

TP 1 : Installation et utilisation d'Apache Hbase

Filière : Data Engineer, INE2

Pr. D. ZAIDOUNI



Présentation de Hbase :

Apache HBase est un système de gestion de bases de données distribué, non-relationnel et orienté colonnes, développé au-dessus du système de fichier HDFS.

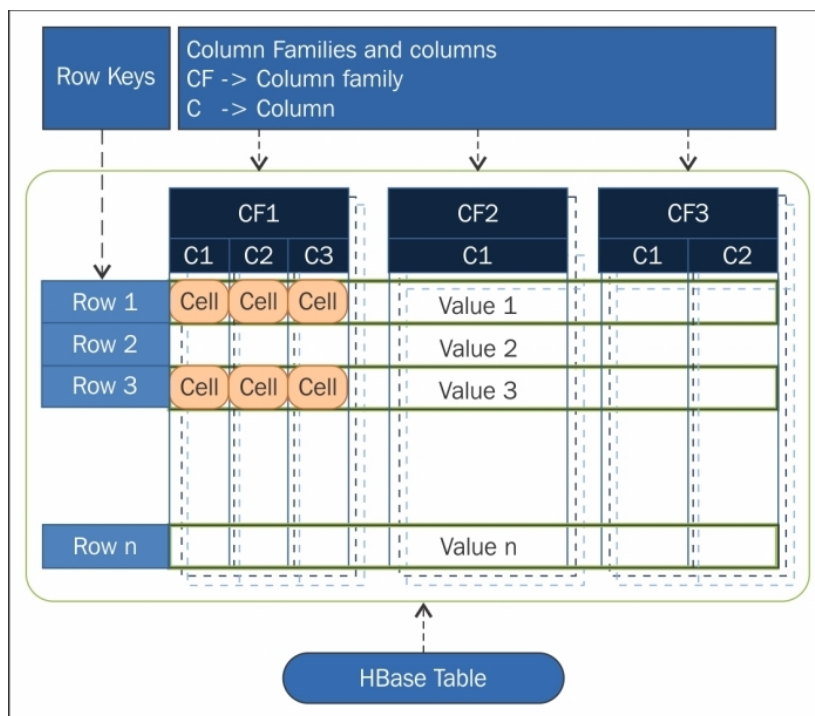
Il permet un accès aléatoire en écriture/lecture en temps réel à un très grand ensemble de données.

Modèle de données :

le modèle se base sur six concepts, qui sont :

- **Table** : dans HBase les données sont organisées dans des tables. Les noms des tables sont des chaînes de caractères.
- **Row** : dans chaque table, les données sont organisées dans des lignes. Une ligne est identifiée par une clé unique (RowKey). La Rowkey n'a pas de type, elle est traitée comme un tableau d'octets.
- **Column Family** : Les données au sein d'une ligne sont regroupées par column family. Chaque ligne de la table a les mêmes column families, qui peuvent être peuplées ou pas. Les column families sont définies à la création de la table dans HBase. Les noms des column families sont des chaînes de caractères.
- **Column qualifier** : L'accès aux données au sein d'une column family se fait via le column qualifier ou column. Ce dernier n'est pas spécifié à la création de la table mais plutôt à l'insertion de la donnée. Comme les rowkeys, le column qualifier n'est pas typé, il est traité comme un tableau d'octets.
- **Cell** : La combinaison du RowKey, de la Column Family ainsi que la Column qualifier identifie d'une manière unique une cellule. Les données stockées dans une cellule sont appelées les valeurs de cette cellule. Les valeurs n'ont pas de type, ils sont toujours considérés comme tableaux d'octets.

- **Version** : Les valeurs au sein d'une cellule sont versionnées. Les versions sont identifiées par leur timestamp (de type long). Le nombre de versions est configuré via la Column Family. Par défaut, ce nombre est égal à trois.



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

- **Table** -----
 - Contains column-families
- **Column family** -----
 - Logical and physical grouping of columns
- **Column** -----
 - Exists only when inserted
 - Can have multiple versions
 - Each row can have different set of columns
 - Each column identified by it's key
- **Row key** -----
 - Implicit primary key
 - Used for storing ordered rows
 - Efficient queries using row key

HBTABLE	
Row key	Value
11111	cf_data: { 'cq_name': 'name1', 'cq_val': 1111 } cf_info: { 'cq_desc': 'desc11111' }
22222	cf_data: { 'cq_name': 'name2', 'cq_val': 2013 @ ts = 2013, 'cq_val': 2012 @ ts = 2012 }

HFile

```

11111 cf_data cq_name name1 @ ts1
11111 cf_data cq_val 1111 @ ts1
22222 cf_data cq_name name2 @ ts1
22222 cf_data cq_val 2013 @ ts1
22222 cf_data cq_val 2012 @ ts2
          
```

HFile

```

11111 cf_info cq_desc desc11111 @ ts1
          
```

Autres caractéristiques de HBase:

HBase n'a pas de schéma prédéfini, sauf qu'il faut définir les familles de colonnes à la création des tables, car elles représentent l'organisation physique des données.

HBase est décrite comme étant un magasin de données clef/valeur, où la clef est la combinaison (row-column family-column-timestamp) représente la clef, et la cell représente la valeur.

