**Les commandes HDFS**

**Les commandes les plus utilisées :**

**Créer un dossier dans HDFS :**

**Commande :**

hadoop fs -mkdir

**Exemple :**

hadoop fs -mkdir /user/monDossier

hadoop fs -mkdir /user/monDossier1  /user/monDossier2  /user/monDossier3

L’option -p est nécessaire si le dossier parent n’existe pas lors de la création d’un sous répertoire.

**Lister le contenu d’un dossier :**

**Commande :**

hadoop fs -ls

**Exemple :**

hadoop fs -ls /user

hadoop fs -ls /user/monDossier

**Charger un ou plusieurs fichiers du local à HDFS :**

**Commande :**

hadoop fs -put

**Exemple :**

hadoop fs -put /home/monFichier.txt /user/monDossier

**Exporter un ou plusieurs fichiers de HDFS au local :**

**Commande :**

hadoop fs -get

**Exemple :**

hadoop fs -get /user/monDossier/monFichier.txt /home

**Copier un ou plusieurs fichiers dans HDFS :**

**Commande :**

hadoop fs -cp

**Exemple :**

hadoop fs -cp /user/monDossier1/monFichier.txt  /user/monDossier2

**Déplacer un ou plusieurs fichiers dans HDFS :**

**Commande :**

hadoop fs -mv

**Exemple :**

hadoop fs -mv /user/monDossier1/monFichier.txt  /user/monDossier2

**Charger un ou plusieurs fichiers du local à HDFS :** Cette commande est similaire à -put

**Commande :**

hadoop fs -copyFromLocal

**Exemple :**

hadoop fs -copyFromLocal /home/monFichier.txt /user/monDossier

**Exporter un ou plusieurs fichiers de HDFS au local:**

**Commande :**

hadoop fs -copyToLocal

**Exemple :**

hadoop fs -copyToLocal /user/monDossier/monFichier.txt /home

**Afficher le contenu d’un fichier :** Cette commande est similaire à -get

**Commande:**

hadoop fs -cat <Path[Filename]>

**Exemple :**

hadoop fs -cat /user/monFichier.txt

**Afficher les dernières lignes d’un fichier :**

**Commande :**

hadoop fs -tail <Path[Filename]>

**Exemple :**

hadoop fs -tail /user/monFichier.txt

**Supprimer un fichier dans HDFS:**

**Commande :**

hadoop fs -rm

**Exemple :**

hadoop fs -rm /user/monFichier.txt

**Suppression récursive dans HDFS :**

**Commande :**

hadoop fs -rmr

**Exemple :**

hadoop fs -rmr /user/

**Tableau récapitulatif des commandes HDFS** **:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Import/export de fichier entre HDFS et le local** | **Déplacer les fichiers** | **Lister le contenu d’un fichier/répertoire** | **Afficher le contenu d’un fichier** | **Supprimer le contenu d’un fichier/répertoire** | **Créer un fichier vide** |
| -put  -Get  -CopyFromLocal  -CopyToLocal | -cp  -mv  -movefromlocal  -movetolocal | -Ls  -lsr | -cat  -tail | -rm  -rmr | -touchz |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Voici quelques commandes à essayer dans la fenêtre ssh :**

• **hdfs dfs -ls** : elle n’affiche rien pour l’instant car elle s’adresse par défaut à votre dossier HDFS personnel /user/votre\_login qui est vide. Le temps de réaction de la commande est dû au grand nombre de jars présents dans son CLASSPATH. On n’y peut rien dans cette version de Hadoop.

• **hdfs dfs -ls /** : affiche ce qu’il y a à la racine HDFS. Vous pouvez descendre inspecter les dossiers que vous voyez.

**Exemple :**

**hdfs dfs -ls /user**. Il n’y a pas de commande équivalente à cd, parce qu’il n’y a pas de notion de dossier courant dans HDFS, donc à chaque fois, il faut remettre le chemin complet. C’est une habitude à prendre.

• **hdfs dfs -ls -R -h /var** : affiche les fichiers des sous-dossiers, avec une taille arrondie en Ko, Mo ou Go.

