

# TP préparatoire Spark

Jonathan Lejeune



## Objectifs

Ce sujet de travaux pratiques vous indique comment installer la plate-forme Spark et vous initiera à son utilisation dans un environnement UNIX.

## Prérequis

Vous devez être familier avec l'utilisation de la plate-forme Hadoop, et en particulier HDFS et YARN. Votre machine devra avoir une installation de Hadoop version 2.7 ou supérieure afin d'assurer le bon fonctionnement de Spark. Vous devez également pouvoir vous connecter par clé ssh aux différentes machines que vous utiliserez. Si tel n'est pas le cas, reportez-vous au TP préparatoire de SSH et Hadoop. Enfin vous devez avoir un environnement de développement scala (ex : plugin Eclipse scala IDE).

## Introduction

Apache Spark est une plate-forme écrite en scala pour exécuter des applications qui traitent des données massives sur une architecture distribuée. Le cœur de Spark repose sur des structures de données appelées Resilient Distributed Dataset (RDD).

Une application spark est constituée de deux types de processus :

- les **executor** : processus qui exécutent les tâches du job
- le **driver** : processus maître exécutant la fonction main de l'application et attribuant les différentes tâches sur les executors.

L'exécution d'une application Spark implique le déploiement de plusieurs tâches sur les machines d'un cluster. Pour gérer ce déploiement, il est nécessaire d'avoir un gestionnaire de ressources distribuées. Il existe quatre plateformes existantes sur lequel on peut exécuter un application Spark :

- **Spark standalone** : un gestionnaire de ressources dédié uniquement aux déploiements d'application Spark et qui fonctionne sur un schéma maître-esclaves.
- **Apache Hadoop Yarn** : un gestionnaire de déploiement de JVM, (cf. cours sur Hadoop)

- **Apache Mesos** : un gestionnaire de déploiement de conteneurs type Docker développé par Apache (que nous n'utiliserons pas dans le cadre de ce module)
- **Kubernetes** : un autre gestionnaire de déploiement de conteneurs développé par Google. C'est un des gestionnaires les plus populaires actuellement dans le monde informatique.

Notons également qu'il existe un mode local où une application Spark s'exécutera sur une seule machine (sans gestionnaire de ressources distribuées). Dans ce mode le gestionnaire de ressource est émulé utilisant le multi-threading de la JVM permettant ainsi profiter d'un minimum de parallélisation. Ce mode reste toutefois réservé aux phases de test et de debugage. En TP la fonctionnalité de vos programmes sera testée à travers ce mode local et JUnit. Il vous est cependant possible de déployer vos programmes sur un gestionnaire Yarn ou Standalone pour le plaisir de la curiosité.

Spark reste très lié à Hadoop :

- il peut s'exécuter sur un cluster Yarn
- il nécessite un moyen de stockage persistant et fiable qui est la plupart du temps un HDFS
- l'interfaçage avec les données à traiter (découpage en splits des données d'entrée, écriture des résultats, formatage des données, etc.) utilise la même API et les mêmes classes d'implantation que celles proposées par Hadoop Map-Reduce.

## Exercice 1 – Installation de Spark

### Étape 1

Tapez la commande :

```
ssh localhost
```

Si on vous demande de taper votre mot de passe refaire le TP sur ssh

### Étape 2

Téléchargez la dernière version (prebuilt pour Hadoop) de Spark depuis le formulaire en ligne à l'URL suivante. Choisir une version de spark 3.x.y pré construit pour la version de Hadoop installée sur votre machine (normalement 3.a.b).

```
http://spark.apache.org/downloads.html
```

.

### Étape 3

Extrayez le contenu de l'archive précédemment téléchargée dans votre home avec la commande (attention aux quotas de votre compte PPTI) :

```
tar xzf spark-x.y.z-bin-hadoopx.y.tgz
```

Comme Hadoop, Spark a besoin que les variables d'environnement `JAVA_HOME` et `MY_HADOOP_HOME` (utilisées en TP Hadoop) soient définies pour indiquer le répertoire d'une jvm et du répertoire d'installation de Hadoop. Cela a du être fait lors du TP Hadoop. Vérifiez cependant en tapant :

```
echo $JAVA_HOME
echo $MY_HADOOP_HOME
```

Si rien ne s'affiche, reportez-vous à l'installation de Hadoop dans le sujet de TP dédié.