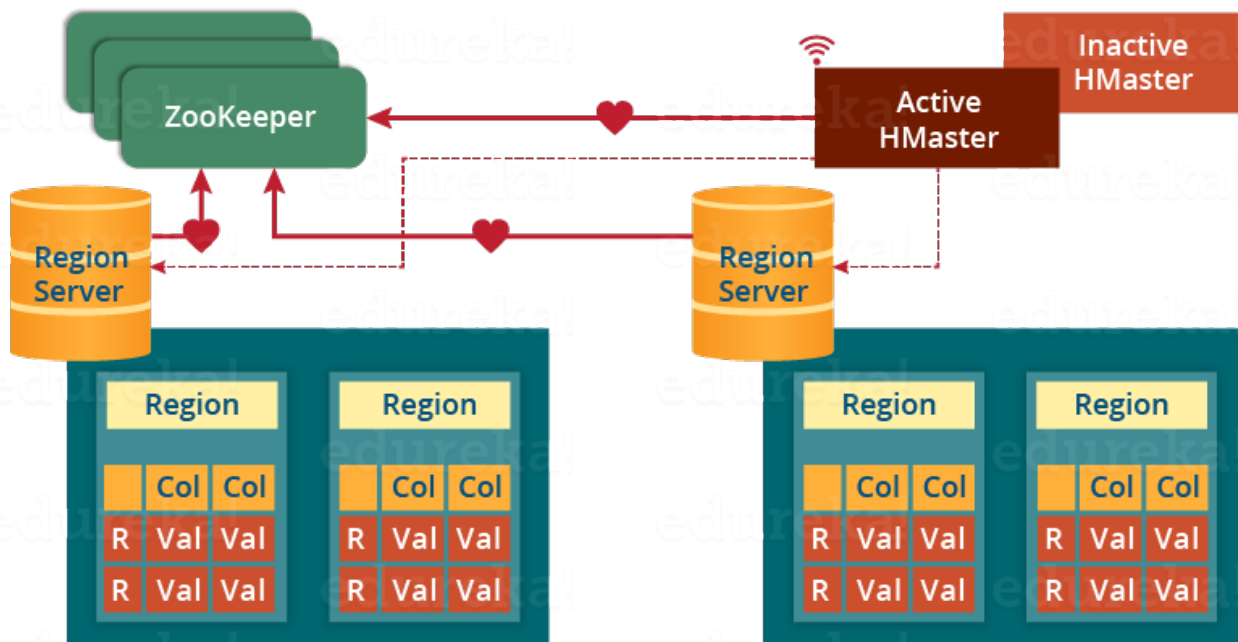
Architecture :

Physiquement, HBase est composé de trois types de serveurs de type Master/Slave :

- **Region Servers:** permettent de fournir les données pour lectures et écritures. Pour accéder aux données, les clients communiquent avec les RegionServers directement.
- **HBase HMaster :** gère l'affectation des régions, les opérations de création et suppression de tables.
- **Zookeeper:** permet de maintenir le cluster en état.

Le DataNode de Hadoop permet de stocker les données que le Region Server gère. Toutes les données de HBase sont stockées dans des fichiers HDFS. Les RegionServers sont colocalisés avec les DataNodes.

Le NameNode permet de maintenir les métadonnées sur tous les blocs physiques qui forment les fichiers.



Les tables HBase sont divisées horizontalement, par row en plusieurs Regions. Une region contient toutes les lignes de la table comprises entre deux clefs données. Les regions sont affectées à des noeuds dans le cluster, appelés Region Servers, qui permettent de servir les données pour la lecture et l'écriture. Un region server peut servir jusqu'à 1000 régions.

Le HBase Master est responsable de coordonner les region servers en assignant les régions au démarrage, les réassignant en cas de récupération ou d'équilibrage de charge, et en faisant le monitoring des instances des region

servers dans le cluster. Il permet également de fournir une interface pour la création, la suppression et la modification des tables.

HBase utilise Zookeeper comme service de coordination pour maintenir l'état du serveur dans le cluster. Zookeeper sait quels serveurs sont actifs et disponibles, et fournit une notification en cas d'échec d'un serveur.

Les objectifs de ce TP sont les suivants :

- Installation et configuration d'Apache Hbase en mode « Standalone » et en mode « Pseudo-distribué »,
- Manipulation de « HBase » (opérations create, put , get, list, scan, disable, drop, exists),
- L'utilisation de l'API Java de Hbase,
- Traitement de données avec Spark.

1) Installation et configuration d'Apache Hbase en mode « Standalone » et en mode « Pseudo-distribué » :

1- Préparation de l'environnement :

Dans ce TP, nous allons nous intéresser à l'utilisation de Hbase avec le système de fichier HDFS de **l'écosystème hadoop** et avec le logiciel de traitement **Spark**. Nous allons donc utiliser une VM avec Apache Hadoop et Apache Spark installé et configuré en mode local.

- Il faut donc tout d'abord **créer une VM0** qui soit configurer avec **Hadoop2.7.4** (version hadoop compatible avec **hbase-1.4.7**) et avec **spark-2.4.3** en mode local. L'installation et la configuration de hadoop et spark sur un seul nœud est déjà vu et expliqué dans le TP1 et TP2 relatifs au cours précédent : « Introduction aux Big data».

- N'oubliez pas d'utiliser les versions suivantes pour ce TP (afin d'éviter des problèmes d'incompatibilité de versions) :

- **hadoop-2.7.4.tar.gz** :

<https://archive.apache.org/dist/hadoop/core/hadoop-2.7.4/hadoop-2.7.4.tar.gz>

- **hbase-1.4.7-bin.tar.gz** :

<https://archive.apache.org/dist/hbase/1.4.7/hbase-1.4.7-bin.tar.gz>

- **spark-2.4.3-bin-hadoop2.7.tgz** :

<https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz>

- **Java version 1.8**

Attention :

Pour savoir quelle version de Hadoop est supportée par quelle version de Hbase, vous pouvez vous référer au tableau ci-dessous :

	HBase-1.2.x	HBase-1.3.x	HBase-2.0.x
Hadoop-2.4.x	S	S	X
Hadoop-2.5.x	S	S	X
Hadoop-2.6.0	X	X	X
Hadoop-2.6.1+	S	S	S
Hadoop-2.7.0	X	X	X
Hadoop-2.7.1+	S	S	S
Hadoop-2.8.[0-1]	X	X	X
Hadoop-2.8.2	NT	NT	NT
Hadoop-2.8.3+	NT	NT	S
Hadoop-2.9.0	X	X	X
Hadoop-3.0.0	NT	NT	NT

