Université Mohammed V- Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique
Filière Génie Informatique et Digitalisation

Fondements du Big Data

Pr. N. EL FADDOULI
nfaddouli@gmail.com

2023-2024

L'ÉCO-SYSTÈME HADOOP

- ❑ INTRODUCTION
- ❑ ARCHITECTURE
- ❑ HDFS
- ❑ MAPRREDUCE/YARN
- ❑ PIG
- ❑ HIVE
- ❑ HBASE
- ❑ SQOOP

Introduction

- ❑ La vitesse de lecture et d'écriture (liée à la *Latence du disque*) ne s'est pas beaucoup amélioré au cours des dernières années (*disque récents* jusqu'à 760 MB/s (selon l'interface SATA/IDE ou SCSI – SSD peut atteindre 1000MB/s))
- ❑ **Problème:** Combien de temps faut-il pour lire 1 To de données?

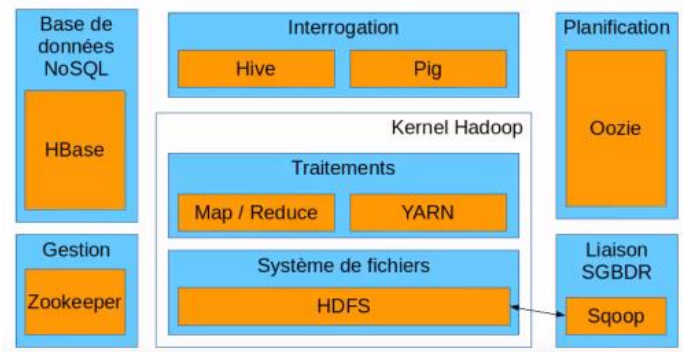
  1 To (à 80 Mb / sec): 1 disque - 3,4 heures
  10 disques - 20 min
  100 disques - 2 min
  1000 disques - 12 sec
- ❑ **Solution:** Le traitement parallèle des données
  - **Calcul distribué**: plusieurs ordinateurs apparaissent comme un super ordinateur, communiquent les uns avec les autres par le passage de messages, opèrent ensemble pour atteindre un but commun.
- ❑ **Défis:** Hétérogénéité (matériel-OS) - Ouverture - Évolutivité - Sécurité - Concurrence - Tolérance aux pannes - Transparence

Hadoop - Historique

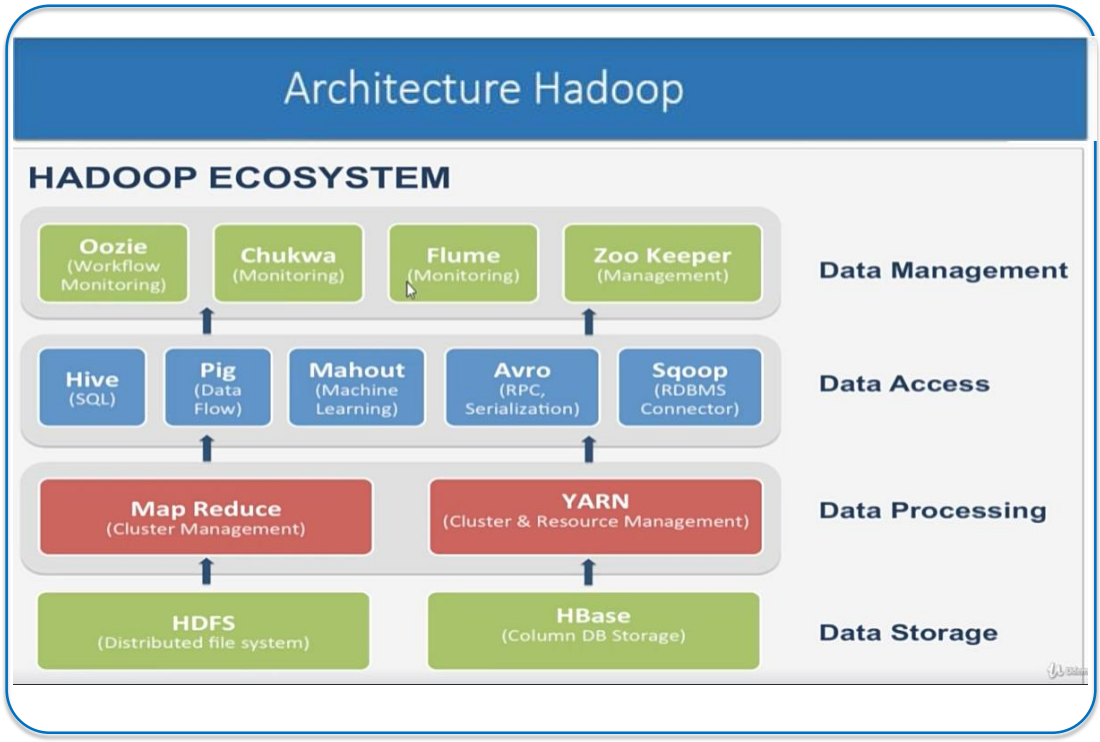
- **2003** Google publie un document sur son "Google File Système" un système de fichiers distribués pour gérer des larges volumes de données distribuées.
- **2004**: Mike Cafarella et Doug Cutting (*créateur de Lucene*) ont commencé à écrire une implémentation open source, le moteur de recherche distribué **Nutch (**son système de fichiers distribué **NDFS**) pour pouvoir indexer plus de 1 Billion pages web.
- **2004**: Google a introduit *MapReduce* pour le traitement distribué.
- **2005**: Implémentation de *MapReduce* dans **Nutch**, tous les principaux algorithmes de **Nutch** fonctionnaient avec MapReduce sur des fichiers gérés par NDFS.
- **2006**: Projet **Hadoop** successeur de **Nutch** indépendant de Lucene.
- **2006**: Doug Cutting a rejoint Yahoo! qui a beaucoup investi pour transformer **Hadoop** en un système fonctionnant à l'échelle du Web.
- **May 2006**: Yahoo! déploie un cluster **Hadoop** de 300 nœuds pour son index de recherche.
- **2008**: Yahoo! a annoncé que son index de recherche était généré par un cluster **Hadoop** à 900 nœuds
- **Mai 2009**: Yahoo! a utilisé son cluster Hadoop pour traiter 1 To en 62 secondes par 1460 nœuds.
- **Juin 2009**: Yahoo rend le code source d'Hadoop public.
- **Septembre 2009**: Doug Cutting rejoint Cloudera.

