TP4 - Stockage de Données dans une Base NOSQL avec HBase

Télécharger PDF

Objectifs du TP

Manipulation de données avec HBase, et traitement co-localisé avec Spark.

Outils et Versions

- Apache HBase Version 2.5.8
- Apache Hadoop Version: 3.3.6
- Apache Spark Version: 3.5.0
- Docker Version *latest*
- Visual Studio Code Version 1.85.1 (ou tout autre IDE de votre choix)

- Java Version 1.8.
- Unix-like ou Unix-based Systems (Divers Linux et MacOS)

Apache HBase

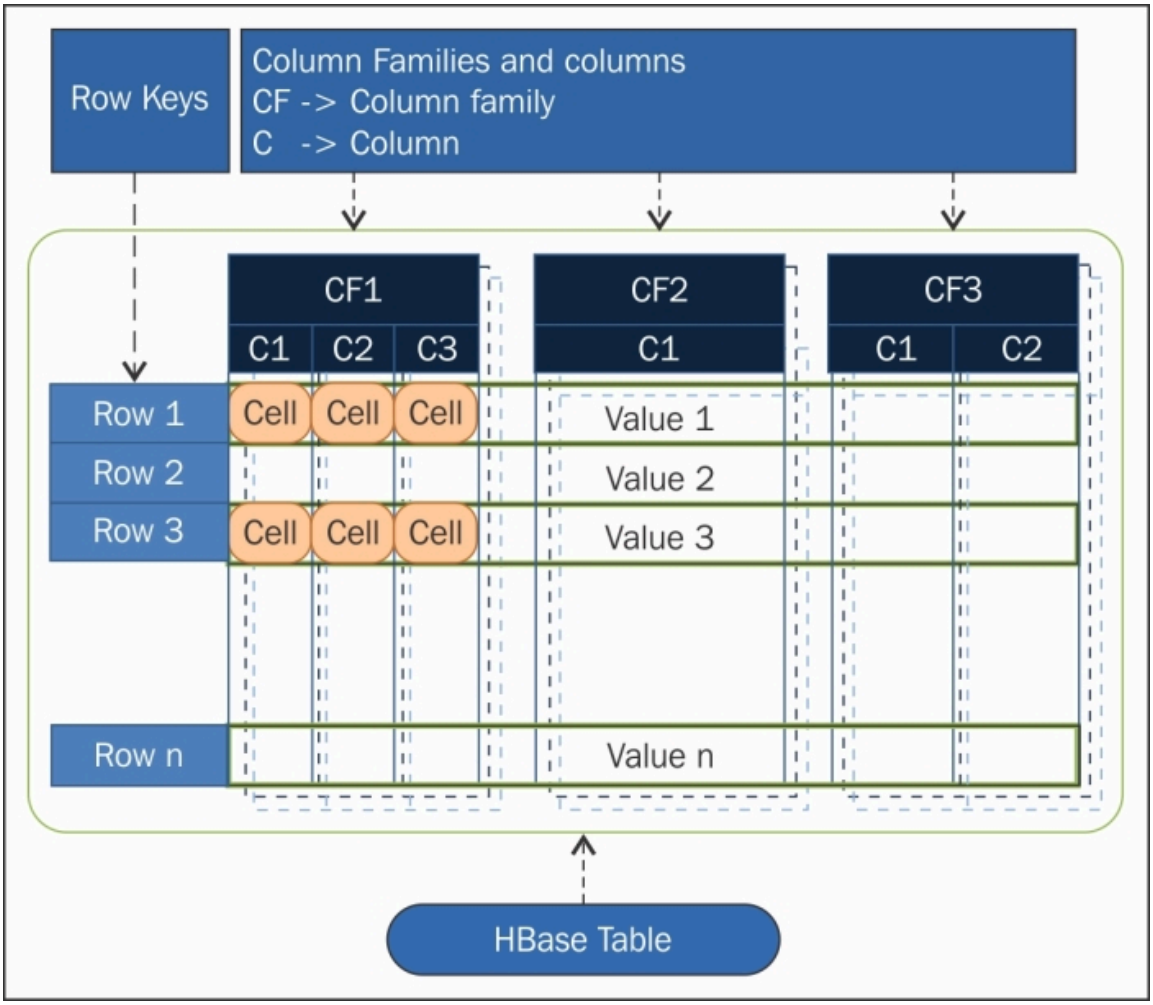
Présentation

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.

