type le plus basique,
- Lire et écrire avec n'importe quelle langage de programmation
- Généralement séparé par des virgule (csv) ou aussi tabulation (tsv)
- Un type lisible par l'être humain,
- Difficulté de stocker des images ou des objets binaire
- Bon interopérabilité, pauvre en performance



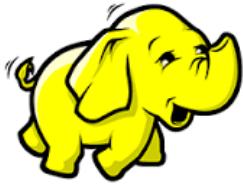
# Hadoop Data Storage Formats

## SequenceFiles

- Stockage key/value sous format binaire
- Possibilité de stocker des images binaires
- Lié fortement à Hadoop
- Bonne performance, pauvre en interopérabilité

## Apache Avro

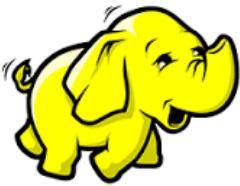
- Stockage efficace parce qu'il est binaire, le bon choix pour stocker en Hadoop
- Supporté par plusieurs outils autre que Hadoop
- Idéal pour un stockage long terme et des données importants
- Schéma Json embarqué dans le fichier
- Excellent interopérabilité et performance



# Hadoop Data Storage Formats

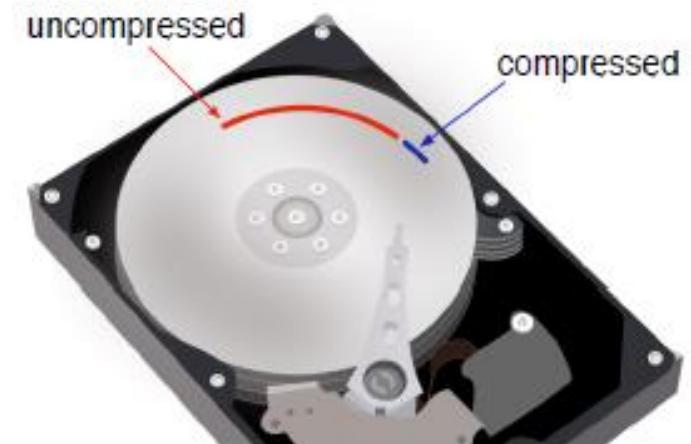
## Parquet

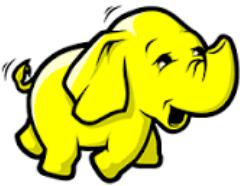
- Support un stockage colonne
- Très efficace pour la sélection des données
- Développé par Cloudera et twitter
- Supporté par : Spark, MapReduce, Hive, Pig et Impala
- Schéma embarque dans le fichier
- Réduire l'espace de stockage, et augmente la performance
- Excellent interopérabilité and performance
- Outil : parquet-tools



# Hadoop Compression de données

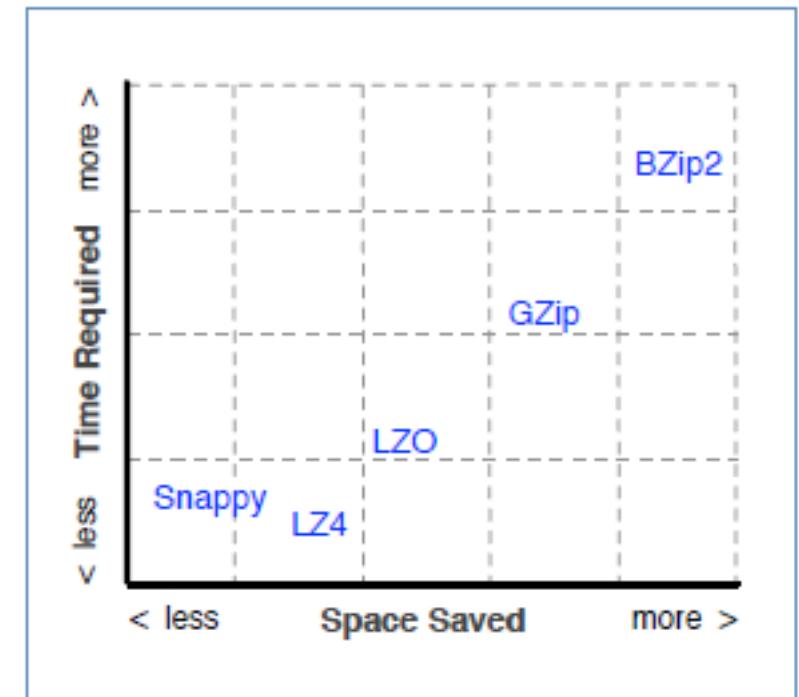
- Cela réduit la quantité d'espace disque nécessaire pour stocker les données
- Compromis entre le temps CPU et la bande passante/espace de stockage
- Les algorithmes agressifs prennent beaucoup de temps, mais économisent plus d'espace
- Les algorithmes moins agressifs économisent moins d'espace mais sont plus rapides
- Peut améliorer considérablement les performances
- Gérer plus de données par opération E/S
- Améliorer les performances des transferts réseau

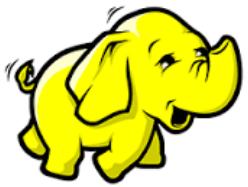




# Hadoop Compression de données

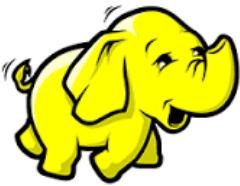
- La mise en œuvre d'un algorithme de compression est connu comme un codec
- De nombreux codecs sont couramment utilisés avec Hadoop, mais tous les outils Hadoop ne sont pas compatibles avec tous les codecs
- BZip2 économise le plus d'espace, mais LZ4 et Snappy sont beaucoup plus rapides
- Impala soutient Snappy mais pas LZ4





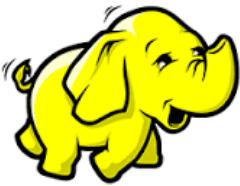
# Plan

## MapReduce



# Présentation de MapReduce

- Publication par Google en 2004 pour résoudre son problème d'indexation
- Traite 20 PB de données par jour ( 1PB = 1000TB)
- Un modèle de programmation massivement parallèle, souvent distribué qui est adapté au traitement de très grandes quantités de données (typiquement supérieures en taille à 1TB)
- PetaBytes des données
- Milliers de nœuds
- Les programmes qui adoptent ce modèle sont automatiquement parallélisés et exécutés sur des clusters d'ordinateurs



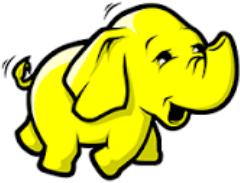
# Objectifs

Évolutivité à de grands volumes de données

- Scan 100 TB sur 1 nœud @ 50 Mo / s = 24 jours
- Scan sur un Cluster de 1000 nœuds = 35 minutes

Coût-efficacité

- Nœuds: des produits de base (pas cher, mais peu fiables)
- Réseau de machines standards
- La tolérance aux pannes automatique (Grace à la redondance de données sur les nœuds, besoins d'administration): une tâche est transférée d'un nœud à l'autre, et si le nœud principal remarque qu'un nœud a été silencieux pendant un intervalle de temps plus long que prévu, le nœud principal assigne à nouveau la tâche à un autre nœud
- Facile à utiliser (moins de programmation parallèle et distribué)



# Exemples d'utilisation

## Chez Google

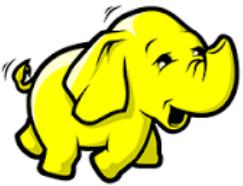
- Indexation pour Google Search
- Regroupement des articles pour Google News
- Statistique de la traduction automatique

## Chez Yahoo

- Indexation pour Yahoo! Search
- Détection de spam pour Yahoo! Mail

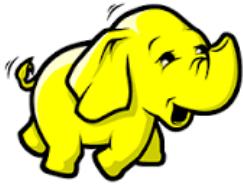
## Chez Facebook

- L'exploration de données
- L'optimisation de la pub
- Détection de spam



# Implémentation

- Des librairies MapReduce existent pour C++, C#, Erlang, Java, Python, Ruby, R
- La plus connue est Apache Hadoop
- Hadoop peut gérer des milliers de nœuds et de péta-octets de données, tout le travail de distribution, de répartition de charge, de synchronisation et de gestion d'erreur est effectué automatiquement
- Principaux composants de Hadoop (open source, distribué par Apache):
  - HDFS est un système de fichier distribué.
  - MapReduce est un modèle de traitement des données distribué.



# Principales fonctionnalités

Parallélisation automatique des programmes Hadoop

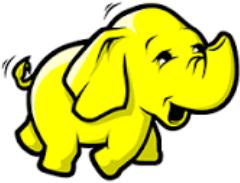
- HDFS se charge de la répartition et de la réPLICATION des données.
- Le maître divise le travail en jobs parallèles et les répartit.
- Le maître collecte les résultats et gère les pannes des nœuds.

Gestion transparente du mode distribué

Tolérance aux pannes

Un programme Hadoop se divise généralement en trois parties

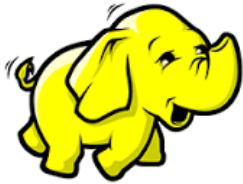
- Le driver, s'exécute sur la machine client et configure le job et le soumettre pour exécution.
- Le Mapper est chargé de lire les données stockées sur HDFS et les traiter.
- Le Reducer chargé de consolider les résultats issus du mapper puis de les écrire sur HDFS



## Principales fonctionnalités

Gestion transparente du mode distribué

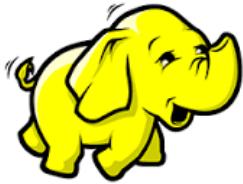
- Transparence d'accès: les ressources locales et distantes doivent être accessibles de la même manière.
- Transparence de localisation: les ressources doivent être accessibles quelque soit leur localisation physique.
- Transparence de concurrence: plusieurs processus doivent opérer de manière concurrentielle sans interférences entre eux.
- Transparence de réplication: plusieurs instances des ressources doivent être déployées pour assurer la fiabilité du système.



## Principales fonctionnalités

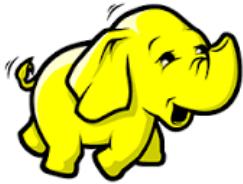
### Gestion transparente du mode distribué

- Transparence de panne : une panne ne doit pas bloquer le fonctionnement global du système.
- Transparence de mobilité : les ressources et clients doivent pouvoir être mobiles sans affecter le fonctionnement global.
- Transparence de performance : le système doit pouvoir être reconfigurable pour assurer les montées en charge.
- Transparence d'échelle : le système et les applications doivent pouvoir supporter les changements d'échelles sans modification interne des algorithmes.

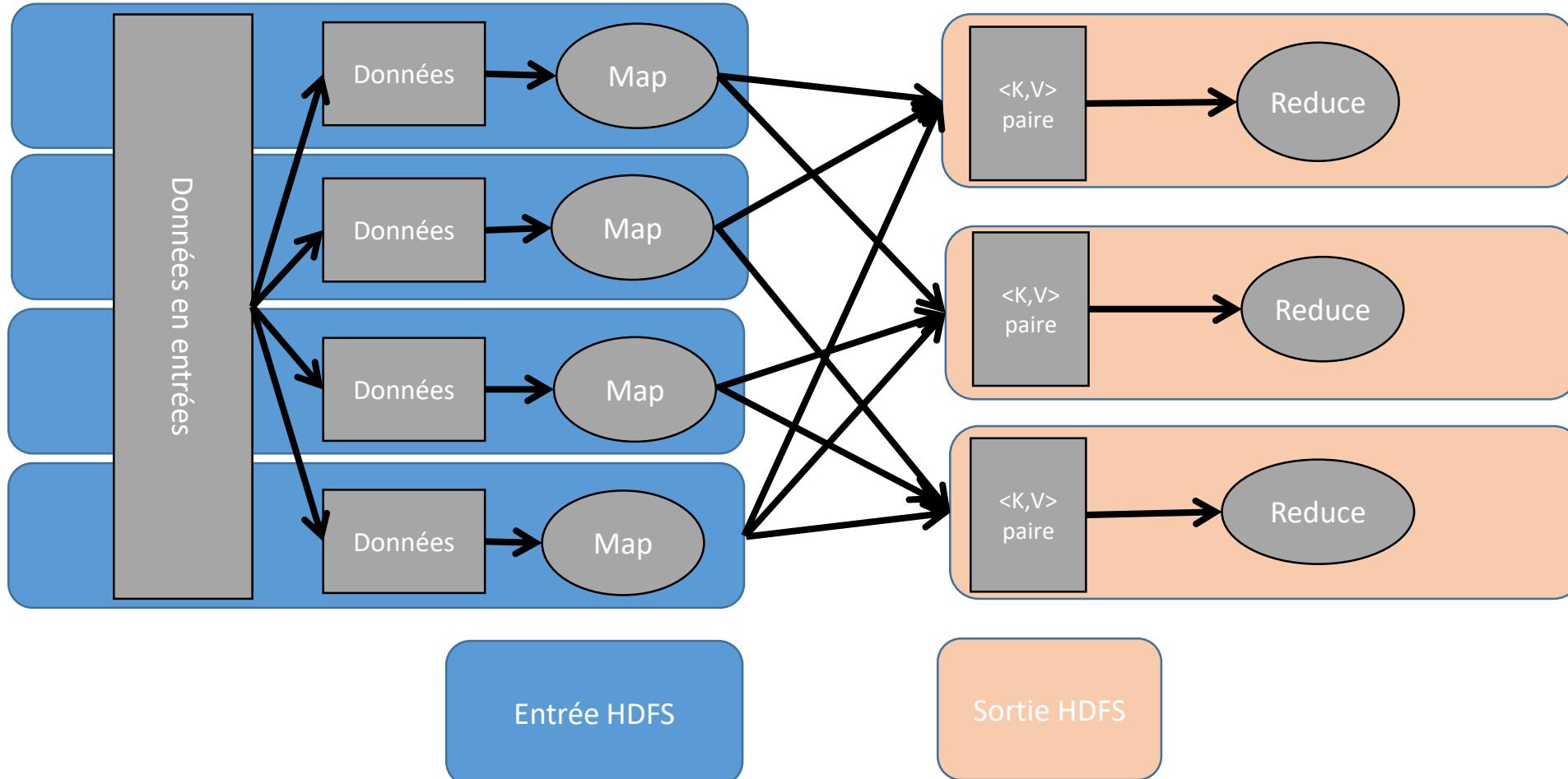


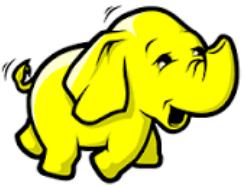
## Schéma général

- MapReduce définit deux opérations distinctes à effectuer sur les données d'entrée:
  - MAP, va transformer les données d'entrée en une série de couples clef/valeur.
  - REDUCE, va appliquer un traitement à toutes les valeurs de chacune des clefs distinctes produite par l'opération MAP.
- MAP va regrouper les données en les associant à des clefs, choisies de telle sorte que les couples clef/valeur aient un sens par rapport au problème à résoudre.
- Par ailleurs, cette opération doit être parallélisable, on doit pouvoir découper les données
- Au terme de l'opération REDUCE, on aura un résultat pour chacune des clefs distinctes.

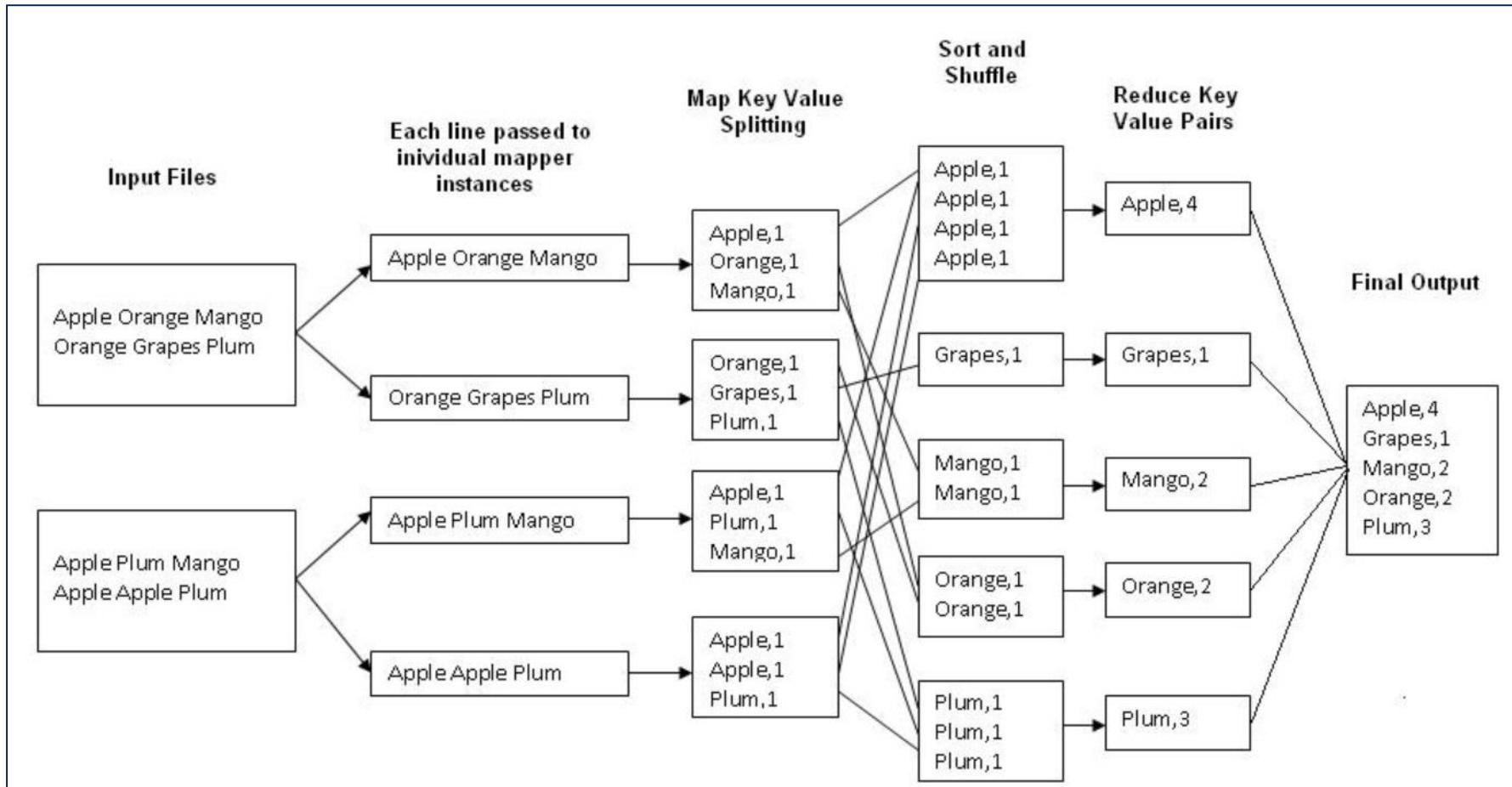


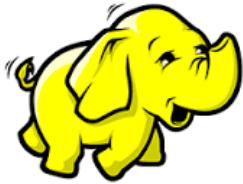
# Schéma général





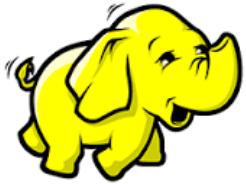
# Exemple de Wordcount



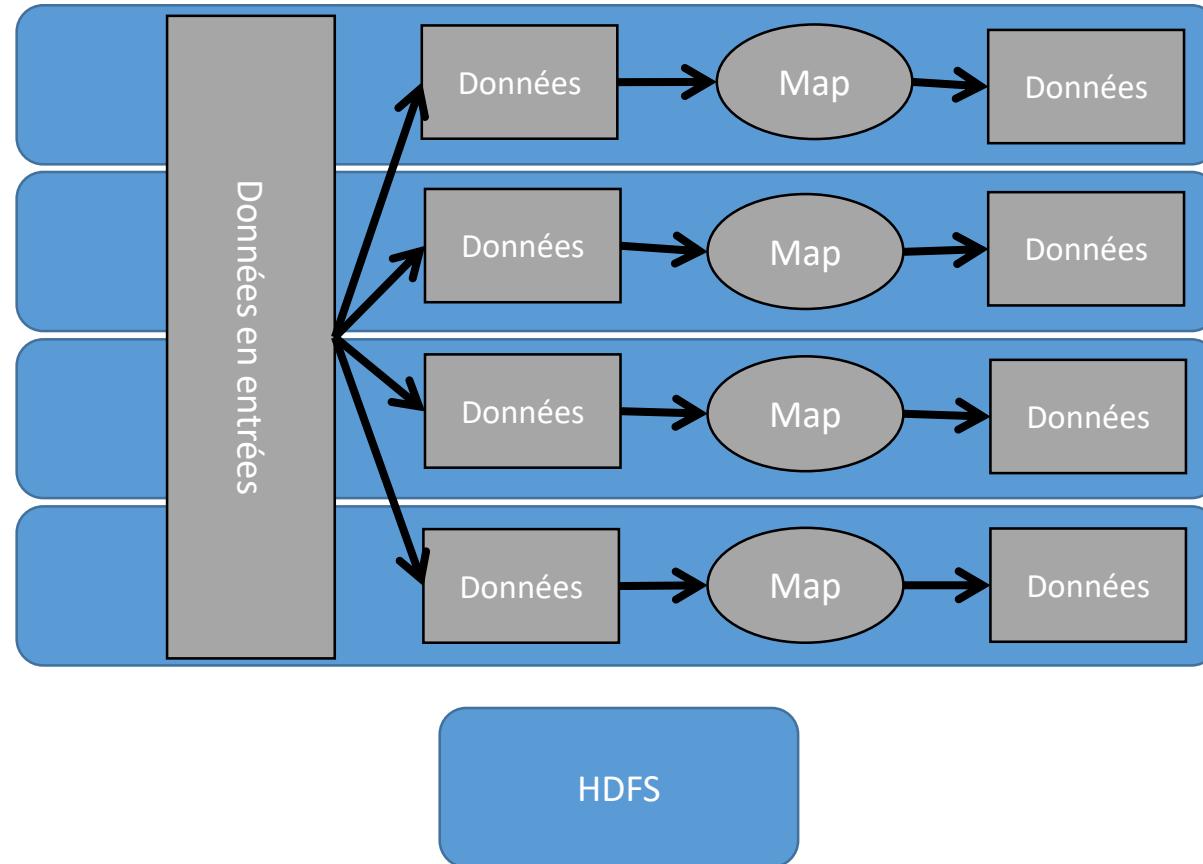


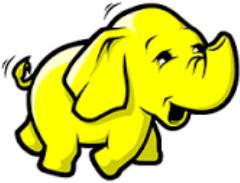
## Exemple de WordCount

- On distingue donc 4 étapes distinctes dans un traitement MapReduce:
  - Découper, *split*, les données d'entrée en plusieurs fragments.
  - Mapper chacun de ces fragments pour obtenir des couples (clef /valeur).
  - Grouper, *shuffle*, ces couples (clef ; valeur) par clef.
  - Réduire, *reduce*, les groupes indexés par clef



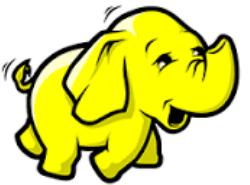
# Schéma sans opération reduce



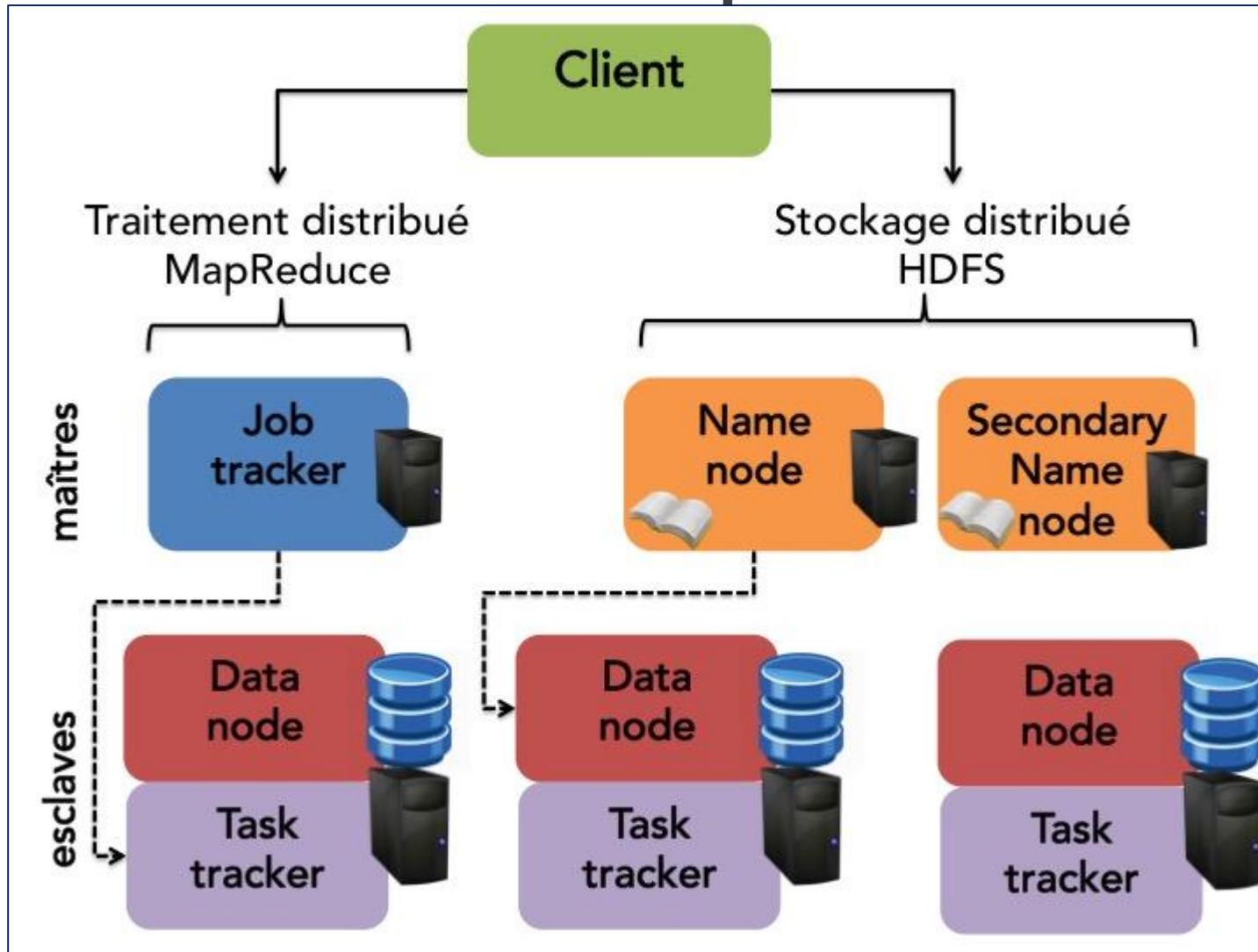


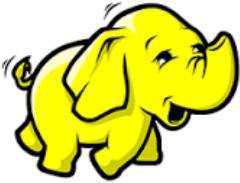
# Architecture MapReduce

- Architecture Master/slave
- Master node contient
  - Job tracker node (MapReduce layer)
  - Name node (HFDS layer)
- Plusieurs Slave nodes contiennent
  - Task tracker node (MapReduce layer)
  - Data node (HFDS layer)
- Job & task tracker nodes pour MapReduce
- Name & data nodes pour HFDS



