TP1: Installation de Spark sous Windows

- ❑ Installation et configuration Spark en local sous Windows
- ❑ Le shell Spark: Spark-shell et PySpark
- ❑ Exécution d'un premier exemple (WordCount)
- ❑ Interface Web de supervision d'application
- ❑ Déployer une application avec Spark-submit.
- ❑ Le package pyspark

TP1: Installation Spark en local sous Windows (1/2)

1. Installer Java

https://www.oracle.com/in/java/technologies/downloads/#jdk19-windows

2. Installer Python (3.11 ou plus)

https://www.python.org/downloads/

- Ajouter ensuite les package **psutil**, **py4j**, **setuptools** et **pyspark** avec les commandes:

pip install psutils

pip install py4j

pip install setuptools

pip install pyspark

3. Télécharger Spark 3.5 ou plus

https://spark.apache.org/downloads.html

N.B: Choisissez "**Pre-built for Apache Hadoop 3.3 and later**"

- Vous aurez, par exemple, le fichier compressé **spark-3.5.0-bin-hadoop3.tgz**

4. Décompressez ce fichier dans le dossier **C:\Spark** à créer

TP1: Installation Spark en local sous Windows (2/2)

5. Télécharger **Winutils**:

- Télécharger le dossier https://github.com/steveloughran/winutils/tree/master/hadoop-3.0.0/bin
- Copier le dossier **bin** dans le dossier **C:\hadoop** à créer
- Copier **hadoop.dll** dans **C:\windows\system32**

6. Créer les variables d'environnement:

A travers l'interface graphique ou avec la commande
setx /M par exemple:
setx /M HADOOP_HOME=C:\hadoop
setx /M Path="%path%;%SPARK_HOME%\bin"

HADOOP_HOME = C:\hadoop

JAVA_HOME = C:\Program Files\Java\jdk-19

SPARK_HOME = C:\spark\spark-3.5.0-bin-hadoop3

PYTHON_HOME= dossier Python par exemple
C:\Users\nfadd\AppData\Local\Programs\Python\Python311

PYSPARK_HOME = % **PYTHON_HOME** %**\python.exe**

Faire une copie de python.exe sous le nom python3.exe dans le même dossier d'installation de Python.

7. Ajouter dans **Path** les entrées suivantes:

%SPARK_HOME%\bin , **%HADOOP_HOME%\bin** , **%JAVA_HOME%\bin** et **PYTHON_HOME**

TP1: Spark-shell et PySpark

1. Le shell Spark en Scala

- Ouvrir un terminal Windows avec la commande **cmd**
- Saisir la commande **spark-shell**
  - ➔Le **Driver** sera démarré et on aura:
    - La version de Spark et celle de Scala
    - Le lien Web pour faire la supervision des Jobs lancés (**http://localhost:4040**)
    - Le nom de l'object **SparkContext** (**sc**)
    - Le nom de l'object **SparkSession** (**spark**)
- Saisir les instructions suivantes une à une:

```
print("Big Data")
print(sc)
print(spark)
```

- Quitter le shell avec la commande **:q**

2. Le shell Spark en Python

- Ouvrir un terminal Windows avec la commande cmd
- Saisir la commande **pyspark**
- Le **Driver** sera démarré et on aura les mêmes informations de **spark-shell** en plus de la version **Python** utilisée
- Saisir les **3** instructions **print** ci-dessus.

TP1: Premier exemple Python (WordCount)

3. On vas tester une petite application de calcul du nombre d'occurrences de chaque mot dans un fichier texte

a) Créer le fichier **c:\test.txt** contenant quelques lignes par exemple:

Nous étudions Apache Spark

C'est pour le traitement Big Data

Nous étudions comment utiliser Spark

b) Saisir les 3 instructions suivantes une à une dans le shell **Pyspark**:

```
rdd1 = sc.textFile("D:\\test.txt")
rdd2 = rdd1.flatMap(lambda ligne: ligne.split(" ")).
       map(lambda mot:(mot, 1)).reduceByKey(lambda a, b: a+b)
print(rdd2.collect())
```

b) La fonction **exit()** permet de quitter **Pyspark**:

TP1: Le client Web de Supervision (Web UI)

1. Utiliser l'URL suivant pour accéder à l'interface Web de monitoring (Web UI) de l'application Pyspark: **http://localhost:4040**
2. Noter dans l'onglet **Executors** le nombre d'Executors lancés par l'application et le nombre de cœurs CPU virtuels de chacun.
3. Dans l'onglet **Job**, on a la chronologie (**TimeLine**) de lancement d'**Executors** et des opérations de **calcul des résultats**.

