Pratique de Hadoop n°04

HBase (en Python)

Table des matières

I. Objectif du TP

- Après le TP02_Hadoop_HDFS et TP03_Hadoop_Map_Reduce_Python
- Manipulation de données avec HBase, et traitement avec Python

II. Apache HBase

A. Présentation

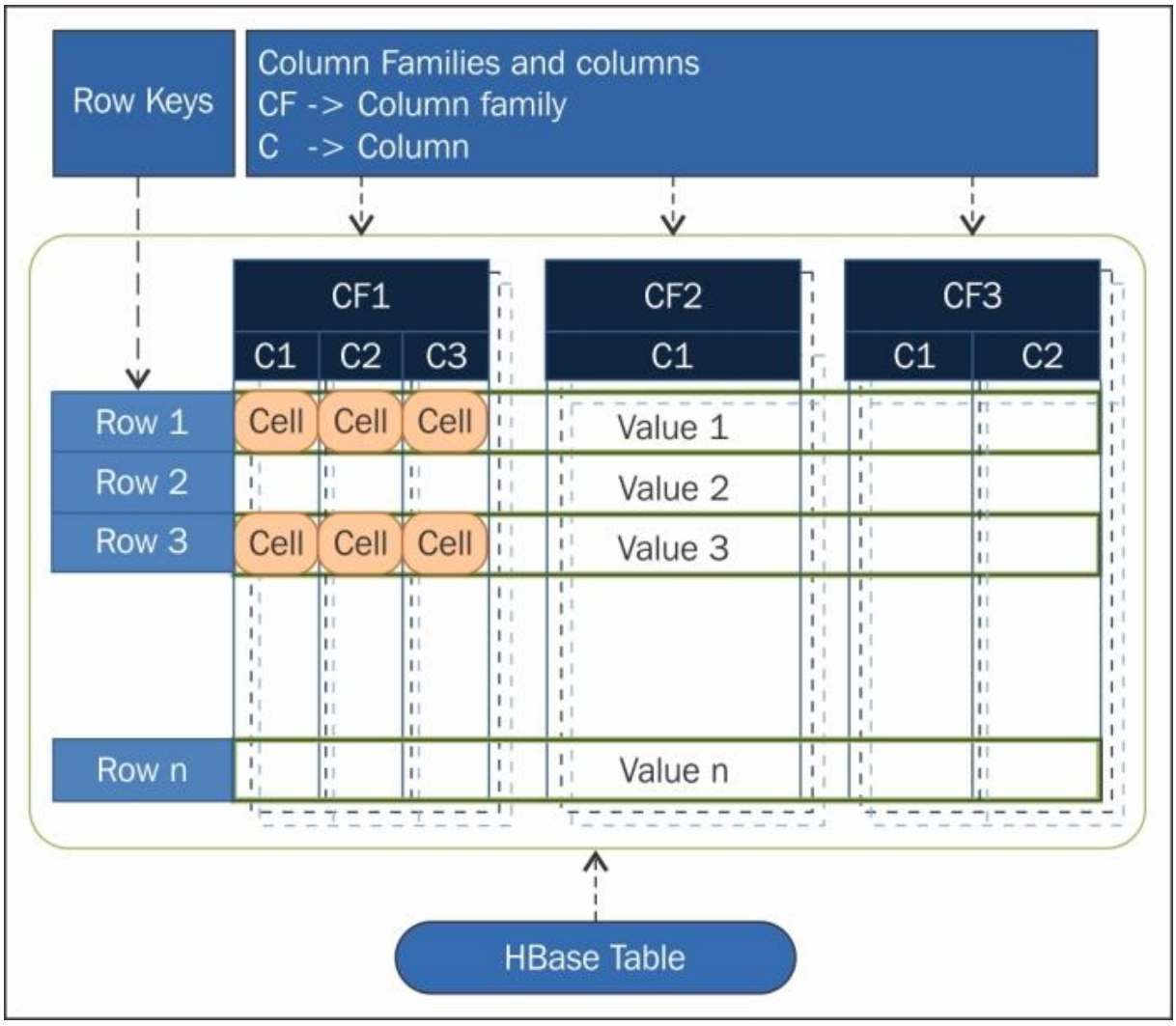
- HBase est un système de gestion de bases de données distribué, non-relationnel et orienté colonnes, développé au-dessus du système de fichier HDFS. Il permet un accès aléatoire en écriture/lecture en temps réel à un très grand ensemble de données.