2-Installation et configuration d'Apache Hbase :

1- Récupérez les fichiers sources de Hbase :

- Téléchargez les fichiers sources de Hbase-1.4.7 à partir du lien fourni précédemment.
- Déposez le fichier « **hbase-1.4.7-bin.tar.gz** » dans le répertoire de votre choix, par exemple : /home/user/Documents/

2- Décompressez le fichier récupéré dans le répertoire de votre choix :

```
user@user:~$ cd /home/user/Documents/
```

```
user@user:~/Documents$ tar -xvzf hbase-1.4.7-bin.tar.gz
```

Un répertoire hbase-1.4.7 a été créé, renommez le et placez le dans le répertoire : /usr/local :

```
user@user:~/Documents$ mv hbase-1.4.7 hbase
```

```
user@user:~/Documents$ sudo mv hbase /usr/local/
```

3- Configurez le PATH dans le .bashrc :

Ouvrez le fichier .bashrc (\$vim .bashrc):

```
user@user:~/Documents$ cd
```

```
user@user:~$ vim .bashrc
```

Modifiez le PATH en ajoutant les chemins suivants dans le .bashrc :

```
export HBASE_HOME=/usr/local/hbase
```

```
export PATH=$PATH:$HBASE_HOME/bin
```

Ensuite, tapez :

```
$ source .bashrc
```

4- Configuration du fichier « hbase-env.sh » :

Hbase nécessite l'installation de Java 8. Dans votre machine virtuelle VM0, il faut installer JAVA 8 et après vérifier son installation en tapant :

```
$ java -version
```

```
user@user:~$ java -version
java version "1.8.0_71"
Java(TM) SE Runtime Environment (build 1.8.0_71-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.71-b15, mixed mode)
```

Il faut maintenant configurer le JAVA_HOME dans le fichier « hbase-env.sh », pour cela tapez :

```
user@user:~$ cd /usr/local/hbase/conf/
```

```
user@user:/usr/local/hbase/conf$ vim hbase-env.sh
```

Insérer la ligne suivante dans le fichier :

```
export JAVA_HOME=/opt/java/jdk1.8.0_71/
```

```
# The java implementation to use. Java 1.8+ required.
export JAVA_HOME=/opt/java/jdk1.8.0_71/
# Extra Java CLASSPATH elements. Optional.
```

5-Modification de /etc/hosts :

Puisque dans ce TP nous allons travailler avec un seul serveur, il faut modifier le fichier /etc/hosts pour modifier l'adresse de votre serveur de 127.0.1.1 à l'adresse 127.0.0.1, comme suit :

```
user@user:~$ sudo vim /etc/hosts
```

Fichier	Édition	Affichage	Rechercher	Terminal	Aide
127.0.0.1		localhost			
127.0.0.1		user			

Pour prendre les modifications en compte, n'oubliez pas de redémarrer votre machine.

user@user:~\$ **sudo reboot**

3-Démarrage du cluster Hadoop configuré dans VM0 :

Une fois la VM démarre, tapez les commandes suivantes pour nettoyer le hadoop_store :

```
cd /usr/local/hadoop_store/
rm -rf *
mkdir -p /usr/local/hadoop_store/hdfs/namenode
mkdir -p /usr/local/hadoop_store/hdfs/datanode
chown -R user /usr/local/hadoop_store/hdfs/namenode
chown -R user /usr/local/hadoop_store/hdfs/datanode
```

Lancez le formatage du namenode et démarrez les démons de Hadoop :

```
cd /usr/local/hadoop/etc/hadoop
hdfs namenode -format
start-all.sh
```

Vérifiez que le cluster de Hadoop fonctionne bien :

Finalement, testez votre configuration en tapant : **\$ hdfs dfsadmin -report**

```
Live datanodes (1):

Name: 127.0.0.1:50010 (localhost)
Hostname: localhost
Decommission Status : Normal
Configured Capacity: 21001486336 (19.56 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 8704499712 (8.11 GB)
DFS Remaining: 11206549504 (10.44 GB)
DFS Used%: 0.00%
DFS Remaining%: 53.36%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Tue Dec 03 10:53:55 WET 2019
```

4- Configuration d'Apache Hbase pour un mode « Standalone » :

Cette section décrit la configuration d'un mode Standalone de HBase sur un seul noeud. Une instance standalone contient tous les démons HBase (HMaster et ZooKeeper) exécutés dans une seule JVM persistante sur le système de fichiers local. C'est le mode de déploiement le plus élémentaire.

Tout d'abord, il faut démarrer les démons de hadoop et s'assurer qu'ils marchent bien.

En tapant : **\$ jps** et **\$ hdfs dfsadmin -report**

Configuration de « hbase-site.xml » pour un mode « Standalone » :

Pour installer HBase en mode Standalone, ajoutez les lignes suivantes dans le « **hbase-site.xml** » :

```
user@user:/usr/local/hadoop/etc/hadoop$ cd /usr/local/hbase/conf/
```

```
user@user:/usr/local/hbase/conf$ vim hbase-site.xml
```

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///home/user/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/user/zookeeper</value>
  </property>
  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>>false</value>
  </property>
</configuration>
```


Lancement de Hbase pour un mode « Standalone » :

```
user@user:/usr/local/hbase$ ./bin/start-hbase.sh
```

```
user@user:/usr/local/hbase$ jps
2562 ResourceManager
2007 NameNode
3512 HMaster
2411 SecondaryNameNode
3950 Jps
2191 DataNode
```

```
user@user:/usr/local/hbase$ hbase shell
```

```
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.2, re6513a76c91cceda95dad7af246ac81d46fa2589, Sat Oct 19 10:10:12 UTC 2019
Took 0.0029 seconds
hbase(main):001:0>
hbase(main):002:0* status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.7736 seconds
```

Pour stopper tous les démons de Hbase, vous pouvez taper :

