

On va découvrir la base de données Cassandra à l'aide de son shell CQL. Il ressemble énormément à SQL mais il y a quelques spécificités cruciales. Ensuite, on s'intéressera au couplage entre Cassandra et Spark.

1. Cassandra en ligne de commande

Pour tout le TP, créez un dossier TP5 et allez travailler dedans.


Et pour cette partie, créez un sous-dossier appelé CQL et descendez dedans.

1.1. Commandes de base

- Connectez-vous sur la machine hadoop par `ssh`.
- Lancez le shell de Cassandra en tapant `cqlsh master`
- Tapez la commande `help`, puis `help describe`

Dans certains cas, la commande `help` lance le navigateur vers une page d'aide HTML complète. Voici son URL : [CQL v5.0.1](#).

1.2. Création d'un keyspace

Un keyspace est un espace où on crée des tables. Vous allez chacun(e) créer le vôtre, portant le nom de votre compte : 

```
CREATE KEYSPACE $LOGNAME  
WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 2 };
```

Avant de taper entrée, modifiez le nom du keyspace avec celui de votre compte. Vous n'allez pas tous créer le même keyspace « \$LOGNAME » ? Et d'autre part, il faut que la commande soit sur une seule ligne.

Vous pouvez voir votre création par :

```
DESCRIBE KEYSPACES;
```

Les données vont être répliquées 2 fois seulement, car nous n'avons que 5 machines. Sur un cluster professionnel, la réplication pourrait être portée à 3 et avec une stratégie adaptée à sa topologie. La réplication est indépendante de HDFS, car Cassandra n'est pas sur HDFS, il/elle a ses partitions de disques durs spécifiques.


Entrez dans votre keyspace – changez le nom :

```
USE $LOGNAME;
```

Notez le changement de *prompt*.

1.3. Création d'une table

On va créer une petite table de données, y ajouter des données et les interroger. C'est juste une petite liste de livres. La table s'appelle `livres`. Ses colonnes sont `isbn` (entier), `auteur` (chaîne), `titre` (chaîne) et `annee` (entier).

En fait, ce qu'il faudrait faire, c'est écrire un petit script `MkLivres.sh` contenant toute la rafale de commandes à faire pour reconstruire la base de zéro. À vous de compléter ce squelette : 

```
#!/bin/bash

# définition de la clé primaire
PRIMARY_KEY='(titi, toto)'

# création de la table
echo "création de la table"
cqlsh -k $LOGNAME master <<data
DROP TABLE IF EXISTS livres;
CREATE TABLE livres (
    ... ,
    PRIMARY KEY $PRIMARY_KEY
);
DESCRIBE KEYSPACE;
data

# pause avant le remplissage (précaution)
sleep 2

# remplissage de la table
echo "remplissage de la table"
cqlsh -k $LOGNAME master <<data
COPY livres (...) FROM 'livres.csv' ...
data
# fin du script
```

Attention, il y a le « BOM Unicode » au début de ce fichier. Ce sont deux octets spéciaux, invisibles qui disent que c'est un fichier en Unicode UTF-8 (*byte order mark*). Il faut ouvrir ce fichier avec `geany` et décocher « écrire le BOM Unicode » puis enregistrer le fichier. Ce BOM est plus ou moins optionnel, il permet au navigateur de connaître l'encodage.

NB: la notation `<<data...data` est une redirection du script lui-même vers la commande, et elle permet de remplacer les variables bash, comme `$LOGNAME` et `$PRIMARY_KEY`. La ligne vide à la fin est nécessaire avec certains éditeurs de texte.

L'option `-k` de `cqlsh` permet de spécifier le keyspace.

N'oubliez pas de le rendre exécutable. On le lance par : `./MkLivres.sh`.

Ce script va être utile parce que nous allons tester plusieurs types de clés primaires :

- soit seulement `isbn`.
- soit le couple `auteur, isbn` dans cet ordre afin que les données soient partitionnées selon l'auteur.
- soit le couple `annee, auteur`.

Le script permet de tester ces différentes configurations assez facilement. On verra l'intérêt dans la partie *Sélections* plus loin. Dans certains cas, il faudra rajouter des index secondaires ; la directive

sera à mettre dans ce script après la création de la table, avant le DESCRIBE.

