

LabWork 4

Installation et Configuration Spark

1. Objectifs

L'objectif de cet atelier est d'installer et de configurer Spark sur une machine Windows. Après configuration et test, on implémentera l'exemple du compteur de mots en Java vu en LabWork2 mais cette fois en utilisant le langage Python. La dernière partie concernera le même exercice de l'analyse des données météorologiques.

2. Environnement

L'environnement d'installation nécessite un système Windows avec Java installé (Java version 8 ou ultérieure).

Puisque l'installation se fera sur une machine virtualisée Linux, on aura besoin de réaliser les téléchargements suivants :


Afin de vérifier le succès de l'installation : **java -version**

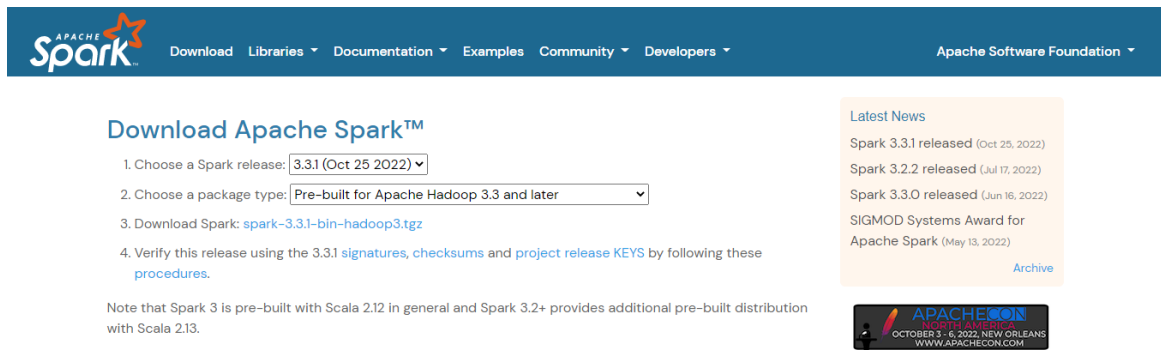
Il faudra également installer une version stable de Python depuis le site : <https://www.python.org/downloads/>

Vérifier que Python est bien installé en tapant la commande '**py**' dans l'invite de commandes.

3. Installation

Apache Spark est livré dans un fichier tar/zip compressé, donc l'installation sur Windows n'est pas vraiment un problème car il suffit de télécharger et de décompresser le fichier. Téléchargez Apache Spark en accédant à la page de téléchargement de Spark. La version actuelle est la version 3.

 spark.apache.org/downloads.html



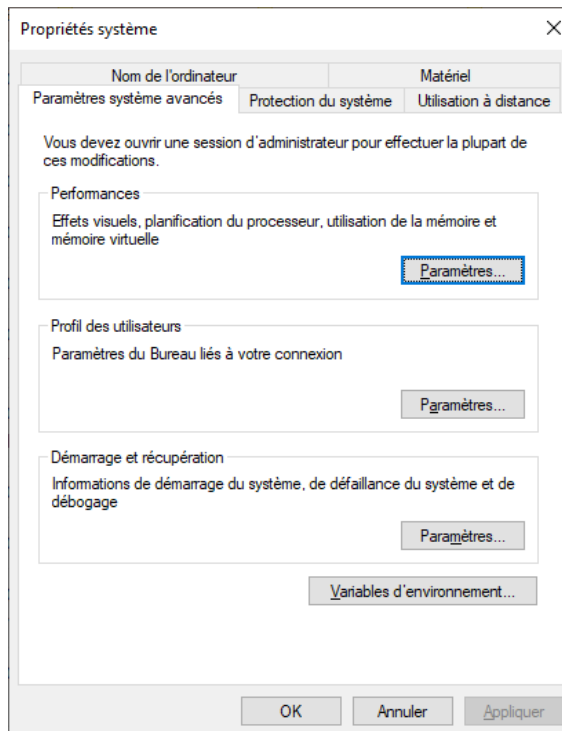
Après le téléchargement, décompresser le fichier à l'aide d'un utilitaire zip pour extraire le fichier zip et copier le répertoire extrait vers `c:\apps\opt\spark-3.3.1-bin-hadoop3`

4. Configuration

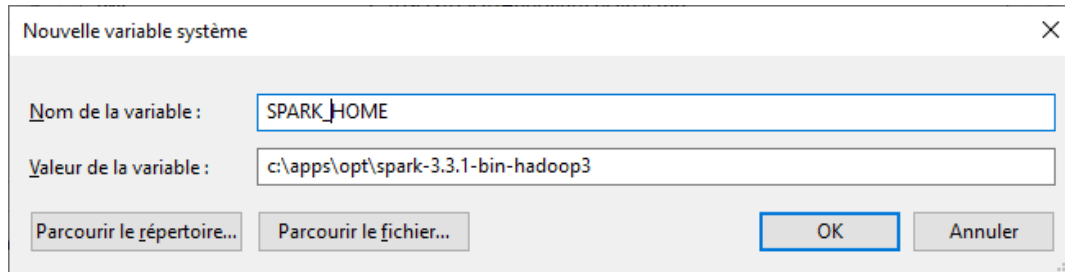
4.1. Variables d'environnement

Après l'installation de Java et Apache Spark sur Windows, on définit les variables d'environnement JAVA_HOME, SPARK_HOME, HADOOP_HOME et PATH.