# Architecture MapReduce





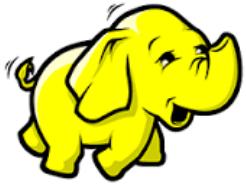
# Architecture MapReduce

## MapReduce

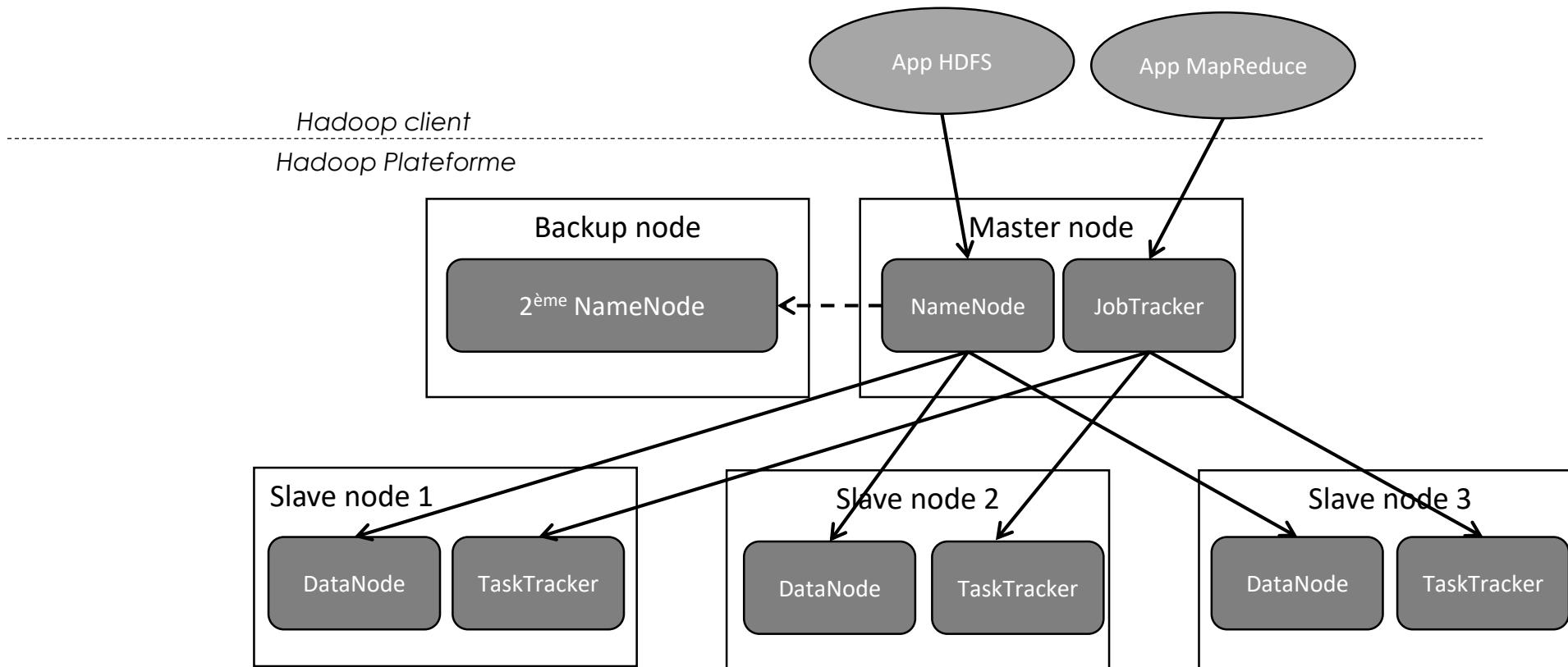
- L'ordonnancement des traitements
- La localisation des fichiers
- La distribution de l'exécution.

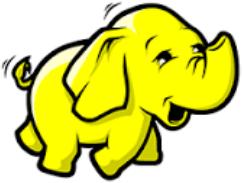
## HDFS

- Distribué: les données sont réparties sur les machines du cluster.
- Répliqué: en cas de panne, aucune donnée n'est perdue.
- Optimisé pour la colocalisation des données et des traitements: la donnée ne bouge pas d'un serveur à l'autre, c'est le code de traitement, d'un volume toujours très faible qui se déplace



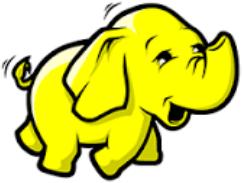
# Architecture MapReduce





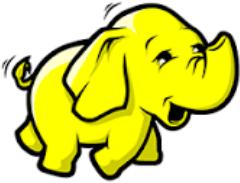
# Architecture MapReduce

- 1- Le client (un outil Hadoop console) va soumettre le travail à effectuer au JobTracker: une archive java.jar implémentant les opérations Map et Reduce. Il va également soumettre le nom des fichiers d'entrée, et l'endroit où stocker les résultats.
- 2- Le JobTracker communique avec le Namenode HDFS pour savoir où se trouvent les blocs correspondant aux noms de fichiers donnés par le client.
- 3- Le JobTracker, à partir de ces informations, détermine quels sont les nœuds TaskTracker les plus appropriés, c'est à dire ceux qui contiennent les données sur lesquelles travailler sur la même machine, ou le plus proche possible (même rack/rack proche).
- 4- Pour chaque « morceau » des données d'entrée, le JobTracker envoie au TaskTracker sélectionné le travail à effectuer (MAP/REDUCE, code Java) et les blocs de données correspondant.



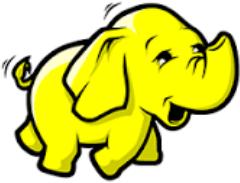
# Architecture MapReduce

- 5- Le JobTracker communique avec les nœuds TaskTracker en train d'exécuter les tâches. Ils envoyent régulièrement un *heartbeat*, un message signalant qu'ils travaillent toujours sur la sous-tâche reçue. Si aucun heartbeat n'est reçu dans une période donnée, le JobTracker considère la tâche comme ayant échouée et donne le même travail à effectuer à un autre TaskTracker.
- 6- Si par hasard une tâche échoue (erreur java, données incorrectes, etc.), le TaskTracker va signaler au JobTracker que la tâche n'a pas pu être exécuté. Le JobTracker va alors décider de la conduite à adopter: redonner la sous-tâche à un autre TaskTracker, demander au même TaskTracker de réessayer, marquer les données concernées comme invalides, etc. il pourra même blacklister le TaskTracker concerné comme non fiable dans certains cas.



# Architecture MapReduce

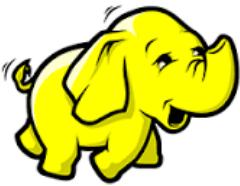
- 7- Une fois que toutes les opérations envoyées aux TaskTracker (MAP + REDUCE) ont été effectuées et confirmées comme effectuées par tous les nœuds, le JobTracker marque la tâche comme « effectuée ». Des informations détaillées sont disponibles (statistiques, TaskTracker ayant posé problème, etc.). Par ailleurs, on peut également obtenir à tout moment de la part du JobTracker des informations sur les tâches en train d'être effectuées: étape actuelle (MAP, SHUFFLE, REDUCE), pourcentage de complétion, etc.
- La soumission du .jar, l'obtention de ces informations, et d'une manière générale toutes les opérations liées à Hadoop s'effectuent avec le même unique client console vu précédemment: hadoop (avec d'autres options que l'option fs vu précédemment).



# Architecture MapReduce

## JobTracker

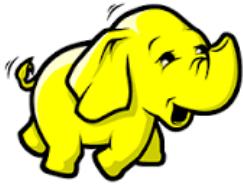
- Est un processus maître qui va se charger de l'ordonnancement des traitements et de la gestion de l'ensemble des ressources du système.
- Il reçoit (du client) la ou les tâches MapReduce à exécuter (un .jar Java) ainsi que les données d'entrée et le répertoire où stocker les données de sorties.
- Il est pour cela en communication avec le Namenode d'HDFS.
- Le JobTracker est en charge de planifier l'exécution des tâches et de les distribuer sur des TaskTrackers.
- Comme il sait où sont situés les blocs de données, il peut optimiser la colocalisation traitements/données.



# Architecture MapReduce

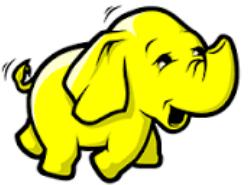
## TaskTracker

- Est une unité de calcul du cluster.
- Il assure, en lançant une nouvelle machine virtuelle java (JVM), l'exécution et le suivi des tâches MAP ou REDUCE s'exécutant sur son nœud et qu'il reçoit du JobTracker.
- Il dispose d'un nombre limité de slots d'exécution (thread) et donc un nombre limité de tâches MAP, REDUCE ou SHUFFLE pouvant s'exécuter simultanément sur le nœud.
- Il est aussi en communication constante avec le JobTracker pour l'informer de l'état d'avancement des tâches (heartbeat call). Et oui, nous sommes toujours confrontés au problème de la tolérance aux pannes car en cas de défaillance, le JobTracker, informé ou sans nouvelle du TaskTracker, doit pouvoir ordonner la réexécution de la tâche.

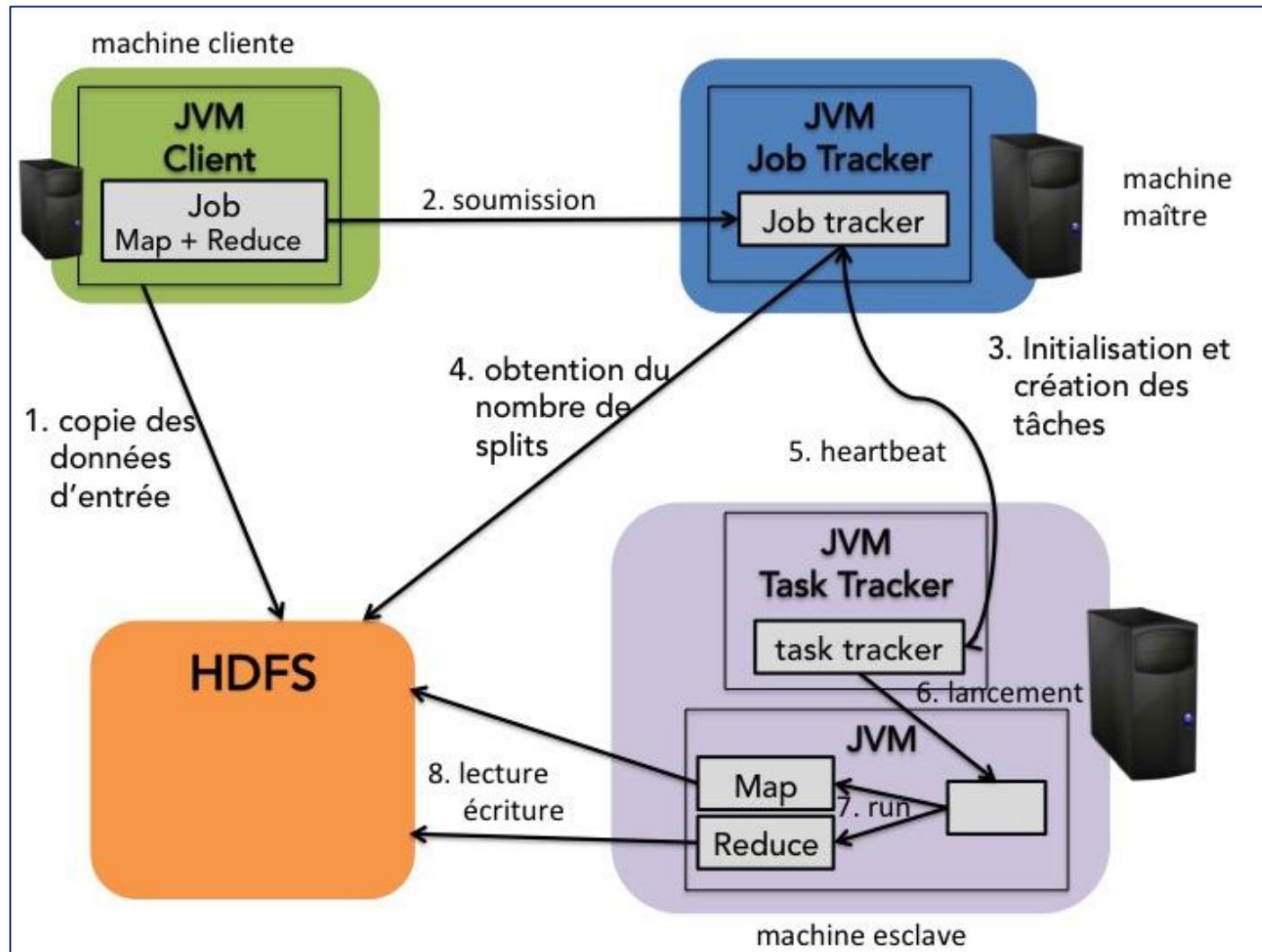


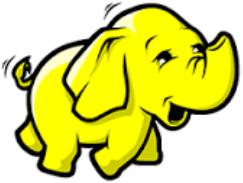
# Utilisation MapReduce en Java

- Un programme Hadoop « natif » est développé en Java.
- Trois classes au minimum au sein d'un programme Hadoop:
  - Classe Mapper: implémentation de la fonction MAP.
  - Classe Reducer: implémentation de la fonction REDUCE.
  - Programme de lancement: principal programme, configurer des différents types et classes utilisés, des fichiers d'entrée/de sortie, etc., ensuite soumet le travail et attend généralement qu'il se termine



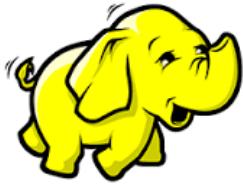
# Utilisation MapReduce en Java





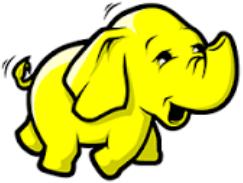
# Utilisation MapReduce en Java

- 1- Un client hadoop copie les données sur HDFS.
- 2- Le client soumet le travail à effectuer au JobTracker sous la forme d'une archive.jar et des noms des fichiers d'entrée et de sortie.
- 3- Le JobTracker demande au Namenode où se trouvent les blocs correspondants aux données d'entrée.
- 4- Il détermine alors quels sont les nœuds TaskTracker les plus appropriés pour exécuter les traitements (colocalisation ou proximité des nœuds). Il envoie alors au TaskTracker sélectionné et pour chaque bloc de données, le travail à effectuer (Map, Reduce ou Shuffle, fichier .jar).
- 5- Les TaskTrackers envoient régulièrement un message (heartbeat) au JobTracker pour l'informer de l'avancement de la tâche et de leur nombre de slots disponibles.



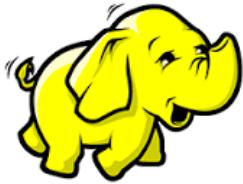
# Utilisation MapReduce en Java

- 6- Quand toutes les opérations envoyées aux TaskTrackers sont confirmées comme étant effectuées, on lance le job
- 7- Appliquer le Map et le Reduce sur le bloc de données
- 8- Ecrire le résultat dans HDFS



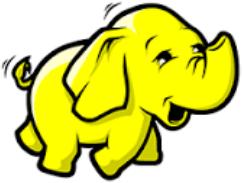
## Classe Mapper

- Une classe Map, implémentant la classe org.apache.hadoop.Mapper de Hadoop
- Le paramétrier avec le type de la clé d'entrée (TypeCleE), le type de la valeur d'entrée (TypeValE), le type de la clé des sorties intermédiaires (TypeCleI) et enfin le type de la valeur des sorties intermédiaires (TypeValI)
- Il est en charge de l'opération MAP correspondant à notre problème en surchargeant la fonction map du Mapper.
- Les valeurs doivent implémenter l'interface *Writable* de l'API Hadoop qui permet la sérialisation et la désérialisation
- Les clés doivent implémenter l'interface *WritableComparable<T>*



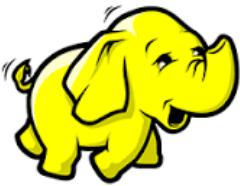
## Classe Mapper

- La classe MAP va être en charge de l'opération MAP de notre programme. Elle doit étendre la classe Hadoop *org.apache.hadoop.mapreduce.Mapper*.
- Il s'agit d'une classe générique qui se paramétre avec quatre types:
  - Un type keyin: le type de clef d'entrée.
  - Un type valuein: le type de valeur d'entrée.
  - Un type keyout: le type de clef de sortie.
  - Un type valueout: le type de valeur de sortie.



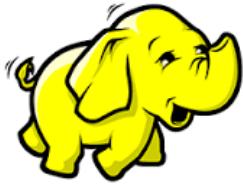
## Classe Reducer

- Une classe Reduce, implémentant la classe `org.apache.hadoop.Reducer` de Hadoop que l'on paramètre avec 4 types comme pour Mapper(deux types étant même identiques).
- Il est en charge de l'opération REDUCE correspondant à notre problème en surchargeant la fonction reduce de Reducer.
- La surcharge d'une méthode ou d'un constructeur permet de définir plusieurs fois une même méthode/constructeur avec des arguments différents. Le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments .



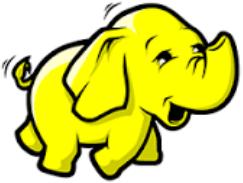
## Classe Reducer

- Elle doit étendre la classe Hadoop `org.apache.hadoop.mapreduce.Reducer`. Il s'agit là aussi d'une classe générique qui se paramétre avec les mêmes quatre types que pour la classe Mapper: keyin, valuein, keyout et valueout.
- On rappelle que l'opération REDUCE recevra en entrée une clef unique, associée à toutes les valeurs pour la clef en question.
- Dans le cas du compteur d'occurrences de mots, on recevra en entrée une valeur unique pour la clef, par exemple `appel`, suivi de toutes les valeurs qui ont été rencontrées à la sortie de l'opération MAP pour la clef `appel` (par exemple cinq fois la valeur 1)



# Programme de lancement

- Une classe qui contient la fonction *main* du programme et qui va permettre de:
  - Récupérer la configuration générale du cluster.
  - Créer un job.
  - Préciser quelles sont les classes Map et Reduce du programme.
  - Préciser les types de clés et de valeur correspondant à notre problème (attention souvent des types propres à Hadoop).
  - Indiquer où sont les données d'entrée et de sortie dans HDFS.
  - Lancer l'exécution de la tâche



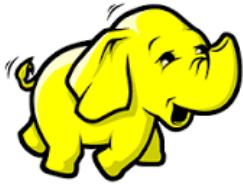
## Compilation et exécution

- Un programme MapReduce se compile comme tout autre programme Java classique, Les classes sont compressées dans un paquet Java .jar (on peut tout à fait y inclure plusieurs MapReduce différents et sélectionner celui dont on a besoin)
- Utiliser le client console Hadoop pour l'exécution, synthèse:

```
hadoop jar [JAR FILE] [LANCEMENT CLASS] [PARAMETERS]
```

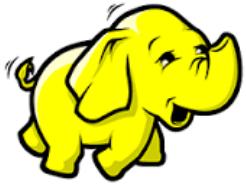
- Pour l'exemple WordCounter:

```
hadoop jar wordcounter.jar org.wordcounter.Wordcounter /input/poeme.txt /results
```



# Résultat de MapReduce

- Hadoop stocke les résultats dans une série de fichiers dont le nom respecte le format:
  - part-r-XXXX
  - avec XXXX un compteur numérique incrémental.
- On a un fichier part-r par opération Reduce exécutée. Le « r » désigne le résultat de l'opération Reduce. On peut également demander la génération de la sortie des opérations map – dans des fichiers part-m-\*.
- En cas de succès, un fichier vide \_SUCCESS est également créé.

