Cluster de Spark Comercio 360

Por: Juan Néstor Franco Domínguez

Github con el contenido:

https://github.com/Junesfran/Trabajo_Spark.git

1. Descripción del problema

Como nos indica el enunciado, la empresa Comercio360 nos ha contratado como “profesionales en Big Data”, ellos desponen de datos de sus clientes, las ventas y productos almacenados en Amazon S3.

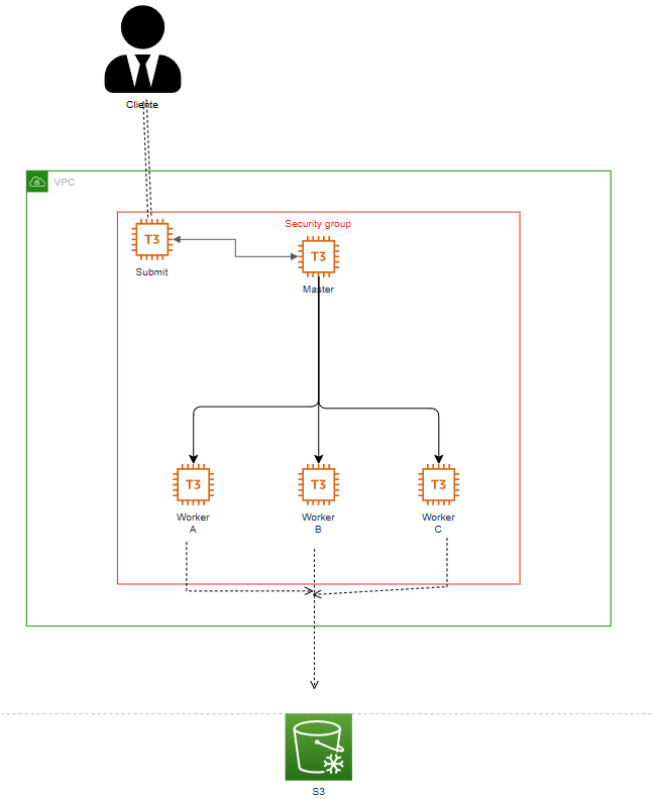
Esta empresa ficticia quiere disponer de una solución escalable y distribuida que le permita como dice el enunciado:

- Procesar mucho volumen de datos
- Calcular métricas de negocio objetivas
- Almacenar resultados analíticos para su posterior explotación

Por lo que como “equipo profesional de Big Data” determinamos que las necesidades de esta empresa se pueden solventar desplegando un apache Spark a través de instancias EC2 haciendo uso de una arquitectura cloud.