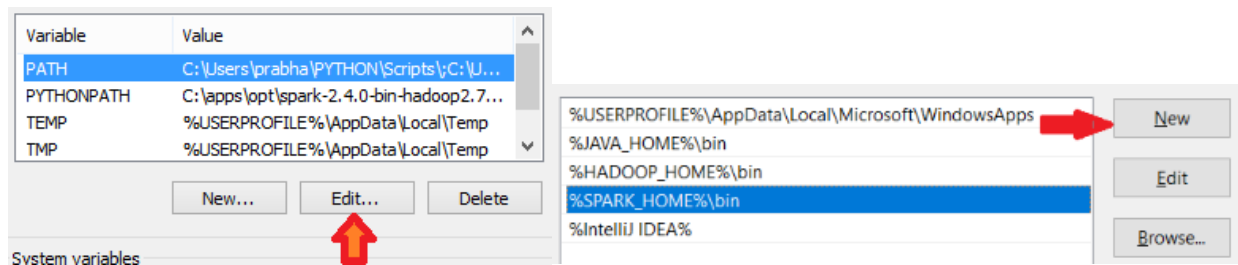
1. Ouvrir la fenêtre Variables d'environnement système et sélectionner Variables d'environnement.



- Sur l' cran Variable d'environnement, ajouter **SPARK_HOME**, **HADOOP_HOME** (la m me que celle de SPARK_HOME), **JAVA_HOME** en s lectionnant l'option Nouveau.



- Modifier maintenant la variable PATH, et ajouter l'emplacement de Spark, Java et Hadoop en s lectionnant l'option Nouveau.



4.2. Spark avec WinUtils.exe

Nombreux pensent qu'Apache Spark a besoin d'un cluster Hadoop install  pour fonctionner, mais ce n'est pas vrai, Spark peut fonctionner sur AWS en utilisant S3, Azure en utilisant le stockage blob sans Hadoop et HDFS etc.

Pour ex cuter Apache Spark sur Windows, on a besoin de winutils.exe car il utilise POSIX comme les op rations d'acc s aux fichiers dans Windows   l'aide de l'API Windows.

winutils.exe permet   Spark d'utiliser des services sp cifiques   Windows, notamment l'ex cution de commandes shell dans un environnement Windows.

On t l charge winutils.exe pour Hadoop3 et on le copie dans le dossier %SPARK_HOME%\bin. Les Winutils sont diff rents pour chaque version de Hadoop, on doit donc t l charger la bonne version en fonction de la distribution Spark vs Hadoop : <https://github.com/steveloughran/winutils>

Ajouter  galement le fichier hadoop.dll dans le m me emplacement (%SPARK_HOME%\bin).

4.3. Apache Spark shell

spark-shell est un utilitaire CLI fourni avec la distribution Apache Spark. On ouvre l'invite de commande, on accède à %SPARK_HOME%/bin et on tape la commande spark-shell pour exécuter le shell Apache Spark. On doit voir quelque chose comme ci-dessous.

```

c:\>cd %SPARK_HOME%\bin
C:\apps\opt\spark-3.0.0-bin-hadoop2.7\bin>spark-shell
20/11/25 16:38:17 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/11/25 16:38:25 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Spark context Web UI available at http://DELL-ESUHA02K0J:4041
Spark context available as 'sc' (master = local[*], app id = local-1606351105374).
Spark session available as 'spark'.
Welcome to

  ____              __
 / _  \            /  |
| |_) |          / ___|
| |_) |         /___ \
|  __/         /___ \
 \_\_\_        /___ \
                |___|

version 3.0.0

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

```

Spark-shell crée également une interface utilisateur Web contextuelle Spark et, par défaut, il peut accéder à partir de <http://localhost:4040>.

Sur la ligne de commande spark-shell, on peut exécuter toutes les instructions Spark telles que la création d'un RDD, l'obtention de la version Spark, etc.

5. Exemple WordCount

Afin de tester l'exemple du compteur de mots, on doit faire en sorte de suivre les étapes suivantes :

1. **Préparer le fichier de données** (le fichier input.txt est fourni en ressources à cet atelier) : ce fichier contient un poème en français.