1.4. Ajout de n-uplets

Voici ce qu'il faut injecter, les isbn sont fictifs :



```
15432,Jules Verne,Le rayon vert,1882
67527,Victor Hugo,La Légende des siècles,1855
98784,Jules Verne,Face au drapeau,1896
76526,Victor Hugo,Les misérables,1862
87876,Jules Verne,L'île mystérieuse,1874
21435,Jules Verne,De la Terre à la Lune,1865
74261,Victor Hugo,Torquemada,1882
```

Enlevez son BOM Unicode s'il y en a un.

Si vous vous débrouillez bien, vous n'aurez pas à retaper ces données, seulement à les placer dans un fichier CSV pour les charger dans Cassandra. Relire le cours concernant l'injection de CSV dans des tables – ajoutez l'instruction dans le script. Sinon, n'oubliez pas de rajouter les '...' autour de toutes les chaînes.

1.5. État du cluster

Dans le terminal, en dehors de `cqlsh`, essayez les commandes :

```
nodetool info
nodetool status
```

Il y a d'autres commandes, mais destinées à l'administrateur seulement. Ne tentez pas de les lancer, car si vous cassez le cluster, plus personne ne pourra travailler.

1.6. Sélections

Créez un nouveau fichier appelé `requetes.cql` et contenant ceci, sans le BOM Unicode :



```
-- Requetes CQL pour tester la table livres
SELECT * FROM livres;
```

On le lance en tapant `cqlsh -k $LOGNAME master < requetes.cql`.

Écrivez des requêtes pour ceci, en commentant chaque fois la requête précédente (une seule non-commentée à la fois), les commentaires se font avec un `--` devant :

- afficher seulement les titres et années des livres,
- afficher les livres de Jules Verne (avertissement car cela oblige à parcourir tous les n-uplets ; il faudra créer un index secondaire, sauf dans le cas où auteur est la clé de partition, c'est à dire la première de la clé primaire, ou alors activer le filtrage),

- afficher les livres écrits entre 1850 et 1870 inclus (on doit activer le filtrage car la condition ne peut pas être indexée),
- afficher les livres écrits en 1882,
- compter les livres (il serait préférable de n'avoir à compter que les livres ayant la même clé de partition),
- afficher la moyenne des années de parution des livres de cette base (idem),
- afficher le titre du livre le plus ancien (ne marchera pas car il y a une requête imbriquée, il faudra faire en deux temps).
- afficher le nombre de livres par année (avec un `GROUP BY`)

Voilà, c'est approximativement tout ce qu'on peut faire. L'intérêt de Cassandra est de pouvoir stocker une énorme quantité de données et y accéder très rapidement avec des clés. Ses performances sont bien au delà des capacités des SGBD classiques, et sont extensibles (il suffit de rajouter des machines si c'est encore trop lent), mais les requêtes sont très limitées.

Vous voyez notamment qu'il faut bien étudier la clé primaire des données. Elle doit être conçue pour les requêtes cruciales de votre application.

Il faut noter que si les données sont volumineuses et les clés mal étudiées, Cassandra peut échouer à répondre à temps (*time-out*).

2. Injection de données existantes (optionnel)

Cette partie est optionnelle, passez à la partie suivante (pySpark sur Cassandra) si vous n'avez pas le temps.

On a vu comment placer des données dans Cassandra à l'aide d'un fichier CSV. C'est limité à de petites quantités et ne permet pas d'y placer, par exemple, les relevés météo. La technique qu'on va utiliser s'appelle le chargement en masse, *bulk loading* en anglais.

On va le faire à l'aide d'un utilitaire de Cassandra qui gère des fichiers au format interne. On aurait également pu utiliser une sorte de MapReduce mais cela fait intervenir Yarn et l'ensemble occupe trop de mémoire sur nos machines.

2.1. Chargement des arbres de Paris

Téléchargez l'archive [ArbresBulkLoad.tar.gz](#), puis déplacez-la du dossier Téléchargements vers TP5. Décompressez-la à cet endroit. Vous allez obtenir un sous-dossier ArbresBulkLoad à côté de CQL. Dans le terminal ssh, allez dans ce nouveau dossier.