• **hdfs dfs -mkdir fichiers** : crée un dossier dans votre espace HDFS, c’est à dire **/user/votre\_login/**fichiers. Notez que la taille d’un dossier sera toujours 0. Créez un fichier appelé bonjour.txt dans votre compte Linux et contenant le mot « bonjour ». • Copier ce fichier sur HDFS par **hdfs dfs -put bonjour.txt**. Utilisez **hdfs dfs -ls -R** pour vérifier.

• **hdfs dfs -cat bonjour.txt** : affiche le contenu. Il n’y a pas de commande more mais vous pouvez faire **hdfs dfs -cat bonjour.txt | more**

• **hdfs dfs -tail bonjour.txt** : affiche le dernier Ko du fichier.

• Supprimer ce fichier de HDFS par **hdfs dfs -rm bonjour.txt.**

**•** Remettre à nouveau ce fichier par **hdfs dfs -copyFromLocal bonjour.txt** (vérifier avec hdfs dfs -ls). Cette commande est similaire à hdfs dfs -put.

• **hdfs dfs -chmod go+w bonjour.txt** (vérifier son propriétaire, son groupe et ses droits avec **hdfs dfs -ls**)

• **hdfs dfs -chmod go-r bonjour.txt** (vérifier les droits)

• **hdfs dfs -mv bonjour.txt fichiers/bonjour.txt** (vérifier avec hdfs dfs -ls -R)

• **hdfs dfs -get fichiers/bonjour.txt demat.txt** : transfère le fichier de HDFS vers votre compte Linux en lui changeant son nom. Cette commande ne serait pas à faire avec de vraies méga-données !

**Afficher la liste des blocs d’un fichier :**

import java.io. \*;

import org. apache.hadoop.conf.Configuration;

import org. apache.hadoop.fs.\*;

public class CompterLignesArbres {

public static void main (String [] args) throws IOException {

// nom complet du fichier

Path nomcomplet = new Path("/share/paris/arbres.csv") ;

// ouvrir le fichier sur HDFS

Configuration conf = new Configuration ();

FileSystem fs = FileSystem.get(conf);

FSDataInputStream inStream = fs. open(nomcomplet); try {

// préparer un lecteur séquentiel

InputStreamReader isr = new InputStreamReader(inStream);

BufferedReader br = new BufferedReader(isr);

// parcourir les lignes une par une

String ligne = br. readLine();

while (ligne != null) {

// traiter la ligne courante System.out.println(ligne) ;

// passer à la ligne suivante (retourne null si fin fichier)

Ligne = br.readLine();

}} finally {

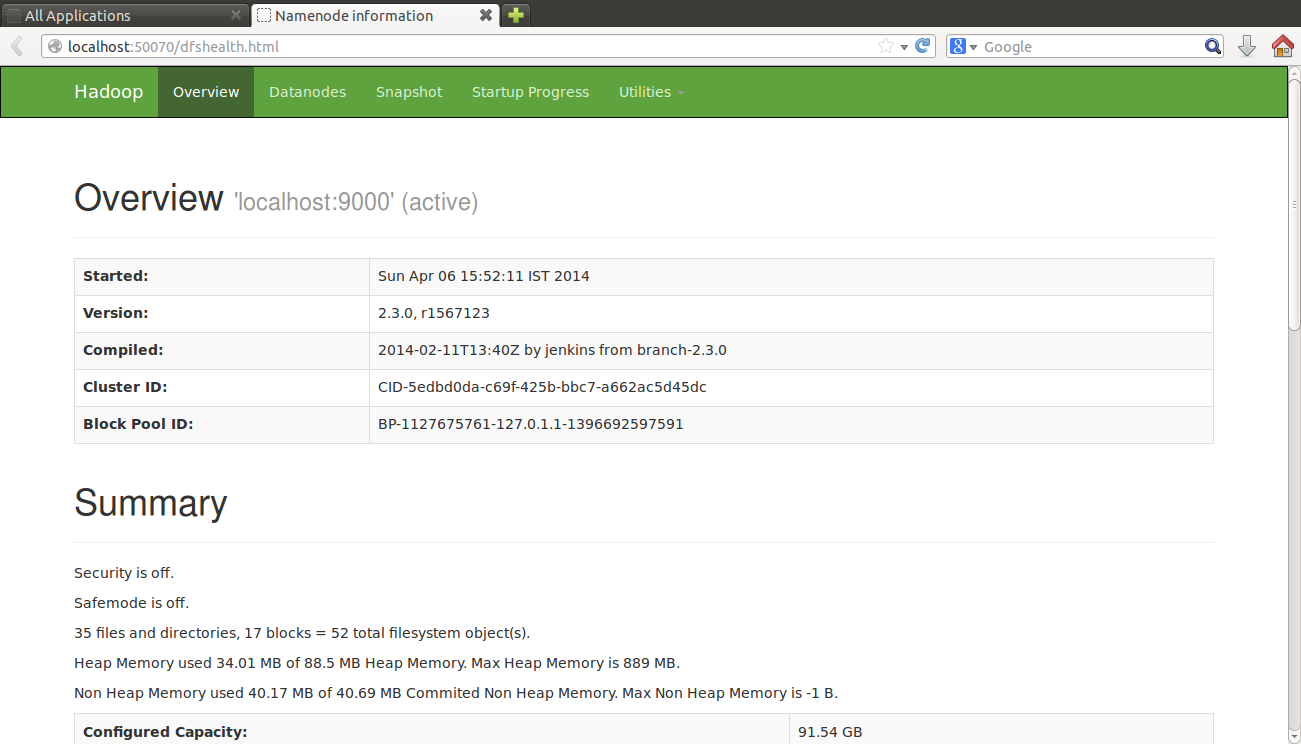
// fermer le fichier quoi qu'il arrive inStream.close() ;