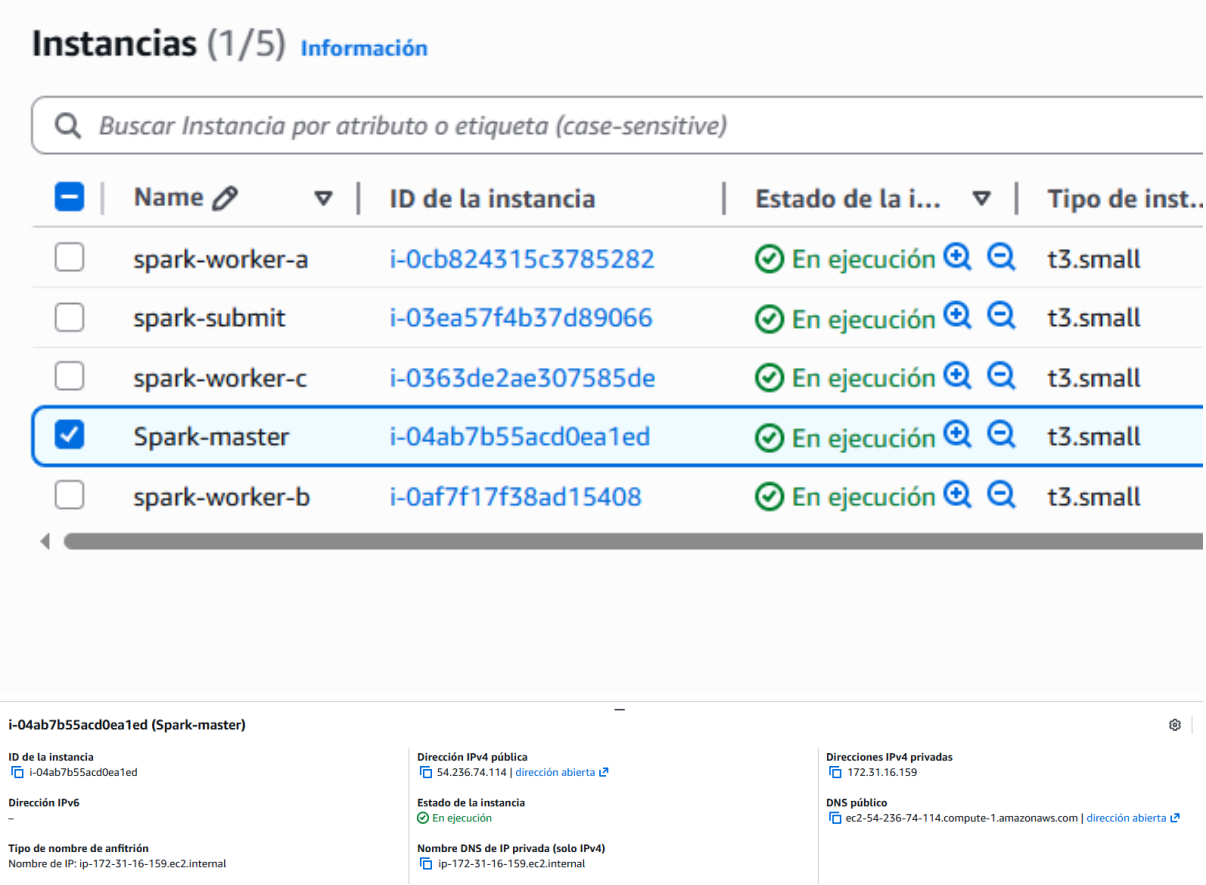
2. Arquitectura del clúster

Para la arquitectura de la red se ha ideado el siguiente esquema a seguir:



En esta distribución mostramos como idea habilitar 5 instancias de AWS EC2, en el que una de las instancias trabajará como nodo máster del clúster de apache Spark, tres instancias de EC2 cumplirán el rol de workers y una última instancia cumplirá el rol de nodo submit, por el que lanzaremos los jobs o trabajos que procesará el cluster.

Todo esto se conectará a los buckets de S3 de la empresa ficticia Comercio360, para sacar los datos pertinentes.



Para la configuración de todo se optó por las siguientes prestaciones para las instancias *(Contando siempre por buscar la mejor opción dentro de la capa gratuita al ser un laboratorio):*

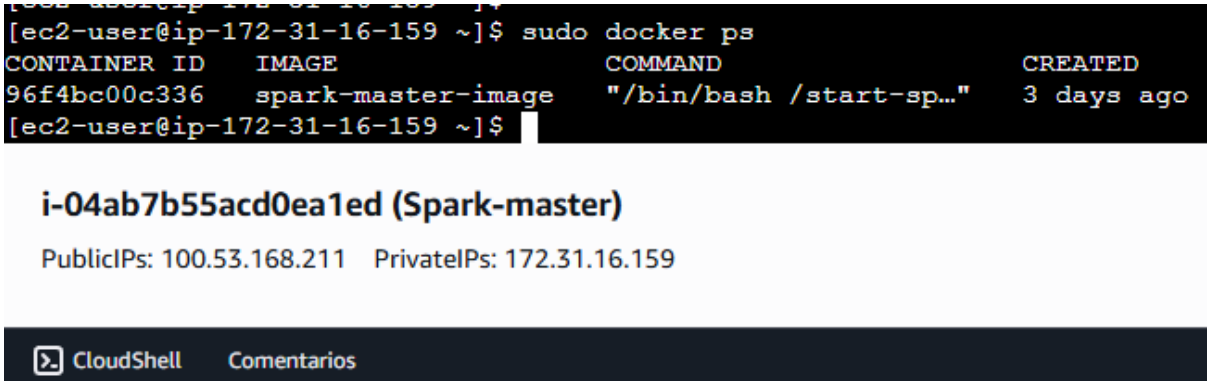
- Tipo de instancia: t3.small .- Se selecciono esta porque dentro de la capa gratuita para el laboratorio esta tenía al menos 2 Gb de memoria y eso es lo mínimo que me requería para funcionar como nodo master de Spark.
- Sistema Operativo: Amazon Linux .- Basado en RedHat. Para el SO se probó a usar también Ubuntu, pero tras varios intentos tratando de configurar apache Spark en Ubuntu, la dificultad de este me estaba comiendo mucho tiempo sin llegar a dar resultados. Por lo tanto, se optó por usar Docker al poder partir del Dockerfile que se nos proporcionó. Como Docker es más sencillo de descargar en una máquina Amazon Linux se optó porque todas las instancias lo usarán.
- Grupo de seguridad VPC .- Todas las instancias están dentro de una VPC y un grupo de seguridad personalizados, para poder controlar los puertos de entrada de datos.

