TD02 – BigData

Chapitre III : Systèmes distribués

Prise en main de Spark avec docker dans un environnement Linux

Ing. Sidi Mahmoud RHIL

Objectif

- Lancer Spark en local via Docker Compose
- Découvrir l'écosystème Apache Spark
- Manipuler les données avec les RDD et Data frames
- Utiliser Spark SQL pour l'analyse des données

1. Mise en place de l'environnement

- **Arborescence du projet**

td_spark/

|— docker-compose.yml

|— notebooks/

| └— ex01_rdd.ipynb

└— ex02_dataframe.ipynb

|— data/

└— ventes.csv

└— magasins.csv

- **docker-compose.yml**

```
services:
 spark-master:
   image: bitnami/spark:latest
   container_name: spark-master
   environment:
    - SPARK_MODE=master
    - SPARK_RPC_AUTHENTICATION_ENABLED=no
    - SPARK_RPC_ENCRYPTION_ENABLED=no
    - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
    - SPARK_SSL_ENABLED=no
```

```
    ports:
      - "7077:7077"   # Port du cluster
      - "8080:8080"   # UI du master
    volumes:
      - ./data:/opt/spark-data

  spark-worker:
    image: bitnami/spark:latest
    container_name: spark-worker
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark-master:7077
    depends_on:
      - spark-master
    ports:
      - "8081:8081"   # UI du worker
    volumes:
      - ./data:/opt/spark-data

  jupyter:
    image: jupyter/pyspark-notebook
    container_name: spark-jupyter
    ports:
      - "8888:8888"   # Interface Jupyter
    volumes:
      - ./notebooks:/home/jovyan/work
      - ./data:/home/jovyan/data
    depends_on:
      - spark-master
    environment:
      - SPARK_MASTER=spark://spark-master:7077
```

- **Jeu de données**

Créer les répertoires ***data*** et ***notebooks*** et puis déposer les fichiers ventes.csv et magasins.csv sous le répertoire local *td_spark/data*

```
sidi@pc-sidi:~/bigdata/spark/td_spark$ mkdir data
sidi@pc-sidi:~/bigdata/spark/td_spark$ mkdir notebooks
```

- **Lancer l'environnement**

```
sidi@pc-sidi:~/bigdata/spark/td_spark$ docker compose up -d
[+] Running 3/3
 ✔Container spark-master   Running
 ✔Container spark-jupyter  Running
 ✔Container spark-worker   Running
sidi@pc-sidi:~/bigdata/spark/td_spark$
```

- **Accès à Jupyter**

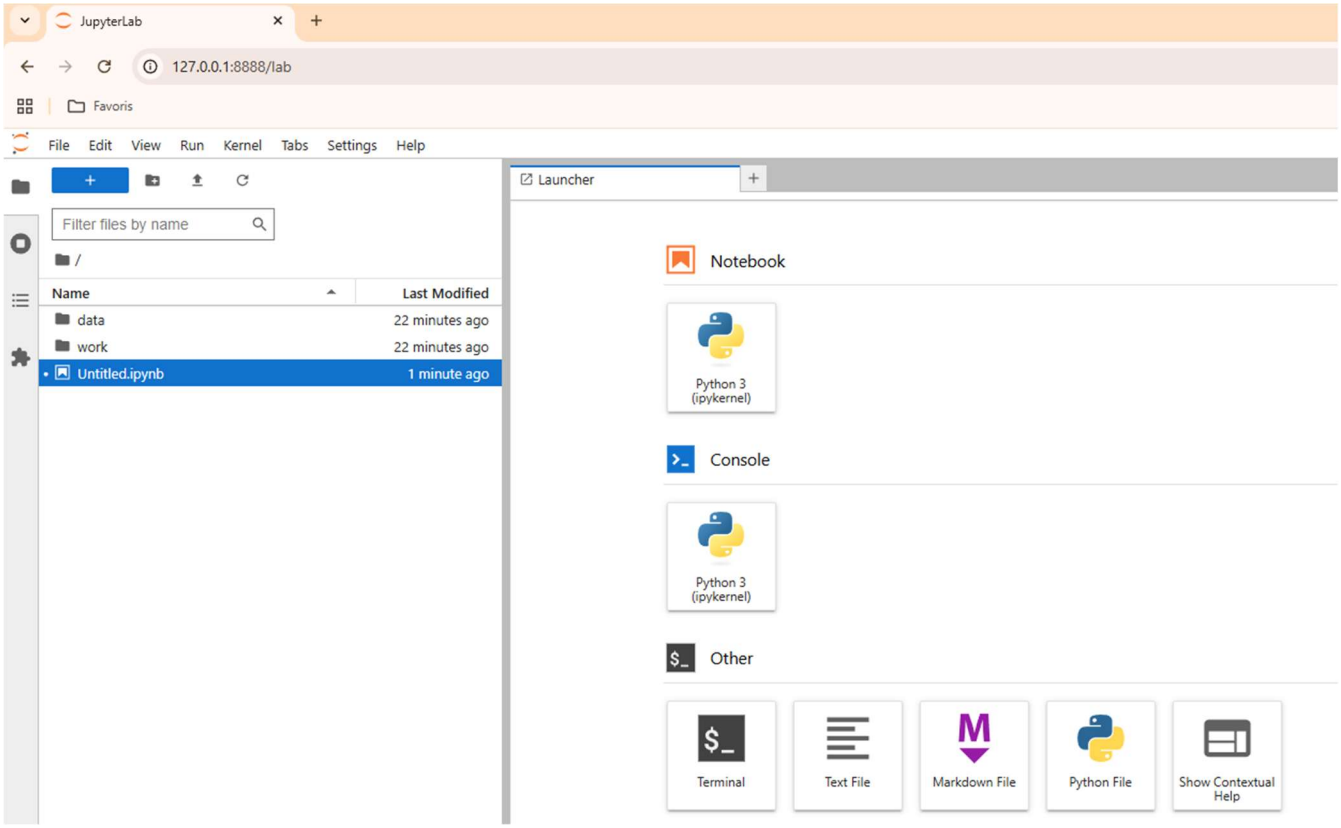
Jupyter est accessible avec cette URL : http://127.0.0.1:8888/lab?token=xxxx