L'écosystème Hadoop

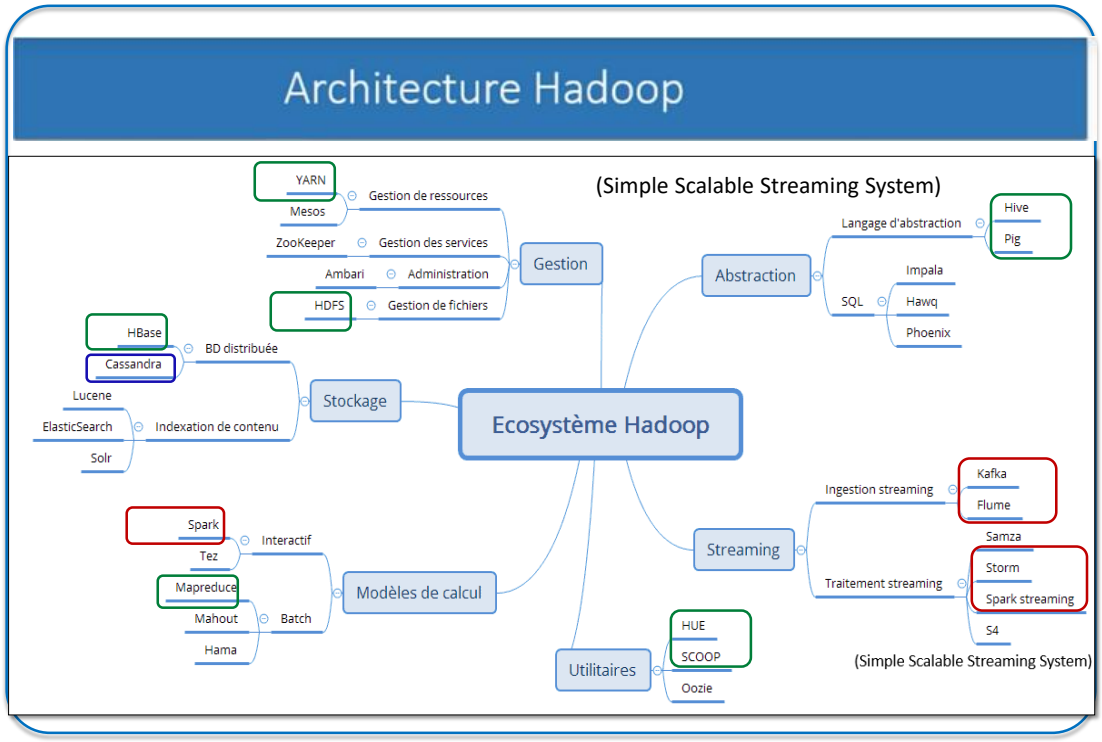
- Hadoop est basé sur:
  - **MapReduce/YARN**: framework de **traitement** distribué
  - **HDFS** : pour un **stockage** distribué
- Hadoop est écrit en **Java** et sous License Apache
- Plusieurs projets sont liés à Hadoop ⇨ Ecosystème Hadoop



Simon Gilliot : Introduction à la plate-forme Hadoop et à son écosystème


Architecture Hadoop
HADOOP ECOSYSTEM
Oozie (Workflow Monitoring)
Chukwa (Monitoring)
Flume (Monitoring)
Zoo Keeper (Management)
Data Management
Hive (SQL)
Pig (Data Flow)
Mahout (Machine Learning)
Avro (RPC, Serialization)
Sqoop (RDBMS Connector)
Data Access
Map Reduce (Cluster Management)
YARN (Cluster & Resource Management)
Data Processing
HDFS (Distributed file system)
HBase (Column DB Storage)
Data Storage

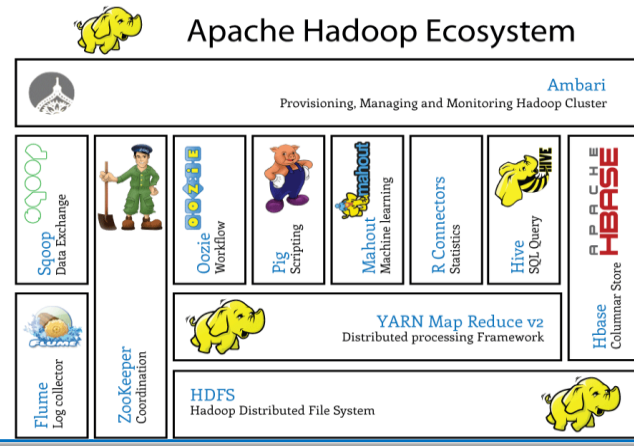
7


Architecture Hadoop
(Simple Scalable Streaming System)
Ecosystème Hadoop
Gestion
YARN
Mesos
Gestion de ressources
ZooKeeper
Gestion des services
Ambari
Administration
HDFS
Gestion de fichiers
Stockage
HBase
Cassandra
BD distribuée
Lucene
ElasticSearch
Solr
Indexation de contenu
Modèles de calcul
Spark
Tez
Interactif
Mapreduce
Mahout
Hama
Batch
Abstraction
Langage d'abstraction
Hive
Pig
SQL
Impala
Hawq
Phoenix
Streaming
Ingestion streaming
Kafka
Flume
Traitement streaming
Samza
Storm
Spark streaming
S4
(Simple Scalable Streaming System)
Utilitaires
HUE
SCOOP
Oozie

8

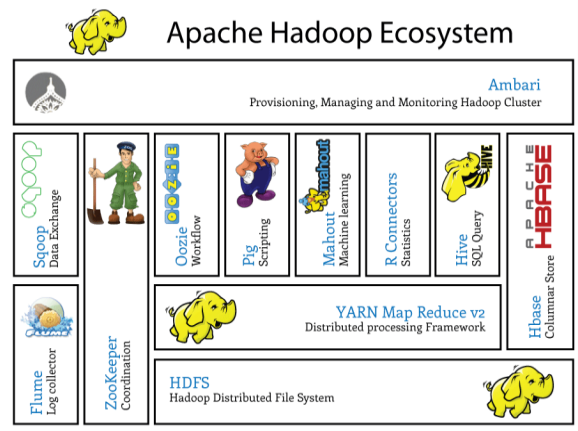
L'écosystème Hadoop

- Composants de base: **HDFS – MapReduce /Yarn**
- D'autres outils existant pour:
  - La simplification des opérations de traitement sur les données
  - La gestion et coordination de la plateforme
  - Le monitoring du cluster



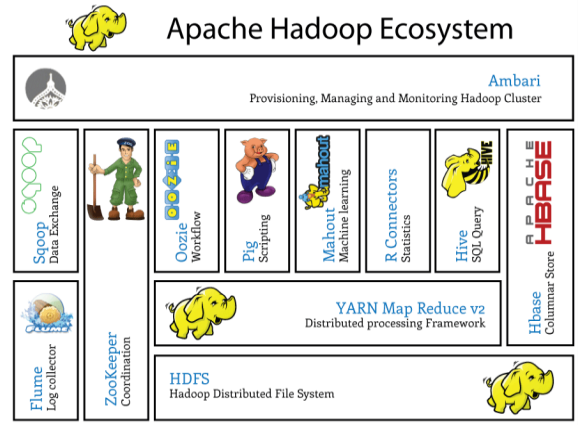
L'écosystème Hadoop

- **Pig**: Plateforme haut niveau pour le traitement de données, basée sur un langage de script Pig Latin
- **Hive**: Environnement de haut niveau pour le traitement de données, basé sur un langage proche de SQL (Hive QL)
- **R Connectors**: permet l'accès à HDFS et l'exécution de requêtes Map/Reduce à partir du langage R
- **Mahout**: bibliothèque de machine learning et mathématiques
- **Oozie**: permet d'ordonnancer les jobs Map Reduce (Java, Python, Pig, Hive, …), en définissant des workflows