B. Modèle de données

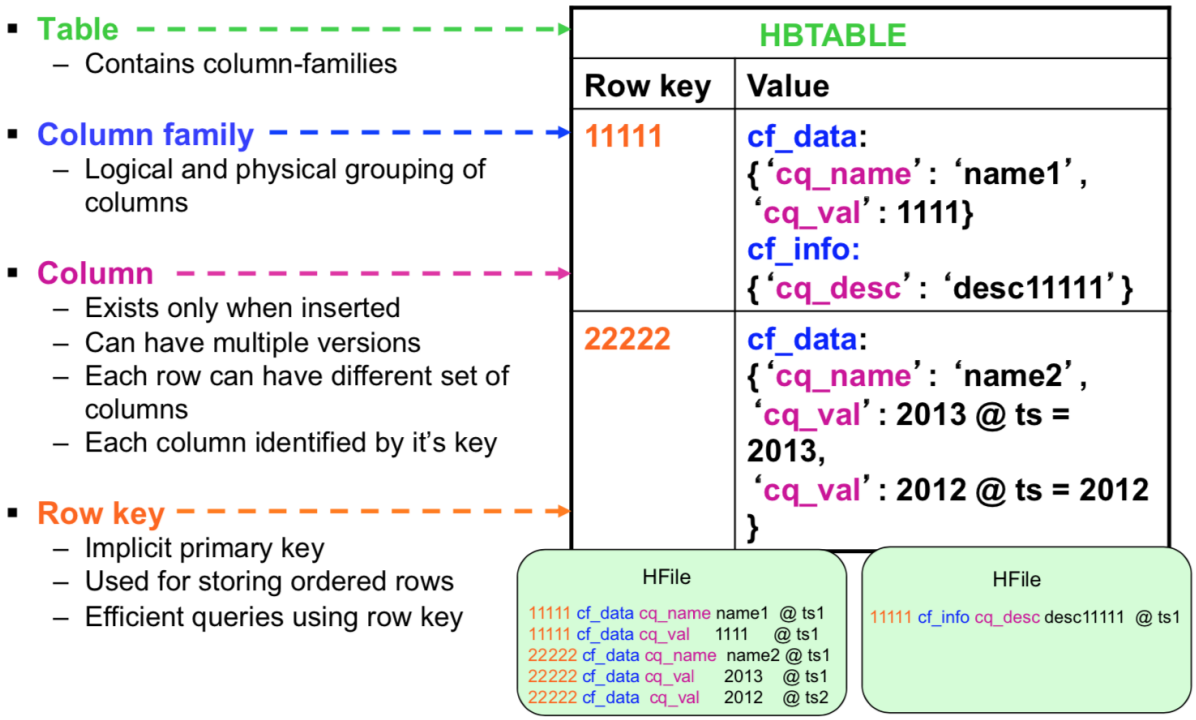
Le modèle se base sur six concepts, qui sont :

- **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.