Le token est visible dans la log du conteneur *spark-jupyter,* pour afficher la log, taper cette commande :

```
sidi@pc-sidi:~/bigdata/spark/td_spark$ docker logs spark-jupyter
```

```
To access the server, open this file in a browser:
    file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
Or copy and paste one of these URLs:
    http://85979a0ad111:8888/lab?token=5106f0f165e5c36aa45fa85338421507574d4a5f6562bd8a
    http://127.0.0.1:8888/lab?token=5106f0f165e5c36aa45fa85338421507574d4a5f6562bd8a
```

Interface web jupyter :



- **Accès aux autres interfaces**

  Master UI → http://localhost:8080
  Worker UI → http://localhost:8081
  Jupyter (PySpark) → http://localhost:8888
  Spark UI (jobs en cours) → http://localhost:4040

2. Manipulation des données avec les RDD

- **Initialisation d'une session spark**

Exécuter ce code dans la première cellule du notebook jupyter :

```
from pyspark.sql import SparkSession

# Création de la session spark
spark = SparkSession.builder \
    .appName("TD Spark PySpark") \
    .getOrCreate()

print("SparkSession démarrée !")
```

- **Charger les fichiers csv en RDD**

```
# Charger ventes.csv
rdd = spark.sparkContext.textFile("/home/jovyan/data/ventes.csv")
header = rdd.first()
ventes_rdd = rdd.filter(lambda line: line != header).map(lambda line: line.split(","))

# Charger magasins.csv
rdd_magasins = spark.sparkContext.textFile("/home/jovyan/data/magasins.csv")
header_m = rdd_magasins.first()
magasins_rdd = rdd_magasins.filter(lambda line: line != header_m).map(lambda line:
line.split(","))

# Vérification
print("Exemple de ventes :", ventes_rdd.take(2))
print("Exemple de magasins :", magasins_rdd.take(2))
```

- **Calculer la quantité totale vendue**

```
quantites = ventes_rdd.map(lambda x: int(x[3]))
print("Quantité totale vendue :", quantites.sum())
```

- **Calculer le chiffre d'affaires total**

```
ca_total = ventes_rdd.map(lambda x: int(x[3]) * int(x[4])).sum()
print("Chiffre d'affaires total :", ca_total)
```

- **Quel est le produit le plus vendu**

```
produits_qte = ventes_rdd.map(lambda x: (x[2], int(x[3])))
produits_aggr = produits_qte.reduceByKey(lambda a, b: a + b)
produit_top = produits_aggr.takeOrdered(1, key=lambda x: -x[1])

print("Produit le plus vendu :", produit_top)
```

- **Jointure RDD ventes et magasins**

```
# (id_magasin, (nom, ville))
magasins_kv = magasins_rdd.map(lambda x: (x[0], (x[1], x[2])))


# (id_magasin, (produit, quantite, prix))
```

```
ventes_kv = ventes_rdd.map(lambda x: (x[5], (x[2], int(x[3]), int(x[4]))))

# Jointure
jointure = ventes_kv.join(magasins_kv)

for row in jointure.take(5):
    print(row)
```

- **Chiffre d'affaires par ville**

```
ca_par_ville = jointure.map(lambda x: (x[1][1][1], x[1][0][1] * x[1][0][2])) \
                .reduceByKey(lambda a, b: a + b)

print("Chiffre d'affaires par ville :")
for row in ca_par_ville.collect():
    print(row)
```

- **Sauvegarder un RDD dans fichier sur disque**

```
# Transforme chaque tuple (ville, CA) en "ville,CA"
ca_par_ville.map(lambda x: f"{x[0]},{x[1]}") \
        .saveAsTextFile("/home/jovyan/data/ca_par_ville")
```

En sortie on aura plusieurs fichiers, un fichier par partition.