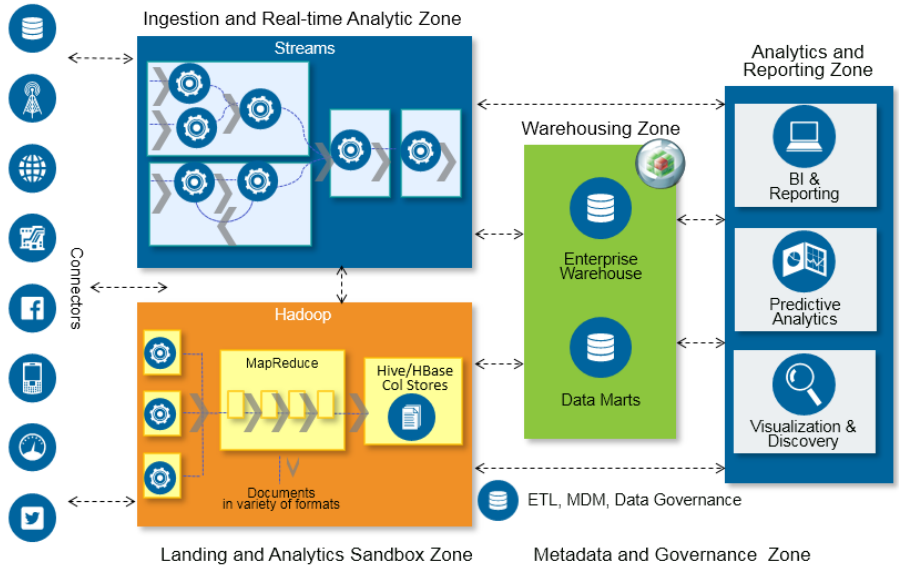


L'écosystème Hadoop

- **Hbase** : Base de données NoSQL orientée colonnes.
- **Sqoop**: Lecture et écriture des données à partir de BD externes
- **Flume**: Collecte de logs et stockage dans HDFS
- **Ambari**: outil pour la gestion et monitoring des clusters
- **Zookeeper**: fournit un service centralisé pour maintenir les information de configuration, de nommage et de synchronisation distribuée



Hadoop dans les entreprises



Course Guide Volume 1: IBM BigInsights Foundation v4.0 (Course code DW613 ERC 1.0)

HDFS

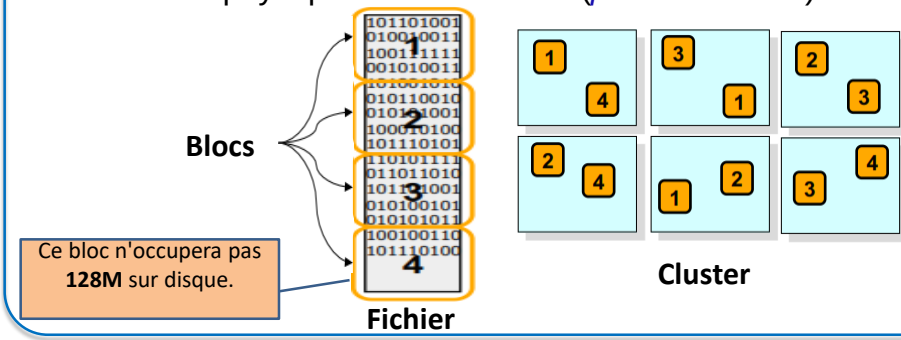
Hadoop: HDFS

HDFS : Système de fichiers conçu pour stocker des fichiers très volumineux dans un cluster d'ordinateurs pas chers.

- **HDFS** stocke les données sur plusieurs nœuds qui forment un **cluster**
- Il existe des clusters **Hadoop** qui stockent des **Eo** de données.
- Ecrire une fois/lire plusieurs fois (***WORM***: *write once read many times*): Les données sont généralement générées ou copiées à partir de la source. Des analyses (batch) sont ensuite effectuées sur cet ensemble de données au fil du temps. Chaque analyse impliquera une grande proportion, sinon la totalité, de l'ensemble de données.
- **HDFS réplique** les données sur plusieurs nœuds afin d'éviter leur perte en cas de panne de certains nœuds ⇨ **Fiabilité**, **Disponibilité**
- Chaque **fichier** est décomposé en plusieurs **blocs** qui seront **répliqués** sur les nœuds.

Hadoop: HDFS

- **Bloc**: taille par défaut 128Mo sur Hadoop 2.x (64Mo sur Hadoop 1.x) (bloc OS = 4k Octets par exemple)
- La taille est grande pour minimiser le temps de recherche (localisation) des blocs.
- Possibilité de configurer cette taille lors de la création d'un fichier
  ➔ Cette fonctionnalité est appelée "Variable Block Size" (VBS)
- Chaque bloc est répliqué sur un petit nombre (***facteur de réplication***) de machines physiquement séparées. (*par défaut trois*)



15

Hadoop: Architecture de HDFS - maitre/esclave

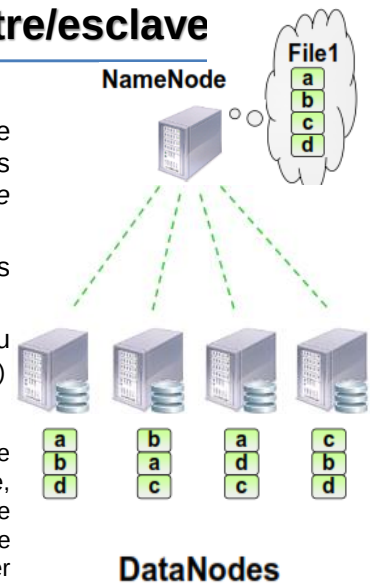
- **Maitre: NameNode**
  - Gère **l'espace de noms du système de fichiers**: Il gère l'arborescence du système de fichiers et les métadonnées de tous les fichiers (*noms, facteurs de réplication, propriétés*) et répertoires de l'arborescence.

Ces informations sont stockées sur le disque local dans deux fichiers:

- **FsImage** (*métadonnées*) chargé en mémoire au démarrage du **NameNode** (Chemin dans: **hdfs-site.xml**)

  Dossier: **hdfs getconf -confKey** dfs.namenode.name.dir
- **EditLog** : pour enregistrer chaque modification apportée aux métadonnées du système de fichiers. Par exemple, création d'un nouveau fichier dans HDFS, le NameNode insère un enregistrement dans le **EditLog** indiquant cela. De même, la modification du facteur de réplication d'un fichier entraîne l'insertion d'un nouvel enregistrement dans **EditLog**

- Régule l'accès aux fichiers par les clients (équilibrer la charge)



Réf: IBM BigInsights Foundation v4.0

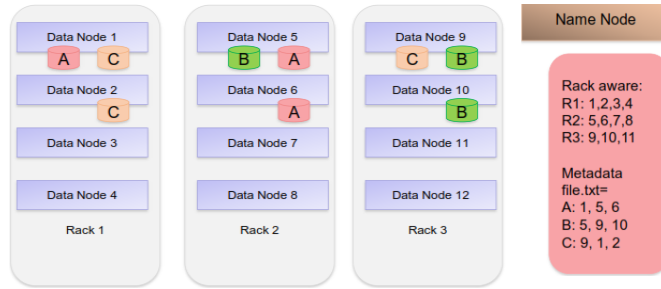
(Course code DW613 ERC 1.0)

16

Hadoop: Architecture de HDFS

- Le **NameNode** conserve les **métadonnées** en **mémoire** pour assurer un accès rapide ce qui nécessite une grande taille mémoire pour son fonctionnement.
- Les blocs de fichiers sont répliqués sur plusieurs nœuds (***DataNodes***).
- Le nombre de copies de chaque bloc est contrôlé par le paramètre de réplication (Par défaut 3)
- Réplication sur plusieurs racks:

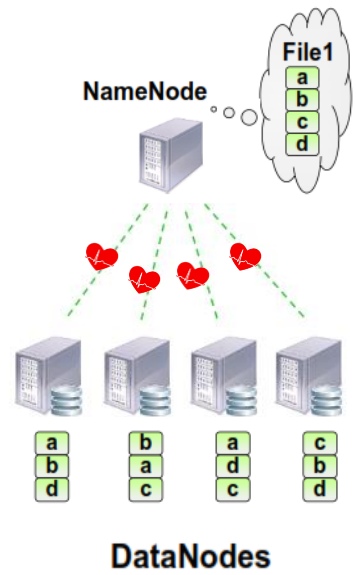
Une copie sur un premier rack - La 2ème et la 3ème copies peuvent être ensemble sur un autre Rack.



