Memoria Practica 1

Architecture de procesamiento de datos masivos

Ksenia Turava Turav

Repositiorio de la entrega Github público: https://github.com/turava/BigDataOps

1. Clonado y arranque del entorno

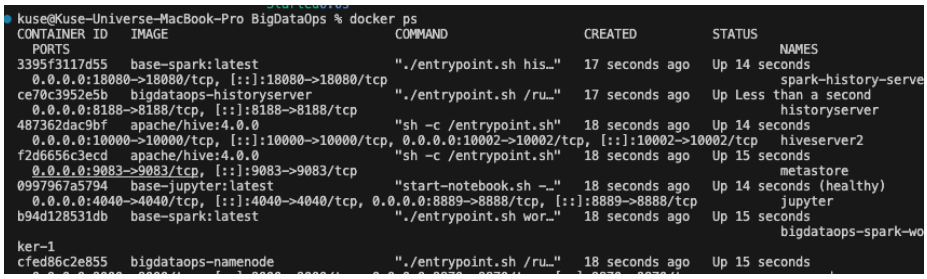
Clonar el repositorio BigDataOps y levantar el entorno distribuido con Docker Compose. Se inicia HDFS (NameNode + DataNodes) y el clúster Spark (Master + Workers). Verificar que todos los contenedores estén activos con docker ps. Este paso asegura el entorno Big Data operativo antes de procesar datos.

Solución

$ docker compose up -d

$ docker ps

HDFS NameNode: http://localhost:9870
Spark Master: http://localhost:8080



2. Carga del dataset en HDFS

Copiar el archivo bus_trips.csv desde el sistema local al contenedor del NameNode y subirlo a HDFS usando hdfs dfs -put. Confirmar su correcta ubicación con hdfs dfs -ls. Este paso garantiza que Spark lea datos desde almacenamiento distribuido real.

Solución

Copiar el CSV al contenedor namenode
$ docker cp datos-uax/practica_2_spark/bus_trips.csv namenode:/tmp/bus_trips.csv
$ Successfully copied 14.8MB to namenode:/tmp/bus_trips.csv
$ docker exec -it namenode bash
$ hdfs dfs -mkdir -p /datos-uax
$ hdfs dfs -ls /datos-uax
Found 3 items

-rw-r--r--   3 root supergroup   14832996 2025-12-15 20:38 /datos-uax/bus_trips.csv
drwxr-xr-x   - root supergroup   0 2025-11-17 22:12 /datos-uax/output_wordcount
-rw-r--r--   3 root supergroup   61 2025-11-17 21:57 /datos-uax/simple_text.txt

$ root@cfed86c2e855:/# ls -l /tmp/bus_trips.csv
-rw-rw-r-- 1 501 dialout 14832996 Dec 15 20:19 /tmp/bus_trips.csv
$ exit

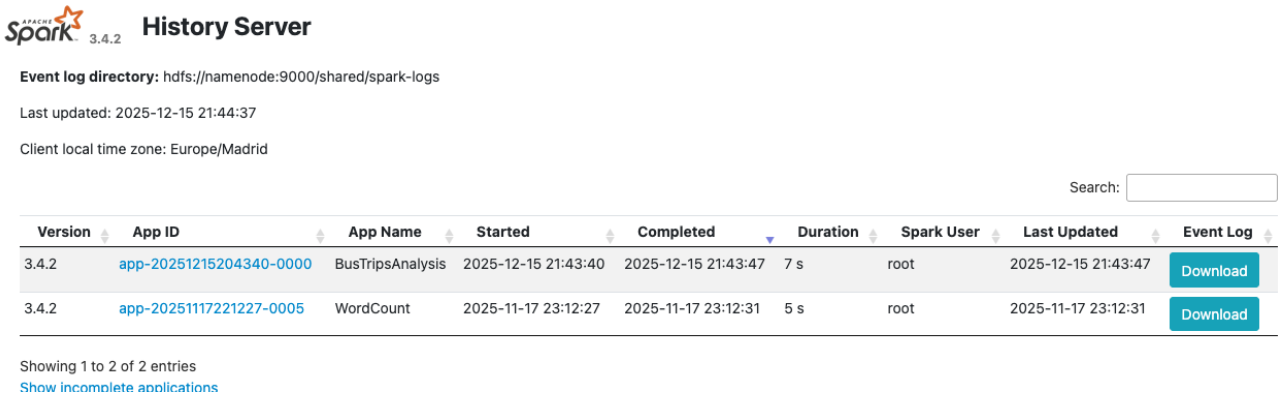
3. Creación del script analyze_bus_trips.py