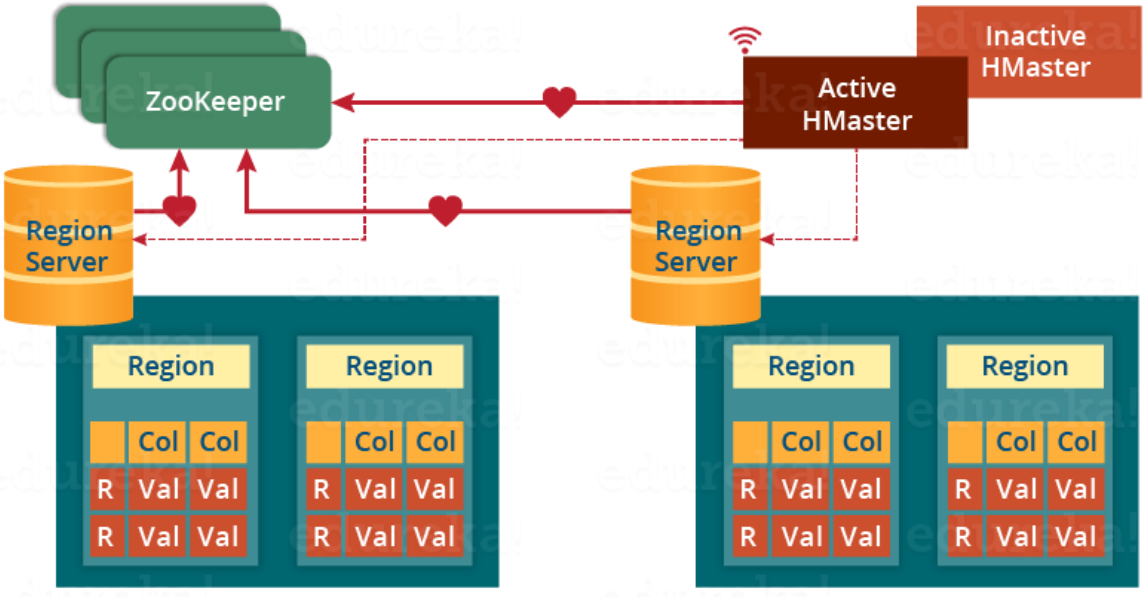
C. Architecture¶

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 nœuds 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.

III. Installation

HBase est installé sur le même cluster que précédemment. Suivre les étapes décrites dans la partie Installation du TP01_Hadoop_HDFS pour télécharger l'image et exécuter les trois conteneurs.

1. Lancer vos machines grâce aux commandes suivantes:

   docker start hadoop-master hadoop-slave1 hadoop-slave2

2. Puis d'entrer dans le conteneur master:

   docker exec -it hadoop-master bash

3. Lancer ensuite les démons yarn et hdfs:

   ./start-hadoop.sh

4. Lancer HBase en tapant :

   start-hbase.sh

5. 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

IV. Première manipulation de HBase

A. 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

V. HBase API

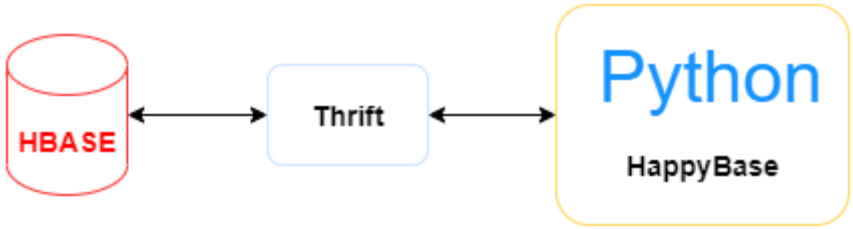
HBase fournit une API en Python : HappyBase pour pouvoir manipuler en développant les données de la base. Nous allons montrer ici un exemple très simple.

On part du principe que les services sont toujours lancés dans votre conteneur.

A. Happybase installation

Happybase est une bibliothèque Python conviviale pour les développeurs permettant d'interagir avec Apache HBase. HappyBase est conçu pour être utilisé dans configurations HBase standard et offre aux développeurs d'applications une API Python pour interagir avec HBase. En dessous de la surface, HappyBase utilise la bibliothèque Python Thrift pour se connecter à HBase à l'aide de sa passerelle Thrift.

Dans le contexte de HBase, Java est le seul langage qui peut accéder directement à HBase. L'interface Thrift agit comme un pont qui permet à d'autres langages d'accéder à HBase à l'aide d'un serveur Thrift qui interagit avec le client Java.



Pour utiliser happybase sur python il faut tout d'abord l'installer en utilisant la commande suivante : *pip install happybase*

Note :

Si la commande *pip* n'est pas installer sur votre python, lancer les commandes suivantes :

1. apt update
2. apt upgrade
3. apt install curl
4. curl https://bootstrap.pypa.io/pip/3.5/get-pip.py --output get-pip.py
5. python3 get-pip.py
6. apt install python3-pip
7. pip –version
8. pip3 --version

Pour que Thrift fonctionne, un autre démon HBase doit être en cours d'exécution pour gérer ces demandes.