17

Hadoop: Architecture de HDFS

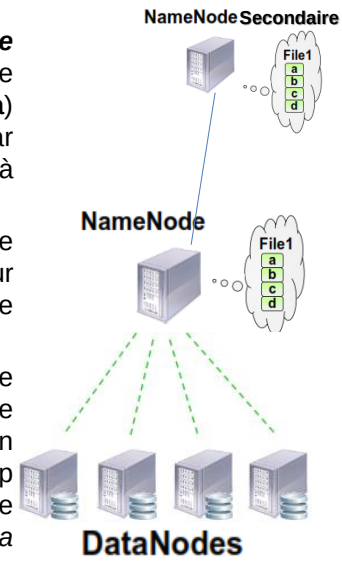
- **Esclaves: DataNode**
  - Nombreux DataNodes par cluster (***un par nœud***)
  - Gère le stockage (read/write) attaché aux nœuds
  - Envoie périodiquement un signal au **NameNode** sous forme de **heartbeat** (id, espace stockage, espace utilisé, …) avec un intervalle indiqué dans **hdfs-site.xml** (**dfs.heartbeat.interval** , par défaut 3s).
  - Renvoie périodiquement (**dfs.blockreport.intervalMsec** par défaut 21600000 Ms=6h ou 3600000Ms=1h) au NameNode un **rapport** sur les blocs qu'il stocke (id, taille, date de création, …)
  - Si un **DataNode** tombe en panne, ses blocs seront répliqués sur d'autres **DataNodes** (**timout** = 2 * heartbeat.recheck.interval *(5min par défaut)* + 10 * heartbeat.interval) = 630s par défaut
  - Effectue également la **création**, la **suppression** et la **réplication** de blocs sur ordre du **NameNode**.
  - Les données sont stockées sur plusieurs nœuds



18

Hadoop: Architecture de HDFS

- **Secondary NameNode**
  - Le **NameNode** est un point de défaillance unique (***Single Point of Failure - SPOF***): Si la machine exécutant le **NameNode** est perdue, toutes les informations (**meta-data**) sur les fichiers de données seraient perdues. Par conséquent, on ne pourra pas reconstruire les fichiers à partir des blocs sur les **DataNodes**.
  - Hadoop sauvegarde les métadonnées du système de fichiers (**FsImage**): **NameNode** écrit son état persistant sur plusieurs systèmes de fichiers (disque local, disque distant).
  - Utiliser un **NameNode secondaire**: n'a pas le même rôle que le **NameNode**. Il **fusionne périodiquement** l'image de l'espaces des noms (**FsImage**) et le journal d'édition (**EditLog**) pour éviter que ce dernier devienne trop volumineux. Il s'exécute sur une machine physique distincte (*nécessite beaucoup de CPU et de RAM pour effectuer la fusion*)



19

Hadoop: Architecture de HDFS

- **Secondary NameNode** *(suite)*
- La **réplication** des métadonnées (FsImage) du **NameNode** et l'utilisation du **NameNode secondaire** permet d'éviter la perte de métadonnées et de conserver des copies récentes.
- Mais elle n'offre pas une **haute disponibilité** du système de fichiers:
  - En cas d'échec du **NameNode**, tous les clients, y compris les jobs MapReduce, ne pourraient pas lire, écrire ou répertorier les fichiers, car le **NameNode** est le seul référentiel des métadonnées et du mapping fichier-bloc.
  - Dans un tel cas, l'ensemble du système Hadoop serait **hors service** jusqu'à ce qu'un nouveau **NameNode** puisse être mis en ligne.
- Pour récupérer un **NameNode défaillant**, l'administrateur démarre un **nouveau NameNode** avec l'une des copies de métadonnées du système de fichiers et configure les **DataNodes** et les applications clientes (du Data Center) pour utiliser ce nouveau **NameNode**.

20

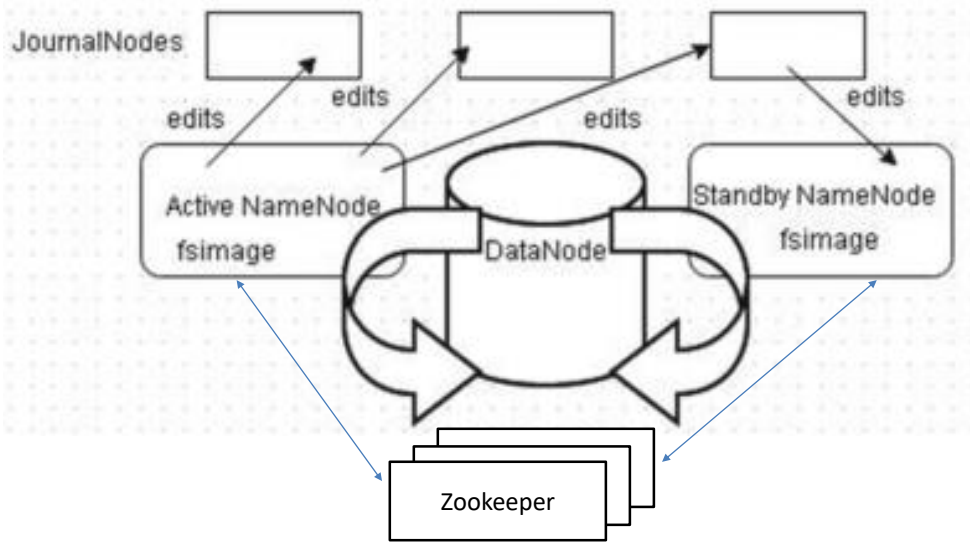
Hadoop: Architecture de HDFS

- **Secondary NameNode** *(suite)*
- Le nouveau **NameNode** ne peut répondre aux demandes des clients tant qu'il n'a pas **(1)** chargé son **FsImage dans la mémoire**, **(2)** relu son journal d'édition (EditLog) , et **(3)** reçu suffisamment de rapports de blocs des **DataNodes** *(par défaut **95%** des DN)* pour quitter le mode sans échec (**Safe Mode**). [Paramètre: dfs.namenode.safemode.threshold-pct]
- Sur les grands clusters avec un grand nombre de fichiers et de blocs, le temps nécessaire pour qu'un **NameNode** démarre à froid peut prendre plusieurs minutes à plusieurs heures.
- **Hadoop 2** a remédié à cette situation en ajoutant la prise en charge de la **haute disponibilité** (**High Availability - HA**) HDFS: il y a un pair de **NameNodes Passif/Actif**. En cas de panne du **Namenode actif**, le **passif** devient **actif** (*à chaud*) pour continuer à répondre aux demandes des clients sans interruption significative.

21

Hadoop: Architecture de HDFS

Haute Disponibilité dans HDFS (high-availability – HA)



22

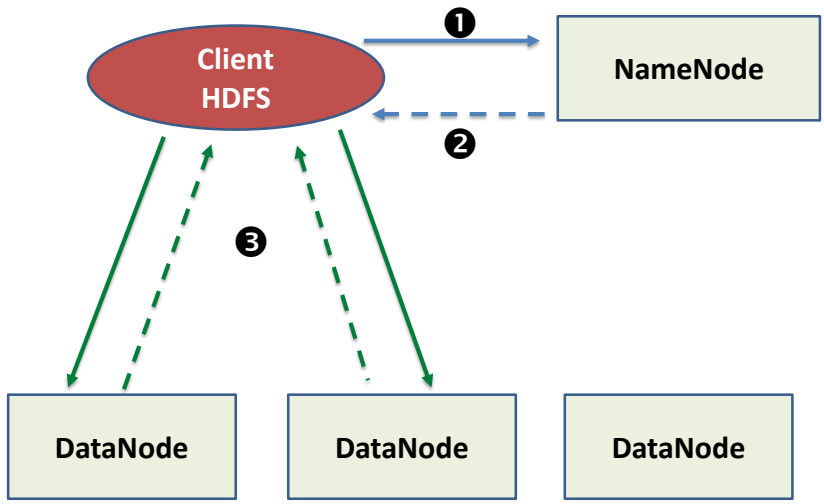
Hadoop: Architecture de HDFS

Haute Disponibilité dans HDFS (high-availability – HA) (Hadoop 2)

