



Travaux Pratiques

**ESILV**
nicolas.travers (at) devinci.fr

# Chapitre

# Table des matières

# Chapitre 1

# Installation

## 1.1 Serveurs

Dans ce document, nous nommerons :

— XXX pour le serveur `Spark` *master* qui contrôle le cluster ;

— YYY le ou les serveurs *slave*.

Pour les instructions suivantes, si ce n'est pas précisé, elles doivent être exécutées sur le *master*.

### 1.1.1 Clé SSH

Sur le *master*, nous souhaitons faciliter la connexion avec les *slaves* en copiant la clé SSH du *master*.

```
cd ~/
ssh-keygen
#(enter x2)
ssh-copy-id administrateur@devicimongodbXXX
ssh-copy-id administrateur@devicimongodbYYY
```

### 1.1.2 Configuration du cluster `Spark`

*1.1.1* Récupérer l'adresse IP du *slave* (se connecter sur le serveur *slave*)

```
ip addr
#2: eth0.... inet 10.0.YYY.YYY
```

*1.1.2* Sur le *master*, rajouter les adresses IP à la fin du fichier de configuration des *slaves* :

```
cp /opt/spark/conf/slaves.template /opt/spark/conf/slaves
vim /opt/spark/conf/slaves

# ...
localhost
10.0.YYY.YYY
```

*1.1.3* Sur le *master*, rajouter l'adresse du *master* dans le fichier de configuration de l'environnement `Spark`

```
cp /opt/spark/conf/spark-defaults.conf.template /opt/spark/conf/spark-defaults.conf
vim  /opt/spark/conf/spark-defaults.conf
#Rajouter à la fin :
spark.master                     spark://devicimongodbXXX:7077
spark.eventLog.enabled           true
spark.serializer                 org.apache.spark.serializer.KryoSerializer
spark.driver.memory              2g
```

*1.1.4* Créer le répertoire temporaire pour spark :

```
mkdir /tmp/spark-events
```

⚠Ce répertoire est supprimée lors de la sortie de l'environnement. Il faut le réécréer à chaque redémarrage.

*1.1.5* Démarrer le cluster `Spark`

```
/opt/spark/sbin/start-all.sh
```

C'est bon, le cluster est démarré !

### 1.1.3 Suivi des logs (optionnel)

Lors du démarrage du cluster, les fichiers de logs (*master* et *slave*) sont donnés pour indication.
Vous pouvez les suivre avec les instructions suivantes (changer les XXX et YYY) :

```
tail -f /opt/spark/logs/spark-administrateur-org.apache.spark.deploy.master.Master-1-devicimongodbXXX.out
tail -f /opt/spark/logs/spark-administrateur-org.apache.spark.deploy.worker.Worker-1-devicimongodbYYY.out
```

### 1.1.4 Interface d'administration

Si vous souhaitez vérifier le bon fonctionnement du cluster, une interface Web est disponible sur le *master* avec le port 8080. Malheureusement, le cluster étant dans un réseau local, vous ne pourrez y accéder.

Pour ce faire, il faut créer un tunnel SSH entre VOTRE machine, et le *master*. Sur **VOTRE** machine avec une invite de commande, taper (en changeant les XXX) :

```
ssh -L 8080:devicimongodbXXX.westeurope.cloudapp.azure.com:8080 administrateur@
                        devicimongodbXXX.westeurope.cloudapp.azure.com
```

Votre machine va chercher le port 8080 sur le *master*. Ouvrir dans un navigateur Web : `http://localhost:8080`

## 1.2 Console de connexion à Spark

Pour interagir avec Spark , nous avons : la console (Scala, Python), Jupyter-notebook ou un driver de connexion.

### 1.2.1 Scala

Console standard de Spark avec le langage Scala pour les puristes :

```
/opt/spark/bin/spark-shell --master spark://devicimongodbXXX:7077
```

### 1.2.2 Python

Console python pour le commun des développeurs :

```
alias python=python3
/opt/spark/bin/pyspark --master spark://devicimongodbXXX:7077
```

Exécution de scripts python :

```
/opt/spark/bin/spark-submit  --verbose --master spark://devicimongodbXXX:7077
                --deploy-mode cluster mon_script_python.py
#Options supplémentaires : --class <main-class>  --conf <key>=<value>
```

### 1.2.3 Jupyter Notebook

Pour permettre à python de retrouver les librairies dans Spark, le MASTER doit initialiser les variables d'environnement :

```
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin
export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.9-src.zip:$PYTHONPATH
alias python=python3
```

On peut par ailleurs éditer le fichier "vim /.bash_profile" pour y intégrer les lignes précédentes pour qu'elles soient exécutées automatiquement à la connexion.