```
user@user:/usr/local/hbase$ ./bin/stop-hbase.sh
```

Le mode Standalone **n'est pas recommandé** pour l'installation de Hbase, nous allons donc lancer hbase en mode pseudo-distribué sur un seul nœud (voir la section suivante).

5- Configuration d'Apache Hbase en mode « Pseudo-distribué » :

Cette section décrit la configuration d'un mode « **pseudo-distribué** ». Puisque nous utilisons un seul nœud (une seule VM), hbase fonctionne alors sur un seul nœud, cela veut dire que chaque démon HBase (HMaster, HRegionServer et ZooKeeper) vont s'exécuter en tant que processus séparé dans le nœud (en utilisant le multi-threading). Si on avait la possibilité d'utiliser un cluster à plusieurs nœuds, (un serveur par exemple) on pourrait configurer Hbase en mode distribué. Pour ce TP, nous allons se contenter d'une configuration « pseudo-distribué » sur une seule VM (à cause des ressources limitées d'un ordinateur portable ordinaire).

1- Création des fichiers nécessaires pour Hbase dans le HDFS :

```
user@user:/usr/local/hbase$ hadoop fs -mkdir -p /hbase
```

```
user@user:/usr/local/hbase$
```

```
hadoop fs -mkdir -p /user/zookeeper
```

Vérifier avec la commande suivante que les répertoires sont bien créés :

```
user@user:/usr/local/hadoop/etc/hadoop$ hadoop fs -ls /
```

Pour installer HBase en mode pseudo-distribué, ajoutez les lignes suivantes dans le « **hbase-site.xml** » :

```
user@user:/usr/local/hbase$ cd conf/
```

```
user@user:/usr/local/hbase/conf$ vim hbase-site.xml
```

```
<configuration>
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:54310/hbase</value>
</property>
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>hdfs://localhost:54310/user/zookeeper</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
</configuration>
```

Remarque :

Attention, il faut utiliser dans hbase-site.xml la même adresse configurée dans le fichier de configuration de hadoop : core-site.xml. (Dans ce TP, nous avons utilisé localhost:54310 dans core-site.xml alors dans le fichier hbase-site.xml nous utilisons aussi la même adresse localhost:54310).

4- Lancement de Hbase :

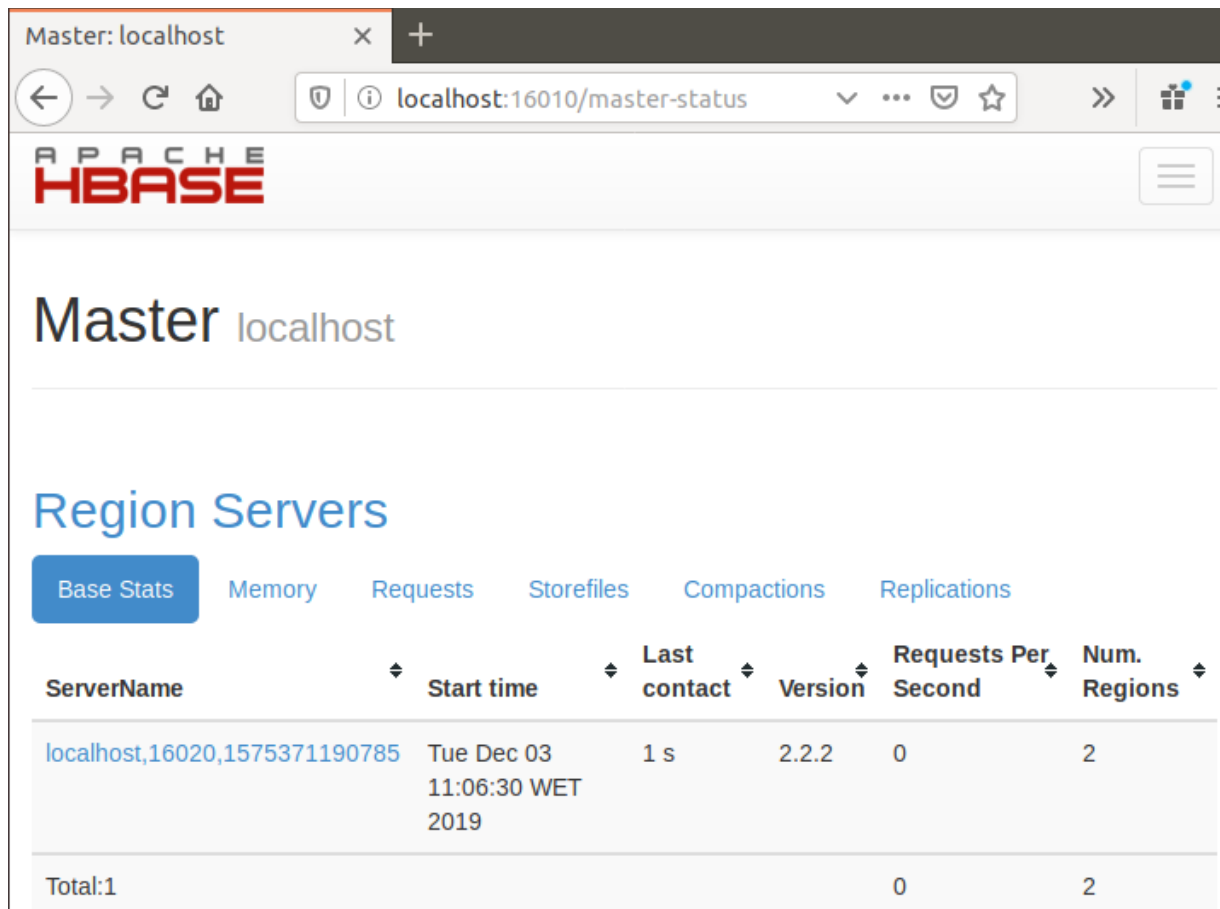
Maintenant, vous pouvez lancer Hbase en tapant :