fs.close(); } } }

**HDFS web UI :**

HDFS expose un serveur Web capable d'effectuer des opérations de surveillance de l'état et de navigation de fichiers de base. Par défaut, cela est exposé sur le port 50070 sur le NameNode. L'accès à http : // namenode : 50070 / avec un navigateur Web renverra une page contenant des informations sur la santé, la capacité et l'utilisation du cluster (similaire aux informations renvoyées par bin / hadoop dfsadmin -report).

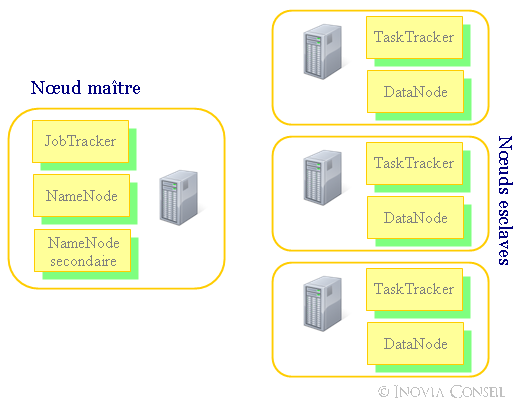
Entrez l'URL en tant que *http : // localhost : 50070 /* dans le navigateur Web et pointez sur le numéro de port 50070 sur le nom d'hôte localhost en mode pseudo-distribué. En mode Entièrement distribué, remplacez «localhost » par le nom d'hôte réel de la machine sur le cluster.



**HADOOP 1.X VS 2.X :**

|  |  |  |
| --- | --- | --- |
| **Sl No** | **Hadoop1** | **Hadoop2** |
| **1** | Prend en charge le modèle de traitement MapReduce (MR) uniquement. Ne prend pas en charge les outils non-MR | Permet de travailler dans MR ainsi que d'autres modèles de calcul distribué comme Spark, Hama, Giraph, Message Passing Interface) coprocesseurs MPI et HBase. |
| **2** | MR effectue à la fois le traitement et la gestion des ressources de cluster. | YARN (Encore un autre négociateur de ressources) gère les ressources de la grappe et le traitement est effectué en utilisant différents modèles de traitement. |
| **3** | A une mise à l'échelle limitée des noeuds. Limité à 4000 nœuds par cluster | A une meilleure évolutivité. Extensible jusqu'à 10000 nœuds par cluster |
| **4** | Fonctionne sur les concepts de slots - les slots peuvent exécuter une tâche Map ou une tâche Reduce uniquement. | Fonctionne sur les concepts de conteneurs.L'utilisation de conteneurs peut exécuter des tâches génériques. |
| **5** | Un Namenode unique pour gérer l'intégralité de l'espace de noms. | Plusieurs serveurs Namenode gèrent plusieurs espaces de noms. |
| **6** | A Single-Point-of-Failure (SPOF) - en raison de Namenode unique - et en cas de défaillance de Namenode, nécessite une intervention manuelle à surmonter. | A pour fonction de surmonter SPOF avec un Namenode de secours et dans le cas d'une panne de Namenode, il est configuré pour la récupération automatique. |
| **7** | L'API MR est compatible avec Hadoop1x. Un programme écrit dans Hadoop1 s'exécute dans Hadoop1x sans aucun fichier supplémentaire. | L'API MR requiert des fichiers supplémentaires pour qu'un programme écrit dans Hadoop1x s'exécute dans Hadoop2x. |
| **8** | A une limitation pour servir de plate-forme pour le traitement des événements, le streaming et les opérations en temps réel. | Peut servir de plate-forme pour une grande variété d'analyses de données, permettant d'exécuter des opérations de traitement d'événements, de diffusion en continu et en temps réel. |
| **9** | Une défaillance de Namenode affecte la pile. | Les piles Hadoop - Hive, Pig, HBase, etc. sont toutes équipées pour gérer les défaillances de Namenode. |
| **Dix** | Ne supporte pas Microsoft Windows | Ajout du support pour Microsoft Windows |