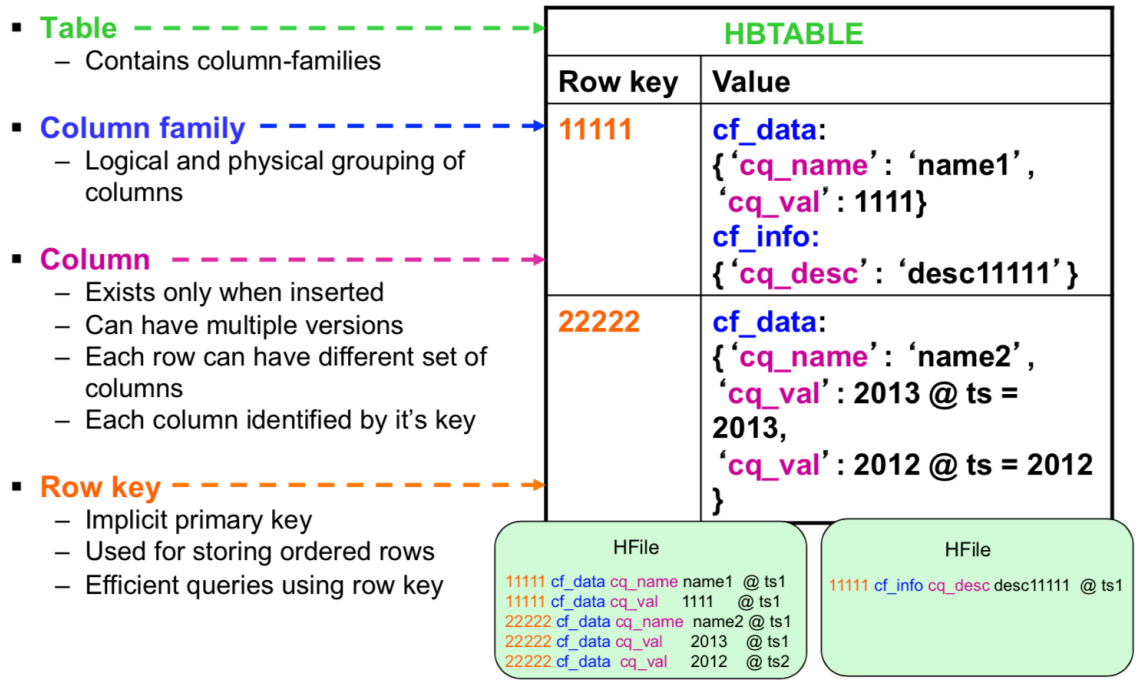
Modèle de données

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 versionnés. Les versions sont identifiés 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.



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.