```
user@user:/usr/local/hbase/conf$ cd ..
```

```
user@user:/usr/local/hbase$ ./bin/start-hbase.sh
```

```
user@user:/usr/local/hbase$ jps
2562 ResourceManager
4931 HMaster
5044 HRegionServer
2007 NameNode
5128 Jps
2411 SecondaryNameNode
4877 HQuorumPeer
2191 DataNode
```

Vous remarquerez que tous les démons Hadoop (NameNode, SecondaryNameNode, DataNode et ResourceManager) ainsi que les démons Hbase (HRegionServer, HMaster et HQuorumPeer (Zookeeper)) sont démarrés.



The screenshot shows the Apache HBase Master web interface for the localhost. The page title is "Master localhost". Below the title, there is a section for "Region Servers". The interface includes a navigation bar with tabs: "Base Stats" (selected), "Memory", "Requests", "Storefiles", "Compactions", and "Replications". A table displays the status of the Region Servers. The table has columns: "ServerName", "Start time", "Last contact", "Version", "Requests Per Second", and "Num. Regions". There is one server listed: "localhost,16020,1575371190785" with a start time of "Tue Dec 03 11:06:30 WET 2019", a last contact of "1 s", a version of "2.2.2", 0 requests per second, and 2 regions. A "Total" row at the bottom shows 1 server, 0 requests per second, and 2 regions.

ServerName	Start time	Last contact	Version	Requests Per Second	Num. Regions
localhost,16020,1575371190785	Tue Dec 03 11:06:30 WET 2019	1 s	2.2.2	0	2
Total:1				0	2

Testez aussi le terminal hbase :

```
user@user:/usr/local/hbase$ hbase shell
```

```
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.2, re6513a76c91cceda95dad7af246ac81d46fa2589, Sat Oct 19 10:10:12 U
TC 2019
Took 0.0141 seconds
hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.9950 seconds
hbase(main):002:0> █
```

2) Manipulation de « HBase »

1-Création d'une BD :

Tout d'abord, pour manipuler une base de données avec le shell interactif, vous devez finaliser la section précédente et ensuite lancer le hbase shell :

```
hduser@user-VirtualBox:/usr/local/hbase/bin$ hbase shell
```

Nous allons créer une base de données qui contient les données suivantes :

Row Key	client		ventes	
Row_ID	nom	ville	produit	montant
101	Mohamed A	Casablanca	Chaises	2000 MAD
102	Amine B	Salé	Lampes	300 MAD
103	Younes C	Rabat	Bureaux	10 000 MAD
104	Othman D	Tanger	Lits	15 000 MAD

1) Créez la table, ainsi que les familles de colonnes associées :

```
create 'registre_ventes','client','ventes'
```

```
hbase(main):002:0> create 'registre_ventes','client','ventes'
Created table registre_ventes
Took 1.4723 seconds
=> Hbase::Table - registre_ventes
hbase(main):003:0> █
```

2) Vérifier que la table est bien créée:

```
list
```

```
hbase(main):003:0> list
TABLE
registre_ventes
1 row(s)
Took 0.0985 seconds
=> ["registre_ventes"]
hbase(main):004:0> █
```

Insérer les différentes lignes de la table « registre_ventes »:

```
put 'registre_ventes', '101', 'client:nom', 'Mohamed A'
put 'registre_ventes', '101', 'client:ville', 'Casablanca'
put 'registre_ventes', '101', 'ventes:produit', 'Chaises'
put 'registre_ventes', '101', 'ventes:montant', '2000,00 MAD'
```

```
put 'registre_ventes', '102', 'client:nom', 'Amine B'
put 'registre_ventes', '102', 'client:ville', 'Salé'
put 'registre_ventes', '102', 'ventes:produit', 'Lampes'
put 'registre_ventes', '102', 'ventes:montant', '300,00 MAD'
```

```
put 'registre_ventes', '103', 'client:nom', 'Younes C'
put 'registre_ventes', '103', 'client:ville', 'Rabat'
put 'registre_ventes', '103', 'ventes:produit', 'Bureaux'
put 'registre_ventes', '103', 'ventes:montant', '10000,00 MAD'
```

```
put 'registre_ventes', '104', 'client:nom', 'Othman D'
put 'registre_ventes', '104', 'client:ville', 'Tanger'
put 'registre_ventes', '104', 'ventes:produit', 'Lits'
put 'registre_ventes', '104', 'ventes:montant', '15000,00 MAD'
```

3) Visualisez le résultat de l'insertion, en tapant:

```
scan 'registre_ventes'
```

ROW	COLUMN+CELL
101	column=client:nom, timestamp=1575371512528, value=Mohamed A
101	column=client:ville, timestamp=1575371512644, value=Casablanca
101	column=ventes:montant, timestamp=1575371517417, value=2000,00 MAD
101	column=ventes:produit, timestamp=1575371512744, value=Chaises
102	column=client:nom, timestamp=1575371525015, value=Amine B
102	column=client:ville, timestamp=1575371525084, value=Sal\xC3\xA9
102	column=ventes:montant, timestamp=1575371527296, value=300,00 MAD
102	column=ventes:produit, timestamp=1575371525174, value=Lampes
103	column=client:nom, timestamp=1575371536354, value=Younes C
103	column=client:ville, timestamp=1575371536441, value=Rabat
103	column=ventes:montant, timestamp=1575371537988, value=10000,00 MAD
103	column=ventes:produit, timestamp=1575371536504, value=Bureaux
104	column=client:nom, timestamp=1575371546962, value=Othman D
104	column=client:ville, timestamp=1575371547095, value=Tanger
104	column=ventes:montant, timestamp=1575371549009, value=15000,00 MAD
104	column=ventes:produit, timestamp=1575371547166, value=Lits

4 row(s)

5) Afficher les valeurs de la colonne « produit » de la ligne 102

get 'registre_ventes','102',{COLUMN => 'ventes:produit'}