1. Le NameNode actif et le NameNode passif communiquent avec un groupe (quorum) de JournalNodes (généralement 3)
2. Le NameNode actif envoie les "edits" (logs) aux JounalNodes pour les partager avec le NameNode passif.
3. Le NameNode passif récupère les "edits" depuis l'un des JournalNodes et les applique à la copie Fsimage qu'il a afin de la synchroniser avec celle du NameNode actif.
4. Les DataNodes envoient périodiquement les rapports sur les blocs aux NameNodes actif et passif.
5. ZooKeeper est un démon de coordination utilisé (***sur plusieurs nœuds***) pour coordonner le basculement rapide du NameNode actif vers le NameNode passif. ZooKeeper a deux fonctions dans un cluster HDFS haute disponibilité:
   - Détecter une défaillance du NameNode actif. Les deux NameNodes (passif et actif) conservent une session active dans ZooKeeper. Lorsqu'il y a défaillance du NameNode actif, ZooKeeper détecte cette défaillance suite à l'interruption de la session du NameNode actif.
   - ZooKeeper initie un basculement vers un NameNode passif et fournit un mécanisme pour l'**élection** du NameNode actif lorsqu'il y a plusieurs candidats (NameNodes passfis)

23

Hadoop: Lecture d'un fichier HDFS (1/4)

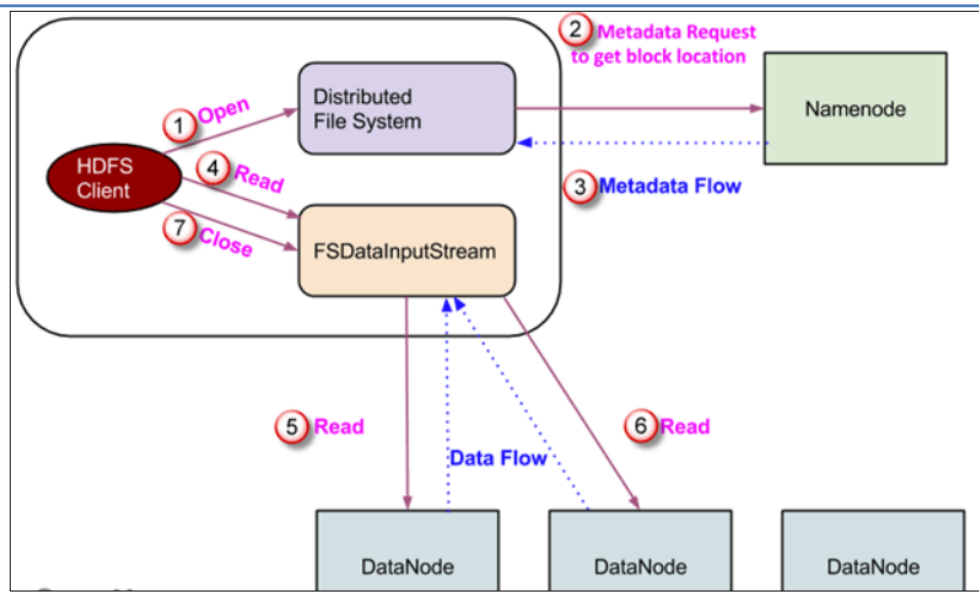


24

Hadoop: Lecture d'un fichier HDFS (2/4)

1. Le client (application HDFS) communique avec le NameNode pour demander la lecture d'un fichier (*ou plusieurs, un répertoire par exemple*)
2. Le NameNode envoie des métadonnées au client avec le détail du nombre de blocs, du nombre de réplications, des identifiants et tailles des blocs ainsi que leur ordre et leurs emplacements.
3. Le client communique avec les DataNodes où les blocs sont présents.
4. Le client commence à lire les blocs en parallèle à partir des DataNodes en se basant sur les métadonnées reçues du NameNode.
5. Pour améliorer les performances de lecture, l'emplacement de chaque bloc est choisi en fonction de sa distance par rapport au client: d'abord lecture des blocs dans le même DataNode, puis un autre DataNode dans le même rack, et enfin un autre DataNode dans un autre rack.
6. Une fois que le client reçoit tous les blocs, il les combine pour former un fichier.

Hadoop: Lecture d'un fichier HDFS (3/4)



Référence: https://www.guru99.com/learn-hdfs-a-beginners-guide.html

Hadoop: Lecture d'un fichier HDFS (4/4)

❶ Le client appelle la méthode **open()** de l'objet **FileSystem** pour lire un fichier HDFS.

❷ Cet objet se connecte au **NameNode** à l'aide d'un appel **RPC** pour avoir les métadonnées du fichier à lire telles que les emplacements des blocs du fichier.

❸ Le NameNode envoie les métadonnées au client avec le détail du nombre de blocs, des identifiants et les tailles de blocs, des emplacements des blocs et du nombre de réplication.

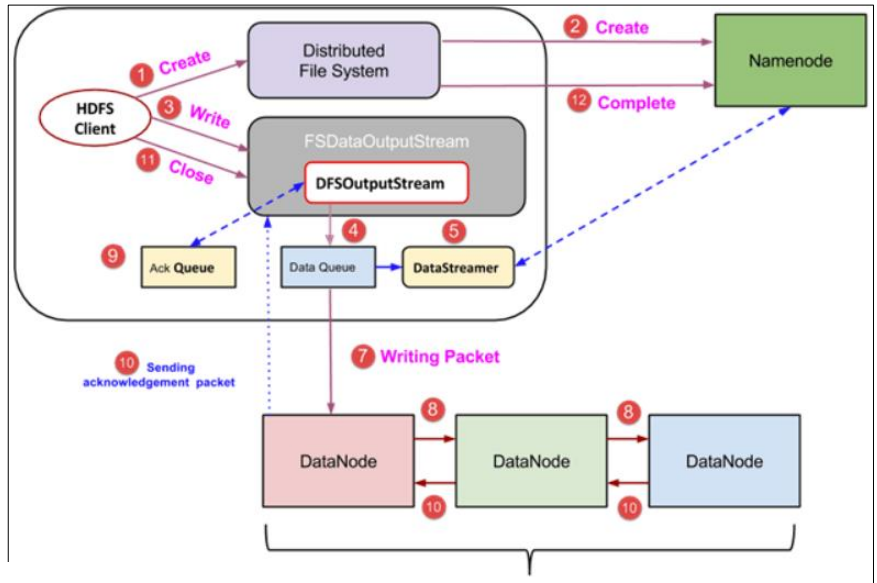
❹ Le client appelle la méthode **read(),** ce qui permet à un objet **FSDataInputStream** d'établir une connexion avec un **DataNode** contenant un bloc du fichier

❺❻Le client lit ce bloc sous forme de flux via des appels répétitifs de **read()**. A la fin d'un bloc, **FSDataInputStream** ferme la connexion avec le **DataNode**.

❼La méthode c**lose()** est appelée à la fin de la lecture de tous les blocs.

Une fois que le client reçoit tous les blocs, il les combine pour former un fichier.