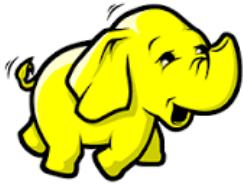


## Résultat de MapReduce

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01  
Hello World, Bye World!  
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02  
Hello Hadoop, Goodbye to hadoop.
```

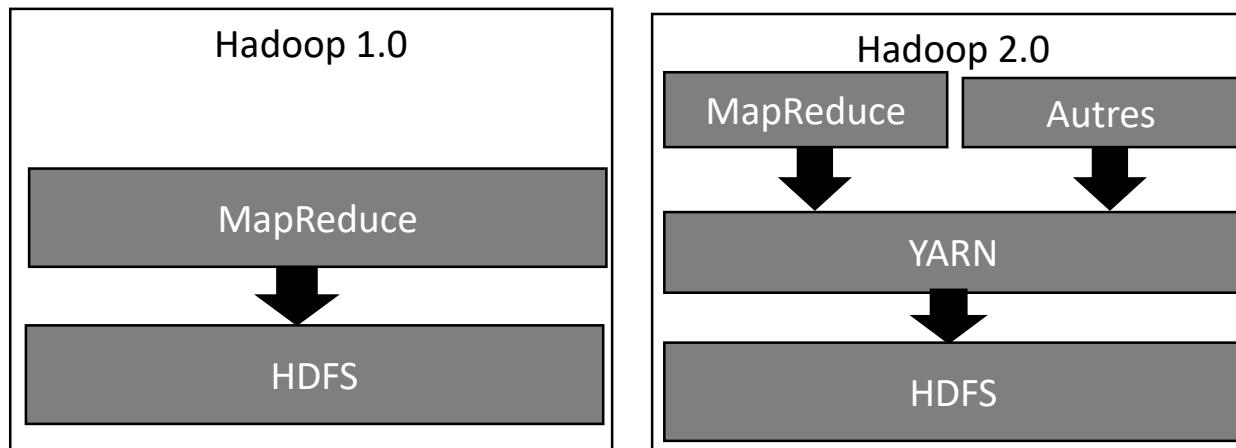
```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount  
/usr/joe/wordcount/input /  
usr/joe/wordcount/output
```

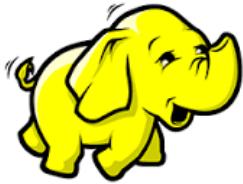
```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000  
Bye 1  
Goodbye 1  
Hadoop, 1  
Hello 2  
World! 1  
World, 1  
hadoop. 1  
to 1
```



# MapReduce dans Hadoop 1.0 & 2.0

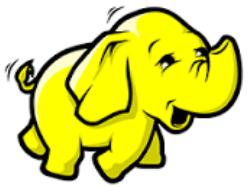
- Un nouveau composant YARN est ajouté dans Hadoop 2.0
- Le changement de l'architecture de Hadoop 1.0 à Hadoop 2.0
- YARN est en charge de la gestion des ressources Cluster. MapReduce est simplifié pour effectuer le traitement de données
- MapReduce existant peut être exécuté dans Hadoop 2.0 sans modification





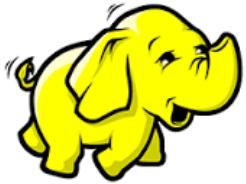
# MapReduce dans Hadoop 1.0 & 2.0

- JobTracker et TaskTracker ont disparu dans Hadoop 2.0,
- Deux composants de YARN (ResourceManager et NodeManager) remplacent ces 2 fonctionnalités
- MapReduce dans Hadoop 2.0 est une des applications distribuées qui s'exécute au-dessus de YARN, il effectue le traitement de données via YARN.
- MapReduce dans Hadoop 1.0 est MR1, la nouvelle version dans Hadoop 2.0 est MR2.

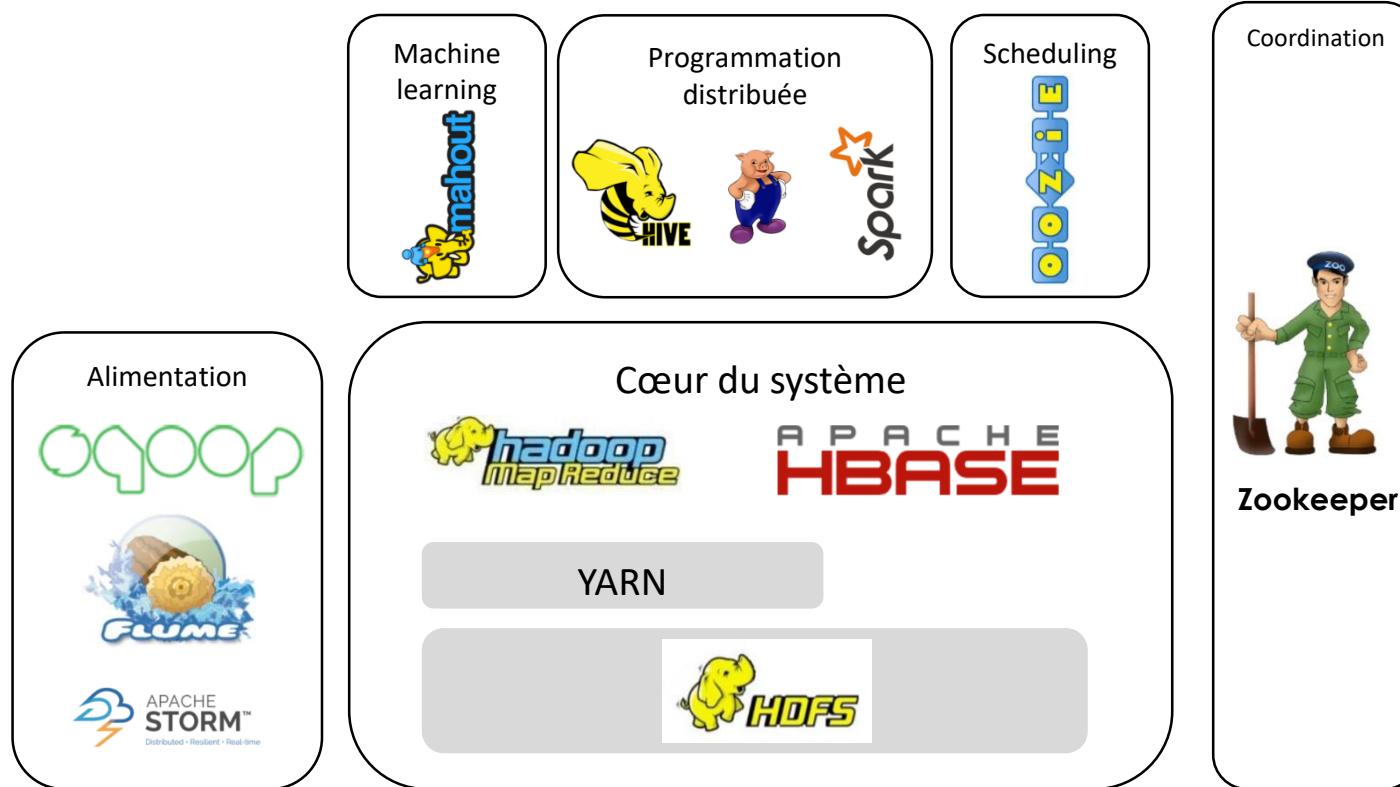


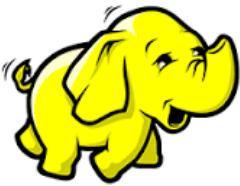
# Plan

## YARN



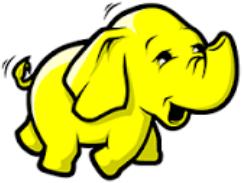
# YARN





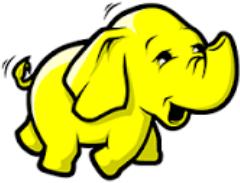
## Limitations de MapReduce1 (MR1)

- L'utilisation du cluster est affaiblie lorsque il y a plusieurs map et reduce qui s'exécutent
- Le partage des ressources avec des applications n'utilisant pas MR est impossible (ex Impala)
- La scalabilité est limité à 4000 nœuds par cluster, avec 1 JobTracker par cluster, et un maximum de 40 000 tâches concurrentes.
- MR1 est un single point of failure: une panne entraînera l'arrêt complet du système.
- Beaucoup d'efforts de transformer un algorithme en MapReduce.
- Pour traiter des problèmes complexes, les deux étapes MAP et REDUCE ne suffisent pas, il est très souvent nécessaire d'enchaîner des séquences de MapReduce ce qui est très coûteux car cela nécessite de démarrer un job MapReduce à chaque fois.
- Le JobTracker a une double responsabilité: Il doit gérer les ressources du cluster et il doit ordonner les jobs. Que se passe-t-il si le JobTracker est défaillant ?



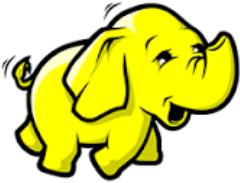
## Concept général de YARN

- YARN est un gestionnaire des ressources d'un cluster Hadoop. Il a été introduit dans Hadoop pour améliorer l'implémentation de MapReduce.
- L'idée fondamentale de YARN est de diviser les deux grandes fonctionnalités du JobTracker, la gestion des ressources et l'ordonnancement de job en deux daemons séparés.
- La réalisation de cette division se fera par le développement de deux daemons:
  - Un ResourceManager qui gère l'utilisation des ressources du cluster. (1 par cluster)
  - Un NodeManager s'exécutant sur tous les nœuds du cluster pour lancer et monitorer les containers.
- YARN propose de séparer la gestion des ressources du cluster et la gestion des jobs MapReduce, permettant ainsi de généraliser cette gestion des ressources à d'autres applications.



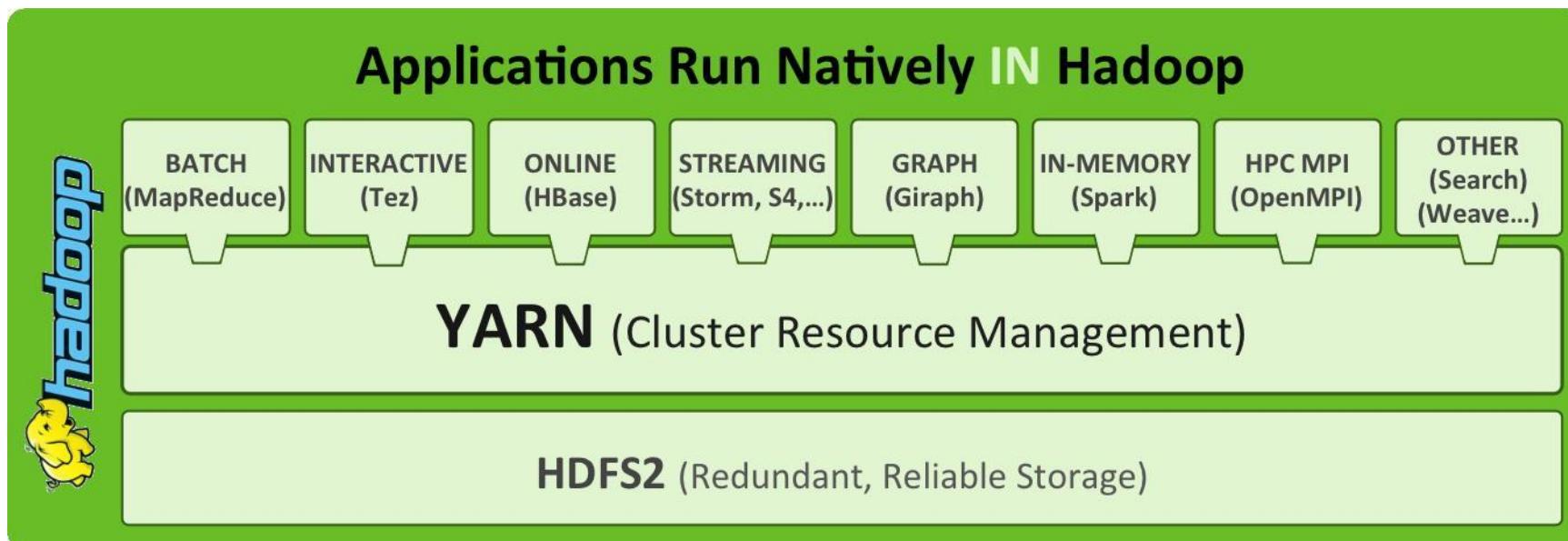
## Concept général de YARN

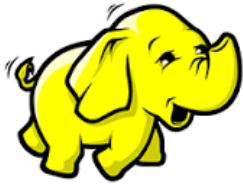
- L'idée principale est de considérer que les nœuds ont des ressources (mémoire et CPU) qui seront allouées aux applications quand elles le demandent.
- Un deamon est un processus ou un ensemble de processus qui s'exécute en arrière-plan plutôt que sous le contrôle direct d'un utilisateur.



# Applications YARN

- Les outils comme Spark, Hive et Pig sont développés sur la base de YARN, qui masque les détails de la gestion des ressources à l'utilisateur.
- Les utilisateurs développent les applications directement sur ces outils sans se soucier de la gestion des ressources.

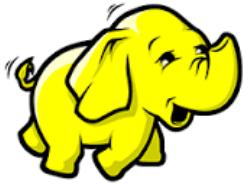




# Anatomie de l'exécution d'une application YARN

Les fonctionnalités du JobTracker sont réparties entre :

- 1- Le ressource manager (RM) qui est le chef d'orchestre des ressources du cluster.
- Il ordonne les requêtes clients et pilote le cluster par l'intermédiaire de node managers qui s'exécutent sur chaque nœud de calcul.
- Il a donc pour rôle de contrôler toutes les ressources du cluster et l'état des machines qui le constituent.
- Il gère donc le cluster en maximisant l'utilisation de ressources.
- 2- L'application master (AM) qui est un processus s'exécutant sur toutes les machines esclaves et qui gère, en discussion avec le ressource manager, les ressources nécessaires au travail soumis.



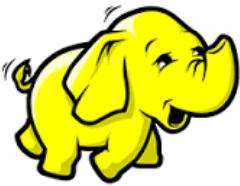
# Anatomie de l'exécution d'une application YARN

Les fonctionnalités du TaskTracker sont aussi réparties sur une même machine entre:

- 1- Des containers qui sont des abstractions de ressources sur un nœud dédiées soit à l'exécution de tâches comme Map et Reduce, soit à l'exécution d'un application master.
- 2- Un node manager qui héberge des containers et gère donc les ressources du nœud. Il est en communication via un heartbeat avec le resource manager.

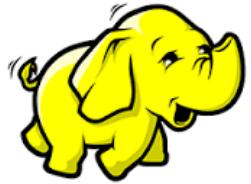
YARN fournit ses propres services à travers deux types de deamons:

- Un ResourceManager (un par cluster) qui gère l'utilisation des ressources du cluster
- Un NodeManager s'exécutant sur tous les nœuds du cluster pour exécuter et moniter les containers. Le container exécute les processus des applications avec des ressources contrôlées par le scheduler (mémoire, CPU, etc).

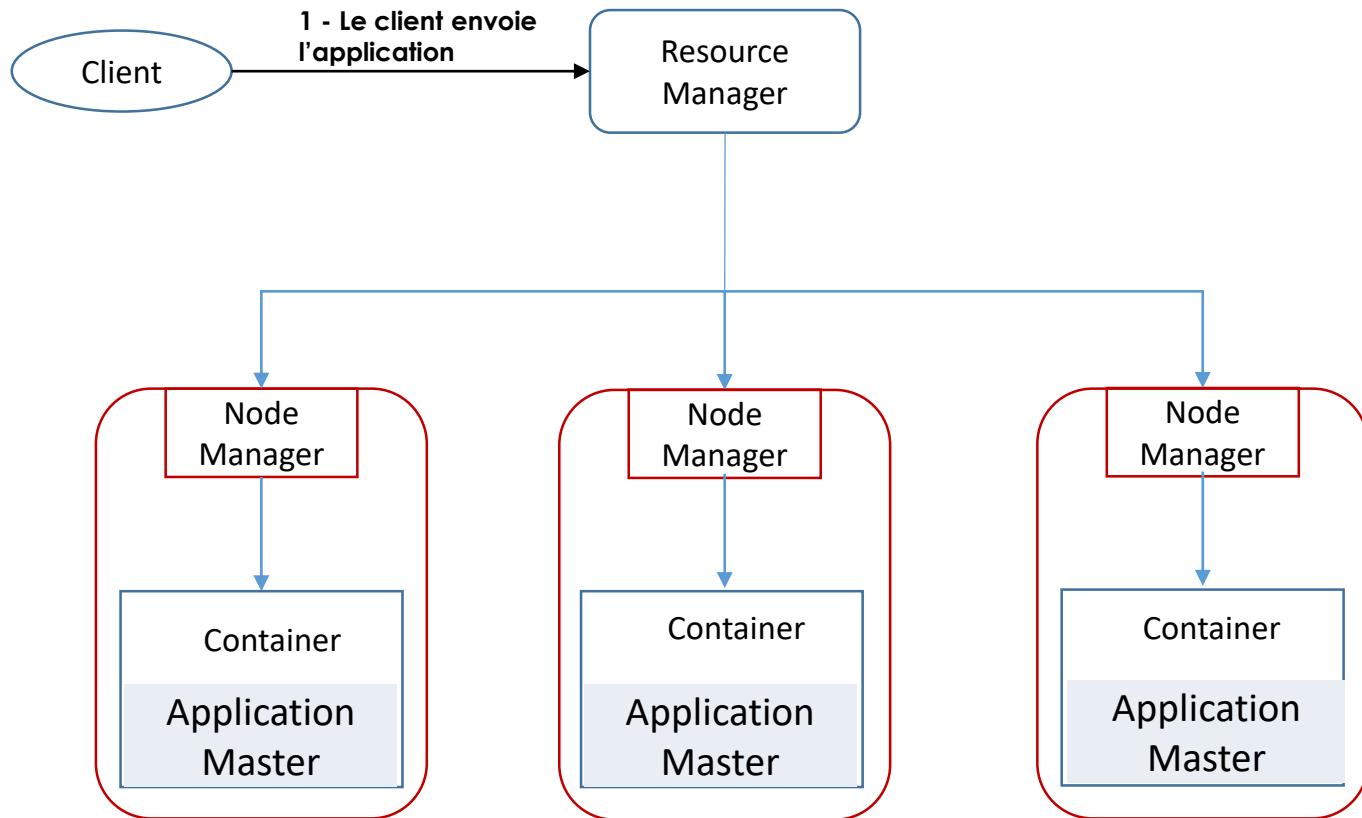


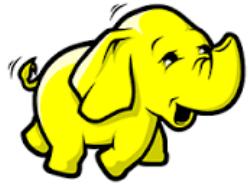
# Anatomie de l'exécution d'une application YARN

- 1- Pour exécuter une application sur YARN, le client contacte le ressource manager et lui demande d'exécuter l'application.
- 2- Le ressource manager trouve alors le node manager qui peut exécuter l'application dans le container.
- Précisément, ce que l'application master fait pendant qu'il s'exécute dépend de l'application.
- Deux cas de figures sont possibles:
- 1- Si l'application peut être exécutée dans le container, l'application master l'exécute dans le container et retourne le résultat au client.
- 2- Si l'application ne peut pas être exécutée à cause de ressources insuffisantes, il peut requérir plus de containers et les utiliser pour exécuter un calcul distribué.

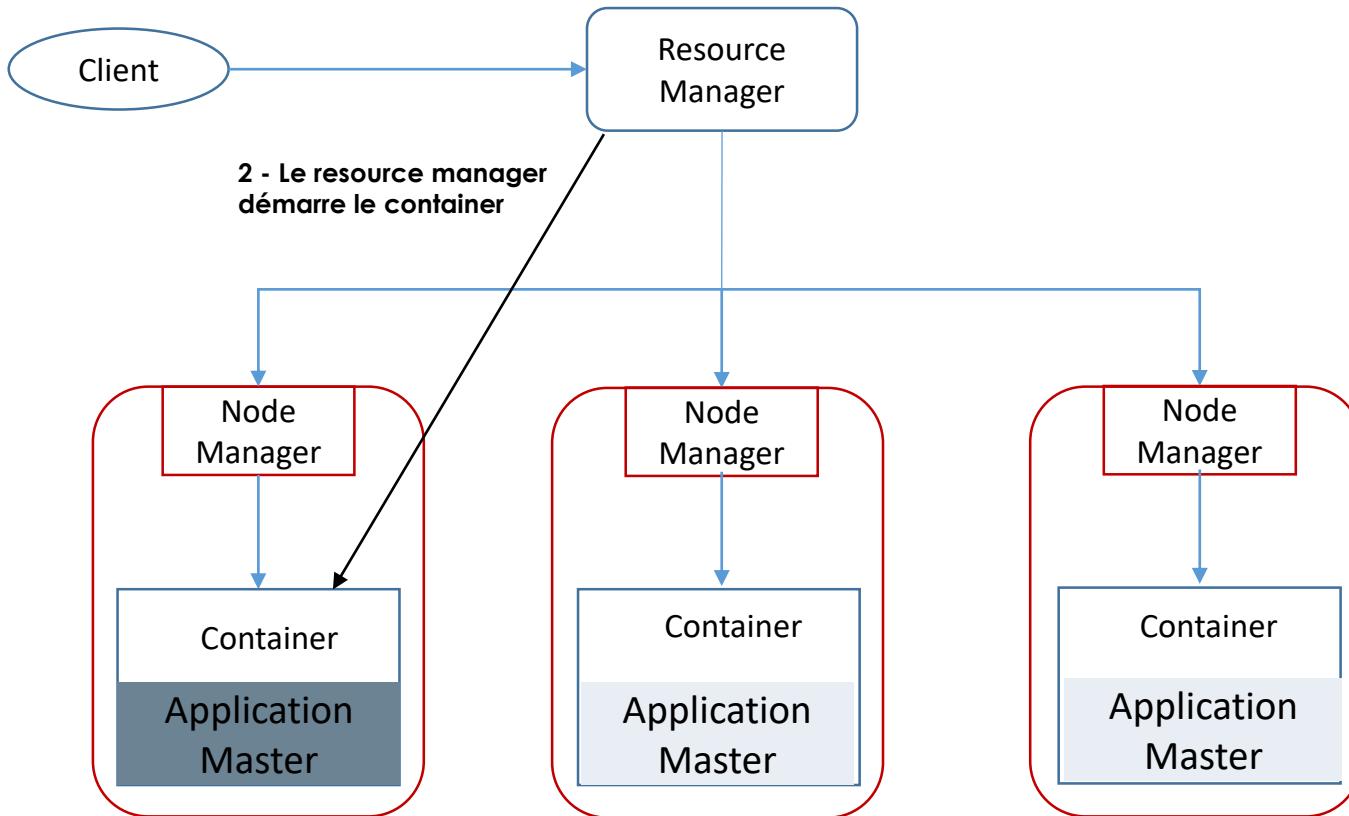


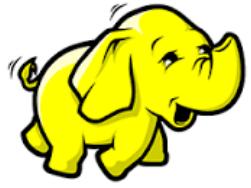
# Anatomie de l'exécution d'une application YARN



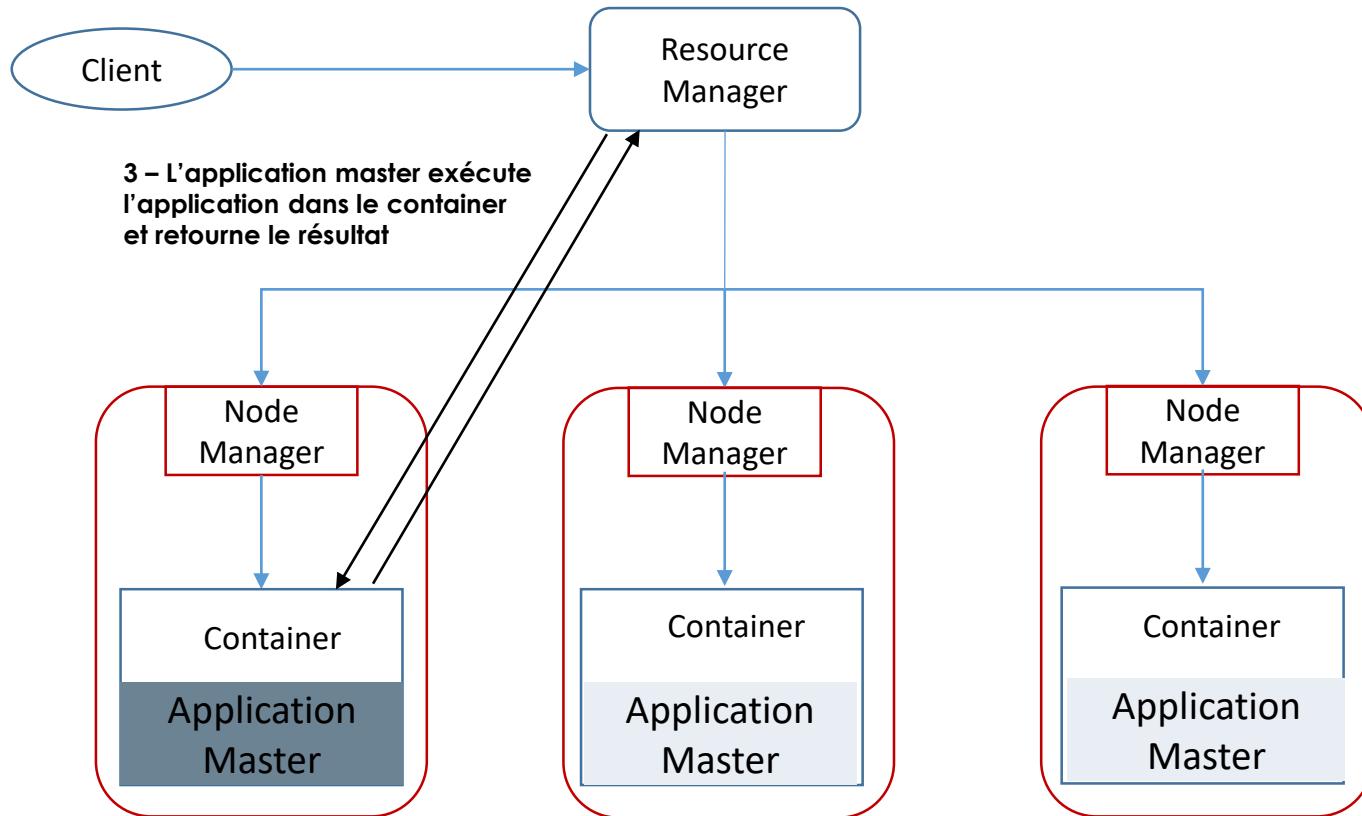


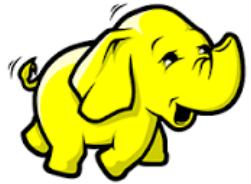
# Anatomie de l'exécution d'une application YARN



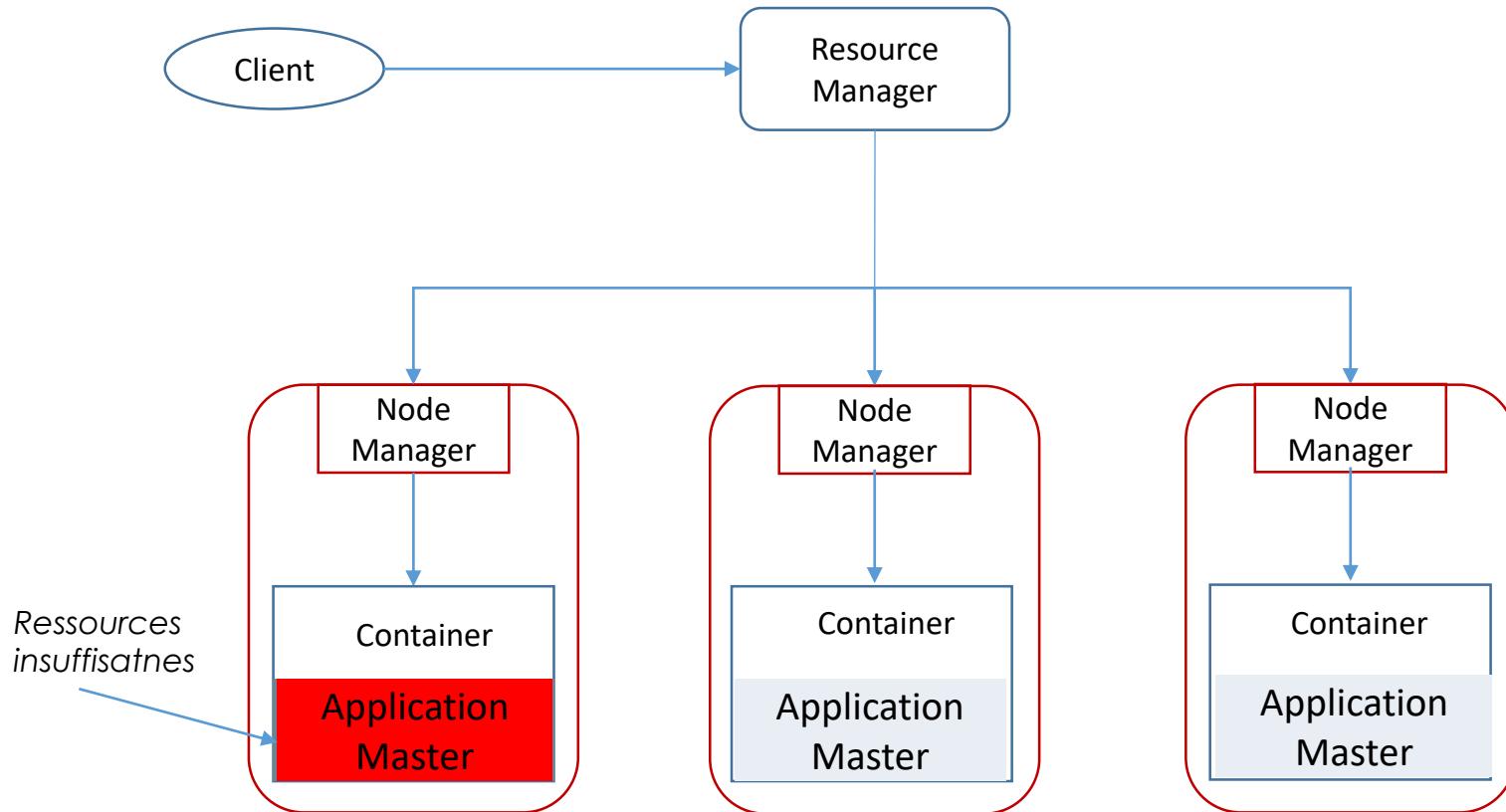


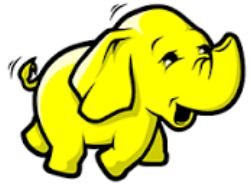
# Anatomie de l'exécution d'une application YARN



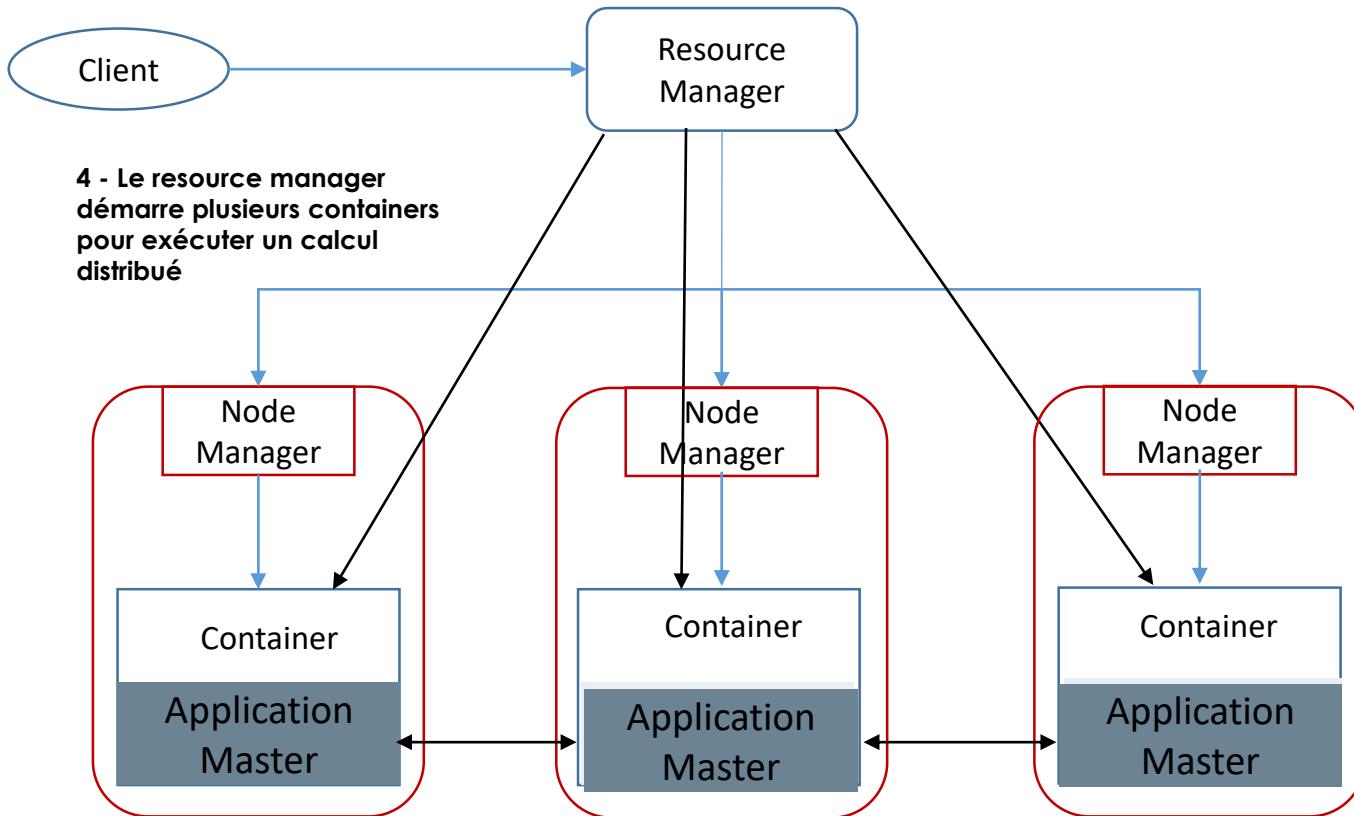


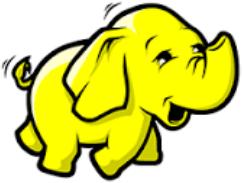
# Anatomie de l'exécution d'une application YARN





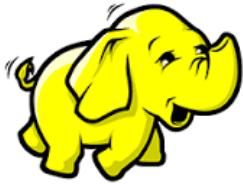
# Anatomie de l'exécution d'une application YARN





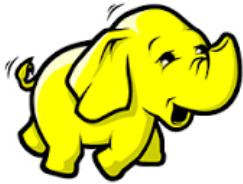
## Les requêtes de ressources

- YARN possède un modèle flexible, pour faire les requêtes de ressources. Une requête pour un ensemble de containers peut exprimer les quantités de ressources (mémoire et CPU) requises.
- YARN permet à l'application de spécifier les contraintes de localité pour le container requis, afin de permettre une utilisation optimale de la bande passante du cluster.
- Parfois, les contraintes de localité ne sont pas rencontrées. Par exemple, si un nœud spécifique a été requis, et qu'aucun container ne peut être démarré dessus (parce que d'autres containers sont déjà en cours d'exécution), YARN essayera de démarrer un container dans un nœud du même rack, ou en cas d'impossibilité, dans un autre nœud du cluster.
- Une application YARN peut faire une requête de ressource à tout moment, même si l'application est en exécution.



# Les avantages d'utilisation de YARN

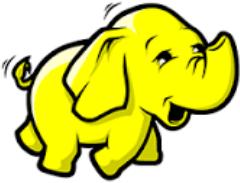
- YARN a un design qui permet de surmonter plusieurs limitations de MapReduce 1, Les bénéfices d'utilisation de YARN permettent un gain en:
  - Scalabilité
  - Disponibilité
  - Le bon utilisation des ressources via les containers et la négociation entre l'application master et le resource manager
  - Colocation: architecture logicielle où plusieurs instances indépendantes d'une ou plusieurs applications opèrent dans un environnement partagé



# Les avantages d'utilisation de YARN

## Scalabilité

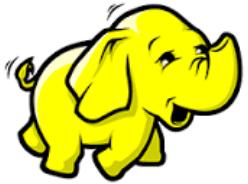
- YARN peut être exécuté sur des clusters plus larges que MR1. Comme sur MR1, le JobTracker doit gérer les jobs et les tâches, sa scalabilité est limitée à 4000 nœuds et à 40 000 tâches.
- L'architecture de YARN, qui tire profit du split: ressource manager/application master, permet de passer à 10 000 nœuds et à 100 000 tâches concurrentes.
- Contrairement au JobTracker, un application master est dédié à chaque instance d'application, et s'exécute pendant la durée de l'application.



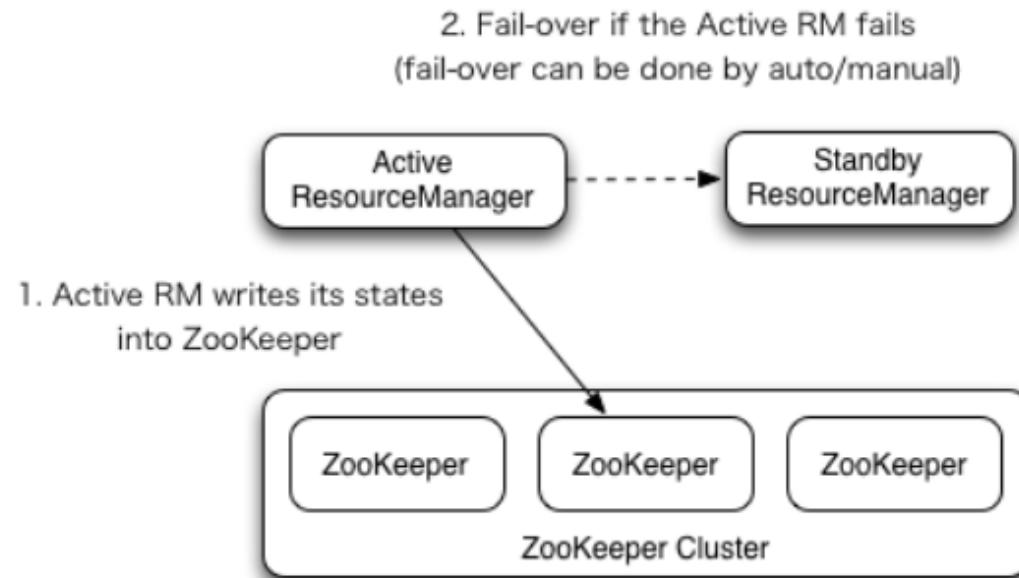
# Les avantages d'utilisation de YARN

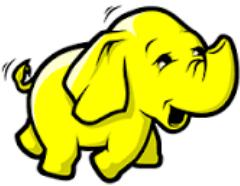
## Disponibilité

- Si le service du daemon (RM ou NM) tombe en panne, la haute disponibilité (High Availability HA) est obtenue en répliquant l'état du daemon RM ou NM.
- ResourceManager est le point de défaillance unique dans un cluster.
- RM est actif, et un ou plusieurs RM sont en attente (Standby) et prennent le relais en cas de problème avec l'Active
- Le déclencheur de la transition vers le Standby provient soit de l'administrateur (via CLI), soit du failover-controller intégré lorsque il est activé.



# Les avantages d'utilisation de YARN

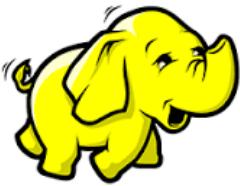




# Les avantages d'utilisation de YARN

## Utilisation

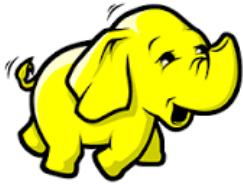
- Dans MR1, chaque TaskTracker est configuré avec une allocation statique de « slots » de taille fixe, qui sont divisés en map slots et en reduce slots au moment de la configuration. Un map slot peut seulement être utilisé pour exécuter une map tâche, et un reduce slot peut seulement être utilisé pour exécuter une reduce tâche.
- YARN gère une panoplie de ressources, à travers le NodeManager.
- Exemple: si MR s'exécute sur YARN, MR ne se confrontera pas à la situation où une tâche reduce doit attendre, parce qu'il n'y a que des map slots disponibles sur le cluster, ce qui peut arriver dans MR1.



# Les avantages d'utilisation de YARN

## Utilisation

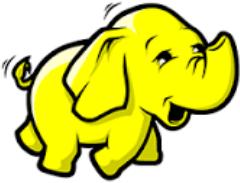
- De plus, les ressources de YARN sont de granularité fine. Une application peut utiliser une ressource pour ce dont elle a besoin, plutôt que d'utiliser un slot indivisible, qui peut être trop grand (et représente une perte de ressources) ou trop petit (et peut causer une panne) pour une tâche particulière.
- Granularité: La notion de granularité définit la taille du plus petit élément, de la plus grande finesse d'un système. Quand on arrive au niveau de granularité d'un système, on ne peut plus découper l'information.



# Les avantages d'utilisation de YARN

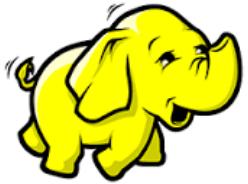
## Colocation

- Le plus grand bénéfice de YARN est qu'il ouvre Hadoop à d'autres types d'applications distribuées hormis MapReduce.
- Il est aussi possible aux utilisateurs d'exécuter différentes versions de MapReduce sur le même cluster YARN, ce qui rend le processus de mise à niveau de MapReduce plus facilement gérable.

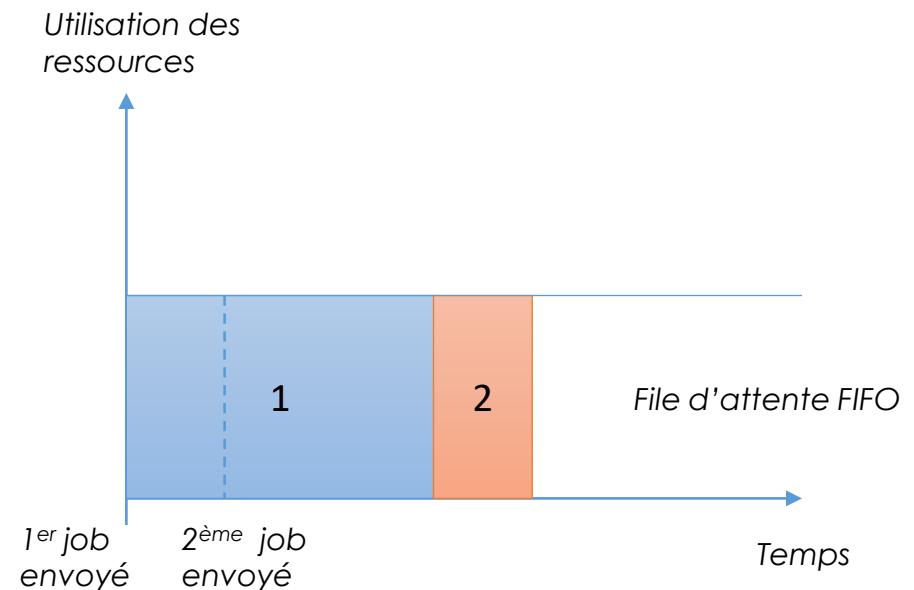


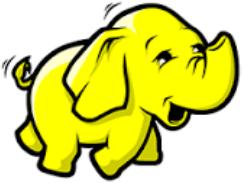
## Scheduling options (ordonnancement)

- Le YARN Scheduler est responsable de l'allocation des ressources suivant des règles prédéfinies. Le Scheduling n'est pas une tâche facile, et il n'y a pas de règles meilleures que d'autres, chacune a ses spécifités.
- Nous dénombrons 3 Schedulers dans YARN:
  - Fifo Scheduler
  - Capacity Scheduler
  - Fair Scheduler



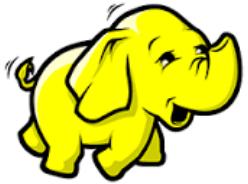
# FIFO Scheduler



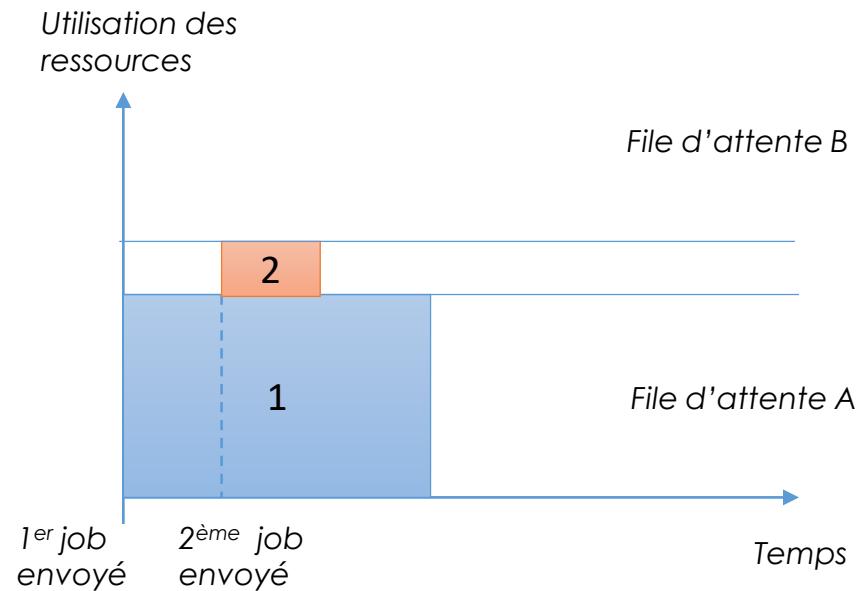


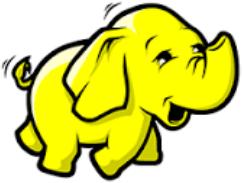
## FIFO Scheduler

- Il place les applications dans une file d'attente et les exécute dans l'ordre de soumission (FIFO: First In First Out)
- Il alloue les ressources pour la première application dans la file d'attente, une fois que cela est réalisé, l'application suivante est servie, etc
- Il a le mérite d'être simple à comprendre et n'a besoin d'aucune configuration, mais il n'est pas recommandé pour des clusters partagés.
- Les grandes applications utiliseront une grande partie voir la totalité des ressources du cluster, donc chaque application doit attendre son tour.



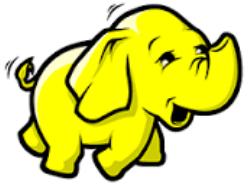
# Capacity Scheduler



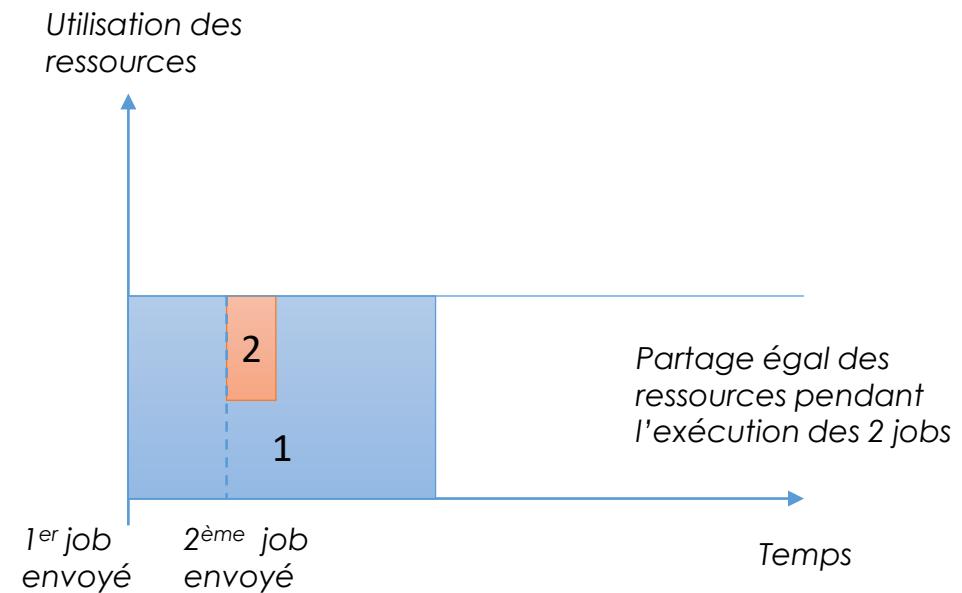


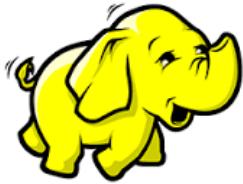
# Capacity Scheduler

- Dans un cluster partagé, il est recommandé d'utiliser le capacity scheduler ou le fair scheduler. Les deux permettent l'exécution de longs jobs en temps voulu, tout en permettant aux utilisateurs qui exécutent de courts jobs, d'obtenir des résultats dans des délais raisonnables.
- Une file d'attente dédiée et séparée permet aux courts jobs de démarrer aussitôt soumis. Cependant, l'utilisation globale du cluster est amoindrie pour un long job, (comparé au FIFO scheduler). En effet, le même long job exécuté sur les deux schedulers se terminera plus rapidement sur le FIFO que sur le capacity.



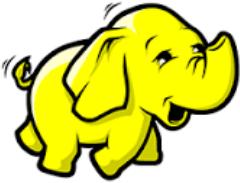
# Fair scheduler





## Fair scheduler

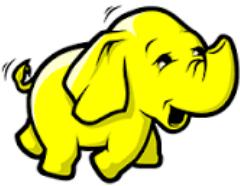
- Les ressources sont allouées dynamiquement entre les jobs en cours d'exécution.
- Sur le schéma, le premier job exécuté est long, seul à être dans la file d'attente, alors il occupe toutes les ressources du cluster. Lorsqu'un second job, (dans notre cas un court job) est exécuté, les ressources sont réparties dynamiquement de manière équitable (50% pour le premier et 50% pour le second). Une fois que le second job (court) termine son exécution, le premier job reprend la totalité des ressources du cluster.



# Spark Submit et YARN

- Les ressources sont allouées
- Le script spark-submit du répertoire bin de Spark est utilisé pour lancer des applications sur un cluster.
- Ce script prend en charge la configuration du classpath avec Spark et ses dépendances. Il peut prendre en charge différents gestionnaires de cluster et modes de déploiement pris en charge par Spark:

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
--deploy-mode <deploy-mode> \
--conf <key>=<value> \
... # other options
<application-jar> \
[application-arguments]
```