Crear el script PySpark que inicializa una SparkSession, lee el CSV desde HDFS usando DataFrames y muestra esquema y primeras filas. Se valida la correcta lectura y tipado inicial de los datos antes de aplicar transformaciones.

Solución

Código subido a repositorio **https://github.com/turava/BigDataOps/blob/main/analyze_bus_trips.py**

$ kuse@Kuse-Universe-MacBook-Pro BigDataOps % docker cp analyze_bus_trips.py spark-master:/tmp/
Successfully copied 3.07kB to spark-master:/tmp/
$ docker exec -it spark-master bash
$ cd /tmp
$ spark-submit analyze_bus_trips.py
$ exit



4.Transformaciones con DataFrames

Seleccionar columnas relevantes (destination, trip_duration_hours) y convertir la duración a tipo numérico (float). Agrupar por destino y contar trayectos. Este paso demuestra el uso de la API estructurada frente a RDDs, con código más conciso y optimizado.

Solución

Se seleccionaron las columnas relevantes y se realizaron transformaciones para convertir tipos de datos y agrupar la información por destino, demostrando el uso de DataFrames frente a RDDs.

5. Uso de Spark SQL

Registrar el DataFrame como vista temporal (createOrReplaceTempView) y ejecutar consultas SQL. Se calcula el número de trayectos por destino y la duración media de trayectos superiores a 60 horas. Se evidencia el uso de SQL sobre datos distribuidos.

Solución

```
df_selected.createOrReplaceTempView("bus_trips")
spark.sql("""
SELECT destination, COUNT(*) AS trip_count
FROM bus_trips
GROUP BY destination
ORDER BY trip_count DESC
""")

spark.sql("""
SELECT destination, AVG(trip_duration_hours) AS avg_duration
FROM bus_trips
GROUP BY destination
HAVING avg_duration > 60
ORDER BY avg_duration DESC
""")
```

6. Persistencia de resultados en HDFS

Guardar los resultados de las consultas en formato **Parquet** dentro de HDFS usando mode("overwrite"). Se generan los directorios trip_analysis.parquet y avg_duration.parquet. Parquet optimiza almacenamiento y lectura en entornos Big Data.

Solución

```
trip_count_sql.write.mode("overwrite") \
  .parquet("hdfs://namenode:9000/datos-uax/trip_analysis.parquet")

avg_duration_sql.write.mode("overwrite") \
```

.parquet("hdfs://namenode:9000/datos-uax/avg_duration.parquet")