Hadoop: Ecriture d'un fichier dans HDFS (1/4)



Référence:
https://www.guru99.com/learn-hdfs-a-beginners-guide.html

Hadoop: Ecriture d'un fichier dans HDFS (2/4)

1. Le client appelle la méthode **create()** de l'objet **DistributedFileSystem** qui permet de de créer un nouveau fichier pour écriture.
2. Cet objet se connecte au **NameNode** à l'aide de **RPC** et lui demande la création d'un nouveau fichier. A ce stade on n'associe aucun bloc à ce fichier. Le **NameNode** vérifie s'il y a déjà un autre fichier de même nom et si le client à les droits nécessaires pour créer le fichier. Dans ce cas, une entrée (dans le **FsImage**) est créée pour le nouveau fichier. Dans le cas contraire, une exception **IOException** est soulevée.
3. Un objet de la classe **FSDataOutputStream** est retourné au client afin de l'utiliser pour écrire le contenu du fichier dans **HDFS** en invoquant la méthode **write()** (en boucle)
4. L'objet **FSDataOutputStream** utilise un autre objet **DFSOutputStream** qui gère la communication avec les **DataNodes** et le **NameNode**. **DFSOutputStream** crée à fur et à mesure des paquets (64k) avec les données écrites par le client. Ces paquets sont mis dans une file d'attente appelée **DataQueue**. **DFSOutputStream** décompose ainsi les fichiers à écrire en paquets.

Hadoop: Ecriture d'un fichier dans HDFS (3/4)

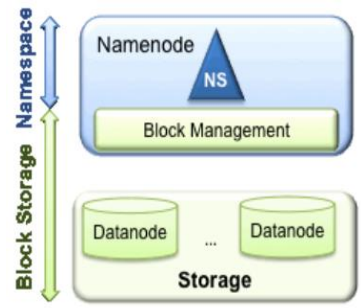
5. Le composant **DataStreamer** consomme les paquets de **DataQueue** et demande au **NameNode** l'allocation de nouveaux blocs. Le **NameNode** envoie les métadonnées au client avec le détail du nombre de blocs, de leurs identifiants, de leurs emplacements et du nombre de réplication. Ainsi les **DataNodes** à utiliser pour la réplication de chaque bloc seront identifiés.
6. Le processus de réplication d'**un bloc** commence par la création d'un pipeline (*par le client*) à l'aide de **DataNodes** identifiés dans l'étape 5. Dans le schéma, le facteur de réplication est 3, il y a donc 3 **DataNodes** dans le pipeline.
7. Le **DataStreamer** envoie des paquets vers le premier **DataNode** du pipeline
8. Chaque **DataNode** d'un pipeline stocke le paquet reçu et le transmet au **DataNode** suivant du pipeline.
9. **DFSOutputStream** gère une autre file d'attente, "**Ack Queue**", pour stocker les paquets qui attendent un accusé de réception de **DataNodes**.

Hadoop: Ecriture d'un fichier dans HDFS (4/4)

10. Lorsque l'accusé de réception d'un paquet dans la file d'attente est reçu de tous les DataNodes du pipeline, il est supprimé de la file «Ack Queue». En cas d'échec de DataNode, les paquets de cette file d'attente sont utilisés pour relancer l'opération.
11. Le client termine l'écriture des données par l'appel de la méthode close (), ce qui entraîne l'envoie des paquets de données restants vers le pipeline, puis l'attente de l'accusé de réception.
12. Une fois l'accusé de réception final reçu, le NameNode est contacté pour lui indiquer que l'opération d'écriture de fichier est terminée.

Architecture de la fédération HDFS (1/2)

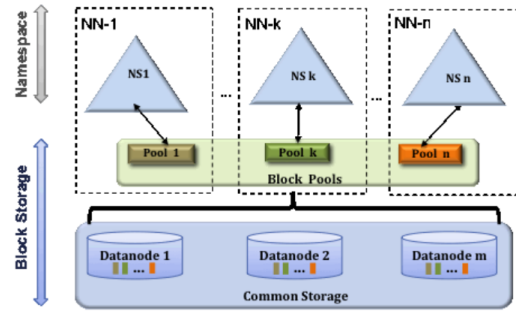
- Les fichiers dans **Hadoop** (**HDFS**) sont divisés en blocs qui sont ensuite stockés dans différents **DataNodes**.
- Pour accéder au contenu d'un fichier, on doit localiser ses blocs pour pouvoir les lire. Le **NameNode** permet cette localisation par la gestion de toutes les **méta-données** des fichiers et leurs blocs.
- Ces **méta-données** sont organisées en **pools de blocs (Blockpools)** et stockées dans la RAM du **NameNode.** Ainsi, on a besoin de les reconstituer en cas de perte (*Edit-Log, NameNode Secondaire, Rapports détaillés des DataNodes, Architecture HA*).



- Dans **Hadoop 1** on a un seul **NameNode** qui répond aux commandes d'accès aux fichiers de tous les utilisateurs.
- **Hadoop 2** a introduit la **fédération HDFS** basée sur l'utilisation de **plusieurs NameNodes**.

Architecture de la fédération HDFS (2/2)

- Dans une architecture de la **Fédération HDFS** (**Hadoop Federation**), on a **multiple namespaces** par décomposition de l'arborescence des fichiers en plusieurs parties.
- **Exemple**: utilisation de trois namespaces **/finance** , **/agriculture** et **/automobile**
- Le **Blockpool** (*métadonnées*) de chaque **namespace** est géré par un **NameNode** spécifique.
- Pour accéder à un fichier spécifique, on s'adresse à un **NameNode** bien déterminé.



- Chaque DataNode s'enregistre auprès de tous les NameNodes (leur envoie périodiquement des pulsations et des rapports de blocs, exécute leurs commandes)
- Les **Blockpools** sont gérés indépendamment les uns des autres. Ainsi, la perte d'un **NameNode** ne cause pas l'arrêt total de l'accès au **cluster Hadoop**.
- Avantages de la fédération HDFS sont:
  1. **Scalabilité et Performance**: l'ajout de NameNodes supplémentaires au cluster augmente le débit des opérations de lecture et écriture du système de fichiers.
  2. **Isolation** : On n'a aucune isolation avec un seul **NameNode** dans un environnement multi-utilisateur. Les tests d'une application peuvent surcharger le NameNode et ralentir les applications en production. L'utilisation de plusieurs NameNodes, des catégories d'applications et d'utilisateurs peuvent être isolées dans différents namespaces.

Hadoop: Modes de fonctionnement

Hadoop possède trois modes d'exécution :

- **Mode local (standalone)** : Hadoop fonctionne sur une seule machine et tout s'exécute dans la même JVM (*Java Virtual Machine*). En mode local le système de gestion de fichiers utilisé est celui du système hôte (Exemple Linux: ext3, ext4 ou xfs) et non HDFS.
- **Mode pseudo-distribué**: Hadoop fonctionne sur une seule machine, mais chacun des daemons principaux s'exécute dans sa propre JVM. Le système de fichier utilisé est HDFS dans ce mode.
- **Mode totalement distribué**: c'est le mode d'exécution réel d'Hadoop. Il permet de faire fonctionner le système de fichiers distribué et les daemons sur un ensemble de machines différentes.