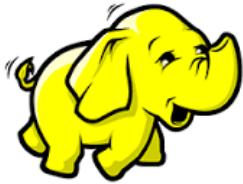


# Spark Submit et YARN

- Les ressources sont allouées

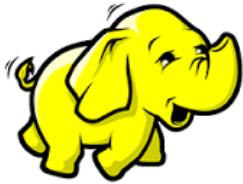
```
# Run application locally on 8 cores
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local[8] \
/path/to/examples.jar \
100

# Run on a YARN cluster
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster # can be client for client mode
--executor-memory 20G \
--num-executors 50 \
/path/to/examples.jar \
1000
```



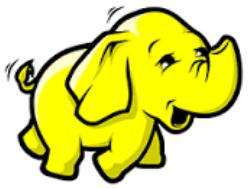
# YARN Tuning: Executor, Cores et Memory

- Données : nb cores, nb nodes, memoire par node
- nb executor = nb cores/ 5
- memoire executor = (memoire par node / nb executor) - Max(0.10 x (memoire par node / nb executor), 384 MB)
- memory in executor : 25% pour interne et 60% pour calcul et stockage spark,  
60%==M==spark.memory.fraction, la partie cache du M est appelé  
R==spark.memory.storageFraction et une troisième partie overhead
- memory\_for\_compute (M - R) < (spark.executor.memory - overhead) x spark.memory.fraction  
x (1 - spark.memory.storage.fraction)
- memory\_per\_task = memory\_for\_compute / spark.executor.cores
- number\_of\_partitions = à trouver par les test : size of shuffle stage / memory per task avec



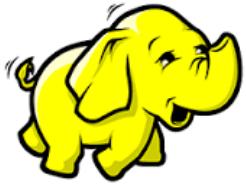
## YARN Tuning: Executor, Cores et Memory

- 6 Nodes et chaqu'un 16 cores et 64 GB RAM :
- 1- Pour chaque node, 1 core et 1 GB utilisé pour le SE et les Daemons Hadoop
- 2- Il reste : 15 cores, 63 GB RAM pour chaque node
- 3- Nombre de cores : 5
- 4- Nombre des Executors:  $17 \Rightarrow 3$  executors par node:  $15/5$ , avec 6 nodes, et 3 executors par node, en totale 18 executors, 1 executor (java process) pour Application Master de YARN.  
Au final : 17 executors
- 5- Mémoire de chaque executor:  $63/3 = 21\text{GB}$  , on a 3 executors par node
- 6- Mémoire overhead:  $\max(384, 0.07 * \text{spark.executor.memory})$ (7% ou 10%)
- $0.07 * 21 = 1.47$ , Puisque  $1.47 \text{ GB} > 384 \text{ MB}$ , le overhead est: 1.47
- Executor memory = 19 GB :  $21 - 1.47 \sim 19 \text{ GB}$
- Conclusion: 17 Executors, 5 Cores, 19 GB Mémoire Executor

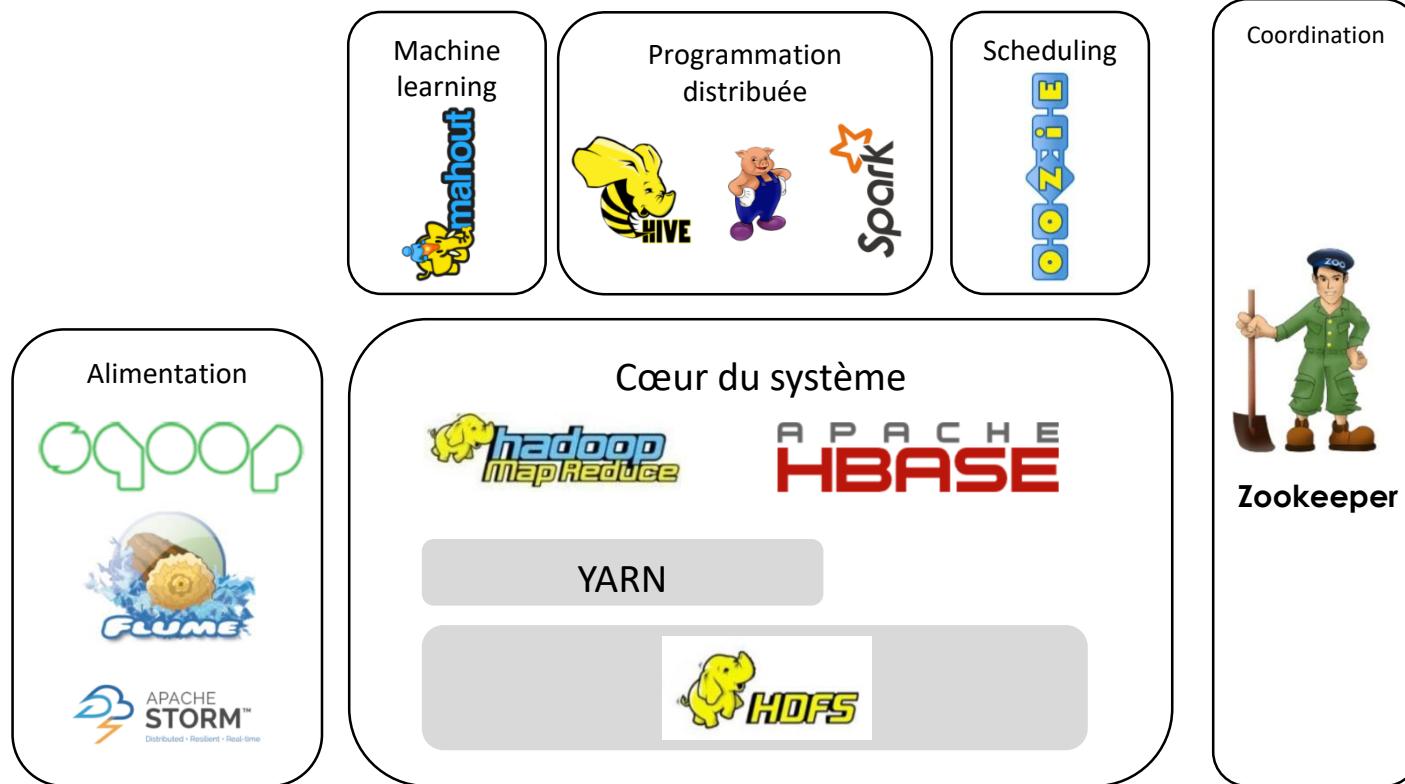


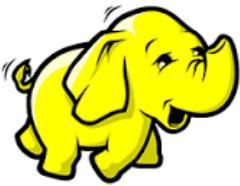
# Plan

# Hive



# Hive

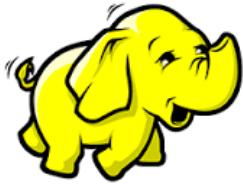




# Qu'est ce que Hive

Hive : Base de données distribuée

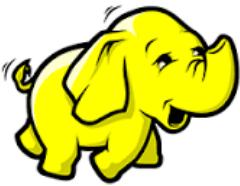
- Créé chez Facebook
- S'inscrit comme un Data Warehouse au dessus de l'architecture Hadoop
- Facilite la lecture, l'écriture et la gestion de grands ensembles de données résidant dans le stockage distribué à l'aide de SQL
- Apache Hive est une infrastructure d'entrepôt de donnée intégrée sur Hadoop permettant l'analyse, le requêtage via un langage proche syntaxiquement de SQL.
- Un outil de ligne de commande et un JDBC driver sont fournis pour connecter les utilisateurs à Hive.



# Qu'est ce que Hive

Hive : Entrepôt de données

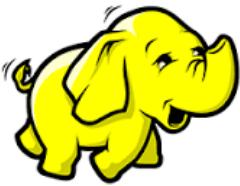
- Le Data Warehouse, ou entrepôt de données dédiée au stockage de l'ensemble des données utilisées dans le cadre de la prise de décision et de l'analyse décisionnelle
- Le terme entrepôt de données désigne une base de données utilisée pour collecter, ordonner, journaliser et stocker des informations provenant de base de données opérationnelles et fournir ainsi un socle à l'aide à la décision en entreprise



# Qu'est ce que Hive

Hive fournit :

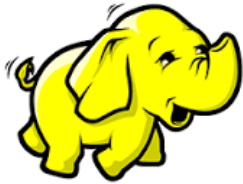
- Une couche d'abstraction entre l'utilisateur et le stockage de la donnée(job MapReduce automatiquement développé)
- Un stockage des données via HDFS ou Hbase
- Un moteur de requêtage et d'analyse se basant sur un langage proche du SQL (HQL: Hive Query Language)
- Des outils facilitant les opérations d'extraction, de transformation et de chargement (ETL)
- La possibilité de réaliser directement des traitements Map Reduce afin d'effectuer des analyses plus poussées ou non supportées par HQL(puisque la data est dans HDFS)
- Hive apporte une couche d'abstraction entre les flux de données et requêtes, et les flux de travail MapReduce complexes qu'ils impliquent.