Sur le shell de spark, nous devons d'abord créer la variable et donner le chemin à notre fichier WordCount :

```
val text = sc.textFile("C:/input.txt")
```

```

scala> val text = sc.textFile("C:/input.txt")
text: org.apache.spark.rdd.RDD[String] = C:/input.txt MapPartitionsRDD[1] at textFile at <console>:23

scala> text.collect
res0: Array[String] = Array(O mon jardin d'eau fra`che et d'ombre, Ma danse d'Ûtre mon c?ur sombre, Mon
ciel des Ûtoiles sans nombre, Ma barque au loin douce Ó ramer, Heureux celui qui devient sourd, Au cha
nt s'il n'est de son amour, Aveugle au jour d'aprs son jour, Ses yeux sur toi seule ferms, Heureux ce
lui qui meurt d'aimer, Heureux celui qui meurt d'aimer, D'aimer si fort ses lvres closes, Qu'il n'ait
besoin de nulle chose, Hormis le souvenir des roses, A jamais de toi parfumes, Celui qui meurt mme Ó
douleur, A qui sans toi le monde est leurre, Et n'en retient que tes couleurs, Il lui suffit qu'il t'ai
t nomme, Heureux celui qui meurt d'aimer, Heureux celui qui meurt d'aimer, Mon enfant dit-il ma chre
me, Le temps de te conna`tre ¶ femme, L'Ûternit n'est qu'une pme, Au feu ...

```

2. Exécuter le traitement MapReduce pour compter le nombre d'occurrences de chaque mot dans le poème.

Pour ceci, on définit d'abord le séparateur entre les différents mots (ici c'est l'espace)

```
val counts= text.flatMap(line => line.split(" "))
```

```
scala> val counts= text.flatMap(line => line.split(" "))
counts: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> counts.collect
res1: Array[String] = Array(0, mon, jardin, d'eau, fra`che, et, d'ombre, Ma, danse, d'Ûtre, mon, c?ur,
sombre, Mon, ciel, des, Ûtoiles, sans, nombre, Ma, barque, au, loin, douce, Ó, ramer, Heureux, celui, q
ui, devient, sourd, Au, chant, s'il, n'est, de, son, amour, Aveugle, au, jour, d'aprPs, son, jour, Ses,
yeux, sur, toi, seule, fermÛs, Heureux, celui, qui, meurt, d'aimer, Heureux, celui, qui, meurt, d'aime
r, D'aimer, si, fort, ses, l?vres, closes, Qu'il, n'ait, besoin, de, nulle, chose, Hormis, le, souvenir
, des, roses, A, jamais, de, toi, parfumÛes, Celui, qui, meurt, mÛme, Ó, douleur, A, qui, sans, toi, le
, monde, est, leurre, Et, n'en, retient, que, tes, couleurs, Il, lui, suffit, qu'il, t'ait, nommÛe, Heu
reux, celui, qui, meurt, d'aimer, Heureux, celui, qui, meurt, d'aimer, Mon, ...
```

Par la suite, on utilise cette commande pour ajouter les clés des mêmes mots

```
val mapf=counts.map(word=>(word,1))
val reducef=mapf.reduceByKey(_+_)
```

```
scala> val mapf=counts.map(word=>(word,1))
mapf: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:23

scala> mapf.collect
res2: Array[(String, Int)] = Array((0,1), (mon,1), (jardin,1), (d'eau,1), (fra`che,1), (et,1), (d'ombre
,1), (Ma,1), (danse,1), (d'Ûtre,1), (mon,1), (c?ur,1), (sombre,1), (Mon,1), (ciel,1), (des,1), (Ûtoiles
,1), (sans,1), (nombre,1), (Ma,1), (barque,1), (au,1), (loin,1), (douce,1), (Ó,1), (ramer,1), (Heureux,
1), (celui,1), (qui,1), (devient,1), (sourd,1), (Au,1), (chant,1), (s'il,1), (n'est,1), (de,1), (son,1)
, (amour,1), (Aveugle,1), (au,1), (jour,1), (d'aprPs,1), (son,1), (jour,1), (Ses,1), (yeux,1), (sur,1),
(toi,1), (seule,1), (fermÛs,1), (Heureux,1), (celui,1), (qui,1), (meurt,1), (d'aimer,1), (Heureux,1),
(celui,1), (qui,1), (meurt,1), (d'aimer,1), (D'aimer,1), (si,1), (fort,1), (ses,1), (l?vres,1), (closes
,1), (Qu'il,1), (n'ait,1), (besoin,1), (de,1), (nulle,1), (chose,1), (Hormis...
```

3. Analyser les fichiers résultats.

On utilise cette commande pour enregistrer les fichiers de sortie dans le lecteur C par exemple (on peut spécifier n'importe quel autre emplacement)

```
reducef.saveAsTextFile("C:/output")
```

Aller à l'emplacement spécifié et ouvrir les fichiers part-0000 et part-0001 générés.

Remarque: la méthode 'collect' permet à chaque fois de visualiser le contenu des variables. Par exemple ; **text.collect** permet d'afficher le contenu du fichier de données spécifié en entrée.

6. Analyse des données météorologiques

Implémentez la même application que dans le TP précédent (LabWork2), mais avec Spark en utilisant les mêmes fichiers de données et en répondant aux mêmes requêtes demandées.