### Étape 4

Ajoutez les lignes suivantes dans votre `.bashrc` après la déclaration des variables Hadoop :

```
export MY_SPARK_HOME=<votre_repertoire_spark>
export PATH=$PATH:$MY_SPARK_HOME/bin
export PATH=$PATH:$MY_SPARK_HOME/sbin
export HADOOP_CONF_DIR="$MY_HADOOP_HOME/etc/hadoop"
```

La variable `HADOOP_CONF_DIR` permettra à Spark de connaître l'actuelle configuration de Hadoop, en particulier pour joindre le NameNode du HDFS et si besoin le ResourceManager si on souhaite utiliser Spark sur Yarn. Pour prendre en compte ces modifications dans votre terminal ouvert, tapez :

```
source ~/.bashrc
```

### Étape 5

Pour vérifier que Spark est opérationnel, tapez :

```
spark-submit --version
```

La sortie terminal de cette commande doit ressembler à ceci :

Welcome to

```

      _ _ _ _ _      _ _
     /  _  /  _ _ _ _ _ /  /  _ _
    _ \  \ /  _ \ /  _ ' /  _ /  , _ /
 / _ _ /  . _ _ \ _ , _ / _ /  / _ \ _
      /  /

```

version x.y.z

Using Scala version 2.....

Branch

Compiled by user .....

## Exercice 2 – Configuration pour Spark standalone

## Étape 1

Pour démarrer ou arrêter la plateforme Spark, les scripts `start-all.sh` et `stop-all.sh` sont fournis dans le dossier `sbin`. Cependant les noms de ces scripts rentrent en conflit dans votre `PATH` avec d'autres scripts de démarrage de Hadoop. Pour lever l'ambiguïté, nous allons juste créer deux liens symboliques dans le dossier `sbin` de Spark :

```
ln -s $MY_SPARK_HOME/sbin/start-all.sh $MY_SPARK_HOME/sbin/start-spark.sh
ln -s $MY_SPARK_HOME/sbin/stop-all.sh $MY_SPARK_HOME/sbin/stop-spark.sh
```

## Étape 2

Tapez :

```
cp $MY_SPARK_HOME/conf/spark-env.sh.template $MY_SPARK_HOME/conf/spark-env.sh
```

### Étape 3

Nous pouvons maintenant éditer le fichier `spark-env.sh` dans le répertoire `conf` en initialisant les variables d'environnement

- `SPARK_LOCAL_IP` à l'adresse IP de votre machine (vous pouvez mettre 127.0.0.1 ou bien l'adresse IP de votre machine sur le réseau)
- `SPARK_MASTER_HOST` à l'adresse IP de la machine maître du cluster Spark (idem, 127.0.0.1 ou bien l'adresse IP de votre machine sur le réseau)

```
export SPARK_LOCAL_IP="127.0.0.1"
export SPARK_MASTER_HOST="127.0.0.1"
```

### Étape 4

Le port par défaut du master est le port 7077. Le port par défaut du serveur Web est 8080. Vous pouvez éventuellement vérifier au préalable que ces ports ne sont pas déjà utilisés sur votre machine :

```
netsat -na | grep tcp | grep LISTEN
```

### Étape 5

Vous pouvez à présent démarrer Spark avec la commande :

```
start-spark.sh
```

### Étape 6

Vous pouvez tester que le déploiement des démons sur votre machine locale s'est bien passé en utilisant le script `check_start.sh` qui vous a été fourni dans les ressources de TP.

```
check_start.sh spark
```

### Étape 7

N'hésitez pas à aller voir l'interface Web du gestionnaire avec votre navigateur sur le port 8080.

### Étape 8

Pour arrêter Spark tapez la commande

```
stop-spark.sh
```

## Exercice 3 – Préparation de l'environnement Eclipse

Vous devez avoir installé le plugin `scala-ide`.

Afin d'utiliser *Eclipse* pour éditer et compiler des programmes pour Spark écrit en Scala, il faut déclarer une nouvelle librairie permettant de déclarer dans le build path d'un projet Eclipse (= le classpath du point de vue de la JVM) les différents fichiers jars de Hadoop.