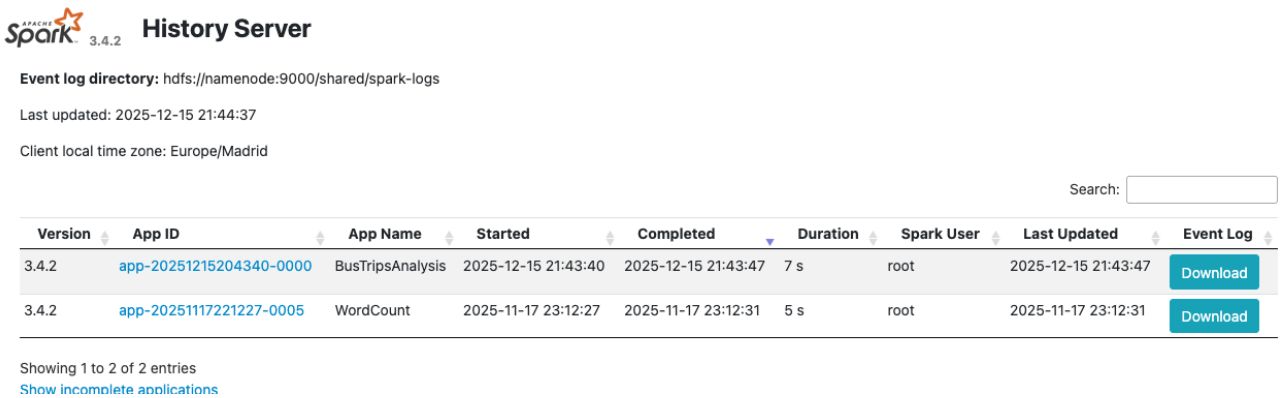
7. Ejecución en el clúster Spark

Copiar el script al contenedor spark-master y ejecutarlo con spark-submit. Se valida la ejecución distribuida y la salida en consola de los resultados. Este paso demuestra el procesamiento real en clúster, no local.

Solución

Paso 3 - Contiene la ejecución

$ spark-submit



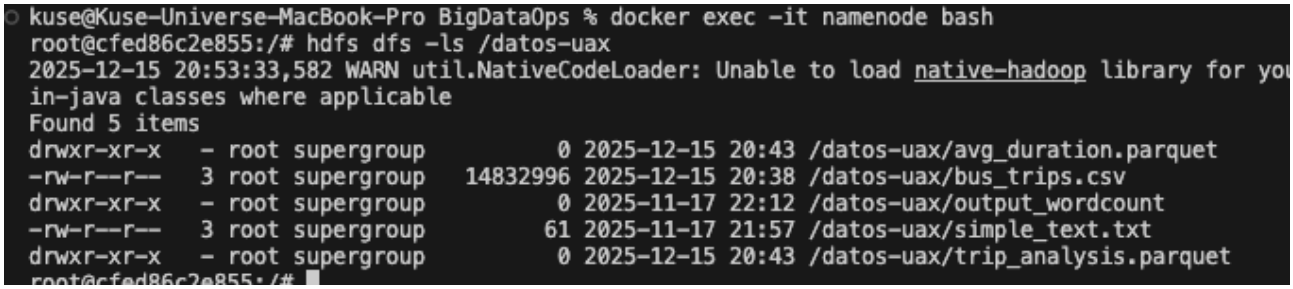
8. Verification de resultados

Se verificó la persistencia y reutilización de los resultados listando los archivos en HDFS y leyéndolos desde PySpark.

Solución

$ docker exec -it namenode bash

$ hdfs dfs -ls /datos-uax



```
kuse@Kuse-Universe-MacBook-Pro BigDataOps % docker exec -it spark-master bash
root@37b1be94422e:/opt/spark# cd /tmp
root@37b1be94422e:/tmp# spark-submit analyze_bus_trips.py
25/12/15 20:43:40 INFO SparkContext: Running Spark version 3.4.2
25/12/15 20:43:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cla
sses where applicable
25/12/15 20:43:40 INFO ResourceUtils: ==============================================================
25/12/15 20:43:40 INFO ResourceUtils: No custom resources configured for spark.driver.
25/12/15 20:43:40 INFO ResourceUtils: ==============================================================
25/12/15 20:43:40 INFO SparkContext: Submitted application: BusTripsAnalysis
25/12/15 20:43:40 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(memory -> name: memory,
amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> n
ame: cpus, amount: 1.0)
25/12/15 20:43:40 INFO ResourceProfile: Limiting resource is cpu
25/12/15 20:43:40 INFO ResourceProfileManager: Added ResourceProfile id: 0
25/12/15 20:43:40 INFO SecurityManager: Changing view acls to: root
25/12/15 20:43:40 INFO SecurityManager: Changing modify acls to: root
25/12/15 20:43:40 INFO SecurityManager: Changing view acls groups to:
25/12/15 20:43:40 INFO SecurityManager: Changing modify acls groups to:
25/12/15 20:43:40 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permi
ssions: root; groups with view permissions: EMPTY; users with modify permissions: root; groups with modify permissions: E
MPTY
25/12/15 20:43:40 INFO Utils: Successfully started service 'sparkDriver' on port 43217.
```