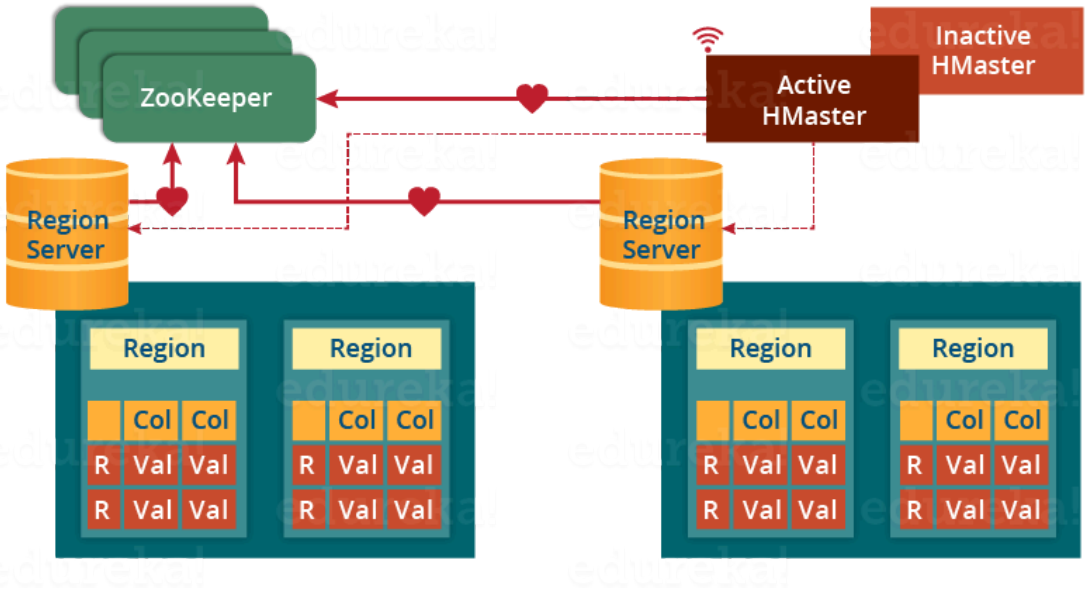
Architecture

Physiquement, HBase est composé de trois types de serveurs de type Master/Worker.

- **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.

Installation

HBase est installé sur le même cluster que précédemment. Suivre les étapes décrites dans la partie *Installation* du TP1 pour télécharger l'image et exécuter les trois contenaires. Si cela est déjà fait, il suffit de lancer vos machines grâce aux commandes suivantes:

```
docker start hadoop-master hadoop-worker1 hadoop-worker2
```

puis d'entrer dans le contenaire master:

```
docker exec -it hadoop-master bash
```

Lancer ensuite les démons yarn et hdfs:

```
./start-hadoop.sh
```

Lancer HBase en tapant :

```
start-hbase.sh
```

Une fois c'est fait, en tapant `jps`, vous devriez avoir un résultat ressemblant au suivant:

```
161 NameNode
1138 HRegionServer
499 ResourceManager
1028 HMaster
966 HQuorumPeer
1499 Jps
348 SecondaryNameNode
```

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

Première manipulation de HBase

HBase Shell

Pour manipuler votre base de données avec son shell interactif, vous devez lancer le script suivant:

```
hbase shell
```

Vous obtiendrez une interface ressemblant à la suivante:

```
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.3, r172373d1f02bbe0e3da37ec25efc97d0ec69fc96, Wed Mar 21 17:21:52 PDT 2018

hbase(main):001:0>
```

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

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

1. Commençons par créer la table, ainsi que les familles de colonnes associées:

```
create 'sales_ledger','customer','sales'
```

2. Vérifier que la table est bien créée:

```
list
```

Vous devriez obtenir le résultat suivant:

```
hbase(main):002:0> list
TABLE
sales_ledger
1 row(s) in 0.0270 seconds
```

3. Insérer les différentes lignes:

```
put 'sales_ledger','101','customer:name','John White'
put 'sales_ledger','101','customer:city','Los Angeles, CA'
put 'sales_ledger','101','sales:product','Chairs'
put 'sales_ledger','101','sales:amount','$400.00'

put 'sales_ledger','102','customer:name','Jane Brown'
put 'sales_ledger','102','customer:city','Atlanta, GA'
put 'sales_ledger','102','sales:product','Lamps'
put 'sales_ledger','102','sales:amount','$200.00'

put 'sales_ledger','103','customer:name','Bill Green'
put 'sales_ledger','103','customer:city','Pittsburgh, PA'
put 'sales_ledger','103','sales:product','Desk'
put 'sales_ledger','103','sales:amount','$500.00'

put 'sales_ledger','104','customer:name','Jack Black'
put 'sales_ledger','104','customer:city','St. Louis, MO'
put 'sales_ledger','104','sales:product','Bed'
put 'sales_ledger','104','sales:amount','$1,600.00'
```