**Job tracker et task tracker :**



* Un **NameNode** (exécuté sur la machine maître) et plusieurs **DataNode** (exécutés sur les machines esclaves)
* Un **JobTracker** (exécuté sur la machine maître) et plusieurs **TaskManager** (exécutés sur les machines esclaves)
* Le **JobTracker** coordonne l’exécution des jobs sur l’ensemble du cluster. Il communique avec les TaskTrackers en leur attribuant des tâches d’exécution (map ou reduce). Dans le cas d’utilisation (théorique) présenté dans le premier article de la série, le JobTracker distribuerait 3 tâches map et 5 tâches reduce.  
  Par ailleurs, il permet d’avoir une vision globale sur la progression ou l’état du traitement distribué via une console d’administration web accessible par défaut sur le port 50030.  
  Le JobTracker est un démon cohabitant avec le NameNode. Il n’y a donc **qu’une instance** par cluster.
* Les **TaskTrackers** exécutent les tâches (map ou reduce) au sein d’une nouvelle JVM instantiée par le TaskTracker. Ainsi, un crash de la machine virtuelle n’impactera pas le TaskTracker.  
  Par ailleurs, ils notifient périodiquement le JobTracker du niveau de progression d’une tâche ou bien le notifient en cas d’erreur afin que celui-ci puissent reprogrammer et assigner une nouvelle tâche.  
  Un TaskTracker est un démon cohabitant avec un DataNode. Il y a donc **autant d’instances que de nœuds esclaves**.

**Yarn :**

Apache Hadoop YARN (Yet Another Resource Negotiator) est une technologie de gestion de clusters. Elle rend l'environnement Hadoop mieux adapté aux applications opérationnelles qui ne peuvent pas attendre la fin des traitements par lots

ARN compte parmi les fonctions clés de [Hadoop 2](http://www.lemagit.fr/definition/Hadoop-2), la deuxième génération de l'infrastructure open source de traitement distribué d'Apache Software Foundation. Décrit à l'origine par Apache comme un gestionnaire de ressources restructuré, YARN est désormais qualifié de système d'exploitation distribué, à grande échelle, destiné aux applications de [Big Data](http://www.lemagit.fr/definition/Big-Data-Gestion).

**les démons maitres esclaves :**

L'idée fondamentale de YARN est de diviser les fonctionnalités de la gestion des ressources et de la planification / surveillance des tâches en démons distincts. L'idée est d'avoir un ResourceManager global (RM) et une ApplicationMaster (AM) par application. Une application est soit un travail unique, soit un DAG d'emplois.

Le ResourceManager et le NodeManager forment la structure de calcul de données. Le ResourceManager est l'autorité ultime qui arbitre les ressources parmi toutes les applications du système. NodeManager est l'agent d'infrastructure par machine qui est responsable des conteneurs, surveillant leur utilisation des ressources (cpu, mémoire, disque, réseau) et en rapportant la même chose au ResourceManager / Scheduler.

Le ApplicationMaster par application est, en effet, une bibliothèque spécifique au framework et est chargé de négocier des ressources depuis le ResourceManager et de travailler avec le (s) NodeManager (s) pour exécuter et surveiller les tâches.