1. start-hbase.sh
2. hbase-daemon.sh start thrift
3. jps

```
root@hadoop-master:~# jps
62018 HMaster
62146 HRegionServer
483 ResourceManager
4919 SecondaryNameNode
168 NameNode
61944 HQuorumPeer
62511 ThriftServer
63391 Jps
root@hadoop-master:~#
```

B. happybase avec Python

Par défaut, le service thrift écoute sur le port 9090. On va lancer une connexion vers la table déjà crée.

```
#!/usr/bin/env python

import happybase
connection = happybase.Connection('127.0.0.1',9090)
connection.open()

table=connection.table('sales_ledger')
print("La liste des tables sur HBASE est : ")
print(connection.tables())
print(" nous avons choisi la table : ")
print(table)
connection.close()
root@hadoop-master:~#
```

Résultat :

```
root@hadoop-master:~# python3 hbase1.py
La liste des tables sur HBASE est :
[b'sales_ledger']
 nous avons choisi la table :
<happybase.table.Table name=b'sales_ledger'>
```

Happybase syntaxes :

1. La création des tables :

```
connection.create_table(
    'mytable',
    {'cf1': dict(max_versions=10),
     'cf2': dict(max_versions=1, block_cache_enabled=False),
     'cf3': dict(),  # use defaults
    }
)
```

La Connection à une table :

```
table = connection.table('mytable')
```

Récupération des données

```
row = table.row('row-key')
print row['cf1:col1']   # prints the value of cf1:col1
```

```
rows = table.rows(['row-key-1', 'row-key-2'])
for key, data in rows:
    print key, data
```

```
for key, data in table.scan():
    print key, data
```

all rows from row *aaa* to the end of the table:

```
for key, data in table.scan(row_start='aaa'):
    print key, data
```

To iterate over all rows from the start of the table up to row *xyz*, use this:

```
for key, data in table.scan(row_stop='xyz'):
    print key, data
```

To iterate over all rows between row *aaa* (included) and *xyz* (not included), supply both:

```
for key, data in table.scan(row_start='aaa', row_stop='xyz'):
    print key, data
```

Stockage des données

```
table.put('row-key', {'cf:col1': 'value1',
                      'cf:col2': 'value2'})
```

```
#!/usr/bin/env python



import happybase
connection = happybase.Connection('127.0.0.1',9090)
connection.open()
print(connection.tables())
table=connection.table('sales_ledger')
rows=table.rows(['101','102'])
for key,data in rows:
    print(key,data)
connection.close()
```

Résultat :

```
root@hadoop-master:~# python3 hbase2.py
[b'sales_ledger']
b'101' {b'sales:amount': b'$400.00', b'customer:city': b'Los Angeles, CA', b'customer:name': b'John White', b'sales:product': b'Chairs'}
b'102' {b'sales:amount': b'$200.00', b'customer:city': b'Atlanta, GA', b'customer:name': b'Jane Brown', b'sales:product': b'Lamps'}
```

VI. Exercice suite au TP

Ecrire le code Python qui permet de :

2. Créer la table bibliothèque avec deux colonnes ‘auteur' (nom et prénom), 'livre' (titre, categ, date).
3. Ajouter les données suivantes :

```
put bibliotheque, 'vhugo', 'auteur:nom', 'Hugo'
put bibliotheque, 'vhugo', 'auteur:prenom', 'Victor'
put bibliotheque, 'vhugo', 'livre:titre', 'La Légende des siècles'
put bibliotheque, 'vhugo', 'livre:categ', 'Poèmes'
put bibliotheque, 'vhugo', 'livre:date', 1855
```

```
put bibliotheque, 'jverne', 'auteur:prenom', 'Jules'
put bibliotheque, 'jverne', 'auteur:nom', 'Verne'
put bibliotheque, 'jverne', 'livre:editeur', 'Hetzel'
put bibliotheque, 'jverne', 'livre:titre', 'Face au drapeau'
put bibliotheque, 'jverne', 'livre:date', 1896
```

4. Afficher toutes les données de la table.
5. Afficher la liste des livres stockés pour vhugo.