4. Visualiser le résultat de l'insertion, en tapant:

```
scan 'sales_ledger'
```

```
hbase(main):022:0> scan 'sales_ledger'
ROW  COLUMN+CELL
 101 column=customer:city, timestamp=1523913843069, value=Los Angeles, CA
 101 column=customer:name, timestamp=1523913464518, value=John White
 101 column=sales:amount, timestamp=1523913843149, value=$400.00
 101 column=sales:product, timestamp=1523913843102, value=Chairs
 102 column=customer:city, timestamp=1523913843219, value=Atlanta, GA
 102 column=customer:name, timestamp=1523913843191, value=Jane Brown
 102 column=sales:amount, timestamp=1523913843272, value=$200.00
 102 column=sales:product, timestamp=1523913843249, value=Lamps
 103 column=customer:city, timestamp=1523913843348, value=Pittsburgh, PA
 103 column=customer:name, timestamp=1523913843308, value=Bill Green
 103 column=sales:amount, timestamp=1523913843413, value=$500.00
 103 column=sales:product, timestamp=1523913843373, value=Desk
 104 column=customer:city, timestamp=1523913843472, value=St. Louis, MO
 104 column=customer:name, timestamp=1523913843438, value=Jack Black
 104 column=sales:amount, timestamp=1523913844965, value=$1,600.00
 104 column=sales:product, timestamp=1523913843495, value=Bed
4 row(s) in 0.0360 seconds
```

5. Afficher les valeurs de la colonne *product* de la ligne *102*

```
get 'sales_ledger','102',{COLUMN => 'sales:product'}
```

Vous obtiendrez:

```
hbase(main):001:0> get 'sales_ledger','102',{COLUMN => 'sales:product'}
COLUMN  CELL
 sales:product timestamp=1523913843249, value=Lamps
1 row(s) in 0.3440 seconds
```

6. Vous pourrez quitter le shell en tapant:

```
exit
```

HBase API

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

1. Dans votre contenaire principal, créer un répertoire *hbase-code* à l'emplacement de votre choix, puis déplacez-vous dedans.

```
mkdir hbase-code
cd hbase-code
```

2. Installer Maven:

```
apt install maven -y
```

3. Créer un projet maven intitulé *hello-hbase* dans ce répertoire:

```
mvn archetype:generate -DgroupId=tn.insat.tp4 -DartifactId=hello-hbase -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Créer et ouvrir le fichier *HelloHBase.java* sous le répertoire tp4 (J'utilise `vim` ici, qui est déjà installé):

```
cd hello-hbase
vim src/main/java/tn/insat/tp4/HelloHBase.java
```

5. Insérer le code suivant dans le fichier:

```
package tn.insat.tp4;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
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();
        try (Connection connection =
ConnectionFactory.createConnection(config);
             Admin admin = connection.getAdmin()) {

            // Create table with column families
            TableDescriptor tableDescriptor =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName))
                .setColumnFamily(ColumnFamilyDescriptorBuilder.of(family1))
                .setColumnFamily(ColumnFamilyDescriptorBuilder.of(family2))
                .build();

            System.out.println("Connecting");
            System.out.println("Creating Table");
            createOrOverwrite(admin, tableDescriptor);
            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(Bytes.toBytes(family1), Bytes.toBytes("name"),
Bytes.toBytes("ahmed"));
                p.addColumn(Bytes.toBytes(family1),
Bytes.toBytes("address"), Bytes.toBytes("tunis"));
                p.addColumn(Bytes.toBytes(family2),
Bytes.toBytes("company"), Bytes.toBytes("biat"));
                p.addColumn(Bytes.toBytes(family2),
Bytes.toBytes("salary"), Bytes.toBytes("10000"));
                table1.put(p);

                System.out.println("Adding user: user2");
                byte[] row2 = Bytes.toBytes("user2");
                Put p2 = new Put(row2);
                p2.addColumn(Bytes.toBytes(family1), Bytes.toBytes("name"),
Bytes.toBytes("imen"));
                p2.addColumn(Bytes.toBytes(family1), Bytes.toBytes("tel"),
Bytes.toBytes("21212121"));
                p2.addColumn(Bytes.toBytes(family2),
Bytes.toBytes("profession"), Bytes.toBytes("educator"));
                p2.addColumn(Bytes.toBytes(family2),
Bytes.toBytes("company"), Bytes.toBytes("insat"));
                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(Bytes.toBytes(family1),
Bytes.toBytes("name"))));
            } catch (Exception e) {
                e.printStackTrace();
            }
        } finally {
            if (table1 != null) {
                table1.close();
            }
        }
    }

