Lab 4: HBase

L'objectif de ce TP est de :

- installation d'apache HBase
- Première utilisation de HBase
- HBase API
- Chargement de fichiers
- Traitement de données avec Spark

Apache HBase est une base de données orientée colonnes open source, distribuée, versionnée et non relationnelle, inspirée de Bigtable de Google. Apache HBase fournit des fonctionnalités de type Bigtable sur la base de Hadoop et HDFS.
Aussi, HBase fournit une API en Java pour pouvoir manipuler programmatiquement les données de la base.

1. Installation HBase

HBase est préconfiguré sur le cluster utilisé précédemment (hadoop-master, hadoop-slave1, hadoop-slave2). Si le cluster n'est plus opérationnel, Suivre les étapes décrites dans la partie Installation du TP en question pour télécharger l'image et exécuter les trois conteneurs.

- Si cela est déjà fait, lancer vos machines grâce à la commande suivante:

```
docker start hadoop-master hadoop-slave1 hadoop-slave2
```

- puis entrer dans le conteneur master:

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

- Lancer ensuite les démons yarn et hdfs:

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

- Lancer HBase en tapant :

```
/usr/local/hbase/bin/start-hbase.sh
```

- si vous ne trouver pas l'executable sur ce chemin utiliser

```
which start-hbase
```

- Une fois c'est fait, vérifier que les tous les processus sont en marche:

```
jps
```

2. Première utilisation HBase Shell

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

```
hbase shell
```

- 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

- Pour ce faire, créer la table sales_ledger, ainsi que les familles de colonnes associées customer et sales:

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

- Vérifier que la table est bien créée:

```
list
```

- 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'
```

- Visualiser le résultat de l'insertion, en tapant:

```
scan 'sales_ledger'
```

- Afficher les valeurs de la colonne *product* de la ligne *102*

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

3. Explorer les filtres dans HBase

Les filtres s'utilisent principalement avec la commande scan. Voici une liste des filtres couramment utilisés.

- **Filter – RowFilter (**Filtrer les lignes en fonction de la clé de ligne.)

Afficher uniquement les lignes où la clé est supérieure à '102' :

```
scan 'sales_ledger', {FILTER => "RowFilter(>, 'binary:102')"}
```

1. Que signifie 'binary:102' dans cette commande ?
2. Que se passe-t-il si vous remplacez > par = ? Testez-le.

- **Filter - FamilyFilter(**Filtrer par famille de colonnes)

Afficher uniquement les données de la famille customer :

```
scan 'sales_ledger', {FILTER => "FamilyFilter(=, 'binary:customer')"}
```

1. Quelles données sont affichées lorsque vous utilisez ce filtre ?

2. Comment modifier le filtre pour exclure la famille customer et afficher uniquement sales ?

- **Filter – QualifierFilter (**Filtrer par nom de colonne (qualifier))

Afficher uniquement les colonnes où le qualifier est 'amount' :

```
scan 'sales_ledger', {FILTER => "QualifierFilter(=, 'binary:amount')"}
```

1. Quelles données sont retournées par ce filtre ?
2. Testez avec un autre qualifier comme 'product'. Quel résultat obtenez-vous ?

- **Filter – ValueFilter** (Filtrer par valeur dans les cellules)

Afficher uniquement les lignes où la valeur est 'Sofa' :

```
scan 'sales_ledger', {FILTER => "ValueFilter(=, 'substring:Sofa')"}
```

1. Quelle ligne est retournée par cette commande ?
2. Que se passe-t-il si vous changez 'substring:Sofa' par 'binary:Sofa' ? Quelle est la différence ?

- **Filter – SingleColumnValueFilter** (Filtrer les lignes en fonction d'une valeur dans une colonne spécifique)

Afficher les lignes où sales:amount est égal à $450.00 :

```
scan 'sales_ledger', {FILTER => "SingleColumnValueFilter('sales', 'amount', =, 'binary:$450.00')"}
```

1. Pourquoi utilise-t-on 'sales' et 'amount' dans la commande ?
2. Testez avec une autre valeur pour la colonne 'sales:amount'. Qu'observez-vous ?

- **Filter – PrefixFilter** (Filtrer les lignes où la clé de ligne commence par un certain préfixe)

Afficher les lignes où la clé de ligne commence par '10' :