Sur votre MASTER, installer :

```
sudo apt-get install jupyter
```

Lancement :

```
cd ~/
jupyter-notebook
```

Comme le serveur est sur le Cloud Azure, le port 8888 utilisé par Jupyter-notebook n'est pas accessible de l'extérieur. Ainsi, pour le voir dans votre navigateur, il faut créer un tunnel SSH entre VOTRE machine, et le *jupyter-notebook* du serveur distant. Sur **VOTRE** machine avec une invite de commande, taper :

```
ssh -L 8888:localhost:8888 administrateur@devicimongodbXXX.westeurope.cloudapp.azure.com
```

Ouvrir dans un navigateur Web : `http://localhost:8888`

## 1.3 Données

Afin d'importer les données dans votre application Spark, les fichiers doivent être présents dans le répertoire "HOME" (en l'occurrence : ~/ ou */home/administrateur*) de TOUS LES SERVEURS (master + slaves).

Cette instruction est donc à exécuter sur chaque serveur :

```
cd ~/
wget https://chewbii.com/wp-content/uploads/2020/12/SparkIMDb.tar.gz
tar xzvf SparkIMDb.tar.gz
```

Dans une application classique, on n'utilise pas un fichier mais une connexion à une BDD. Ce sera fait dans le chapitre 4.

## 1.4 Futures exécutions

Une fois l'environnement démarré, les fois suivantes, il suffira de faire :

```
# Tunnel SSH pour se connecter :
ssh -L 8888:localhost:8888 administrateur@devicimongodbXXX.westeurope.cloudapp.azure.com

# Répertoire temporaire
mkdir /tmp/spark-events

# Démarrage
/opt/spark/sbin/start-all.sh

# Notebook
jupyter-notebook
```

# Chapitre 2

# Manipulation avec Python

Pour chaque application dans JupyterNotebook, il faut mettre en place l'environnement Spark (section 2.1).

## 2.1 Connexion JupyterNotebook

Pour faciliter les manipulations, nous utiliserons le JupyterNotebook que nous avons mis en place précédemment, que l'on peut consulter grâce au tunnel SSH sur le port 8888 (section 1.2.3).

### 2.1.1 Programme Spark en python

Pour tout programme, les instructions suivantes sont à intégrer au notebook :

— Importation des librairies Spark

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
```

— Variables de votre application Spark (changer le nom et le *master*)

```
app_name = "IMDb"
nb_cores = 2
parallelism = 2
memory = 3
master = "spark://devicimongodbXXX:7077"
```

— Initialisation de l'application

```
conf = SparkConf()
conf = conf.setAppName(app_name)
conf = conf.setMaster(master)
conf = conf.set("Spark.serializer", "org.apache.spark.serializer.KryoSerializer")
conf = conf.set("Spark.cores.max", "%s" %(nb_cores))
conf = conf.set("Spark.executor.memory", "%sg" %memory)
conf = conf.set("Spark.driver.memory", "%sg" %memory)
conf = conf.set("Spark.kryoserializer.buffer.max", "1024m")
conf = conf.set("Spark.driver.maxResultSize", "10g")
conf = conf.set("Spark.default.parallelism", "%s" %(nb_cores*parallelism))
sc = SparkContext(conf=conf)
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

### 2.1.2 Importation de notebook

Plusieurs notebook ont été importés en même temps que les données dans le répertoire HOME du serveur.

— Spark/SparkContext_RDD.ipynb

— Spark/SparkContext_DataFrame_SQL.ipynb

— Spark/SparkContext_MongoDB.ipynb

Chargez le premier fichier "SparkContext_RDD.ipynb" pour prendre l'environnement comme présenté précédemment.

Exécutez les instructions 1 à 3 pour lancer l'environnement.

## 2.2 Instructions simples

*2.2.1* Compter le nombre de lignes lues ;

*2.2.2* Afficher les 5 premières lignes du fichier ;

*2.2.3* Filtrer pour compter que le nombre de lignes en 1979 ;

*2.2.4* Nombre de lignes dont la colonne ”primaryTitle” contient le mot ”Star Wars” ;

*2.2.5* Compter par année le nombre de lignes contenant le mot ”Star Wars” ;

*2.2.6* Rendre persistant le DataFrame précédent (avant de calculer le group by) ;

*2.2.7* Charger le deuxième fichier ”./Spark/IMDb_title.akas_short.tsv” et créer le DataFrame ”moviesTitle” ;

*2.2.8* Joindre les deux DataFrame SW et rendre le résultat persistant ;

# Chapitre 3

# Spark SQL

## 3.1 Mise en place

Importer le notebook : `Spark/SparkContext_DataFrame_SQL.ipynb`
Une fois le fichier "IMDb_title.basics" chargé, transformez le en table relationnelle

```
movies.createOrReplaceTempView("movies")
```

Si vous avez besoin de raffraichir le contenu d'un DataFrame rendu persistant :

```
spark.sql("REFRESH TABLE movies")
```

## 3.2 Requêtes en Spark SQL

*3.2.1* Compter le nombre de tuples (en SQL) ;

*3.2.2* Trouver les tuples dont le titre contient le mot "Star Wars" ;

*3.2.3* Transformer le résultat en une nouvelle table "SW" et la rendre persistante ;

*3.2.4* Trouver les instances SW de type "movie", puis "videoGame", puis "tvSeries" ;

*3.2.5* Par `titleType` de "SW", donner le nombre d'instances ;

*3.2.6* Importer le fichier "IMDb_title.akas_short.tsv",créer une table "moviesTitle" puis joindre les deux tables dans une requête SQL (tconst = moviesTitle) et créer une table "SW_titles" ;

*3.2.7* Par valeur de "`language`" de "SW", donner le nombre d'instances ;

*3.2.8* Par valeur de "`region`" de "SW" pour ceux dont "titleType" a pour valeur 'tvSeries', donner le nombre d'instances par ordre décroissant ;

Nous pouvons nous connecter à une base de données pour éviter d'avoir à copier le fichier sur tous les serveurs. Les noeuds n'auront qu'à se connecter au serveur MongoDB pour chercher les données.