    public static void createOrOverwrite(Admin admin, TableDescriptor
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();
    }
}
```

Ce code Java utilise l'API Apache HBase pour interagir avec une base de données HBase :

- **Création ou mise à jour d'une table** : Il crée ou remplace une table HBase nommée user qui contient deux familles de colonnes, PersonalData et ProfessionalData.
- **Insertion de données** : Il ajoute deux enregistrements dans la table user. Le premier enregistrement pour l'utilisateur user1 inclut des informations personnelles et professionnelles comme le nom, l'adresse, la compagnie et le salaire. Le deuxième enregistrement, pour l'utilisateur user2, inclut également des informations personnelles et professionnelles comme le nom, le téléphone, la profession et la compagnie.
- **Lecture de données** : Après l'insertion, il lit et affiche la valeur de la colonne name sous la famille de colonnes PersonalData pour l'utilisateur user1.

Ce script gère également la connexion et la fermeture de la table ainsi que la gestion administrative de la base de données HBase, s'assurant que la table existe, la créant si nécessaire, et la recréant si elle existait déjà.

6. Ouvrir le fichier *pom.xml* et saisir les informations suivantes:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tn.insat</groupId>
  <artifactId>hello-hbase</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello-hbase</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>2.0.13</version>
    </dependency>

    <!-- HBase dependencies -->
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-client</artifactId>
        <version>2.5.8</version>
    <exclusions>
            <exclusion>
                    <groupId>org.apache.hadoop</groupId>
                    <artifactId>*</artifactId>
            </exclusion>
        </exclusions>
```

```
    </dependency>
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-common</artifactId>
        <version>2.5.8</version>
    <exclusions>
            <exclusion>
                    <groupId>org.apache.hadoop</groupId>
                    <artifactId>*</artifactId>
            </exclusion>
        </exclusions>

    </dependency>
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-server</artifactId>
        <version>2.5.8</version>
    <exclusions>
            <exclusion>
                    <groupId>org.apache.hadoop</groupId>
                    <artifactId>*</artifactId>
            </exclusion>
        </exclusions>

    </dependency>

    <!-- Hadoop dependencies -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.3.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-hdfs-client</artifactId>
        <version>3.3.6</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.2.0</version>
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>true</addClasspath>
```

```
                        <mainClass>tn.insat.tp4.HelloHBase</mainClass>
                    </manifest>
                </archive>
            </configuration>
    </plugin>
    <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.1.2</version>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>prepare-package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>lib</outputDirectory>
                    </configuration>
                </execution>
            </executions>
    </plugin>
 </plugins>
</build>

</project>
```

Ce fichier de configuration permet de télécharger toutes les dépendances nécessaires au projet. De plus, il définit les plugins suivants:

- Le plugin `maven-compiler-plugin` est configuré pour utiliser Java 8 (source et target 1.8).
- Le plugin `maven-jar-plugin` définit la classe principale et ajuste le chemin de classe pour l'exécution.
- Le plugin `maven-dependency-plugin` est utilisé pour copier toutes les dépendances dans un répertoire lib, facilitant l'exécution et la distribution.

7. Tout en restant sous le répertoire *hbase-code*, compiler et générer l'exécutable du projet:

```
mvn clean package
```

8. Exécuter ce code:

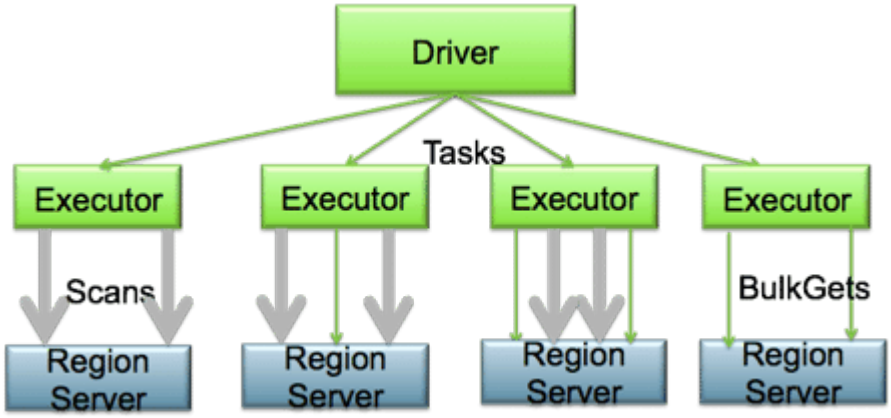
```
java -cp "target/hello-hbase-1.0-SNAPSHOT.jar:lib/*"
tn.insat.tp4.HelloHBase
```

Le résultat devrait ressembler au suivant:

```
connecting
Creating Table
Done......
Adding user: user1
Adding user: user2
reading data...
ahmed
```

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.