```
hbase(main):022:0* get 'registre_ventes','102',{COLUMN => 'ventes:produit'}
COLUMN          CELL
ventes:produit   timestamp=1575371525174, value=Lampes
1 row(s)
Took 0.1269 seconds
```

6) Pour supprimer une table, il faut faire les étapes suivantes (disable puis drop) :

disable 'registre_ventes'

drop 'registre_ventes'

Pour vérifier que la table n'existe plus, tapez :

exists 'registre_ventes'

Pour maîtriser d'autres manipulations de la BD Hbase, vous pouvez consulter le tutoriel suivant :

<https://www.tutorialspoint.com/hbase/index.htm>

3) L'utilisation de l'API Java de HBase :

HBase fournit une API en Java pour pouvoir manipuler les données de la base. Nous allons montrer ici un exemple simple.

Pour compiler sans problèmes votre code java, il faut inclure les librairies de Hbase le classpath par défaut, grâce à la variable d'environnement \$CLASSPATH, pour cela :

Configurez le CLASSPATH, dans le .bashrc :

Ouvrez le fichier .bashrc (\$vim .bashrc):

```
user@user:~/Documents$ cd
```

```
user@user:~$ vim .bashrc
```

Modifiez le PATH en ajoutant les chemins relatifs au classpath dans le .bashrc :

```
export
```

```
CLASSPATH=/usr/local/hbase/lib/*:/usr/local/hadoop/share/hadoop/common/*
```

Ensuite, tapez : \$ source .bashrc

- Dans le répertoire de votre choix, (par exemple : /home/user/Document), créez un répertoire hbase-code, puis déplacez-vous dedans.

```
user@user:/usr/local/hbase$ cd /home/user/Documents/
```

```
user@user:~/Documents$ mkdir hbase-code
```

```
user@user:~/Documents$ cd hbase-code/
```

Créez et ouvrez le fichier **HelloHBase.java** sous le répertoire :

```
user@user:~/Documents/hbase-code$ vim HelloHBase.java
```

- Copiez le code encadré ci-dessous dans le fichier « HelloHBase.java ».

Ce code permet de réaliser les opérations suivantes:

- Créer une table appelée "**user**" contenant deux familles de colonnes: "*PersonalData*" et "*ProfessionalData*". Si cette table existe déjà, elle sera écrasée.
- Insérer deux enregistrements dans cette table.
- Lire la valeur de la colonne 'PersonalData:name' de la ligne 'user1'.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;

public class HelloHBase {
    private Table table1;
    private String tableName = "user";
    private String family1 = "PersonalData";
    private String family2 = "ProfessionalData";

    public void createHbaseTable() throws IOException {
        Configuration config = HBaseConfiguration.create();
        Connection connection = ConnectionFactory.createConnection(config);
        Admin admin = connection.getAdmin();

        HTableDescriptor ht = new HTableDescriptor(TableName.valueOf(tableName));
        ht.addFamily(new HColumnDescriptor(family1));
        ht.addFamily(new HColumnDescriptor(family2));
        System.out.println("connecting");

        System.out.println("Creating Table");
        createOrOverwrite(admin, ht);
        System.out.println("Done.....");
    }
}
```



```

    table1 = connection.getTable(TableName.valueOf(tableName));
    try {
        System.out.println("Adding user: user1");
        byte[] row1 = Bytes.toBytes("user1");
        Put p = new Put(row1);

        p.addColumn(family1.getBytes(), "name".getBytes(), Bytes.toBytes("mohamed"));
        p.addColumn(family1.getBytes(), "address".getBytes(), Bytes.toBytes("maroc"));
        p.addColumn(family2.getBytes(), "company".getBytes(), Bytes.toBytes("corp"));
        p.addColumn(family2.getBytes(), "salary".getBytes(), Bytes.toBytes("10000"));
        table1.put(p);

        System.out.println("Adding user: user2");
        byte[] row2 = Bytes.toBytes("user2");
        Put p2 = new Put(row2);
        p2.addColumn(family1.getBytes(), "name".getBytes(), Bytes.toBytes("younes"));
        p2.addColumn(family1.getBytes(), "tel".getBytes(), Bytes.toBytes("21212121"));
        p2.addColumn(family2.getBytes(), "profession".getBytes(), Bytes.toBytes("Engineer"));
        p2.addColumn(family2.getBytes(), "company".getBytes(), Bytes.toBytes("entrep"));
        table1.put(p2);

        System.out.println("reading data...");
        Get g = new Get(row1);

        Result r = table1.get(g);
        System.out.println(Bytes.toString(r.getValue(family1.getBytes(), "name".getBytes())));
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        table1.close();
        connection.close();
    }
}

```

```

public static void createOrOverwrite(Admin admin, HTableDescriptor table) throws
IOException {
    if (admin.tableExists(table.getTableName())) {
        admin.disableTable(table.getTableName());
        admin.deleteTable(table.getTableName());
    }
    admin.createTable(table);
}

public static void main(String[] args) throws IOException {
    HelloHBase admin = new HelloHBase();
    admin.createHbaseTable();
}
}

```

- Compilez et exécutez cette classe en tapant:

```
user@user:~/Documents/hbase-code$ javac HelloHBase.java
```

```
user@user:~/Documents/hbase-code$ java -cp ./usr/local/hbase/lib/* HelloHBase
```