### Étape 1

Pour ce faire :

- Dans la barre de menu de Eclipse cliquez sur *Window*
- Cliquez sur *Preferences*
- Déroulez l'onglet *Java* sur le panneau de gauche
- Déroulez l'onglet *Build Path*

- Cliquez sur le menu *User Libraries*
- Cliquez sur le bouton *New...*
- Nommez la librairie `spark-3.x.y` (où `x` et `y` sont à remplacer par la version que vous avez téléchargée)
- Sélectionnez la nouvelle librairie créée et cliquez sur le bouton *Add External JARs...*
- Ajoutez l'ensemble des fichiers jars qui se trouvent dans le répertoire `jars` de votre dossier d'installation de Spark.
- cliquez sur *Apply and Close* pour valider.

## Étape 2

Créez un projet scala nommé *exercices\_spark*.

## Étape 3

Ajouter la librairie créée précédemment afin de programmer avec l'API scala de Spark. Pour ce faire :

- Sélectionnez *Build Path* dans le menu contextuel (clic droit sur le dossier) du projet puis *Add Libraries...*
- Sélectionnez *User Library* puis *Next >*
- Cochez la librairie puis cliquez sur *Finish*.
- Vérifiez que la librairie a bien été ajoutée au projet dans le Package Explorer d'Eclipse.

## Étape 4

Pour Spark 3.x.y, il est nécessaire que la version de scala soit la 2.12. Pour éventuellement changer la version des librairies du plugin Eclipse scala, aller sur la fenêtre de configuration du Build Path (onglet *Libraries*) → cliquez sur *Edit* → choisissez la version de scala appropriée.

## Étape 5

Dans le package `datacloud.spark`, créez un fichier `SparkPi.scala` dont le contenu est donné ci-dessous. Assurez-vous que la compilation se passe bien.

```
package datacloud.spark
1
2
import scala.math.random
3
import org.apache.spark._
4
5
object SparkPi extends App{
6
7
    val conf = new SparkConf().setAppName("Spark Pi")
8
    val spark = new SparkContext(conf)
9
    val slices = if (args.length > 0) args(0).toInt else 5
10
    val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid overflow
11
    val count = spark.parallelize(1 until n, slices).map { i =>
12
        val x = random * 2 - 1
13
        val y = random * 2 - 1
14
        if (x*x + y*y < 1) 1 else 0
15
    }.reduce(_ + _)
16
    println("Pi is roughly " + 4.0 * count / n)
17
    spark.stop()
18
19
```

## Exercice 4 – Déploiement de jobs depuis un terminal

### Étape 1

Avant toute chose, assurez-vous que le HDFS, Spark et Yarn soient bien démarrés (utilisez le script `check_start.sh` si nécessaire).

```
check_start.sh hdfs
check_start.sh yarn
check_start.sh spark
```

### Étape 2

Ouvrez votre navigateur sur le port 8080 et notez bien l'URL complète du master Spark en haut à gauche de l'écran (qui doit ressembler à `spark://127.0.0.1:7077`).

### Étape 3

Créez un fichier jar à partir des fichiers `.class` de votre programme

```
jar cvf monProgSpark.jar -C chemin/vers/projet/eclipse/bin .
```

### Étape 4

Pour déployer un jar représentant un programme Spark vous devez utiliser la commande suivante `spark-submit`. Cette commande possède trois options principales à spécifier :

- `--master` : indique l'url du master du gestionnaire de ressources que vous souhaitez utiliser (spark, yarn ou mesos). Il est également possible de spécifier en mode local l'émulation un nombre de `n` nœuds esclaves spark (executor), avec la valeur `local[n]` (`n` est en général égal au nombre de processeurs de la machine locale)
- `--deploy-mode` : indique si vous souhaitez déployer votre programme en mode client (valeur par défaut) ou en mode cluster.
- `--class` : indique le nom de la classe qui contient le main à exécuter dans votre jar

L'option `-h` vous affiche plus d'informations sur le fonctionnement de la commande `spark-submit`.