Nous allons réaliser un traitement simple pour montrer comment greffer spark sur HBase.

1. Créer un nouveau projet Maven.

```
mvn archetype:generate -DgroupId=tn.insat.tp4 -DartifactId=hbase-spark  -
DinteractiveMode=false
```

2. Utiliser le fichier POM suivant:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tn.insat.tp4</groupId>
  <artifactId>hbase-spark</artifactId>
  <packaging>jar</packaging>
  <version>1</version>
  <name>hbase-spark</name>
  <url>http://maven.apache.org</url>
  <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>

    <dependencies>

      <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase</artifactId>
            <version>2.5.8</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_2.13</artifactId>
            <version>3.5.0</version>
        </dependency>
    </dependencies>
</project>
```

3. Créer la classe tn.insat.tp4.HbaseSparkProcess dont le code est le suivant:

```
package tn.insat.tp4;

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,"sales_ledger");
```

```
        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();
    }

}
```

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

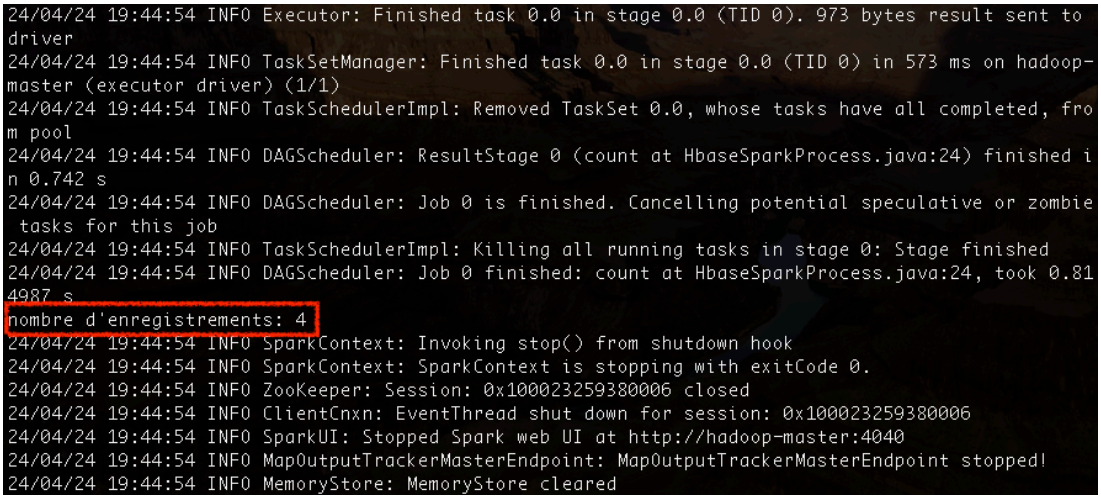
4. Faire un `mvn clean package` sur le projet. Un fichier *hbase-spark-1.jar* sera créé sous le répertoire target du projet.
5. Copier tous les fichiers de la bibliothèque hbase dans le répertoire jars de spark:

```
cp -r $HBASE_HOME/lib/* $SPARK_HOME/jars
```

6. Exécuter ce fichier grâce à *spark-submit* comme suit:

```
spark-submit  --class tn.insat.tp4.HbaseSparkProcess --master yarn --
deploy-mode client target/hbase-spark-1.jar
```

Le résultat qui devra s'afficher ressemblera au suivant:



Activité

Explorer l'utilitaire d'import de données *ImportTsv* de HBase, et donner les étapes nécessaires pour l'intégrer et le tester dans votre environnement.

Projet

Étape 4

Pour la phase finale de votre projet, intégrez une base de données NOSQL. Cela peut être HBase, ou bien toute autre base telle que MongoDB ou Cassandra. L'objectif est de stocker dans cette base le résultat des traitements réalisés précédemment. Il est ensuite demandé d'utiliser un outil de visualisation de votre choix pour afficher des courbes ou graphes.

Last update: 2024-05-19