Le paramètre -cp : définit les chemins qu'on doit ajouter dans la variable CLASSPATH (répertoire courant où se trouve HelloHbase.class et le répertoire où se trouvent les librairies de Hbase).

```

user@user:/usr/local/hbase$ cd /home/user/Documents/
user@user:~/Documents$ javac HelloHBase.java
user@user:~/Documents$ java -cp ./usr/local/hbase/lib/* HelloHBase
debut.....
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell
).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more i
nfo.
connecting
Creating Table
Done.....
Adding user: user1
Adding user: user2
reading data...
ahmed

```

Chargement de fichiers :

Il est possible de charger des fichiers volumineux dans la base HBase, à partir de HDFS. Pour cela, je vous fournis le fichier **purchases2.txt** que vous pouvez télécharger sur le lien

<https://www.dropbox.com/s/1aobaf5ibm5e7gm/purchases2.txt?dl=0>

1- Commencez par charger le fichier dans le répertoire *input* de HDFS (mais d'abord, créer ce répertoire car il n'existe pas déjà) :

```
hadoop fs -mkdir -p /input
```

```
hadoop fs -put /home/user/Documents/data_purchase/purchases2.txt /input
```

```
Hadoop fs -ls /input
```

2- Créer la base *products* avec une famille de colonnes '*cf*' :

Pour cela, accédez au terminal « hbase shell » et tapez :

```
user@user:/usr/local/hbase$ hbase shell
```

```
create 'products', 'cf'
```

```
exit
```

```
hbase(main):002:0> create 'products', 'cf'
0 row(s) in 1.5010 seconds

=> Hbase::Table - products
hbase(main):003:0> exit
```

3- Exécuter la commande suivante. **ImportTsv** est une utilité qui permet de charger des données au format tsv dans HBase. Elle permet de déclencher une opération MapReduce sur le fichier principal stocké dans HDFS, pour lire les données puis les insérer via des *put* dans la base.

```
user@user:/usr/local/hbase$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \
-Dimporttsv.separator=',' \
```

```
-Dimporttsv.columns=HBASE_ROW_KEY,cf:date,cf:time,cf:town,cf:product,cf:price,cf:payment \
products /input
```

```
2019-12-03 14:00:05,534 INFO [Thread-46] mapred.LocalJobRunner: map task executor complete.
2019-12-03 14:00:06,177 INFO [main] mapreduce.Job: map 100% reduce 0%
2019-12-03 14:00:06,178 INFO [main] mapreduce.Job: Job job_local454440959_0001 completed successfully
2019-12-03 14:00:06,660 INFO [main] mapreduce.Job: Counters: 21
    File System Counters
        FILE: Number of bytes read=60102777
        FILE: Number of bytes written=61277028
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=377535548
        HDFS: Number of bytes written=0
        HDFS: Number of read operations=7
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=0
Map-Reduce Framework
```

4-Vérifiez que la base a bien été créée en consultant la ville de l'enregistrement numéro 2000:

\$ hbase shell

get 'products','2000',{COLUMN => 'cf:town'}

```
hbase(main):001:0> get 'products','2000',{COLUMN => 'cf:town'}
COLUMN          CELL
cf:town          timestamp=1575381309034, value=Oklahoma City
1 row(s) in 0.7880 seconds

hbase(main):002:0> get 'products','2000',{COLUMN => 'cf:town'}
COLUMN          CELL
cf:town          timestamp=1575381309034, value=Oklahoma City
1 row(s) in 0.0360 seconds
```

4) Traitement de données avec Spark :

Installé sur le même cluster que HBase, Spark peut être utilisé pour réaliser des traitements complexes sur les données de HBase. Pour cela, les différents « Executors » de Spark seront co-localisés avec les region servers, et pourront réaliser des traitements parallèles directement là où les données sont stockées.

Préparation de l'environnement :

1- Téléchargez maven-3.5.0 à partir du lien :

<https://archive.apache.org/dist/maven/maven-3/3.5.0/binaries/apache-maven-3.5.0-bin.tar.gz>

Installez Apache maven :

```
$ cd /home/user/Documents/
```

```
$ tar -zxvf apache-maven-3.5.0-bin.tar.gz
```

```
$ sudo mv /home/user/Documents/apache-maven-3.5.0 /opt/
```

Pour mettre en place de manière permanente la variable d'environnement PATH pour tous les utilisateurs :

Ouvrez le fichier **/etc/profile** et modifiez le PATH en ajoutant le chemin où se trouve le bin de maven dans export PATH :

```
export PATH=$PATH:/opt/java/jdk1.8.0_71/bin:/opt/java/jdk1.8.0_71/jre/bin::/opt/apache-maven-3.5.0/bin
```

Après avoir enregistré le fichier profile, exécutez la commande source : **\$ source /etc/profile**

```
user@user:~$ source /etc/profile
user@user:~$ mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T20:39:06+01:00)
Maven home: /opt/apache-maven-3.5.0
Java version: 1.8.0_71, vendor: Oracle Corporation
Java home: /opt/java/jdk1.8.0_71/jre
Default locale: fr_FR, platform encoding: UTF-8
OS name: "linux", version: "5.0.0-37-generic", arch: "amd64", family: "unix"
user@user:~$ █
```

2 - Créez un projet Maven avec la commande suivante :

```
user@user :~$ mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1
```

Utilisez les paramètres suivants :

```
Define value for property 'groupId': Tp.myapp
Define value for property 'artifactId': myapp
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' Tp.myapp: : Tp.myapp
Confirm properties configuration:
groupId: Tp.myapp
artifactId: myapp
version: 1.0-SNAPSHOT
package: Tp.myapp
Y: : Y
```