- **Pour avoir un seul fichier**

```
# Transforme chaque tuple (ville, CA) en "ville,CA"
ca_par_ville.coalesce(1).map(lambda x: f"{x[0]},{x[1]}") \
        .saveAsTextFile("/home/jovyan/data/ca_par_ville_unique")
```

- **Afficher le nombre de partitions**

```
print("Nombre de partitions :", ca_par_ville.getNumPartitions())
```

3. Manipulation des données avec les DataFrames

- **Charger les fichiers en DataFrame**

```
ventes = spark.read.csv("/home/jovyan/data/ventes.csv", header=True, inferSchema=True)
magasins = spark.read.csv("/home/jovyan/data/magasins.csv", header=True,
inferSchema=True)

ventes.show()
magasins.show()
```

- **Exploration des DataFrames**

```
ventes.printSchema()
magasins.printSchema()

print("Nombre de ventes :", ventes.count())
print("Nombre de magasins :", magasins.count())
```

- **Manipulations des DataFrames**

```
from pyspark.sql.functions import col, sum

# Ventes de Laptop
ventes.filter(col("produit") == "Laptop").show()

# Produit le plus vendu
ventes.groupBy("produit") \
  .agg(sum("quantite").alias("quantite_totale")) \
  .orderBy(col("quantite_totale").desc()) \
  .show()

# Chiffre d'affaires total
ventes.withColumn("total", col("quantite") * col("prix")) \
  .agg(sum("total").alias("CA_total")) \
  .show()
```

- **Jointure des DataFrames**

```
df_join = ventes.join(magasins, ventes.id_magasin == magasins.id, "inner")
df_join.select("date", "produit", "quantite", "prix", "nom", "ville").show()
```

- **Agrégation : CA par magasin**

```
df_join.withColumn("total", col("quantite") * col("prix")) \
   .groupBy("nom", "ville") \
   .agg(sum("total").alias("CA_magasin")) \
   .orderBy(col("CA_magasin").desc()) \
   .show()
```

- **Sauvegarder un DataFrame sur disque**

```
from pyspark.sql.functions import col, sum

resultat = (
   df_join
   .withColumn("total", col("quantite") * col("prix"))
   .groupBy("nom", "ville")
   .agg(sum("total").alias("CA_magasin"))
   .orderBy(col("CA_magasin").desc())
)

# Affiche à l'écran
resultat.show()
# Sauvegarde en CSV
resultat.write.mode("overwrite").option("header",
True).csv("/home/jovyan/data/ca_par_magasin")

# Ou en Parquet (format plus efficace)
resultat.write.mode("overwrite").parquet("/home/jovyan/data/ca_par_magasin_parquet")
```

Attention : Spark écrit **un dossier** avec plusieurs fichiers (part-0000…) → c'est normal, car on a un fichier par partition.

- **Pour avoir un seul fichier csv**

```
# Regroupe en 1 seule partition avant l'écriture
resultat.coalesce(1) \
   .write.mode("overwrite") \
   .option("header", True) \
   .csv("/home/jovyan/data/ca_par_magasin_unique")
```

- **Agrégations : produit le plus vendu par ville**

```
from pyspark.sql.functions import row_number
from pyspark.sql.window import Window

vente_par_ville = df_join.groupBy("ville", "produit") \
   .agg(sum("quantite").alias("quantite_totale"))

window = Window.partitionBy("ville").orderBy(col("quantite_totale").desc())

vente_top = vente_par_ville.withColumn("rang", row_number().over(window)) \
   .filter(col("rang") == 1)

vente_top.show()
```

4. Manipulation des données avec SparkSQL

- **Créer des vues SQL**

```
ventes.createOrReplaceTempView("ventes")
magasins.createOrReplaceTempView("magasins")
```

- **Exemple de requête SQL : CA par magasin**

```
spark.sql("""
SELECT m.nom, m.ville, SUM(v.quantite * v.prix) AS CA_magasin
FROM ventes v
JOIN magasins m ON v.id_magasin = m.id
GROUP BY m.nom, m.ville
ORDER BY CA_magasin DESC
""").show()
```