# Qu'est ce que Hive

Hive est conçu pour :

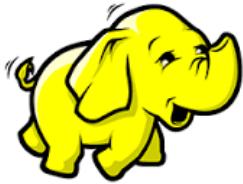
- Gérer et interroger une très grande volumétrie de données
- Etre utilisé par des applications nécessitant un Data Warehouse contenant des données relativement statiques (Pas de temps réel)
- Faciliter l'adaptation des développeurs et des utilisateurs
- Interaction avec les données via un SQL-like
- Eviter d'écrire des programmes Map Reduce longs et complexes
- Pouvoir facilement porter des traitements des bases de données traditionnelles
- Pouvoir utiliser des traitements Map Reduce personnalisés
- Le redéveloppement en HQL des traitements existants n'étant ni pratique ni performant
- Avoir un champ d'action étendu grâce à des fonctions personnalisées (user defined functions = procédure en SQL)



## Qu'est ce que Hive

Hive N'est PAS conçu pour :

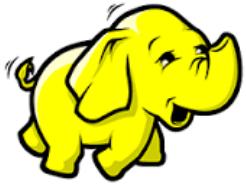
- Gérer les données de manière transactionnelle
- Beaucoup d'insertions/mises à jour enregistrement par enregistrement
- Pas de temps réel
- Du fait du lancement de traitements Map Reduce et de la quantité de données traitées



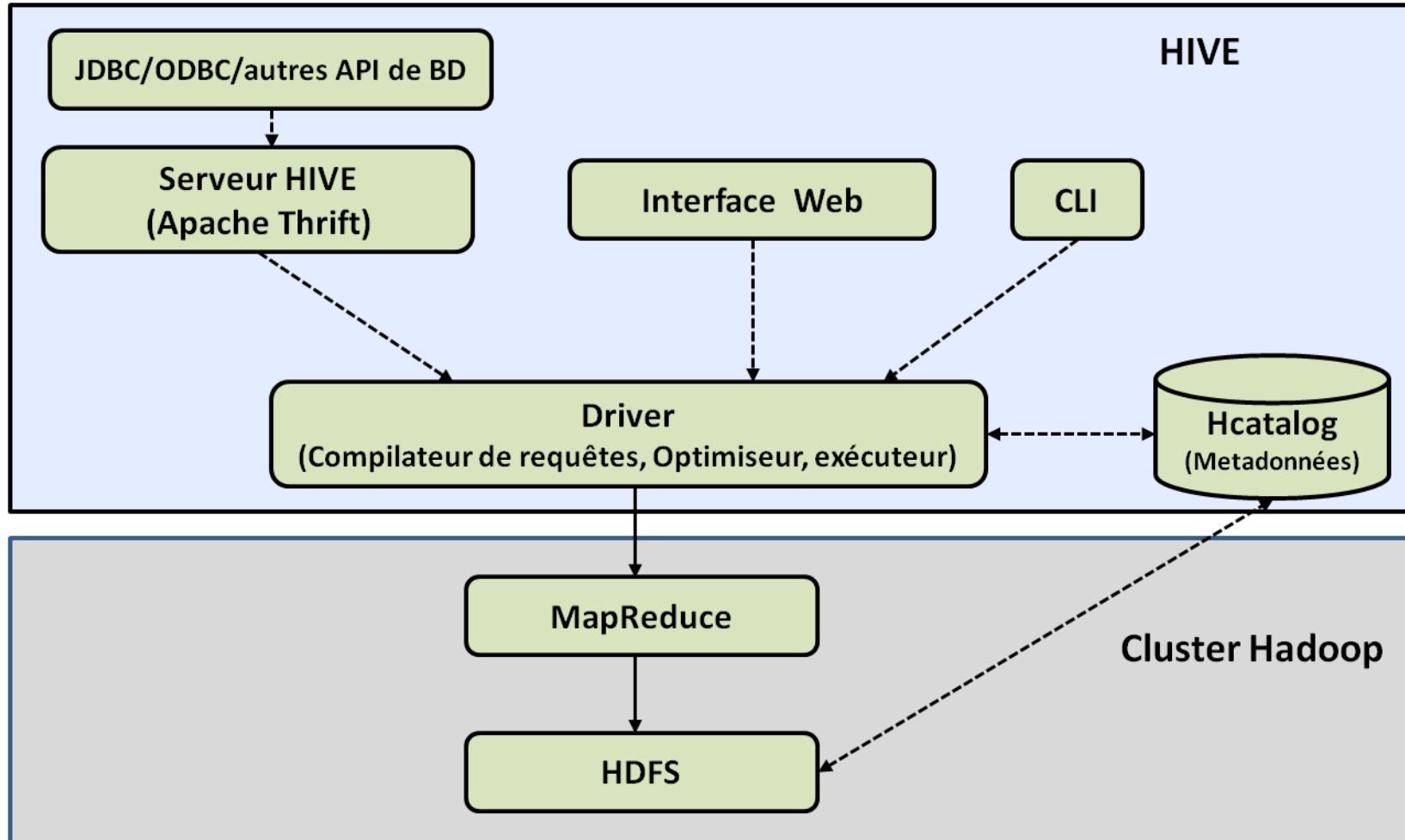
# Qu'est ce que Hive

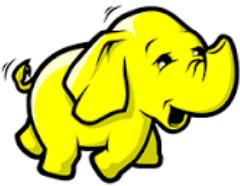
Cas d'utilisation:

- Traitement de logs
- Text mining, analyse sémantique
- Indexation automatique de documents
- Modèle prédictif
- Déduction de profil client



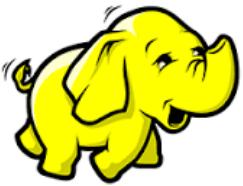
# Architecture





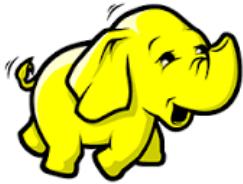
# Architecture

- Hive fournit un langage de requête basé sur le SQL appelé HiveQL
- HiveQL utilisé pour adresser des requêtes aux données stockées sur le HDFS.
- HiveQL permet également aux utilisateurs avancés/développeurs d'intégrer des fonctions Map et Reduce directement à leurs requêtes pour couvrir une plus large palette de problèmes de gestion de données.
- Cette capacité de plug-in du MapReduce sur le HiveQL s'appelle les UDF (User Defined Function).
- Une requête en HiveQL est transformée en job MapReduce et soumise au JobTracker pour exécution par Hive.



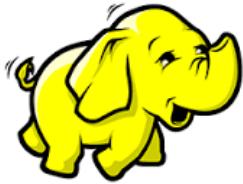
# Architecture

- Les données de la requête ne sont pas stockées dans une table comme dans le cas d'une requête SQL classique.
- Hive s'appuie sur une couche de stockage de données installée sur Hadoop à l'exemple d'HCatalog, pas pour le stockage des données du HDFS, mais pour le stockage des métadonnées des requêtes des utilisateurs.
- Les tables HCatalog ne stockent pas les données, elles font juste référence de pointeurs aux données qui sont sur le HDFS.
- HCatalog est indépendant du HDFS et du Hive, il peut être utilisé comme intermédiaire de connexion aux données contenues dans les systèmes de Business Intelligence de l'entreprise, c'est le but de la présence des connecteurs ODBC/JDBC dans l'infrastructure Hive.



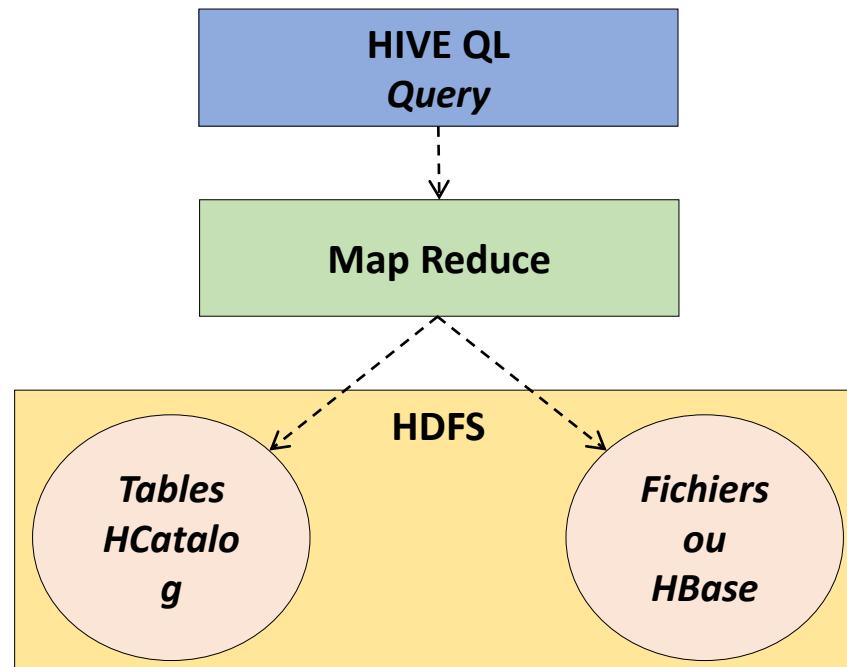
# Architecture

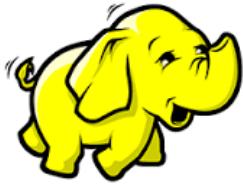
- Pour l'intégration directe des fonctions Map et Reduce sous forme d'UDF (User Defined Functions) dans la requête HiveQL, HIVE s'appuie sur Apache Thrift qui lui permet entre autres d'écrire les UDF en plusieurs langages de programmation (Java, Python, Ruby...)



# Architecture

## Schema Process

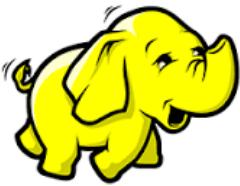




# Architecture

Composants de Hive :

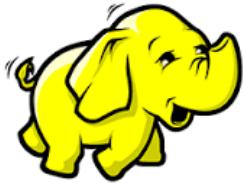
- Interface utilisateur
  - Permet de soumettre les requêtes à exécuter
- Drivers
  - Réception des requêtes à exécuter
  - Gestion des sessions
  - Interface ODBC/JDBC



# Architecture

Composants de Hive :

- Compilateur
  - Parse les requêtes (analyse sémantique des blocs de la requête)
  - Récupère les métadonnées
  - Génère les plans d'exécution des requêtes
- Metastore
  - Stocke la structure de l'information des différentes tables (colonnes, types de données, ...) et partitions de la base de données
  - Stocke les serializers/deserializers (SerDe) nécessaire pour lire et écrire la donnée dans les fichiers HDFS concernés
- Moteur d'exécution: Exécute le plan d'exécution



# Modèle de données

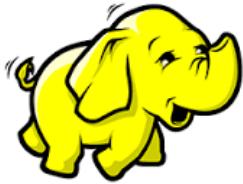
## Tables

- Colonnes, types de données (int, float, string, ...)
- Egalement list, map pour les données de type JSON

## Partitions

- Permet de répartir/classer les données par plage (Ex: plage de date – une partition par mois)

```
CREATE TABLE logs (ts BIGINT, line STRING)  
PARTITIONED BY (dt STRING, country STRING);
```

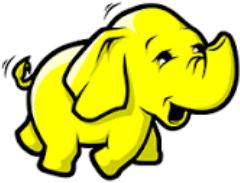


# Modèle de données

## Buckets

- Colonnes, Permet de sous-diviser les partition par clé de hash

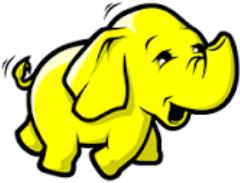
```
CREATE TABLE bucketed_users (id INT, name STRING)  
CLUSTERED BY (id) INTO 4 BUCKETS;
```



# Modèle de données

## Table

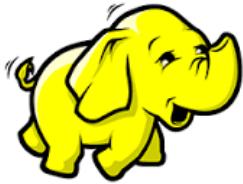
- Une table Hive est constitué des données stockées et des métadonnées associées.
- Les données sont stockées généralement dans HDFS, ou dans le système de fichier local ou dans un stockage objet tel S3.
- Les métadonnées quand à eux sont stockées dans une base de données relationnelle.
- Chaque table Hive est matérialisée dans HDFS par un répertoire.
- Lorsque une table est créée dans Hive, par défaut les données sont stocké dans répertoire de l'entrepôt de données géré par Hive



# Modèle de données

## Partitions

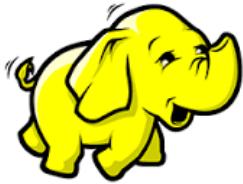
- Hive Organise les tables en partitions. Les partitions se font à base de valeur de colonne de partitionnement. Par exemple la date. L'utilisation de cette technique permet un accès plus rapide aux données (requête sur une ou des tranches de données).
- Chaque table peut avoir une ou plusieurs partitions qui déterminent la distribution des données dans des sous-répertoires du répertoire de la table.



# Modèle de données

## Buckets

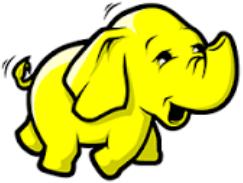
- Deux raisons principales :
- 1- Requêtes plus efficaces : Le Bucketing impose une structure supplémentaire sur la table. La jointure de deux tables est plus.
- 2- Échantillonnages plus efficace : Lorsque on travaille avec un grand ensemble de données, il est très pratique de tester la requête sur une fraction



# Modèle de données

## Arborescence

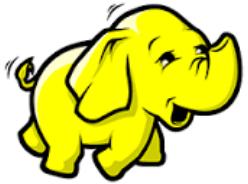
- Répertoire Warehouse dans le système de fichier HDFS: /home/hive/warehouse
- Tables stockées dans des sous-répertoires du warehouse
- Partitions, buckets dans des sous-répertoires des tables
- Données stockées dans des fichiers plats =simple fichier text(délimités ou séquentiels)
- Possibilité d'utiliser d'autres formats de données(parquet par exemple)



# Hive QL

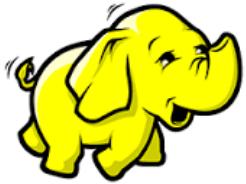
## Types de données

- Les types de données primitifs:
  - TINYINT, SMALLINT, INT, BIGINT
  - BOOLEAN
  - FLOAT, DOUBLE
  - STRING
  - TIMESTAMP
  - BINARY
- Les types de données complexes :
  - Struct : Encapsule plusieurs champs
  - Map : Tuples clé-valeur
  - Array



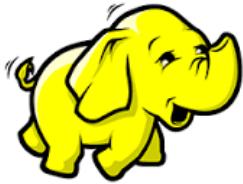
# Hive QL

```
CREATE TABLE test (
    col1 ARRAY<INT>,
    col2 MAP<STRING, INT>,
    col3 STRUCT<a:STRING, b:INT, c:DOUBLE>
) ;
```



# Hive QL

Caractéristiques	SQL	HiveQL
<b>Updates</b>	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (alimente toute la table ou la partition)
<b>Transactions</b>	Supporté	Non supporté
<b>Indexes</b>	Supporté	Non supporté
<b>Fonctions</b>	>100 fonctions internes	≈15 fonctions internes
<b>Multitable inserts</b>	Non supporté	Supporté
<b>Create table as select</b>	Dépend des BDD	Supporté
<b>Clause HAVING</b>	Supporté	Non supporté
<b>Jointures</b>	Inner joins, outer joins	Inner joins, outer joins, semi joins



# Hive QL

## INNER JOIN

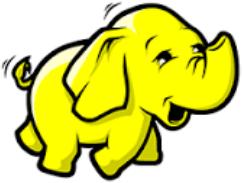
- partie commun entre deux ensembles

## outer joins

- full jointe: la partie commune et le reste aussi

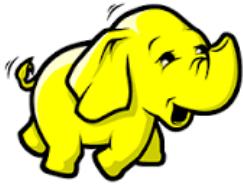
## semi joins

- le résultat ne filtre que les attributs de l'une des deux



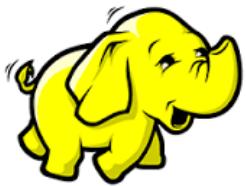
# Hive QL

- HiveQL supporte les opération de :
  - Selection
  - Jointure
  - Aggregation
  - Union
  - Ainsi que les sous-requêtes
- HiveQL supporte le langage de définition de données (DDL):
  - Tables avec des formats de sérialisation spécifiques
  - Partitions
  - Buckets
- Hive permet à travers le langage de manipulation de données (DML) le chargement des données dans les tables gérées par Hive. Cette manipulation par les commandes load et insert.



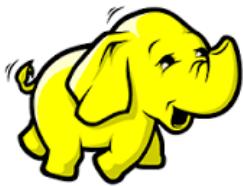
# Format de fichier

- Le format de fichier détermine le format des champs d'une ligne.
- Le format le plus simple est le format texte
  - STORED AS TEXTFILE
  - STORED AS SEQUENCEFILE
  - STORED AS PARQUET
  - STORED AS ORC
  - STORED AS AVRO
  - ...



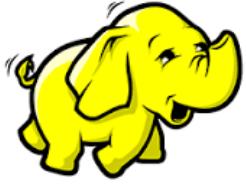
# Hive QL

Function	MySQL	HiveQL
Retrieving information	<code>SELECT from_columns FROM table WHERE conditions;</code>	<code>SELECT from_columns FROM table WHERE conditions;</code>
All values	<code>SELECT * FROM table;</code>	<code>SELECT * FROM table;</code>
Some values	<code>SELECT * FROM table WHERE rec_name = "value";</code>	<code>SELECT * FROM table WHERE rec_name = "value";</code>
Multiple criteria	<code>SELECT * FROM table WHERE rec1="value1" AND rec2="value2";</code>	<code>SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";</code>
Selecting specific columns	<code>SELECT column_name FROM table;</code>	<code>SELECT column_name FROM table;</code>
Retrieving unique output records	<code>SELECT DISTINCT column_name FROM table;</code>	<code>SELECT DISTINCT column_name FROM table;</code>
Sorting	<code>SELECT col1, col2 FROM table ORDER BY col2;</code>	<code>SELECT col1, col2 FROM table ORDER BY col2;</code>
Sorting backward	<code>SELECT col1, col2 FROM table ORDER BY col2 DESC;</code>	<code>SELECT col1, col2 FROM table ORDER BY col2 DESC;</code>
Counting rows	<code>SELECT COUNT(*) FROM table;</code>	<code>SELECT COUNT(*) FROM table;</code>
Grouping with counting	<code>SELECT owner, COUNT(*) FROM table GROUP BY owner;</code>	<code>SELECT owner, COUNT(*) FROM table GROUP BY owner;</code>
Maximum value	<code>SELECT MAX(col_name) AS label FROM table;</code>	<code>SELECT MAX(col_name) AS label FROM table;</code>
Selecting from multiple tables (Join same table using alias w/"AS")	<code>SELECT pet.name, comment FROM pet, event WHERE pet.name = event.name;</code>	<code>SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name);</code>



# Hive QL

Function	MySQL	HiveQL
Selecting a database	USE database;	USE database;
Listing databases	SHOW DATABASES;	SHOW DATABASES;
Listing tables in a database	SHOW TABLES;	SHOW TABLES;
Describing the format of a table	DESCRIBE table;	DESCRIBE (FORMATTED EXTENDED) table;
Creating a database	CREATE DATABASE db_name;	CREATE DATABASE db_name;
Dropping a database	DROP DATABASE db_name;	DROP DATABASE db_name (CASCADE);

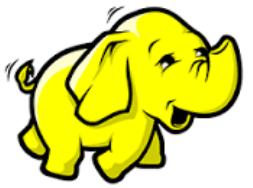


# Hive avec Spark

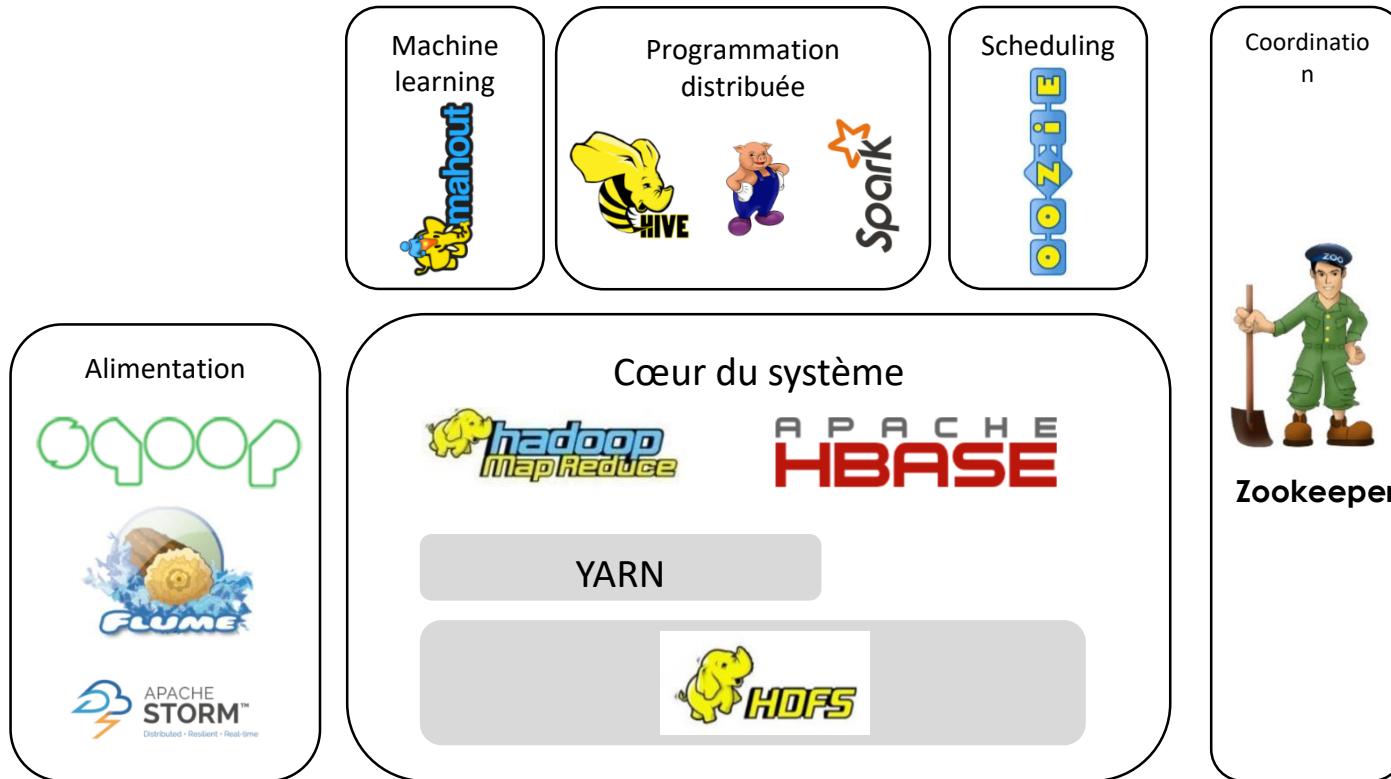
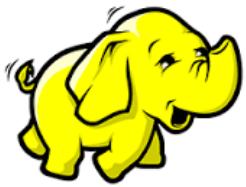
```
val spark = SparkSession  
  .builder()  
  .appName("SparkHiveApplication")  
  .enableHiveSupport()  
  .getOrCreate()
```

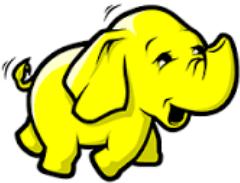
```
sql("SELECT * FROM src").show()
```

```
val df = spark.table("database.table")  
df.write.mode(SaveMode.Overwrite).saveAsTable("hive_records")
```



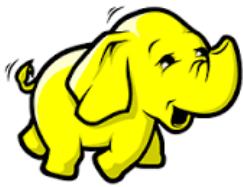
# Sqoop





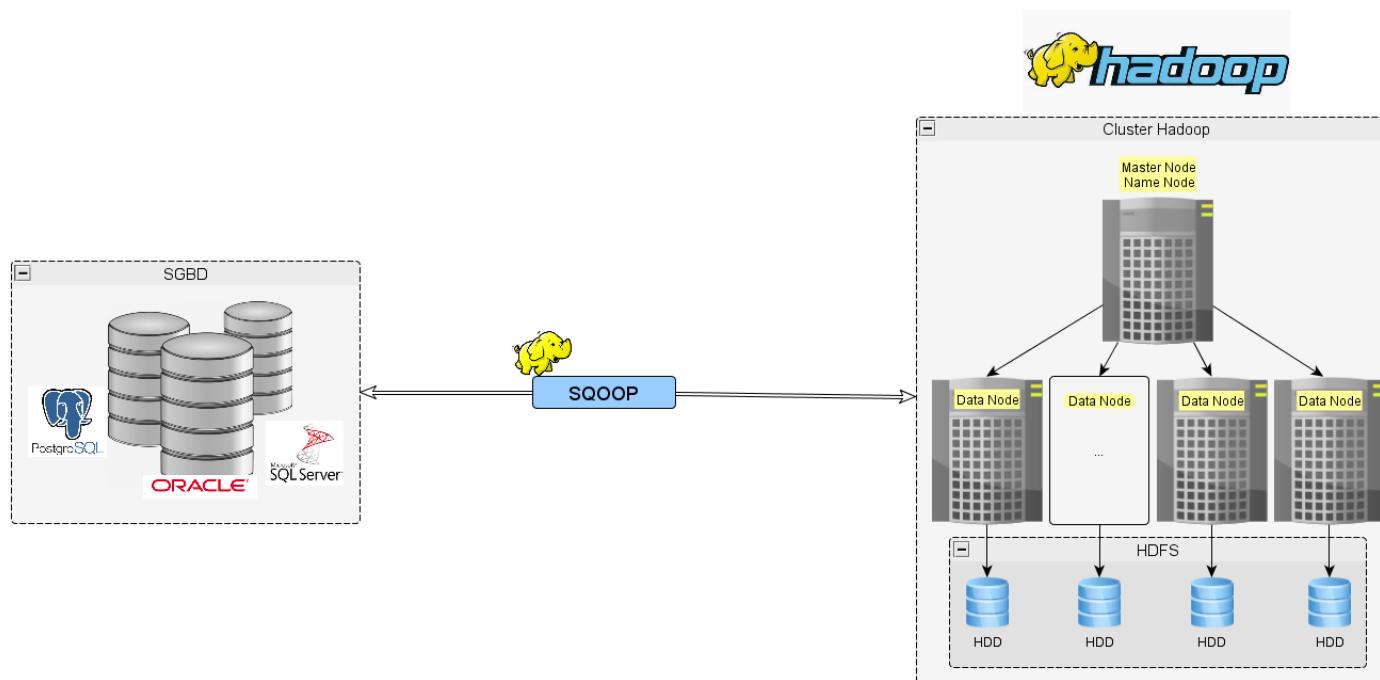
## Sqoop: qu'est-ce ?