### Étape 5

Testez le programme SparkPi en mode **client** et en local avec 4 threads et en ordonnant 5000 tâches

```
spark-submit --master local[4] --class datacloud.spark.SparkPi /chemin/vers/jar 15
```

Vous pouvez suivre l'avancement de votre job local depuis une interface web éphémère dédiée sur le port 4040 (si plusieurs job locaux en parallèle, le numéro de port de chaque job est : 4041, 4042, etc..). Vous pourrez remarquer sur les interfaces Web de Spark (port 8080) et de Yarn (port 8088) qu'aucun job/application n'a été soumis

### Étape 6

Testez le programme SparkPi en mode **client** sur le gestionnaire Spark sur l'url notée en question 2.

```
spark-submit --master spark://.... --class datacloud.spark.SparkPi /chemin/vers/jar 50
```

Vous pourrez remarquer sur l'interface Web de Spark (port 8080) qu'un job Spark est en cours d'exécution.

### Étape 7

Testez maintenant SparkPi en mode **cluster** sur Yarn.

```
spark-submit --master yarn
              --deploy-mode cluster
              --class datacloud.spark.SparkPi
              /chemin/vers/jar
              50
```

Contrôlez que le job s'est bien terminé sur l'interface Web de Yarn. La valeur de la sortie standard créé par le programme spark pourra être trouvée dans le dossier de logs de Hadoop en particulier dans la sortie standard du container qui a hébergé le appMaster.


### Étape 8

Il est vous est également possible de lancer une console interactive scala avec l'API de Spark en lançant la commande **spark-shell**. Le spark shell est une application Spark comme les autres et peut très bien être exécuté en mode local ou en mode distribué. L'interpréteur interactif fait donc office de driver et ne peut se lancer qu'en mode client.

## Exercice 5 – Déploiement de jobs depuis Eclipse

Pour éviter les manipulations fastidieuses du terminal pendant la phase de debug (commandes jar et spark-submit), il vous sera possible de lancer votre programme Spark directement depuis Eclipse.

### Étape 1

Pour le mode local, il est possible de lancer directement votre programme avec le bouton classique de lancement de programme (bouton *Run As...* ). Pour ce faire il faut spécifier le nom du master du gestionnaire dans l'objet SparkConf, via la méthode **setMaster**. il faudra donc écrire

```
val conf = new SparkConf().setAppName("Mon app Spark").setMaster("local[n]")
```

où **n** est a remplacer par le nombre de thread que vous souhaitez considérer (la valeur \* prendra la valeur du nombre de cœurs de calcul de la machine).

### Étape 2

Pour l'utilisation d'un véritable gestionnaire de ressources il est nécessaire d'éditer un script bash qui automatise la création du jar et la soumission du job. Il sera par la suite possible d'exécuter ce script et de le paramétrer à partir d'Eclipse. Pour cela suivre la procédure suivante :

- Créer un script bash dans votre projet : Clic droit sur le projet → *New* → *File*. Nommons-le "*sparkjoblauncher.sh*". Pour une exécution sur Spark Standalone (en mode local sur le loopback), le contenu du fichier sera :

```
#!/bin/bash

if [ $# -lt 2 ]
then
    echo "usage : <java_project_root> <main_class> <arg1> <arg2> ..."
    exit 1
```

```

fi

project_root=$1
main_class=$2
file_jar=/tmp/spark_prog.jar


shift 2

jar cvf $file_jar -C $project_root .
spark-submit --master spark://127.0.0.1:7077 --class $main_class $file_jar $@

```

- Donner les droits en exécution à ce fichier. Sous Eclipse il est possible de le faire en faisant un clic droit sur le fichier → *Properties*. Dans le menu *Resource*, cocher les droits en exécution pour le propriétaire. Cliquer sur le bouton *Apply and Close*.
- Aller dans le menu *Run* → *External Tools* → *External Tools Configurations...*
- Cliquer sur l'icône *New launch configuration*
- Dans le champ *Name* renseigner le nom de votre lanceur (par exemple *Spark Standalone*)
- Dans le champ *Location*, renseigner le chemin du fichier du script bash.
- Dans le champs arguments les deux premiers arguments seront  
`${workspace_loc}/${project_path}/bin`  
`${java_type_name}`

le reste des arguments dépendra du programme spark que vous souhaitez exécuter. Dans l'exemple de **SparkPi**, on pourra renseigner un troisième argument entier.

- Pour exécuter le script, sélectionner dans le package explorer de Eclipse le fichier *main* de l'application scala Spark puis cliquer sur la petite flèche noire à côté de l'icône Run External Tool (le deuxième à droite en partant du bouton d'exécution habituelle )). Sélectionner le lanceur précédemment créé.