```
scan 'sales_ledger', {FILTER => "PrefixFilter('10')"}
```

1. Quelles clés de lignes sont retournées par ce filtre ?
2. Changez le préfixe en '101'. Quel résultat obtenez-vous ?

- **Filter – PageFilter** (Limiter le nombre de lignes retournées.)

Afficher uniquement 2 lignes dans la table :

```
scan 'sales_ledger', {FILTER => "PageFilter(2)"}
```

1. Quelles lignes sont retournées ?
2. Comment modifier la commande pour afficher uniquement 1 ligne ?

- quitter le shell en tapant:

```
exit
```

4. 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.

- Dans votre conteneur principal, créer un répertoire hbase-code à l'emplacement de votre choix, puis ouvrez-le

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

- Crée un sous dossier **out** pour enregistrer le fichier .class généré:

```
mkdir out
```

- Créer et ouvrir le fichier HelloHBase.java sous le répertoire *tp*:

```
vim HelloHBase.java
```

L'objectif de cette classe est de :

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

- **Taper le code suivant dans le fichier:**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
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("rabat"));
            p.addColumn(family2.getBytes(), "company".getBytes(), Bytes.toBytes("sarl1"));
            p.addColumn(family2.getBytes(), "salary".getBytes(), Bytes.toBytes("12000"));
            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("dane"));
            p2.addColumn(family1.getBytes(), "tel".getBytes(), Bytes.toBytes("21212121"));
            p2.addColumn(family2.getBytes(), "profession".getBytes(), Bytes.toBytes("developer"));
            p2.addColumn(family2.getBytes(), "company".getBytes(), Bytes.toBytes("sarl2"));
            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();
    }
}
```

- Tout en restant sous le répertoire hbase-code, compiler cette classe:

```
javac -cp "/usr/local/hbase/lib/*" -d ./out ./HelloHBase.java
```

- Exécuter la classe:

```
java -cp "./out:/usr/local/hbase/lib/*" HelloHBase
```

5. Chargement de fichiers

Il est possible de charger des fichiers volumineux dans la base HBase, à partir de HDFS. Nous supposons que nous avons un gros fichier de données de ventes contenant les données de

ventes de chaque magasin d'une chaine connu au cours d'une année donnée. Le fichier en question est nommé purchases2.txt

- Inspecter le fichier et vérifier sa structure ainsi que les information qu'il contient.
- créer ce répertoire s'il n'existe pas déjà

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

- Copier le fichier purchases2.txt dans le dossier hadoop_project de votre machine
- Charger le fichier dans le répertoire input de HDFS

```
hadoop fs -put /shared_volume/purchases2.txt input
```

- Créer la base products avec une famille de colonnes cf

```
hbase shell
```

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

ImportTsv est un utilitaire 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.

- Exécuter la commande suivante :

```
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:paym
ent \
products /user/root/input/purchases2.txt
```

- Vérifier que la base a bien été créée en consultant la ville de l'enregistrement numéro 8:

```
hbase shell
```

```
get 'products','8',{COLUMN => 'cf:town'}
```

6. Traitement de données avec Spark

On veut créer une classe java qui permet de lire la table products 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.

- Créer la classe bigdata.hbase.tp.HbaseSparkProcess :

```
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.*;

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();
    }
}
```

Ajouter un Commenter le code

Dans votre IDE, importer dans une librairie les jars nécessaires hbase, hbase-spark et spark

- Créer un jar portant le nom processing-hbase.jar
- Copier ce fichier dans votre conteneur:

```
docker cp processing-hbase.jar hadoop-master:/root/
```

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

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

- Exécuter ce fichier grâce à spark-submit

```
spark-submit  --class HbaseSparkProcess --master yarn --deploy-mode client processing-hbase.jar
```

7. Activité supplémentaire

Modifier ce code pour qu'il puisse faire la somme des ventes de tous les produits.