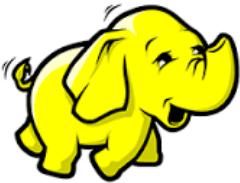
- Outil Open source qui est sorti de l'incubateur Apache en Mars 2012.
- Sqoop est un outil conçu pour transférer des données entre Hadoop et des bases de données relationnelles.
- Objectif Principal :
  - Assurer des performances élevées pour des opérations d'import ou d'export de données massif.
  - Permet de procéder à la collecte de données au sein d'applications traditionnelles n'ayant pas la capacité de se connecter au cluster
  - Peu adapté au temps réel



# Sqoop : quelle utilité ?

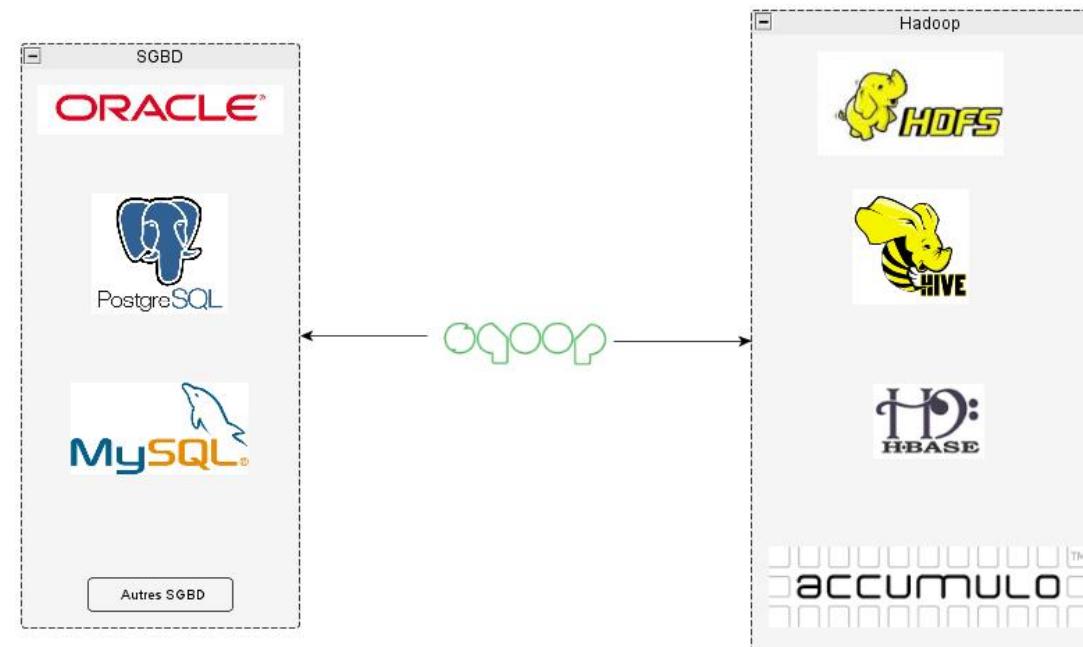
- Outil de transfert de données depuis une base Relationnelle Vers Hadoop et inversement

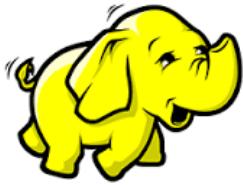




# Cible des imports

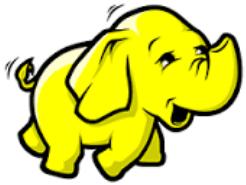
- Sqoop permet d'importer des données dans le HDFS, Hive, Hbase et Accumulo
- Apache accumulo : base de données NoSQL, de type clé/valeur





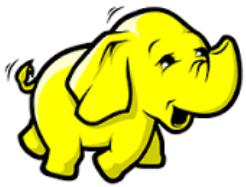
## Sqoop 1 versus Sqoop 2

- Sqoop 1 est la version stable actuellement : Outil en lignes de commandes uniquement (pas d'API Java, etc...)
- Sqoop 2 est la réécriture de Sqoop 1, très actif en ce moment :
  - Dispose d'un composant serveur qui exécute les jobs
  - Dispose toujours d'une Interface en lignes de commandes
  - Web UI
  - Rest API Java API
  - Possibilité d'utiliser un moteur d'exécution alternatif (Spark par exemple)



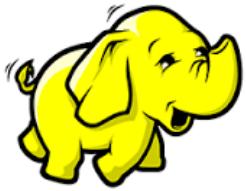
## Fonctionnement : les connecteurs

- Sqoop permet de se connecter aux bases relationnelles au travers de connecteurs (Drivers JDBC)
  - MySQL, PostgreSQL, Oracle, SQL Server, DB2, Netezza,...
  - Un driver JDBC générique également présent
  - Possibilité de télécharger des drivers spécifiques
  - Fonctionnement optimisé pour les bases de données MySQL, PostgreSQL, Oracle et Netezza



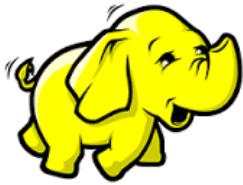
## Fonctionnement: sources des imports

- Sqoop permet d'importer, depuis une base de donnée relationnelle :
  - Toutes les tables d'une base de données
  - Une ou plusieurs tables
  - Une vue
  - Le résultat d'une requête SQL



## MapReduce et HDFS

- Le chargement de données avec Sqoop génère un job MapReduce parallélisé (4 par défaut, paramétrable grâce à l'option –m <Nb de tâches parallèles>)
- Le fichier généré est distribué sur le cluster HDFS (chaque processus MAP REDUCE écrit son morceau de fichier)
- De plus, HDFS redonde ces morceaux de fichiers sur plusieurs nœuds différents. En cas de perte d'un nœud, le master utilisera une des copies d'un autre nœud.



## Where et imports incrémentaux

Argument --Where : Il est possible de n'importer qu'une partie d'une table en appliquant une clause « Where »

--where Table\_id > 5000

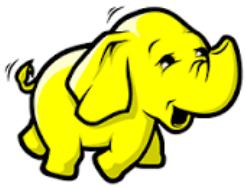
Imports incrémentaux :

- mode "append" (argument --incremental append)

Permet de n'importer que les nouveaux enregistrements mais ne met pas à jour les éventuels enregistrements déjà importés et modifiés depuis.

- mode " lastmodified" (argument --incremental lastmodified)

En plus d'importer les nouveaux enregistrements, permet la mise à jour des enregistrements préalablement importés et modifiés depuis sur la base source

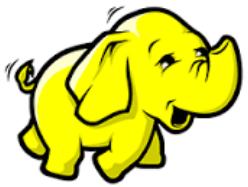


# Sqoop help

```
admin@master: ~
admin@master:~$ sqoop help
Warning: /opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/bin/.../lib/sqoop/.../acc
umulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
15/11/19 14:54:52 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5-cdh5.4.3
usage: sqoop COMMAND [ARGS]

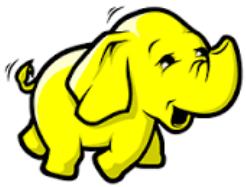
Available commands:
codegen          Generate code to interact with database records
create-hive-table Import a table definition into Hive
eval             Evaluate a SQL statement and display the results
export            Export an HDFS directory to a database table
help              List available commands
import             Import a table from a database to HDFS
import-all-tables Import tables from a database to HDFS
import-mainframe  Import datasets from a mainframe server to HDFS
job                Work with saved jobs
list-databases    List available databases on a server
list-tables        List available tables in a database
merge             Merge results of incremental imports
metastore          Run a standalone Sqoop metastore
version           Display version information

See 'sqoop help COMMAND' for information on a specific command.
admin@master:~$
```



```
admin@master:~$ sqoop help import
Warning: /opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/bin/../lib/sqoop/../acc
umulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
15/11/19 14:56:29 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5-cdh5.4.3
usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]

Common arguments:
  --connect <jdbc-uri>                                Specify JDBC connect
                                                       string
  --connection-manager <class-name>                      Specify connection manager
                                                       class name
  --connection-param-file <properties-file>              Specify connection
                                                       parameters file
  --driver <class-name>                                  Manually specify JDBC
                                                       driver class to use
  --hadoop-home <hdir>                                 Override
                                                       $HADOOP_MAPRED_HOME_ARG
  --hadoop-mapred-home <dir>                           Override
                                                       $HADOOP_MAPRED_HOME_ARG
  --help                                                 Print usage instructions
-P                                                    Read password from console
  --password <password>                                Set authentication
                                                       password
  --password-alias <password-alias>                     Credential provider
                                                       password alias
  --password-file <password-file>                      Set authentication
                                                       password file path
  --relaxed-isolation                                Use read-uncommitted
                                                       isolation for imports
```



# Exemples d'imports vers HDFS

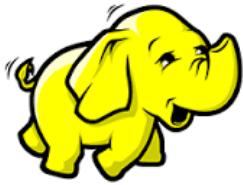
- Exemple d'import de données issues d'une base PostgreSQL
- Données contenues dans la table :

A screenshot of a terminal window titled "admin@master: /". The window displays a PostgreSQL query result. The command run is "select \* from calendrier\_ouvre;". The output shows 10 rows of data from January 4, 2016, to January 15, 2016, mapping dates to days of the week and months.

id_jour	jour_ouvre	mois	annee	libelle_jour	libelle_mois	jour_semaine
2016-01-04	20160104	201601	2016	lundi 4 janvier 2016	janv-16	lundi
2016-01-05	20160105	201601	2016	mardi 5 janvier 2016	janv-16	mardi
2016-01-06	20160106	201601	2016	mercredi 6 janvier 2016	janv-16	mercredi
2016-01-07	20160107	201601	2016	jeudi 7 janvier 2016	janv-16	jeudi
2016-01-08	20160108	201601	2016	vendredi 8 janvier 2016	janv-16	vendredi
2016-01-11	20160111	201601	2016	lundi 11 janvier 2016	janv-16	lundi
2016-01-12	20160112	201601	2016	mardi 12 janvier 2016	janv-16	mardi
2016-01-13	20160113	201601	2016	mercredi 13 janvier 2016	janv-16	mercredi
2016-01-14	20160114	201601	2016	jeudi 14 janvier 2016	janv-16	jeudi
2016-01-15	20160115	201601	2016	vendredi 15 janvier 2016	janv-16	vendredi

(10 rows)

```
postgres=#
```



# Exemples d'imports vers HDFS

Import d'une table :

## Ligne de commande Sqoop :

```
sqoop import --connect jdbc:postgresql://master:5432/test \
--username postgres --password postgres \
--table calendrier_ouvre \
--delete-target-dir --target-dir /user/admin/calendrier/Table_calendrier
```

## Détails :

La commande peut être lancée sur plusieurs lignes grâce au caractère \

--connect : suivi de la chaîne de connexion <jdbc-uri>

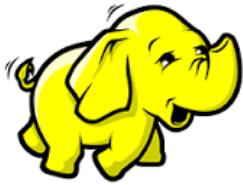
--username et --password : User et mot de passe de la base de données

--table : permet de spécifier la table à importer

--delete-target-dir : permet d'effacer le répertoire de destination

--target-dir : spécifie l'emplacement du fichier cible

--split-by : permet de préciser la colonne utilisée pour la parallélisation



# Exemples d'imports vers HDFS

Import d'une requête SQL :

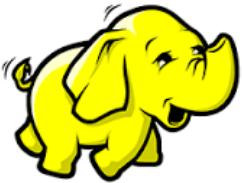
## Ligne de commande Sqoop :

```
sqoop import --connect jdbc:postgresql://127.0.0.1:5432/test \  
--username postgres --password postgres \  
--query 'select ID_JOUR,JOUR_OUVRE,MOIS,ANNEE,LIBELLE_JOUR, \  
LIBELLE_MOIS,JOUR_SEMAINE from calendrier_ouvre WHERE $CONDITIONS' \  
--delete-target-dir --target-dir /user/admin/calendrier/Table_calendrier \  
--split-by ID_JOUR
```

## Détails :

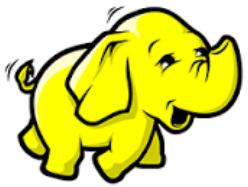
--query : permet de spécifier une requête SQL au lieu d'une table

--split-by : permet de préciser la colonne utilisée pour la parallélisation



# Exemples d'imports : vers HDFS

```
admin@master:~$ sqoop import --connect jdbc:postgresql://master:5432/test --username postgres --password postgres --table calendrier_ouvre --delete-target-dir --target-dir /user/admin/calendrier/Table_calendrier
Warning: /opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/bin/../lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
15/11/20 15:03:53 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5-cdh5.4.3
15/11/20 15:03:53 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
15/11/20 15:03:54 INFO manager.SqlManager: Using default fetchSize of 1000
15/11/20 15:03:54 INFO tool.CodeGenTool: Beginning code generation
15/11/20 15:03:54 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM "calendrier_ouvre" AS t LIMIT 1
15/11/20 15:03:54 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce
Note: /tmp/sqoop-admin/compile/ecd0fad767a0ae72dc658306b10c71de/calendrier_ouvre.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
15/11/20 15:03:56 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-admin/compile/ecd0fad767a0ae72dc658306b10c71de/calendrier_ouvre.jar
15/11/20 15:03:58 INFO tool.ImportTool: Destination directory /user/admin/calendrier/Table_calendrier deleted.
15/11/20 15:03:58 WARN manager.PostgresSQLManager: It looks like you are importing from postgresql.
15/11/20 15:03:58 WARN manager.PostgresSQLManager: This transfer can be faster! Use the --direct
15/11/20 15:03:58 WARN manager.PostgresSQLManager: option to exercise a postgresql-specific fast path.
15/11/20 15:03:58 INFO mapreduce.ImportJobBase: Beginning import of calendrier_ouvre
15/11/20 15:03:58 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
15/11/20 15:03:58 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
15/11/20 15:03:58 INFO client.RMProxy: Connecting to ResourceManager at master/167.114.255.92:8032
15/11/20 15:04:02 INFO db.DBInputFormat: Using read committed transaction isolation
15/11/20 15:04:02 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN("id_jour"), MAX("id_jour") FROM "calendrier_ouvre"
15/11/20 15:04:02 INFO mapreduce.JobSubmitter: number of splits:4
15/11/20 15:04:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1447943110228_0018
15/11/20 15:04:02 INFO impl.YarnClientImpl: Submitted application application_1447943110228_0018
15/11/20 15:04:02 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1447943110228_0018/
15/11/20 15:04:02 INFO mapreduce.Job: Running job: job_1447943110228_0018
15/11/20 15:04:11 INFO mapreduce.Job: Job job_1447943110228_0018 running in uber mode : false
15/11/20 15:04:11 INFO mapreduce.Job: map 0% reduce 0%
15/11/20 15:04:18 INFO mapreduce.Job: map 50% reduce 0%
```

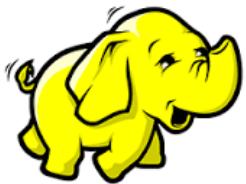


# Exemple d'imports vers HDFS

- Fichiers générés : on remarque qu'il y a 4 fichiers ( 4 tâches parallèles)

A screenshot of a Windows Command Prompt window titled "admin@master: /". The window displays the output of the command "hadoop fs -ls /user/admin/calendrier/Table\_calendrier". The output shows five items found, including four data files and one SUCCESS file. The data files are named part-m-00000, part-m-00001, part-m-00002, and part-m-00003, each with a size of approximately 327, 498, 167, and 664 respectively, all modified on 2015-11-20 at 15:04. A green cursor is visible at the bottom left of the terminal window.

```
admin@master:/$ hadoop fs -ls /user/admin/calendrier/Table_calendrier
Found 5 items
-rw-r--r--  2 admin admin      0 2015-11-20 15:04 /user/admin/calendrier/Table_calendrier/_SUCCESS
-rw-r--r--  2 admin admin   327 2015-11-20 15:04 /user/admin/calendrier/Table_calendrier/part-m-00000
-rw-r--r--  2 admin admin   498 2015-11-20 15:04 /user/admin/calendrier/Table_calendrier/part-m-00001
-rw-r--r--  2 admin admin   167 2015-11-20 15:04 /user/admin/calendrier/Table_calendrier/part-m-00002
-rw-r--r--  2 admin admin   664 2015-11-20 15:04 /user/admin/calendrier/Table_calendrier/part-m-00003
admin@master:/$
```



## Exemple d'imports vers HDFS

A screenshot of a desktop environment showing four separate terminal windows, each representing a different part of an HDFS file. The windows are stacked vertically. Each window has a title bar with a blue icon, the text "admin@master: /", and a red close button. The terminal content is displayed in white text on a black background.

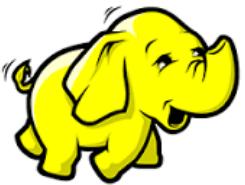
- The top window shows the command "hadoop fs -cat /user/admin/calendrier/Table\_calendrier/part-m-00000" and its output:

```
2016-01-04,20160104,201601,2016,lundi 4 janvier 2016,janv-16,lundi  
2016-01-05,20160105,201601,2016,mardi 5 janvier 2016,janv-16,mardi
```
- The second window shows the command "hadoop fs -cat /user/admin/calendrier/Table\_calendrier/part-m-00001" and its output:

```
2016-01-06,20160106,201601,2016,mercredi 6 janvier 2016,janv-16,mercredi  
2016-01-07,20160107,201601,2016,jeudi 7 janvier 2016,janv-16,jeudi  
2016-01-08,20160108,201601,2016,vendredi 8 janvier 2016,janv-16,vendredi
```
- The third window shows the command "hadoop fs -cat /user/admin/calendrier/Table\_calendrier/part-m-00002" and its output:

```
2016-01-11,20160111,201601,2016,lundi 11 janvier 2016,janv-16,lundi
```
- The bottom window shows the command "hadoop fs -cat /user/admin/calendrier/Table\_calendrier/part-m-00003" and its output:

```
2016-01-12,20160112,201601,2016,mardi 12 janvier 2016,janv-16,mardi  
2016-01-13,20160113,201601,2016,mercredi 13 janvier 2016,janv-16,mercredi  
2016-01-14,20160114,201601,2016,jeudi 14 janvier 2016,janv-16,jeudi  
2016-01-15,20160115,201601,2016,vendredi 15 janvier 2016,janv-16,vendredi
```



## Exemples d'imports vers Hive

Importer des données dans la base Hive

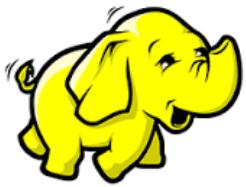
Ligne de commande Sqoop :

```
sqoop import --connect jdbc:postgresql://master:5432/postgres --delete-target-dir  
--table calendrier_ouvre --hive-database exemples --hive-import --hive-drop-import-delims  
--hive-overwrite --hive-table calendrier_ouvre  
• --hive-drop-import-delims : supprimer les délimiteur lors de l'import dans une table hive
```

A screenshot of a terminal window titled "admin@master: ~". The window contains a command-line interface with the following text:

```
postgres@master:/home/admin$ sqoop import --connect jdbc:postgresql://master:5432/postgr  
es --delete-target-dir --table calendrier_ouvre --hive-database exemples --hive-import  
--hive-drop-import-delims --hive-overwrite --hive-table calendrier_ouvre
```

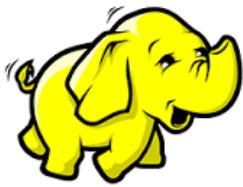
The command is intended to import data from a PostgreSQL database table named "calendrier\_ouvre" into a Hive table with the same name, located in the "exemples" database. The "--hive-drop-import-delims" option is used to remove delimiters during the import process.



# Exemples d'imports vers Hive

```
admin@master: ~
admin@master:~$ hive

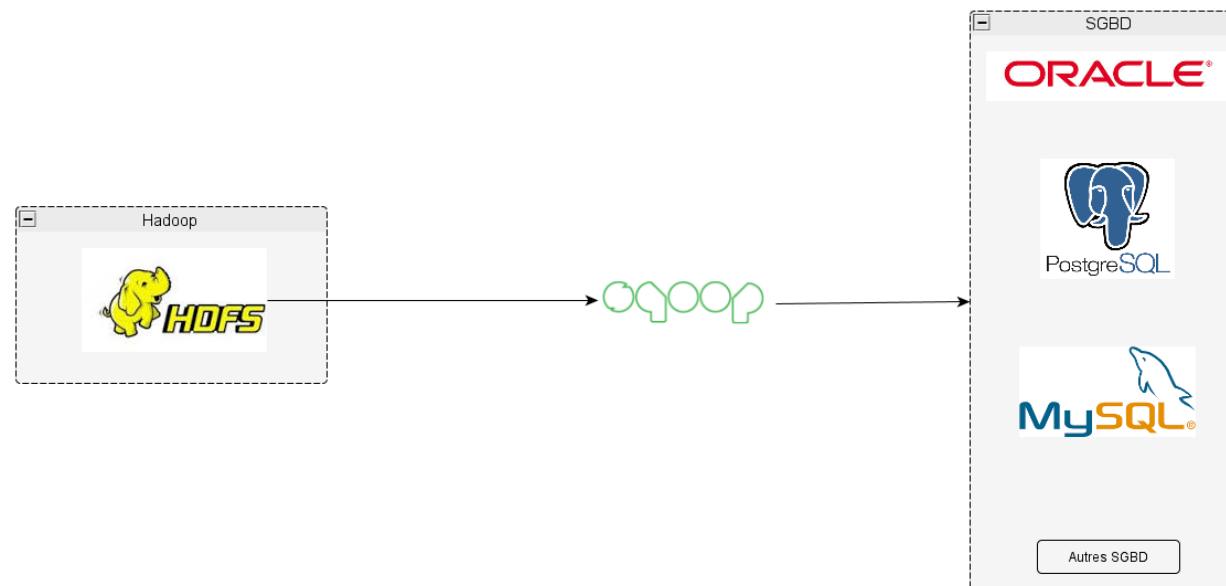
Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/jars/hive-common-1.1.0-cdh5.4.3.jar!/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> show databases;
OK
default
exemples
Time taken: 1.721 seconds, Fetched: 2 row(s)
hive> use exemples;
OK
Time taken: 0.036 seconds
hive> show tables;
OK
calendrier_ouvre
Time taken: 0.056 seconds, Fetched: 1 row(s)
hive> select * from calendrier_ouvre;
OK
2016-01-04      20160104      201601  2016    lundi 4 janvier 2016    janv-16 lundi
2016-01-05      20160105      201601  2016    mardi 5 janvier 2016    janv-16 mardi
2016-01-06      20160106      201601  2016    mercredi 6 janvier 2016 janv-16 mercredi
2016-01-07      20160107      201601  2016    jeudi 7 janvier 2016    janv-16 jeudi
2016-01-08      20160108      201601  2016    vendredi 8 janvier 2016 janv-16 vendredi
2016-01-11      20160111      201601  2016    lundi 11 janvier 2016 janv-16 lundi
2016-01-12      20160112      201601  2016    mardi 12 janvier 2016 janv-16 mardi
2016-01-13      20160113      201601  2016    mercredi 13 janvier 2016 janv-16 mercredi
2016-01-14      20160114      201601  2016    jeudi 14 janvier 2016    janv-16 jeudi
2016-01-15      20160115      201601  2016    vendredi 15 janvier 2016 janv-16 vendredi
Time taken: 0.807 seconds, Fetched: 10 row(s)
hive> █
```

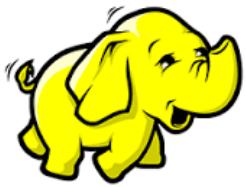


## Exemples d'exports : source possible

A la différence de l'import de Sqoop :

- L'export Sqoop ne permet de se baser que sur un fichier présent dans le HDFS
- Il n'est pas possible de se baser sur une base Hive ou Hbase par exemple.
- Et que sur un table déjà créée



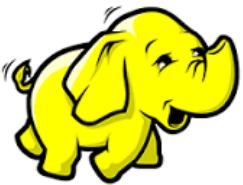


## Exemple d'exports : fichier source

- Exemple d'export depuis un fichier sur le HDFS vers une base PostgreSQL
- Données contenues dans le fichier source :

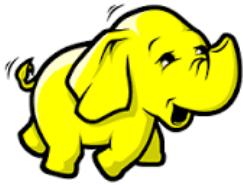
A screenshot of a terminal window titled "admin@master: ~". The window displays the output of a command: "hadoop fs -cat /user/admin/departements/part-m-00000". The output lists 31 French departments with their names, regions, and codes.

```
postgres@master:/home/admin$ hadoop fs -cat /user/admin/departements/part-m-00000
01,Ain,Bourg-en-Bresse,Rhône-Alpes,82,FR-V
02,Aisne,Laon,Picardie,22,FR-S
03,Allier,Moulins,Auvergne,83,FR-C
04,Alpes de Hautes-Provence,Digne,Provence-Alpes-Côte d Azur,93,FR-U
05,Hautes-Alpes,Gap,Provence-Alpes-Côte d Azur,93,FR-U
06,Alpes-Maritimes,Nice,Provence-Alpes-Côte d Azur,93,FR-U
07,Ardèche,Prives,Rhône-Alpes,82,FR-V
08,Ardennes,Charleville-Mézières,Champagne-Ardenne,21,FR-G
09,Ariège,Foix,Midi-Pyrénées,73,FR-N
10,Aube,Troyes,Champagne-Ardenne,21,FR-G
11,Aude,Carcassonne,Languedoc-Roussillon,91,FR-K
12,Aveyron,Rodez,Midi-Pyrénées,73,FR-N
13,Bouches-du-Rhône,Marseille,Provence-Alpes-Côte d Azur,93,FR-U
14,Calvados,Caen,Basse-Normandie,25,FR-P
15,Cantal,Aurillac,Auvergne,83,FR-C
16,Charente,Angoulême,Poitou-Charentes,54,FR-T
17,Charente-Maritime,La Rochelle,Poitou-Charentes,54,FR-T
18,Cher,Bourges,Centre,24,FR-F
19,Corrèze,Tulle,Limousin,74,FR-L
2A,Corse-du-Sud,Ajaccio,Corse,94,FR-H
2B,Haute-Corse,Bastia,Corse,94,FR-H
21,Côte-d Or,Dijon,Bourgogne,26,FR-D
22,Côtes d Armor,Saint-Brieuc,Bretagne,53,FR-E
23,Creuse,Guéret,Limousin,74,FR-L
24,Dordogne,Périgueux,Aquitaine,72,FR-B
25,Doubs,Besançon,Franche-Comté,43,FR-I
26,Drôme,Valence,Rhône-Alpes,82,FR-V
27,Eure,Évreux,Haute-Normandie,23,FR-Q
28,Eure-et-Loir,Chartres,Centre,24,FR-F
29,Finistère,Quimper,Bretagne,53,FR-E
30,Gard,Nîmes,Languedoc-Roussillon,91,FR-K
31,Haute-Garonne,Toulouse,Midi-Pyrénées,73,FR-N
```