A la fin de la génération du projet maven, un message de « Build success » s'affiche.

```
-----
[INFO] Parameter: basedir, Value: /home/user
[INFO] Parameter: package, Value: Tp.myapp
[INFO] Parameter: groupId, Value: Tp.myapp
[INFO] Parameter: artifactId, Value: myapp
[INFO] Parameter: packageName, Value: Tp.myapp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/user/myapp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:19 min
[INFO] Finished at: 2019-12-03T14:19:03Z
[INFO] Final Memory: 17M/60M
[INFO] -----
```

```
user@user:~$ cd myapp/
```

```
user@user:~/myapp$ mvn package
```

```
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar (184 kB at 209 kB/s)
[INFO] Building jar: /home/user/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 32.981 s
[INFO] Finished at: 2019-12-03T14:21:17Z
[INFO] Final Memory: 19M/60M
[INFO] -----
```

- Reconfiguration du projet Maven :

```
user@user:~$ cd myapp/
```

```
user@user:~/myapp$ vim pom.xml
```

Dans le fichier pom.xml ajouter le contenu suivant après **</properties>** :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Ajouter entre **<dependencies>** et **</dependencies>**, le contenu suivant :

```
<dependency>
  <groupId>org.apache.hbase</groupId>
  <artifactId>hbase</artifactId>
  <version>2.1.3</version>
  <type>pom</type>
</dependency>
<dependency>
  <groupId>org.apache.hbase</groupId>
  <artifactId>hbase-spark</artifactId>
  <version>2.0.0-alpha4</version>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>2.2.1</version>
</dependency>
```

Dans le répertoire : /home/user/myapp/src/main/java/DataEngineer/myapp :

- Renommer App.java en WordCountTask.java :

```
user@user:~/myapp$ cd /home/hduser/myapp/src/main/java/Tp/myapp
```

```
user@user:~/myapp/src/main/java/DataEngineer/myapp$ mv App.java  
HbaseSparkProcess.java
```

```
user@user:~/myapp/src/main/java/DataEngineer/myapp$ gedit  
HbaseSparkProcess.java
```

- Copier le contenu du traitement « WordCountTask » ci-dessous dans « HbaseSparkProcess.java » :

```
package Tp.myapp;
```

```
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.hbase.HBaseConfiguration;  
import org.apache.hadoop.hbase.client.Result;  
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;  
import org.apache.hadoop.hbase.mapreduce.TableInputFormat;  
import org.apache.spark.SparkConf;  
import org.apache.spark.api.java.JavaPairRDD;  
import org.apache.spark.api.java.JavaSparkContext;
```

```
public class HbaseSparkProcess {
```

```
    public void createHbaseTable() {  
        Configuration config = HBaseConfiguration.create();  
        SparkConf sparkConf = new  
SparkConf().setAppName("SparkHBaseTest").setMaster("local[4]");  
        JavaSparkContext jsc = new JavaSparkContext(sparkConf);  
        config.set(TableInputFormat.INPUT_TABLE, "products");  
        JavaPairRDD<ImmutableBytesWritable, Result> hBaseRDD =  
            jsc.newAPIHadoopRDD(config, TableInputFormat.class, ImmutableBytesWritable.class,  
Result.class);  
        System.out.println("nombre d'enregistrements: "+hBaseRDD.count());
```

```

    }
    public static void main(String[] args){
        HbaseSparkProcess admin = new HbaseSparkProcess();
        admin.createHbaseTable();
    }
}

```

Enregistrez **HbaseSparkProcess.java**, et lancer la commande :

```
user@user:~/myapp/src/main/java/Tp/myapp$ cd /home/user/myapp/
```

```
user@user:~/myapp$ mvn package
```

```

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myapp ---
[INFO] Building jar: /home/user/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:52 min
[INFO] Finished at: 2019-12-03T14:30:46Z
[INFO] Final Memory: 47M/114M
[INFO] -----

```

Ce code permet de lire la table products que nous avons précédemment créée, puis de créer un RDD en mémoire la représentant. Un Job spark permettra de compter le nombre d'éléments dans la base.

Faire un mvn install package sur le projet. Un fichier **myapp-1.0-SNAPSHOT.jar** sera créé dans le répertoire target du projet :

```
user@user:~/myapp$ mvn install package
```

```

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myapp ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.790 s
[INFO] Finished at: 2019-12-02T15:36:25Z
[INFO] Final Memory: 48M/115M

```


Copier le fichier **myapp-1.0-SNAPSHOT.jar** dans votre répertoire `/usr/local/spark` :

```
user@user:~/myapp/target$ cp myapp-1.0-SNAPSHOT.jar /usr/local/spark/
```

Copier tous les fichiers de la bibliothèque hbase dans le répertoire jars de spark:

```
user@user:~$ cp -r /usr/local/hbase/lib/* /usr/local/spark/jars
```

Exécuter ce fichier grâce à spark-submit comme suit :

```
user@user:/usr/local/spark$ spark-submit --class Tp.myapp.HbaseSparkProcess myapp-1.0-SNAPSHOT.jar
```

Le résultat qui devra s'afficher ressemblera au suivant:

```
nombre d'enregistrements: 4138476
19/12/03 14:43:08 INFO spark.SparkContext: Invoking stop() from shutdown hook
19/12/03 14:43:08 INFO server.AbstractConnector: Stopped Spark@7e9fad7f{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
19/12/03 14:43:08 INFO ui.SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
19/12/03 14:43:08 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/12/03 14:43:08 INFO memory.MemoryStore: MemoryStore cleared
19/12/03 14:43:08 INFO storage.BlockManager: BlockManager stopped
19/12/03 14:43:08 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
19/12/03 14:43:08 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/12/03 14:43:08 INFO spark.SparkContext: Successfully stopped SparkContext
19/12/03 14:43:08 INFO util.ShutdownHookManager: Shutdown hook called
19/12/03 14:43:08 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-46a0b0c7-948a-4d2f-bf01-38eecfff842d
19/12/03 14:43:08 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-cc4d8a10-e2a0-4225-a7b1-02961cea7441
user@user:/usr/local/spark$ █
```

Rapport du TP1 :

Rédigez un rapport détaillé contenant les captures d'écran ainsi que les explications des différentes manipulations effectuées dans ce TP .