   Combien de Jobs ont été généré par le code de l'application?
4. Relancer l'exécution de l'instruction print(rdd2.**collect()**) dans Pyspark. Actualiser la page du Web UI**. Que remarque-t-on à propos du nombre de Jobs ?**
5. Lancer l'exécution de print(rdd2.**count()**) pour avoir le nombre de mots du fichier **test.txt**. **Quel est le nombre de Jobs maintenant ?**
6. **Que peut-on conclure lorsqu'une instruction de récupération ou de calcul de résultat est lancée?**

TP1: Déployer une application avec Spark-submit

1. Créer un fichier Python **wordcount.py** contenant le code suivant:

```
# Importer SparkContext
        from pyspark import SparkContext
# Créer un objet SparkContext
        sc = SparkContext("local[*]", "Exemple WordCount")
# Charger un fichier  texte local
        rdd1 = sc.textFile("D:\\test.txt")
# Nombre d'occurrences de chaque mot
        rdd2 = rdd1.flatMap(lambda ligne: ligne.split(" ")).map(lambda mot:(mot, 1)).
                                                        reduceByKey(lambda a, b: a+b)
# Récupérer le résultat et l'afficher
        res = rdd2.collect()
        print("Résultat", res)
```

Pour désactiver les messages **INFO lors de l'exécution**:

1. Faire une copie de C:\spark\conf**log4j.properties.template** sous C:\spark\conf**log4j.properties**
2. Remplacer la ligne **rootLogger.level = INFO** par **rootLogger.level = ERROR**

2. Lancer l'exécution de ce fichier sur **Spark** avec la commande : **spark-submit** ***wordcount.py***

Remarque: Utiliser la commande **spark-submit** sans paramètre pour avoir sa syntaxe complete

TP1: Le package pyspark (1/3)

- **PySpark** est une bibliothèque d'analyse de données qui supporte quasiment toutes les fonctionnalités de Spark (**Spark SQL**, **Spark Streaming**, **Mlib**, **GraphX**).
- On peut **combiner ces différentes fonctionnalités (librairies)** de façon transparente dans une même application.
- **PySpark** offre des **structures de données** Spark permettant de manipuler des jeux de données (**datasets**) distribués de manière transparente.
- **PySpark** permet d'interagir avec Spark: on envoie des instructions au **Driver** responsable de la coordination de toutes les opérations.
- La communication avec le **Driver** est assurée par un objet **SparkContext** qui l'entrée de l'application.

TP1: Le package pyspark (2/3)

❑ Exemple d'utilisation **pyspark** dans l'IDE **PyCharm**:

wordcount.py

```
# Importer SparkContext
from pyspark import SparkContext

# Créer un objet SparkContext
sc = SparkContext("local[*]", "Exemple Map")

# Charger un fichier  texte local
rdd1 = sc.textFile("D:\\test.txt")
# Nombre d'occurrences de chaque mot
rdd2 = rdd1.flatMap(lambda ligne: ligne.split(" ")).\
            map(lambda mot:(mot, 1)).\
            reduceByKey(lambda a, b: a+b)
# Récupérer le résultat sous forma d'un tableau de pairs (mot, nombre)
res = rdd2.collect()
# Afficher le résultat
print("Résultat", res)
```


TP1: Le package pyspark (3/3)

❑ Lancer l'exécution de votre application avec le raccourci ▶ pour avoir le résultat:

```
"D:\DOC_PERSO\Cours\Traitement Big Data\code\exemple_map_RDD\venv\Scripts\python.exe" "D:\DOC_PERSO\Cours\Traitement Big
 Data\code\exemple_map_RDD\wordcount.py"
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Résultat [('Nous', 2), ('étudions', 2), ("C'est", 1), ('traitement', 1), ('Big', 1), ('comment', 1), ('Apache', 1), ('Spark', 2), ('pour', 1),
 ('le', 1), ('Data', 1), ('utiliser', 1)]

Process finished with exit code 0
```

Remarque: Utiliser l''instruction **sc.setLogLevel**(***niveau***) dans votre code Python pour personnaliser le niveau de journalisation des messages générés par **Spark** lors de l'exécution de l'application. Il peut être "**ALL**", "**DEBUG**", "**INFO**", "**WARN**", "**ERROR**", "**FATAL**" ou "**OFF**" si on veut désactiver la journalisation.