Vous devez d'abord modifier les sources. Si vous connaissez `sed`, tapez :



```
sed -i -e "s/undefined/$LOGNAME/g" Makefile schema.cql src/*.java
```

Sinon, vous pouvez faire ça avec le navigateur de fichiers et éditeur de votre poste :

- `Makefile` : mettez votre nom de compte ligne 2, dans `KEYSPACE = undefined`
- `schema.cql` : remplacez partout `undefined` par votre nom de compte (2 occurrences).
- `src/InjectTableBulk.java` : mettez votre nom de compte ligne 18, `KEYSPACE = "undefined"`; et enlevez le `FIXME`

Ce logiciel se lance tout simplement en tapant `make InjectTableBulk` dans le terminal `ssh`, là où se trouve le `Makefile`. Il crée une table `arbres` contenant les arbres remarquables de Paris. Il recopie les données de HDFS vers Cassandra en créant des fichiers dans `/data/tmp/vous`, puis en les envoyant dans Cassandra avec `sstableloader`. À ce stade, vous verrez les données se propager d'une machine à l'autre.

2.2. Chargement des stations météo

Ce programme est assez facilement adaptable à d'autres données. Il suffit de deux choses :

- le schéma de la table dans le fichier `schema.cql` et dans la variable `CREATE_STMT` de `src/InjectTableBulk.java`,
- une requête d'insertion préparée `INSERT_STMT` dans le programme Java qui correspond aux instructions de lecture des champs du fichier CSV.

Il faut également disposer de la classe qui sert à lire les données, par exemple `Arbre.java`. Tout doit être lié : schémas et requête préparée. Vous remarquerez qu'il n'est pas nécessaire d'avoir une table contenant tous les champs du fichier CSV. Par exemple, on aurait pu ne pas lire les coordonnées géographiques. Il aurait suffi de les enlever du schéma, de la requête préparée et ne pas lire la colonne concernée.

On voudrait adapter ce programme pour stocker les stations météo. Copiez tout ce projet sous le nom `StationsBulkLoad`.

Ce qu'on voudrait, c'est compter les stations par pays. Pour pouvoir le faire, il faut que la clé primaire soit bien conçue pour ne pas obliger à un parcours exhaustif. Également pour d'autres exercices, il faudra mémoriser le code USAF, le nom, le pays, la latitude, la longitude et l'altitude de chaque station. Ces dernières valeurs sont de type `double`. Rajoutez également l'année de la première mesure et l'année de la dernière. Ces deux entiers sont dans des champs `date` complets, mais extrayez seulement l'année.

Vous avez déjà programmé une classe `Station.java` dans l'un des projets du TP3, ou téléchargez [Station.java](#), copiez-la à la place de `Arbre.java`. Il faut savoir que les méthodes de cette classe entraînent des exceptions, et la totalité de la ligne est ignorée. Dans le cas des arbres, les valeurs manquantes se traduisent par des `null` dans la table ; ici, on ignorera de telles lignes.

Il faut adapter `InjectTableBulk.java`. N'oubliez pas de faire appel à la classe `Station` et non pas `Arbre`, également dans le `Makefile` (variable `CHAMPS`). Le fichier HDFS à lire est `/share/noaa/isd-history.txt`. Il contient 22 lignes d'entête qu'il faudra ignorer à la lecture.

Ça sera satisfaisant si la requête `SELECT pays, count(*) AS nombre FROM stations GROUP BY pays` ; marche parfaitement.

3. pySpark sur Cassandra

Cassandra se manipule généralement avec une API : toutes les actions sont faites par des appels de procédures distantes (`RPC`) en Java ou d'autres langages. Il en existe plusieurs, mais on ne pourra pas expérimenter faute de temps. En revanche, Spark, en version Python, peut utiliser des données Spark et c'est extrêmement facile.

Créez un nouveau sous-dossier de TP5, appelez-le `Spark` puis descendez dedans. Téléchargez le

script Python [livres.py](#).

Il faut définir le keyspace dans `livres.py` : si vous avez fait toute la partie précédente (Injection de données), alors changez le keyspace "undefined" par le vôtre. Sinon, mettez "share". Les données sont déjà insérées correctement dans ce keyspace, veillez à ne pas les détruire.