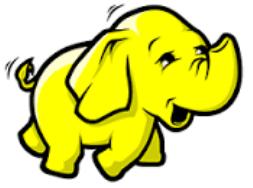
## Exemple d'exports: ligne de commande

```
admin@master: ~
postgres@master:/home/admin$ sqoop export --connect jdbc:postgresql://master:5432/postgres --username postgres --password postgres --table departements --export-dir /user/admin/departements
Warning: /opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/bin/../../lib/sqoop/../../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
15/11/23 12:30:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5-cdh5.4.3
15/11/23 12:30:32 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
15/11/23 12:30:32 INFO manager.SqlManager: Using default fetchSize of 1000
15/11/23 12:30:32 INFO tool.CodeGenTool: Beginning code generation
15/11/23 12:30:32 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM "departements" AS t LIMIT 1
15/11/23 12:30:32 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce
Note: /tmp/sqoop-postgres/compile/67e3b4dc41f32172a4751671c4306bdd/departements.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
15/11/23 12:30:34 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-postgres/compile/67e3b4dc41f32172a4751671c4306bdd/departements.jar
15/11/23 12:30:34 INFO mapreduce.ExportJobBase: Beginning export of departements
15/11/23 12:30:35 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
15/11/23 12:30:36 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative
15/11/23 12:30:36 INFO Configuration.deprecation: mapred.map.tasks.speculative.execution is deprecated. Instead, use mapreduce.map.speculative
15/11/23 12:30:36 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
15/11/23 12:30:36 INFO client.RMProxy: Connecting to ResourceManager at master/167.114.2.55:8032
15/11/23 12:30:39 INFO input.FileInputFormat: Total input paths to process : 1
15/11/23 12:30:39 INFO input.FileInputFormat: Input paths to process : 1
```

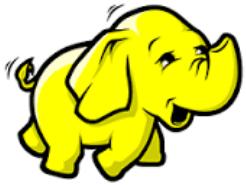


# Fonctions principales

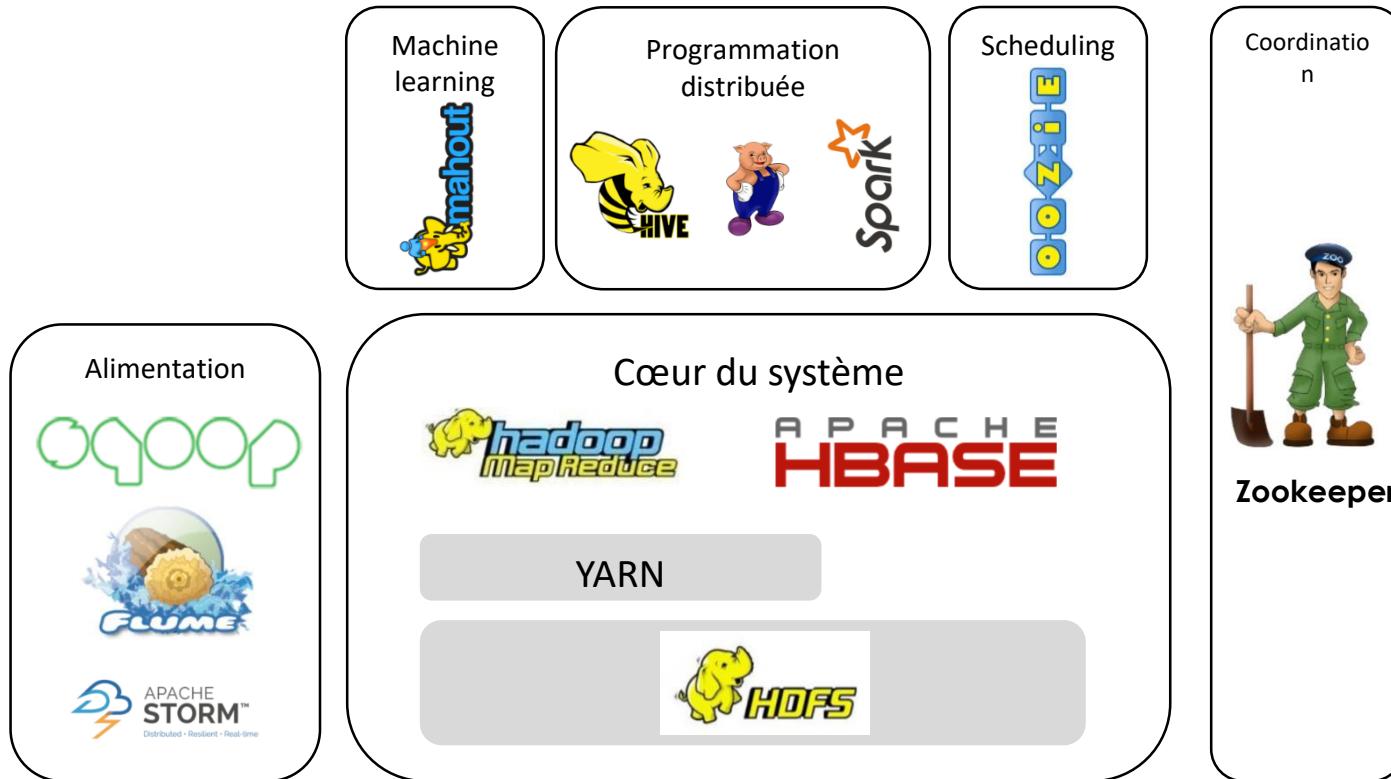
- `sqoop version`
- `sqoop import-all-tables --connect jdbc:mysql://db.foo.com/corp`
- `sqoop eval --connect jdbc:mysql://db.example.com/corp \ --query "SELECT * FROM employees LIMIT 10"`
- `sqoop list-databases --connect jdbc:mysql://database.example.com`
- `sqoop list-tables --connect jdbc:mysql://database.example.com/corp`
- `sqoop codegen --connect jdbc:mysql://db.example.com/corp --table employees`

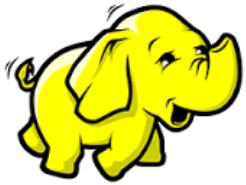


# Oozie



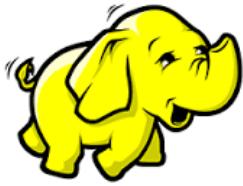
# Oozie





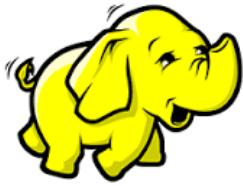
# Introduction

- Apache Oozie est un système de planification permettant d'exécuter et de gérer des tâches Hadoop dans un environnement distribué.
- Oozie est une application Web Java Open Source disponible sous licence Apache 2.0. Il est responsable du déclenchement des actions de workflow. Client-serveur
- Il permet de combiner plusieurs tâches complexes dans un ordre séquentiel pour accomplir une tâche plus importante.
- Dans une séquence de tâches, deux ou plusieurs job peuvent également être programmés pour fonctionner parallèlement les uns aux autres.
- Les définitions des workflows Oozie sont écrites en hPDL (un langage de définition du processus XML) À l'origine.
- Prend en charge divers job Hadoop tels que Hive, Pig, Sqoop, ainsi que des tâches spécifiques au système telles que Java et Shell.
- conçus par Yahoo! pour leur recherche complexe sur les flux



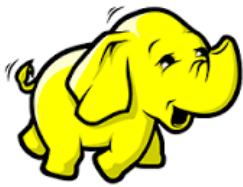
## Pourquoi l'utiliser?

- Conçu pour des traitements Big Data.
- Permet de faire des actions dans un workflow.
- Permet de gérer les actions avec : START,FORK ,JOIN,DECISION,KILL,END
- Permet de gérer les jobs avec les commandes : Stop, Start, Suspend, Resume, Run..

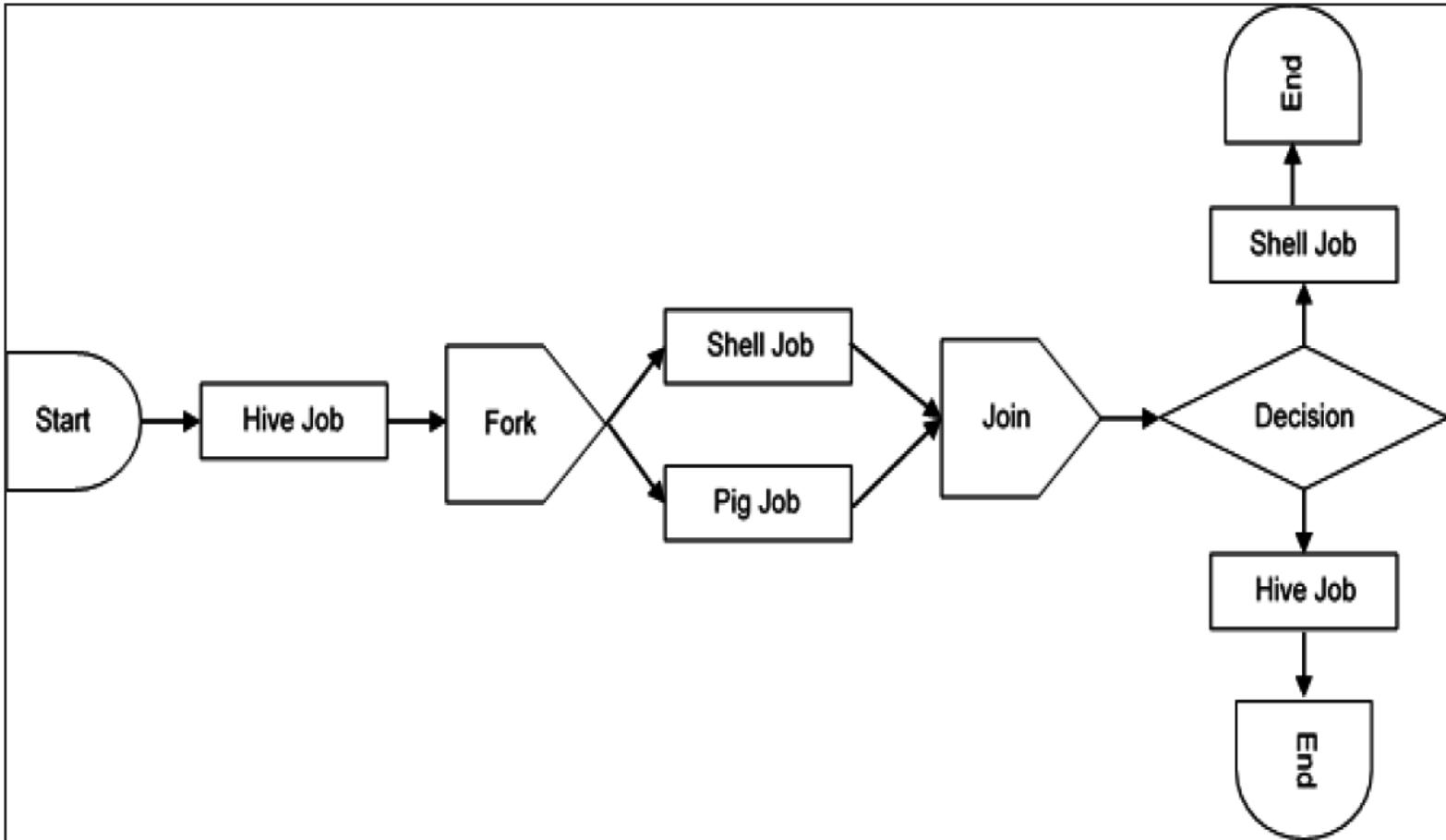


## Types de job

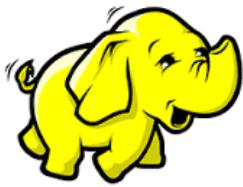
- Oozie Workflow Jobs : représenté en tant que graphes acycliques dirigés (DAG) pour spécifier une séquence d'actions à exécuter.
- Oozie Coordinator Jobs : consistent en des tâches de workflow déclenchées par l'heure et la disponibilité des données.
- Oozie Bundle : un package de plusieurs jobs de coordinator et de workflow.



# Exemple workflow

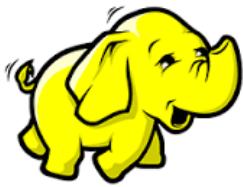


- Controls : Start, Decision, Fork, Join and End
- Actions : Hive, Shell, Pig



## Oozie Editors

- Ecrivez les flux de travail en utilisant n'importe quel éditeur de texte populaire (comme Notepad ++, Sublime ou Atom)
- Créer un flux de travail par la méthode drag and drop pour gérer un workflow.xml. Le plus populaire parmi les éditeurs Oozie est Hue.
- Oozie Eclipse Plugin (OEP) : éditer graphiquement les workflows
- site officiel : <http://oep.mashin.io/>

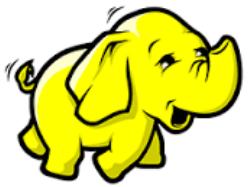


# Oozie Editors : Hue

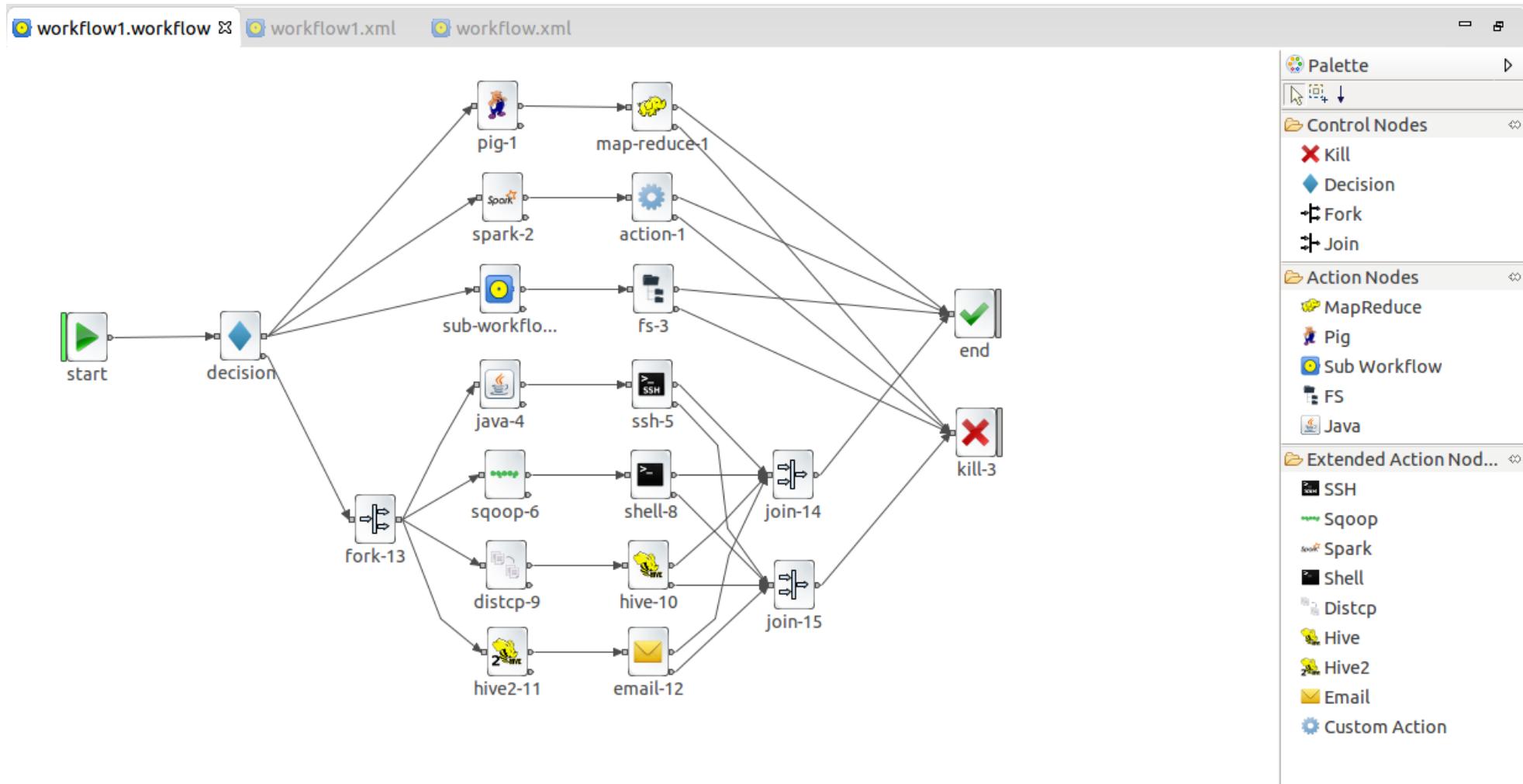
The screenshot shows the Hue Oozie Editor interface. At the top, there is a navigation bar with tabs for "Oozie Editor" (selected), "Workflows", "Coordinators", and "Bundles". On the right side of the bar are buttons for "Unsaved" (with a pen icon), settings, and file operations. Below the navigation bar is a toolbar labeled "ACTIONS" with various icons: a target node, a piggy bank, a star, a folder, a message bubble, a file, a greater than sign, and a double arrow. The main workspace is titled "My Workflow" and contains a placeholder text "Add a description...". Three workflow components are visible:

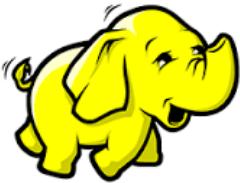
- A top-level node represented by a circle icon.
- An action node represented by a flag icon.
- A coordinator node represented by a square icon with a gear icon in the bottom right corner.

Below each node is a dashed-line box with the text "Drop your action here".



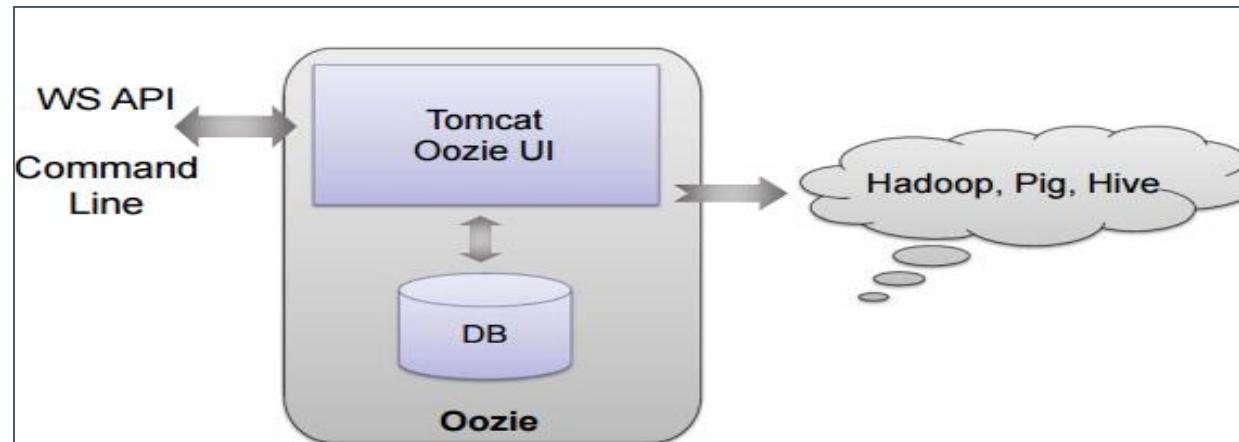
# Oozie Editors : OEP

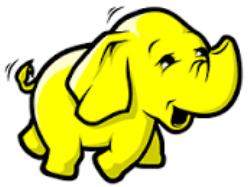




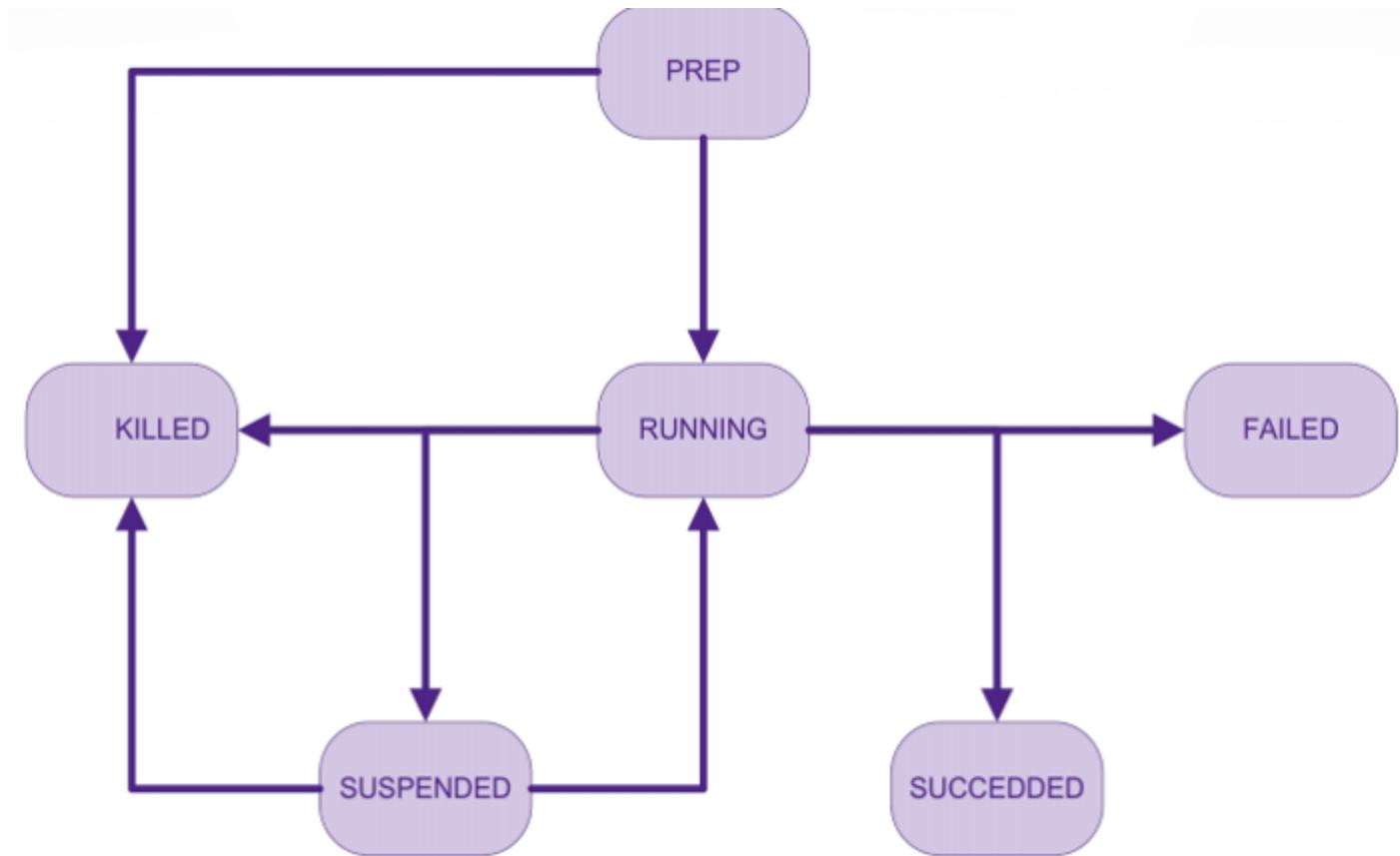
# Fonctionnement de Oozie

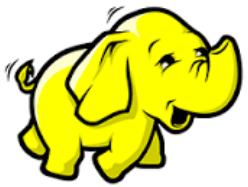
- Client Oozie :
  - Lancer les traitements
  - Communiquer avec le serveur
- Serveur Oozie :
  - Contrôle les jobs
  - Exécute les jobs