Hadoop: Commandes HDFS

- Deux façons pour manipuler HDFS
  - via l'API Java
  - via les commandes depuis un terminal
- Nous utiliserons Hadoop avec un seul nœud pour tester et utiliser quelques commandes HDFS.
- Les clients HDFS utilisent la propriété **fs.default.name** ou **fs.defaultFS** de Hadoop pour déterminer l'URL de l'hôte et le port du **NameNode**.
- Nous l'utiliserons sur localhost comme **fs.defaultFS** = ***hdfs***://localhost: **8020** avec le port **8020** HDFS par défaut, **fs.defaultFS** = ***s3a:***//bucket_name pour le système de fichier Amazon Simple Storage Service (S3)**,** ou autre.
- Les propriétés et paramètres Hadoop sont dans des fichiers de configuration:
  /etc/hadoop/conf/**core-site.xml**
  /etc/hadoop/conf/**hdfs-site.xml:**

  Facteur de réplication (**dfs.replication**),
  NameNode secondaire (**dfs.namenode.secondary.http-address**), …
  ……

35

Hadoop: Ligne de commande HDFS

- Syntaxe utilisé: $ **hadoop fs** *-commande*

  ou

  $ **hdfs dfs** *-commande*

  fs : Interpréteur (Shell) de commandes qui supporte plusieurs commandes: **mkdir**, **ls**, **copyFromLocal**, ….
- Pour avoir la liste des commandes: $ **hdfs dfs -help**
- Pour avoir la description d'une commande: $ **hdfs dfs -help** *commande*

  Exemple: $ **hdfs dfs -help** *mkdir*
- Pour avoir la syntaxe d'utilisation d'une commande: $ **hdfs dfs -usage** *commande*

  Exemple: $ **hdfs dfs -usage** put
- Pour vérifier l'état du contenu d'un dossier HDFS: $ **hdfs fsck** *chemin* **-***options*
  Exemples: $ **hdfs fsck** /user/tp **-files**
  $ **hdfs fsck** /user/tp **-files -blocks -locations**
  $ **hdfs fsck** /user/tp/file1.csv **-blocks -locations**
- Pour avoir la syntaxe détaillée de hdfs fsck: $ **hdfs fsck -help**

36

Hadoop: Ligne de commande HDFS

Exemples:

- Afficher l'arborescence de la racine HDFS: $ hdfs dfs **-ls** **/**

  Ou $ hdfs dfs **-ls** **hdfs://localhost:8020/**

- Créer un dossier dans HDFS: $ hdfs dfs **-mkdir** /**user**/amine

- Copier un fichier local vers HDFS:

  -put

  $ **hdfs dfs -copyFromLocal** */home/cloudera/file1.txt* */**user**/amine/file1.txt*

- Copier le contenu d'un dossier local vers HDFS:

  $ **hdfs dfs -put** */home/cloudera/**bank*** */**user**/amine/*

- Copier un fichier vers HDFS en précisant la taille de bloc différente de celle par défaut:

$ **hdfs dfs** ***-D dfs.blocksize=33554432*** ***-put*** */home/cloudera/file1.txt* */**user**/amine/file1.txt*

Hadoop: Ligne de commande HDFS

Exemples:

- Afficher le contenu d'un dossier HDFS:

  $ **hdfs dfs - ls** /**user**/amine

- Copier un fichier HDFS vers le système local

  -get

$ hdfs dfs **-copyToLocal** */**user**/amine/quangle.txt* */home/cloudera/quangle.txt*

- Afficher les dernières lignes (1Ko) d'un fichier:

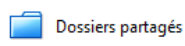
  $ hdfs dfs **-tail** */**user**/amine/quangle.txt*

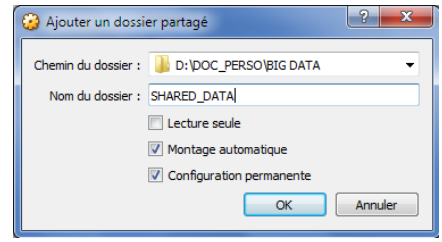
- Affiche un rapport sur les blocs des fichiers d'un dossier donné:

  $ **hdfs fsck** */**user**/amine* **-files -blocks -locations**

- Avoir l'emplacement du FsImage**: hdfs getconf -confKey dfs.namenode.name.dir**

Atelier 1 *(1/10)*

- Configuration du dossier partagé entre la VM Cloudera et le système hôte.
  - Créer un dossier et copier dedans les ressources de TP(fichier texte, csv, …)
  - Clic droit sur le nom da la VM  /  Configuration / Général / Avancé / Presse-papier partagé : **Bidirectionnel**
  - Clic sur  / Clic sur  / Choisir un dossier, donner lui un nom de partage et cocher les deux cases *Montage automatique* et *Configuration permanente*
  - Démarrer votre VM et lancer un terminal
  - Modifier le clavier en azerty: **setxkbmap fr**
  - Taper la commande: **mkdir** *labfile*
  - Taper la commande:



sudo mount -t vboxsf *shared_data labfile*

  - Un raccourci vers le dossier partagé est ajouté sur le bureau de la VM

Atelier 1 *(2/10):* Configuration de Hadoop et état des services

1. Dans un terminal, afficher le contenu du fichier **/etc/hadoop/conf/core-site.xml**
2. Quel est l'URL du processus serveur **NameNode** de HDFS ainsi que le port?
3. Est-ce que c'est équivalent à *localhost* ? Pourquoi ?
4. Quel est la valeur du facteur de réplication?
5. Utiliser la commande **service** *nomservice* status pour vérifier l'état des services suivants:

   hadoop-hdfs-namenode
   hadoop-hdfs-secondarynamenode
   hadoop-hdfs-datanode
   hadoop-yarn-resourcemanager
   hadoop-yarn-nodemanager
6. Utiliser l'outil **jps** pour lister les processus Java en cours d'exécution:

   $ sudo jps -l

Atelier 1 *(3/10):* Commandes HDFS

1. Lister le contenu du **dossier HDFS**: **/user**
2. Créer un dossier HDFS **tp** dans **/user/cloudera**
3. Copier le fichier local ***FL_insurance.csv*** dans le dossier HDFS **/user/cloudera/tp**
4. Afficher le contenu du dossier HDFS: **/user/cloudera/tp**
5. Afficher (commande **tail**) les 10 dernières lignes du fichier HDFS: ***FL_insurance.csv***
6. Arrêter le **NameNode**: **service hadoop-hdfs-namenode stop**
7. Envoyer une commande HDFS au **NameNode**, **ls** par exemple. Que remarque-t-on?
8. Redémarrer le **NameNode**
9. Récupérer la taille d'un bloc HDFS: **hdfs getconf -confKey dfs.blocksize**
10. Récupérer le facteur de réplication: **hdfs getconf -confKey dfs.replication**
11. Copier le fichier local ***purchases.txt*** vers le dossier HDFS **/user/cloudera/tp**
12. Utiliser la commande **hdfs fsck** pour afficher un rapport détaillé sur le fichier ***purchases.txt*** dans HDFS. Identifier: *le nombre de blocs, le nombre de datanodes, le facteur de réplication, …*

Atelier 1 *(4/10):* Commandes HDFS

- ***Over-replicated blocks****: Les blocs dont la réplication dépasse celle du fichier auquel ils appartiennent. Normalement, la sur-réplication n'est pas un problème et HDFS supprimera automatiquement les copies en excès.*
- ***Under-replicated blocks****: Les blocs qui ne répondent pas à au facteur de réplication du fichier auquel ils appartiennent. HDFS créera automatiquement de nouvelles répliques de blocs sous-répliqués jusqu'à ce qu'ils atteignent la réplication cible.*
- ***Misreplicated blocks****: Les blocs qui ne satisfont pas à la politique de placement de répliques de bloc. Par exemple, pour un facteur de réplication de trois dans un cluster multirack, si les trois répliques d'un bloc se trouvent sur le même rack, le bloc est mal répliqué car les copies doivent être réparties sur au moins deux racks. HDFS répliquera automatiquement les blocs mal répliqués afin qu'ils satisfassent la politique de placement en rack.*
- ***Corrupt blocks****: Les blocs dont les répliques sont toutes corrompues. Les blocs avec au moins une réplique non corrompue ne sont pas signalés comme corrompus; le NameNode répliquera la réplique non corrompue jusqu'à ce que la réplication cible soit atteinte.*
- ***Missing replicas****: Ce sont des blocs sans répliques nulle part dans le cluster.*