Étudiez ce script : après avoir créé le contexte Spark, il ouvre une table Cassandra sous forme d'un DataFrame, c'est à dire un RDD avec des colonnes typées et nommées, ensuite il lance un calcul comme ceux que vous connaissez de Spark.

Pour le lancer (à regrouper sur une ligne) :



```
spark-submit \  
  --py-files /usr/lib/spark/jars/pyspark-cassandra-0.7.0.jar \  
  livres.py
```

À noter qu'il affiche deux avertissements, « Expected a closure » qui viennent du plugin Cassandra pour Spark.

3.1. Stations

Copiez ce script sous le nom `stations.py` et adaptez-le pour :

- Compter le nombre de stations par hémisphère.
- Calculer l'altitude moyenne des stations. Vous pourrez employer la méthode `mean()` qui calcule la moyenne d'une collection de nombres.
- Afficher les pays des stations ayant au moins 100 ans d'activité.

Ce sont les exercices du TP précédent que vous reprendrez directement. Vous constaterez que les temps de calculs sont très raisonnables.

3.2. Relevés

Des données ont été placées dans un keyspace appelé `share`. Il s'agit des relevés météo de quelques stations. Vous commencerez par regarder quel est le schéma des données de ce keyspace avec les commandes CQLSH. Faute de temps, de RAM et de puissance de calcul, vous n'aurez pas à injecter les données. En revanche, voici quelques problèmes à traiter avec Spark, dans un nouveau script appelé `releves.py`.

- Pour un petit échauffement, compter le nombre de mesures de chaque station. L'identifiant de la station est le champ `usaf` de chaque relevé.
- Calculer le nombre de stations différentes. C'est assez simple si vous extrayez seulement l'identifiant de station et que vous utilisez la méthode `distinct` [documentation](#).
- Calculer le nombre d'années de mesures des stations. Vous pouvez soit calculer le *min* et le *max* des années de chaque station, ou alors compter vraiment les années où il y a des mesures. Cette seconde approche est très élégante et plus simple que la première si vous pensez à composer des couples appropriés, puis à utiliser `distinct` puis `countByKey`. Quelle que soit l'approche, vous aurez besoin d'extraire l'année des relevés. C'est ultra simple : leur champ `datetime` possède une propriété `year`. Vous écrivez donc `releve.datetime.year`.

Voici un mini projet. Le but est de calculer une regression linéaire sur les températures en fonction

de l'année afin de déterminer s'il y a une hausse au cours du temps. C'est comme dans le TP3, il faut gérer des n-uplets (n , S_x , S_x^2 , S_y , S_y^2 , S_{xy}) et terminer par quelques calculs simples.

- Pour faire les additions des n-uplets assez facilement, il suffit d'utiliser la méthode `aggregateByKey` [documentation](#). Vous devez l'appeler sur un RDD constitué de couples (clé, n-uplet). Elle demande trois paramètres : une valeur nulle et une fonction d'addition à fournir en deuxième et troisième paramètres. Quelle est la valeur nulle des n-uplets à additionner et comment définissez-vous la fonction qui additionne deux de ces n-uplets ?
- Ce qui nous intéresse avec le résultat, c'est la pente de la droite de regression. Elle indique s'il y a une augmentation de la température avec le temps. Son calcul est simple, c'est V_{xy}/V_x , nommé pente dans le TP3. Essayez de voir si elle est généralement positive (ex: 0.01 indique une augmentation de 1° par siècle) et s'il y a des exceptions (les stations ne sont pas forcément bien réparties dans le monde).

Vous pourrez apprécier la puissance de calcul et la vitesse d'accès aux données, cent fois supérieures à celles offertes par le couple YARN et HDFS. Vous apprécierez peut-être aussi l'élégance et la puissance de la programmation fonctionnelle pour exprimer des traitements complexes.

4. Travail à rendre

Remontez au dessus du dossier TP5 et compressez-le en `.tar.gz`. Votre archive doit contenir tous les scripts CQL, les sources Java et Python que vous avez faits et modifiés. Vous pouvez supprimer les dossiers `bin` pour gagner de la place. Déposez-la sur Moodle dans la zone de dépôt prévue.