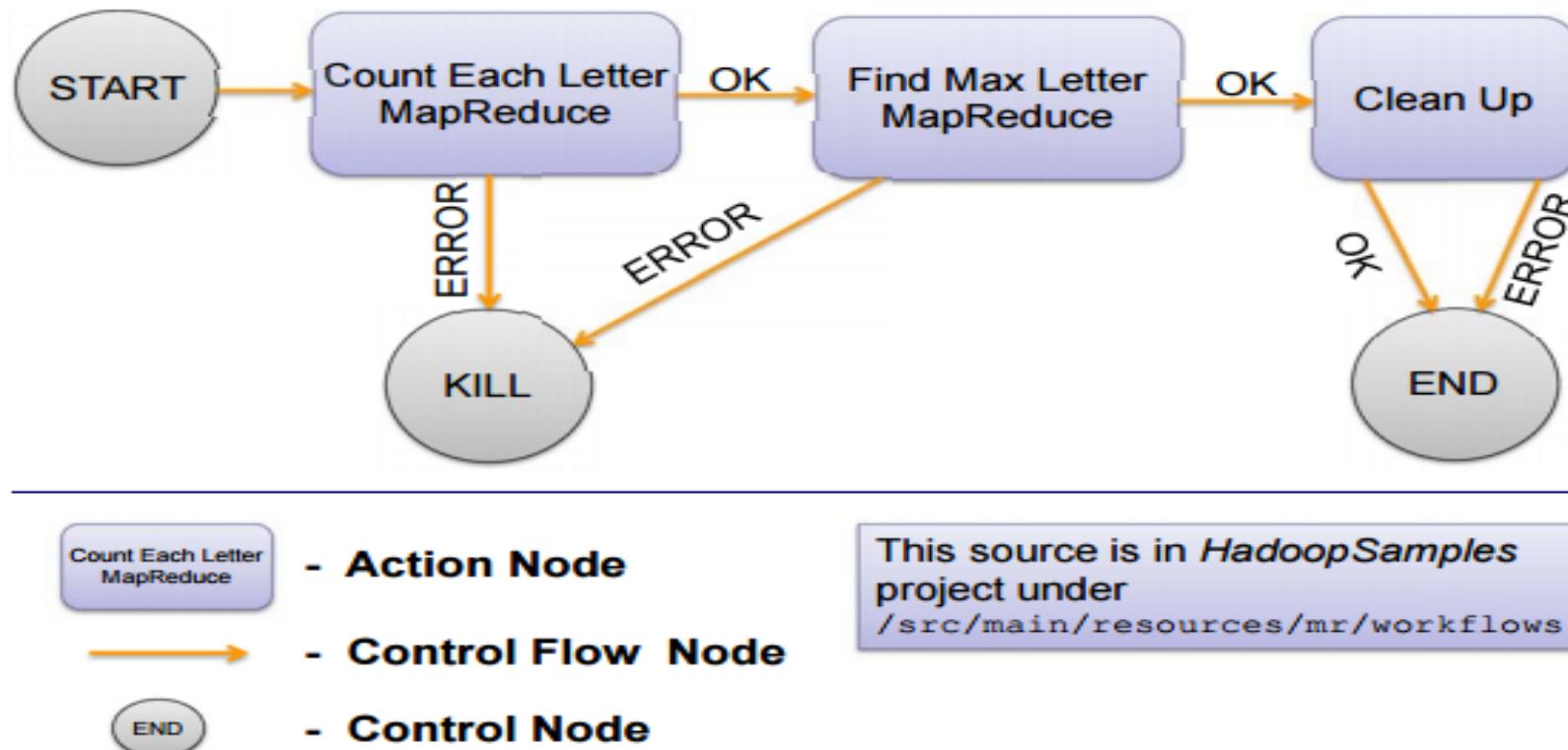


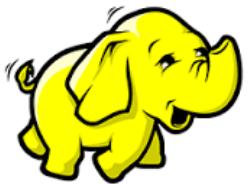
# Fonctionnement de Oozie



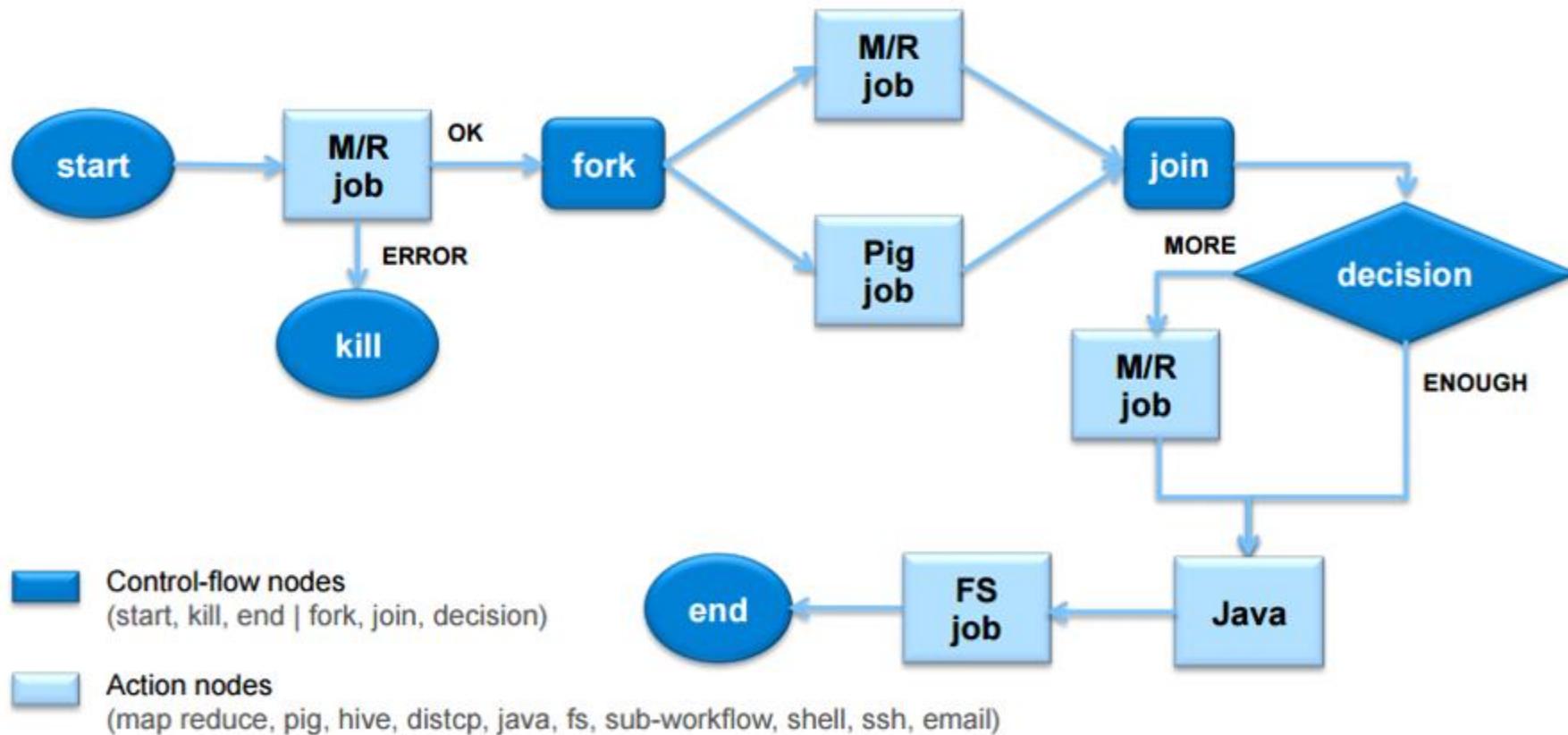


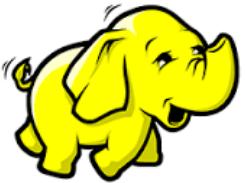
# Fonctionnement de Oozie





# Fonctionnement de Oozie





# Fonctionnement de Oozie

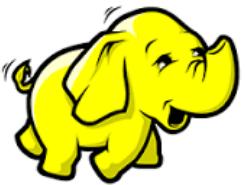
```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="ooziedemo-wf">
<start to="Create_External_Table" />
  <action name="Create_External_Table">
    </action>
</workflow-app>
```

Start contrôle nœud :

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="ooziedemo-wf">
<end name="end" />
</workflow-app>
```

End contrôle nœud :

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="ooziedemo-wf">
<kill name="fail">
  <message>Sqoop failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
</workflow-app>
```



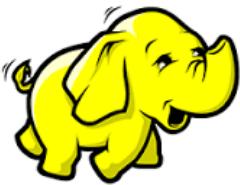
# Fonctionnement de Oozie

Decision control node :

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="ooziedemo-wf">
<start to="external_table_exists" />
<decision name="external_table_exists">
  <switch>
    </switch>
  </decision>
</workflow-app>
```

Fork control node:

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="ooziedemo-wf">
<start to="fork_node" />
<fork name="fork_node">
  <path start="Create_External_Table"/>
  <path start="Create_orc_Table"/>
</fork>
</workflow-app>
```

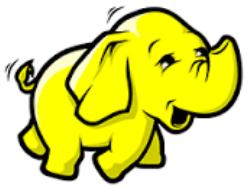


# Action Oozie



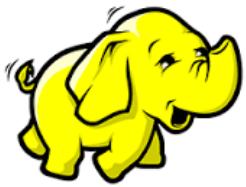
**HDFS Commands:  
Hadoop Shell Commands to Manage HDFS**

```
<action name="timeCheck">
  <shell xmlns="uri:oozie:shell-action:0.1">
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>${queueName}</value>
      </property>
    </configuration>
    <exec>${EXEC}</exec>
    <argument>${tableName}</argument>
    <file>${EXEC}#${EXEC}</file>
    <capture-output/>
  </shell>
  <ok to="sqoopIncrImport"/>
  <error to="fail"/>
</action>
```



# Action Oozie

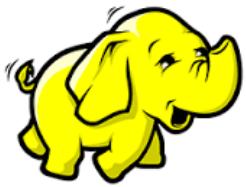
```
<action name="sqoopIncrImport">
  <sqoop xmlns="uri:oozie:sqoop-action:0.2">
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
      <delete path="${s3BucketLoc}/${tableName}/incr"/>
      <mkdir path="${s3BucketLoc}/${tableName}"/>
    </prepare>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>${queueName}</value>
      </property>
    </configuration>
    <arg>import</arg>
    <arg>--connect</arg>
    .....
    <arg>${wf:actionData('timeCheck')}['sqoopLstUpd']</arg>
    <arg>--m</arg>
    <arg>1</arg>
  </sqoop>
  <ok to="sqoopMetaUpdate"/>
  <error to="fail"/>
</action>
```



# Action Oozie



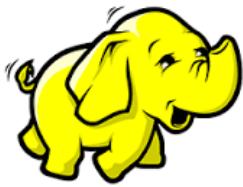
```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  <action name="[NODE-NAME]">
    <java>
      <job-tracker>[JOB-TRACKER]</job-tracker>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]"/>
        <mkdir path="[PATH]"/>
      </prepare>
      <job-xml>[JOB-XML]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
      </configuration>
      <main-class>[MAIN-CLASS]</main-class>
      <java-opts>[JAVA-STARTUP-OPTS]</java-opts>
      <arg>ARGUMENT</arg>
      <file>[FILE-PATH]</file>
      <archive>[FILE-PATH]</archive>
      <capture-output />
    </java>
```



## Action Oozie

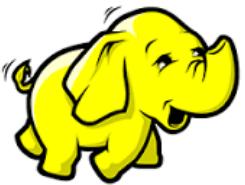
```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.2">
  <action name="[NODE-NAME]">
    <pig>
      <job-tracker>[JOB-TRACKER]</job-tracker>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]"/>
        <mkdir path="[PATH]"/>
      </prepare>

      <script>[PIG-SCRIPT]</script>
      <param>[PARAM-VALUE]</param>
        <param>[PARAM-VALUE]</param>
      <argument>[ARGUMENT-VALUE]</argument>
        <argument>[ARGUMENT-VALUE]</argument>
      <file>[FILE-PATH]</file>
        <archive>[FILE-PATH]</archive>
    </pig>
```



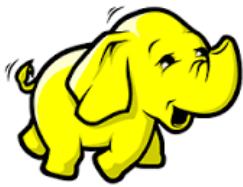
# Action Oozie

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  <action name="[NODE-NAME]">
    <hive xmlns="uri:oozie:hive-action:0.2">
      <job-tracker>[JOB-TRACKER]</job-tracker>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]"/>
        <mkdir path="[PATH]"/>
      </prepare>
      <job-xml>[HIVE SETTINGS FILE]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
      </configuration>
      <script>[HIVE-SCRIPT]</script>
      <param>[PARAM-VALUE]</param>
      <param>[PARAM-VALUE]</param>
      <file>[FILE-PATH]</file>
      <archive>[FILE-PATH]</archive>
    </hive>
```



## Action Oozie : mail

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">  
    ...  
    <action name="an-email">  
        <email xmlns="uri:oozie:email-action:0.1">  
            <to>julie@xyz.com,max@abc.com</to>  
            <cc>jax@xyz.com</cc>  
            <subject>Email notifications for ${wf:id()}</subject>  
            <body>The wf ${wf:id()} successfully completed.</body>  
        </email>  
        <ok to="main_job"/>  
        <error to="kill_job"/>  
    </action>  
    ...  
</workflow-app>
```

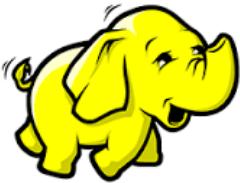


# Oozie Job

- Job.properties
- Workflow.xml
- Coordinateur.xml

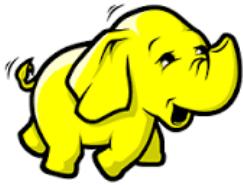
❖ Lancement :

```
oozie job -oozie http://quickstart.cloudera:11000/oozie - config job.properties -run
```



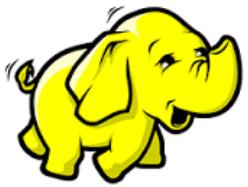
# Oozie Workflow

- Workflow dans Oozie est une séquence d'actions disposées dans une dépendance de contrôle DAG (Direct Acyclic Graph)
- Les actions sont en dépendance contrôlée car l'action suivante ne peut s'exécuter que selon la sortie de l'action en cours.
- Les actions suivantes dépendent de son action précédente
- Une action de workflow peut être une action Hive, une action Pig, une action Java, une action Shell, etc.
- Il peut y avoir des arbres de décision pour décider comment et dans quelle condition un travail doit s'exécuter.



# Oozie Workflow

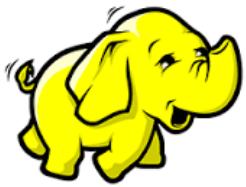
- Fork est utilisée pour exécuter plusieurs jobs en parallèle
- Workflow peut être paramétrés (des variables telles que \${nameNode} peuvent être transmises dans la définition de workflow.
- Ces paramètres proviennent d'un fichier de configuration appelé property file
- Chaque type d'action peut avoir son propre type de tags
- La balise Param définit les valeurs que nous transmettrons au script



# Oozie Workflow

```
Create external table external_table
(
  name string,
  age int,
  address string,
  zip int
)
row format delimited
fields terminated by ','
stored as textfile
location '/test/abc';
```

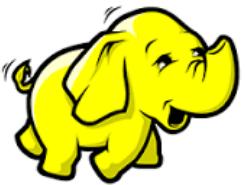
external.hive



# Oozie Workflow

```
use ${database_name}; -- input from Oozie
insert into table orc_table
select
concat(first_name, ' ', last_name) as name,
yearofbirth,
year(from_unixtime) --yearofbirth as age,
address,
zip
from external_table
;
```

Copydata.hql

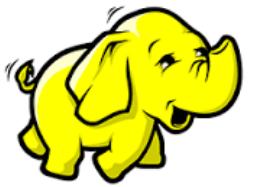


# Oozie Workflow

```
<!-- This is a comment -->

<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">
    <start to="Create_External_Table" />

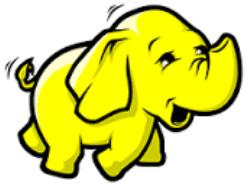
    <!--Step 1 -->
    <action name="Create_External_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>xyz.com:8088</job-tracker>
            <name-node>hdfs://rootname</name-node>
            <script>hdfs_path_of_script/external.hive</script>
        </hive>
        <ok to="Create_orc_Table" />
        <error to="kill_job" />
    </action>
```



# Oozie Workflow

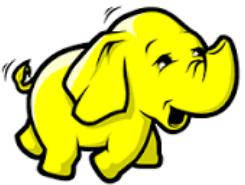
```
<!--Step 2 -->

<action name="Insert_into_Table">
    <hive xmlns="uri:oozie:hive-action:0.4">
        <job-tracker>xyz.com:8088</job-tracker>
        <name-node>hdfs://rootname</name-node>
        <script>hdfs_path_of_script/Copydata.hive</script>
        <param>database_name</param>
    </hive>
    <ok to="end" />
    <error to="kill_job" />
</action>
<kill name="kill_job">
    <message>Job failed</message>
</kill>
<end name="end" />
</workflow-app>
```



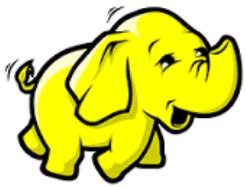
# Oozie Workflow

```
oozie job --oozie http://host_name:8080/oozie -D  
oozie.wf.application.path=hdfs://namenodepath/pathof_workflow_xml/workflow.xml  
-run|
```



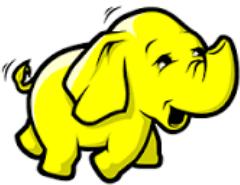
## Oozie Workflow :Fork

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">  
    <start to="fork_node" />  
    <fork name="fork_node">  
        <path start="Create_External_Table"/>  
        <path start="Create_orc_Table"/>  
    </fork>  
    <action name="Create_External_Table">  
        --  
    </action>  
    <action name="Create_orc_Table">  
        --  
    </action>  
    <end name="end" />  
</workflow-app>
```



# Oozie Workflow :Decision

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">
    <start to="external_table_exists" />
    <decision name="external_table_exists">
        <switch>
            <case to="Create_External_Table">${fs:exists('/test/abc')} eq
'false'</case>
            <default to="orc_table_exists" />
        </switch>
    </decision>
    <action name="Create_External_Table">
    </action>
    .
    <action name="Create_orc_Table">
    </action>
    <end name="end" />
</workflow-app>
```



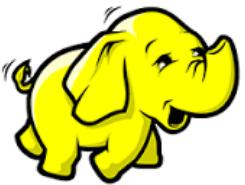
# Oozie Property File

## File name -- job1.properties

```
# properties  
nameNode=hdfs://rootname  
jobTracker=xyz.com:8088  
script_name_external=hdfs_path_of_script/external.hive  
script_name_orc=hdfs_path_of_script/orc.hive  
script_name_copy=hdfs_path_of_script/Copydata.hive  
database=database_name
```

Nous devrons passer le -config lors de l'exécution du workflow.

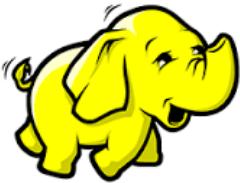
```
oozie job --oozie http://host_name:8080/oozie --config  
edgenode_path/job1.properties -D  
oozie.wf.application.path=hdfs://Namenodepath/pathof_workflow_xml/workflow.xml  
-run
```



# Oozie Property File

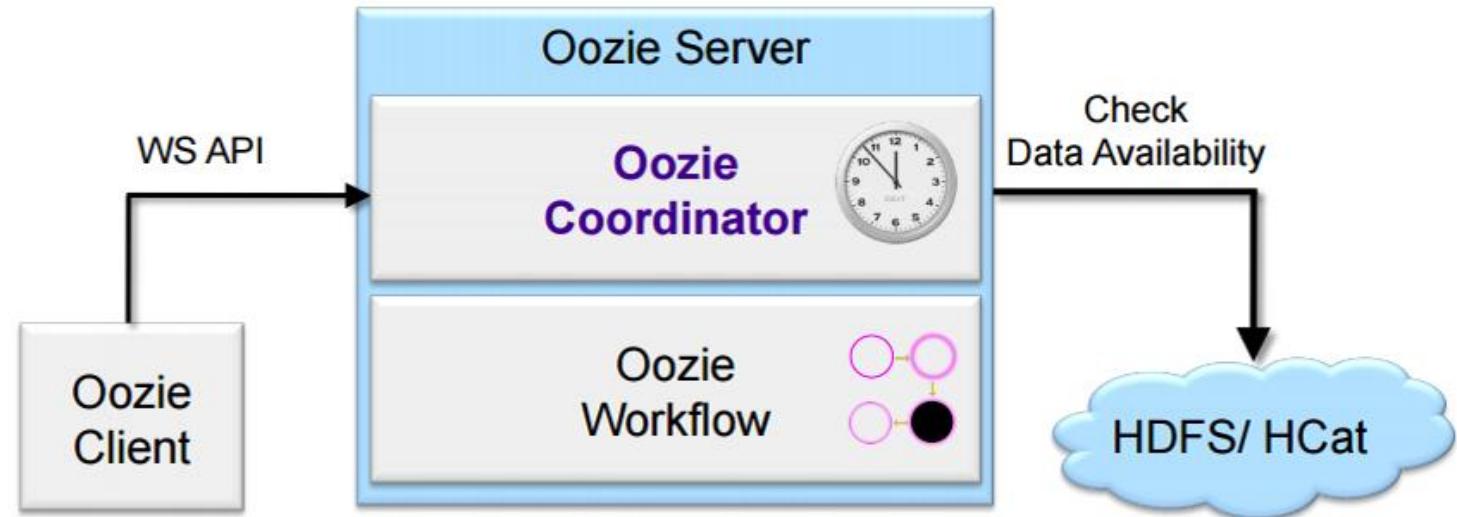
```
<!-- This is a comment -->

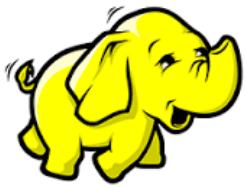
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-Workflow">
    <start to="Create_External_Table" />
    <action name="Create_External_Table">
        <hive xmlns="uri:oozie:hive-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>${script_name_external}</script>
        </hive>
        <ok to="Create_orc_Table" />
        <error to="kill_job" />
    </action>
</workflow-app>
```



# Oozie Coordinator

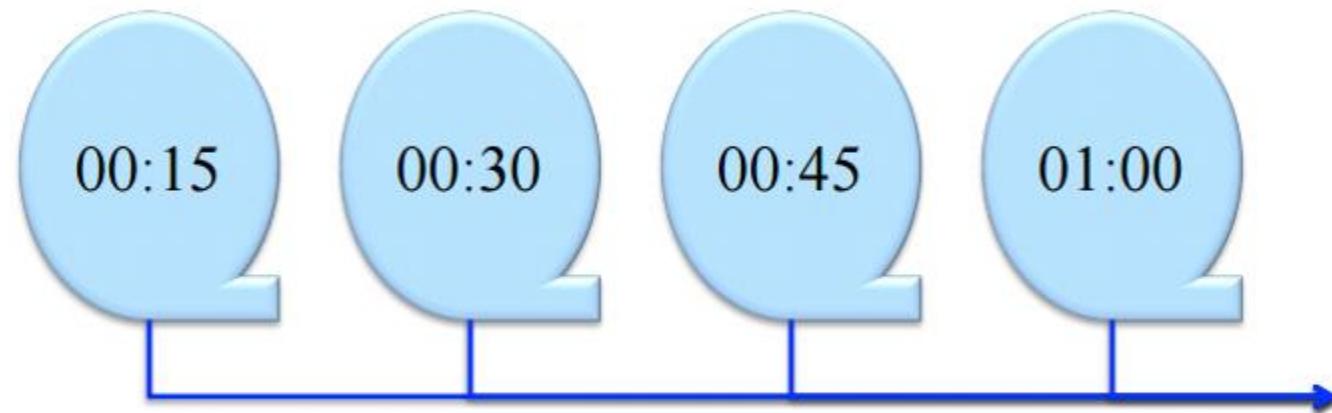
- Oozie Coordinator modélise les déclencheurs d'exécution du workflow sous la forme de prédictats de temps, de données ou d'événements.
- Le travail de workflow mentionné dans le coordinateur est démarré uniquement après que les conditions données ont été satisfaites
- Oozie exécute un workflow basé su
  - Dépendance temporelle (fréquence)
  - Dépendance des données

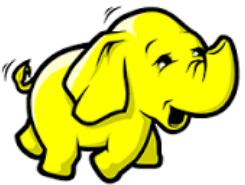




## Oozie Coordinator

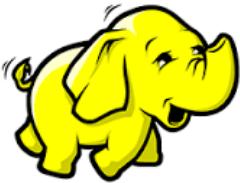
- Exemple de lancement chaque 15 minutes





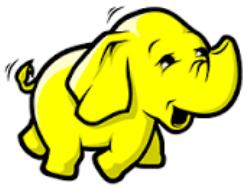
# Oozie Coordinator

```
<coordinator-app xmlns="uri:oozie:coordinator:0.2"
  name="coord_copydata_from_external_orc" frequency="5 * * * *" start="2016-00-
  18T01:00Z" end="2025-12-31T00:00Z" timezone="America/Los_Angeles">
  <controls>
    <timeout>1</timeout>
      <concurrency>1</concurrency>
      <execution>FIFO</execution>
      <throttle>1</throttle>
    </controls>
    <action>
      <workflow>
        <app-path>pathof_workflow_xml/workflow.xml</app-path>
      </workflow>
    </action>
</coordinator-app>
```



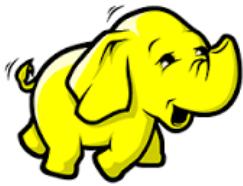
# Oozie Coordinator

- Start : la date et l'heure de début du job
- End : la date et l'heure de fin du travail
- Timezone : le timezone.
- Frequency : fréquence d'exécution du job en minutes.
- Concurrency : Le nombre maximal d'actions pour ce travail pouvant être exécuté simultanément. Cette valeur permet de matérialiser et de soumettre plusieurs instances de l'application de coordination, et permet aux opérations de rattraper le retard de traitement. Valeur par défaut est 1.
- Execution : Spécifie l'ordre d'exécution si plusieurs instances du job de coordinateur ont satisfait leurs critères d'exécution : FIFO LIFO LAST\_ONLY
- Throttle: combien d'actions de coordinateur maximum sont autorisées à être dans l'état WAITING simultanément.



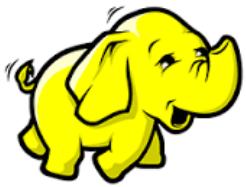
## Oozie Coordinator

- Timeout : La durée maximale, en minutes, qu'une action matérialisée attendra que les conditions supplémentaires soient satisfaites avant d'être rejetée.
- Un timeout de 0 indique qu'au moment de la matérialisation toutes les autres conditions doivent être satisfaites, sinon l'action sera rejetée.
- Un timeout de -1 indique aucun timeout, l'action matérialisée attendra toujours pour que les autres conditions soient satisfaites. La valeur par défaut est -1.



# Oozie Coordinator

- Executer coordinator :
- oozie job --oozie http://host\_name:8080/oozie --config job.properties -D oozie.wf.application.path=hdfs://Namenodepath/pathof\_coordinator\_xml/coordinator.xml -d "2 minute" -run
- “2 minute” : le coordinator commence seulement après 2 minutes du moment où le job a été soumis.



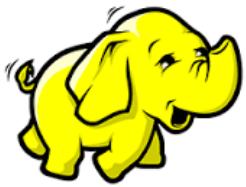
# Monitoring

- Pour vérifier l'état du traitement, vous pouvez accéder à la console Web Oozie - [http://nom\\_hôte:  
8080 /](http://nom_hôte:8080/)

The screenshot shows the Oozie Web Console interface in Mozilla Firefox. The title bar reads "Oozie Web Console - Mozilla Firefox". The address bar shows the URL "http://localhost:8080/oozie". The main content area displays a table of workflow jobs. The table has columns: Job Id, Name, Status, Run, User, Group, Created, Started, Last Modified, and Ended. There is one row visible in the table:

Job Id	Name	Status	Run	User	Group	Created	Started	Last Modified	Ended
1 000000-160118021306335-oozie-oozi-W	simple-Workflow	RUNNING	0	root		Mon, 18 Jan 2016 02:57:51 GMT	Mon, 18 Jan 2016 02:57:51 GMT	Mon, 18 Jan 2016 02:57:55 GMT	

The navigation menu at the top includes "Workflow Jobs", "Coordinator Jobs", "Bundle Jobs", "System Info", "Instrumentation", and "Settings". The status bar at the bottom right indicates "Server version [4.1.0.22.0.0-2041]".



# Monitoring

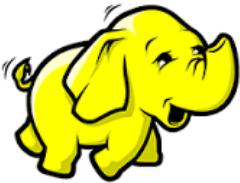
- Via l'interface HUE

The screenshot shows the HUE Oozie Dashboard with the 'Coordinators' tab selected. The interface displays two sections: 'Running' and 'Completed'.  
**Running Workflows:**

Completion	Status	Name	Duration	Submitter	Frequency	Start Time	Id
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000044-141003113244314-oozie-oozi-C
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000042-141003113244314-oozie-oozi-C
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000040-141003113244314-oozie-oozi-C
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000038-141003113244314-oozie-oozi-C
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000036-141003113244314-oozie-oozi-C
Sun, 02 Jun 2013 17:00:00	RUNNING	DailySleep	0d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000034-141003113244314-oozie-oozi-C

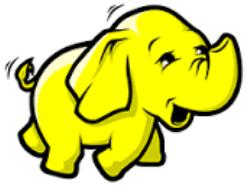
  
**Completed Workflows:**

Completion	Status	Name	Duration	Submitter	Frequency	Start Time	Id
Tue, 04 Jun 2013 18:00:00	KILLED	DailyAnalytics-2	2d:0h:0m:0s	romain	Every day at midnight	Sun, 02 Jun 2013 18:00:00	0000007-140804182536588-oozie-oozi-C
Tue, 04 Jun 2013 18:00:00	KILLED	DailySleep-1	4d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 18:00:00	0000006-140804182536588-oozie-oozi-C
Tue, 04 Jun 2013 17:00:00	KILLED	DailySleep	4d:0h:0m:0s	romain	Every day at midnight	Fri, 31 May 2013 17:00:00	0000023-141003113244314-oozie-oozi-C



# CLI

- oozie version : show client version
- -D <property=value> : set/override value for given property
- -info <arg> : info of a job
- -kill <arg> : kill a job
- -localtime : use local time (default GMT)
- -log <arg> : job log
- -run : run a job
- -start <arg> : start a job
- oozie validate myApp/workflow.xml : Validation d'un workflow XML



# Plan

**FIN ... QUESTIONS ... MERCI**