Il faut que le driver de connexion soit bien installé (section 4.1).

## 4.1 Connecteur Spark/MongoDB

Afin de mettre en place le connecteur MongoDB, il vous faut connecter les packages mongodb avec l'environnement Spark.

Pour cela, modifier le fichier de configuration de Spark :

```
vim /opt/spark/conf/spark-defaults.conf

#Rajouter les lignes suivantes (UNE SEULE ligne à chaque fois, packages uniquement séparés par des ":"):
spark.driver.extraClassPath     /home/administrateur/.ivy2/jars/org.mongodb.spark_mongo-spark-connector_2.12-3.0.0.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_bson-4.0.5.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_mongodb-driver-core-4.0.5.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_mongodb-driver-sync-4.0.5.jar

spark.executor.extraClassPath   /home/administrateur/.ivy2/jars/org.mongodb.spark_mongo-spark-connector_2.12-3.0.0.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_bson-4.0.5.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_mongodb-driver-core-4.0.5.jar:
        /home/administrateur/.ivy2/jars/org.mongodb_mongodb-driver-sync-4.0.5.jar
```

⚠L'environnement Spark / Python / MongoDB est très sensible. Il faut vérifier que CHAQUE version de package est compatible. Vous pourrez remarquer que le présent document utilise :

- Spark 3.0.0
- Connector avec Scala 2.12.3 (d'où le org.mongodb.spark_mongo-spark-connector_2.12-3.0.0.jar)
- MongoDB 4.0.5 (tous les packages mongoDB sont en 4.0.5)

Ces packages sont téléchargés automatiquement par Maven dans le répertoire "/home/administrateur/.ivy2/jars".

## 4.2 Environnement Python

Pour initialiser le *SparkContext* avec l'importation des librairies :

```python
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession

app_name = "MongoDB_app"
nb_cores = 2
parallelism = 2
memory = 2
master = "spark://devicimongodbXXX:7077"
conf = SparkConf()
conf = conf.setAppName(app_name)
conf = conf.setMaster(master)
conf = conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
conf = conf.set("spark.cores.max", "%s" %(nb_cores))
conf = conf.set("spark.executor.memory", "%sg" %memory)
conf = conf.set("spark.driver.memory", "%sg" %memory)
conf = conf.set("spark.kryoserializer.buffer.max", "1024m")
conf = conf.set("spark.driver.maxResultSize", "10g")
conf = conf.set("spark.default.parallelism", "%s" %(nb_cores*parallelism))
```

Pour préparer la connexion au serveur MongoDB, avec la base de données et la collection ciblée :

```
server = "devicimongodbXXX:27017"
db = "ottawa"
collection = "permits"
mongodb_connection = "mongodb://"+server+"/"+db+"."+collection

conf = conf.set("spark.mongodb.input.uri",mongodb_connection)
conf = conf.set("spark.mongodb.output.uri",mongodb_connection)
conf = conf.set("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.0")
```

⚠Les DataFrame de Spark font de l'inférence de schéma sur le contenu des documents produits par MongoDB. De fait, ils doivent avoir le même schéma pour pouvoir être lu dans le DataFrame. **La base "TourPedia.paris" ne peut donc être utilisée tel quel**[1]. Nous vous proposons d'importer vos données dans une base de données "ottawa" avec le dataset suivant : `https://chewbii.com/wp-content/uploads/2015/11/ottawa-json.zip` ⚠

Chargement du contexte et de la session :

```
sc = SparkContext(conf=conf)
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

## 4.3 Filtrages / Requêtes

*4.3.1* Lancer la connexion et les premières instructions de test :

```
df = spark.read.format("mongo").load()
df.count()
df.printSchema()
```

*4.3.2* Etant donné que Spark importe les données **DÉJÀ** filtrées par une requête, celle-ci doit être exécuté **DANS** le contexte et non après. Dans le cas contraire, vous perdez tous les avantages de l'utilisation d'une base de données.

Ainsi, changez le contexte et la session par le pipeline d'agrégation :

```
match = "{$match: {'road' : 'Somerset'}}"
df = spark.read.option("pipeline","[" + match + "]").format("mongo").load()
df.count()
```

*4.3.3* Afficher le nombre de "permits" par route ;

*4.3.4* Vous pouvez faire le test de performance en effecturant une requête de filtre dans MongoDB, puis de faire le même filtre sur un DataFrame contenant la collection.

1. On peut appliquer un pipeline d'agrégation pour transformer le contenu pour avoir un schéma unique