Atelier 1 *(5/10):* Commandes HDFS

13. Créer et exécuter un script shell pour:
    a) Supprimer **récursivement** le dossier HDFS **/user/cloudera/data** s'il existe
    b) Créer les dossiers HDFS: **/user/cloudera/data** et **/user/cloudera/data/tpmr**
    c) Copier les fichiers **purchases.txt** et ***FL_insurance.csv*** du dossier local **shared_data** vers le dossier HDFS **/user/cloudera/data/tpmr**
    d) Afficher de manière récursive l'arborescence du dossier HDFS **/user/cloudera/data**
14. Utiliser la commande **df** pour consulter l'espace de stockage: **hdfs dfs –df -h**
15. Pour Modifier le facteur de réplication, valeur 2, du fichier **purchases.txt** dans HDFS: $ **hadoop fs -setrep -w** 2 /user/cloudera/tp/purchases.txt

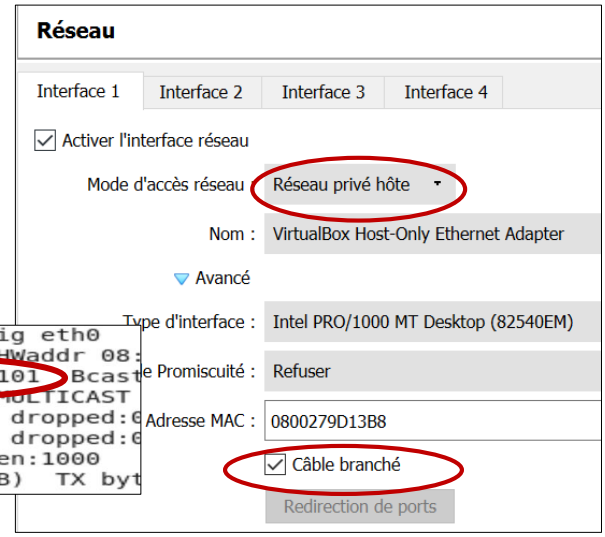
N.B: **N'exécutez pas cette commande**

- **Sur un pseudo-cluster, le facteur de réplication pour un fichier ne peut pas prendre une valeur supérieure à 1.**
- **Cette commande prend un temps considérable si le fichier est volumineux.**
- **L'option –w oblige Hadoop (le NameNode) à attendre jusqu'à ce que la réplication soit effectuée.**

Atelier 1 *(6/10):* Connexion SSH la VM

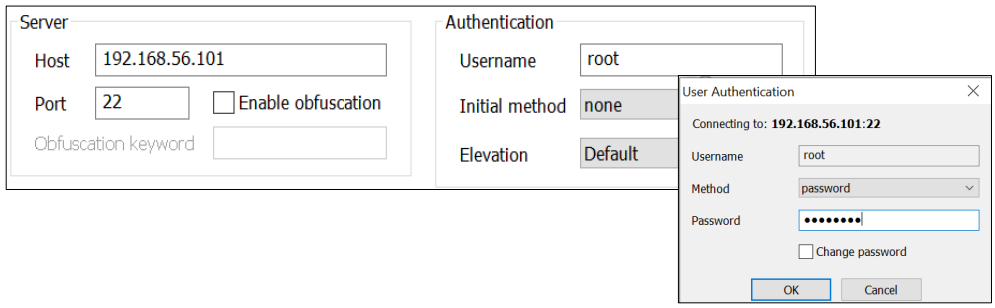
15. Dans les paramètres Réseau de VM Cloudera: modifier le "**Mode d'accès réseau**"
16. Redémarrer votre VM
17. Dans un terminal de votre VM, afficher son adresse IP: $ **ifconfig eth0**

```
[cloudera@quickstart ~]$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:
          inet addr:192.168.56.101  Bcast
          UP BROADCAST RUNNING MULTICAST
          RX packets:5 errors:0 dropped:0
          TX packets:6 errors:0 dropped:0
          collisions:0 txqueuelen:1000
          RX bytes:2420 (2.3 KiB)  TX byt
```



Atelier 1 *(7/10):* Connexion SSH la VM

18. Dans votre système hôte, démarrer un client SSH (Bitvise par exemple) et utiliser l'adresse IP pour se connecter à la VM avec le compte **root/cloudera**



19. Utiliser la fenêtre FTP pour transférer vos fichiers de données vers un sous-dossier de **/home/cloudera** dans votre VM
20. Utiliser la fenêtre de terminal pour lancer des commandes Linux ou HDFS

Atelier 1 *(8/10):* Accès à des services via un navigateur Web

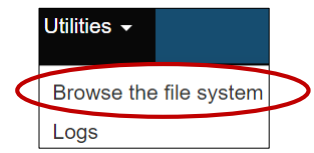
21. Dans un navigateur web de votre système hôte, accéder à l'URL: http:// *adresse_IP_VM* **:50070**

    Par exemple: http://192.168.56.101:50070

La page d'accueil fournit des informations sur le **NameNode**: **Overview**, **Summary**, ….

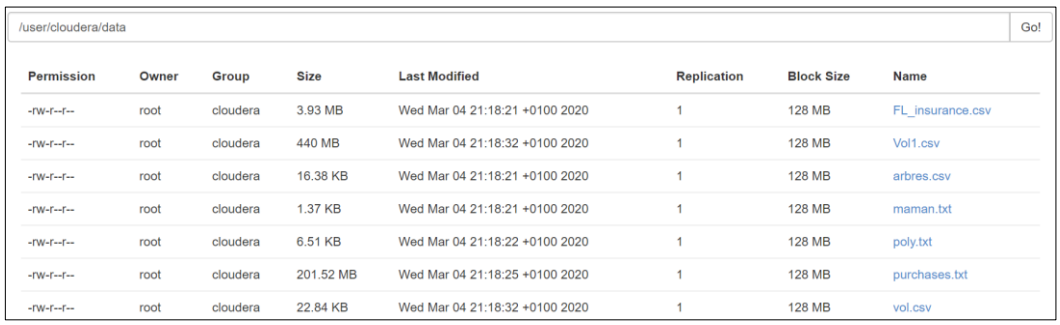
Le menu principal permet d'avoir des informations sur les DataNodes, le Snapshot de métadonnées, …

22. Pour parcourir l'arborescence HDFS, utiliser l'option **Utilities/Browse the file system**:



Atelier 1 *(9/10):* Accès à des services via un navigateur Web

23. Les propriétés des éléments de chaque dossier sont indiquées!



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	cloudera	3.93 MB	Wed Mar 04 21:18:21 +0100 2020	1	128 MB	FL_insurance.csv
-rw-r--r--	root	cloudera	440 MB	Wed Mar 04 21:18:32 +0100 2020	1	128 MB	Vol1.csv
-rw-r--r--	root	cloudera	16.38 KB	Wed Mar 04 21:18:21 +0100 2020	1	128 MB	arbres.csv
-rw-r--r--	root	cloudera	1.37 KB	Wed Mar 04 21:18:21 +0100 2020	1	128 MB	maman.txt
-rw-r--r--	root	cloudera	6.51 KB	Wed Mar 04 21:18:22 +0100 2020	1	128 MB	poly.txt
-rw-r--r--	root	cloudera	201.52 MB	Wed Mar 04 21:18:25 +0100 2020	1	128 MB	purchases.txt
-rw-r--r--	root	cloudera	22.84 KB	Wed Mar 04 21:18:32 +0100 2020	1	128 MB	vol.csv

Atelier 1 *(10/10):* Accès à des services via un navigateur Web

24. Un clic sur un fichier permet d'avoir les propriétés de ses blocs.