$ docker exec -it spark-master bash

$ pyspark

```
kuse@Kuse-Universe-MacBook-Pro BigDataOps % docker exec -it spark-master bash
root@37b1be94422e:/opt/spark# pyspark
Python 3.11.6 (main, Nov 29 2023, 03:58:21) [GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/15 20:54:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
sses where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.4.2
      /_/

Using Python version 3.11.6 (main, Nov 29 2023 03:58:21)
Spark context Web UI available at http://37b1be94422e:4040
Spark context available as 'sc' (master = spark://spark-master:7077, app id = app-20251215205441-0001).
SparkSession available as 'spark'.
>>>
```

$ spark.read.parquet("hdfs://namenode:9000/datos-uax/trip_analysis.parquet").show()

```
>>> spark.read.parquet("hdfs://namenode:9000/datos-uax/trip_analysis.parquet").show()
+--------------------+----------+
|         destination|trip_count|
+--------------------+----------+
|           SAO PAULO|     17102|
|      RIO DE JANEIRO|     14299|
|      BELO HORIZONTE|      4626|
|             RECIFE|      2874|
|       FLORIANOPOLIS|      2857|
|            CURITIBA|      2743|
|        PORTO ALEGRE|      2305|
|        JUIZ DE FORA|      2090|
|           TERESINA|      1985|
|           CAMPINAS|      1647|
|     POCOS DE CALDAS|      1569|
|            CRUZEIRO|      1557|
|              SANTOS|      1536|
|              NATAL|      1507|
|            VITORIA|      1461|
|            MANTENA|      1392|
|        CAMPO GRANDE|      1380|
|           BOA VISTA|       984|
|             ARACAJU|       952|
|SAO JOSE DO RIO P...|       912|
+--------------------+----------+
only showing top 20 rows
```

$ spark.read.parquet(“hdfs://namenode:9000/datos-uax/avg_duration.parquet").show()

```
>>> spark.read.parquet("hdfs://namenode:9000/datos-uax/avg_duration.parquet").show()
+-----------+------------------+
|destination|      avg_duration|
+-----------+------------------+
|ESPERANTINA|107.14499759674072|
+-----------+------------------+
```

9. Conclusiones

Reflexión sobre el uso de DataFrames y Spark SQL para la manipulación de datos en comparación con los RDDs, resaltando la facilidad en la ejecución de consultas SQL. Discutir los principales desafíos enfrentados durante el procesamiento de los datos y las ventajas de utilizar PySpark en un entorno distribuido

El uso de DataFrames y Spark SQL ha facilitado significativamente el procesamiento de los datos frente a un enfoque basado en RDDs. La API estructurada de Spark permite trabajar con datos de forma más intuitiva, reduciendo la complejidad del código y haciendo el flujo de procesamiento más legible. Además, el uso de Spark SQL aporta una capa de abstracción muy útil, ya que permite realizar análisis complejos mediante consultas SQL sin necesidad de implementar lógica adicional en Python.

Durante el desarrollo del proyecto, uno de los principales retos fue la gestión del entorno distribuido en Docker, especialmente en sistemas Apple Silicon, donde algunas herramientas no funcionan de forma nativa. Aun así, el uso de la línea de comandos y la ejecución mediante spark-submit permitió completar todo el flujo de trabajo sin afectar al resultado final. Este enfoque ayudó a entender mejor cómo interactúan HDFS y Spark en un entorno real.

En conjunto, el uso de PySpark en un entorno distribuido ha demostrado ser una solución eficiente para el análisis de datos a gran escala. La posibilidad de combinar transformaciones con DataFrames, consultas SQL y almacenamiento en formatos optimizados como Parquet refuerza la utilidad de Spark como herramienta clave dentro de arquitecturas Big Data.