El resto de datos de las instancias EC2 fueron los valores predeterminados.

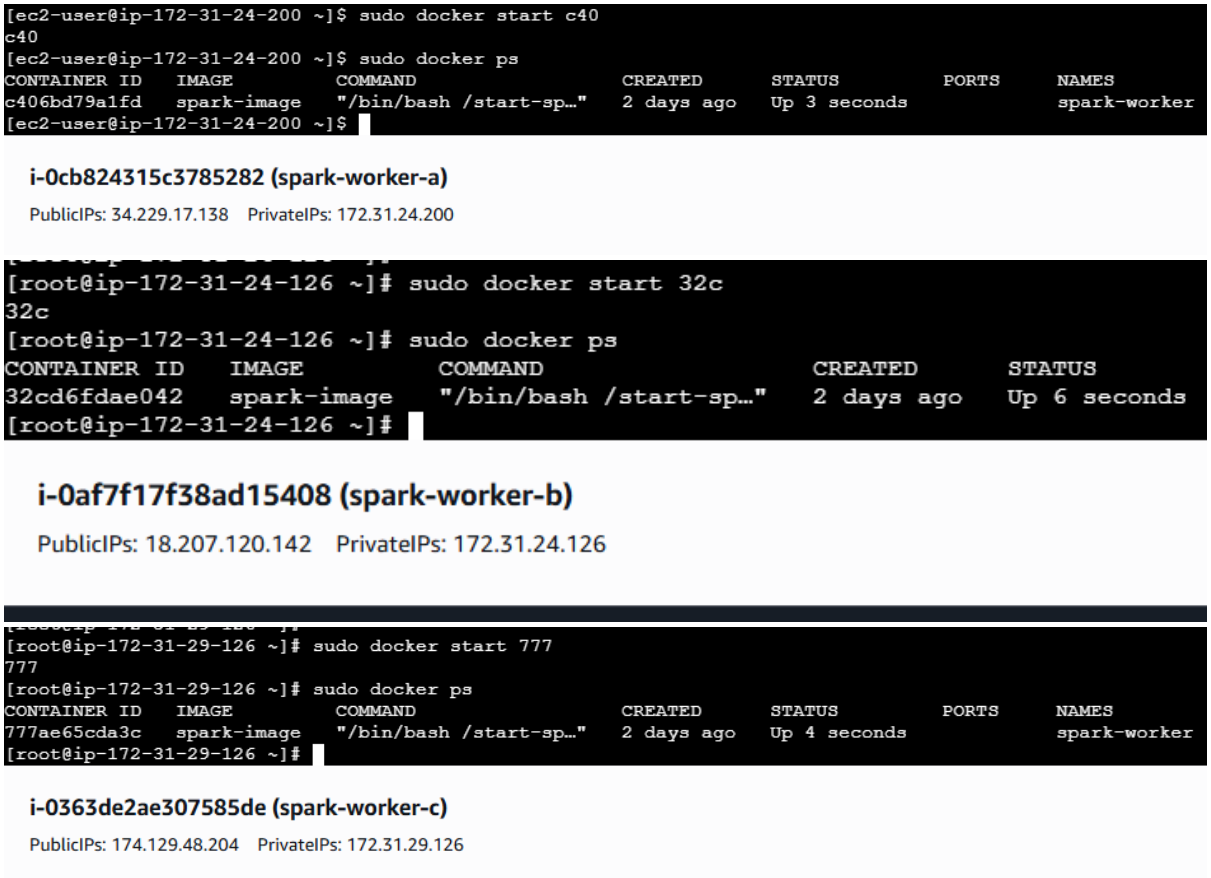
Una vez creadas las instancias como ya se ha comentado se optó por usar Docker, por lo tanto, cada instancia tendrá un único contenedor con su trabajo concreto.

Como la configuración de Docker en Amazon Linux es simple se descargó y configuro a mano para el master y el primer worker. Una vez conectado el primer worker al cluster se creó una imagen de la instancia que sirvió para lanzar los otros dos workers y como base para el submit, ya que comparte la misma imagen de contenedor, solo cambio la configuración de cómo se lanza el contenedor.

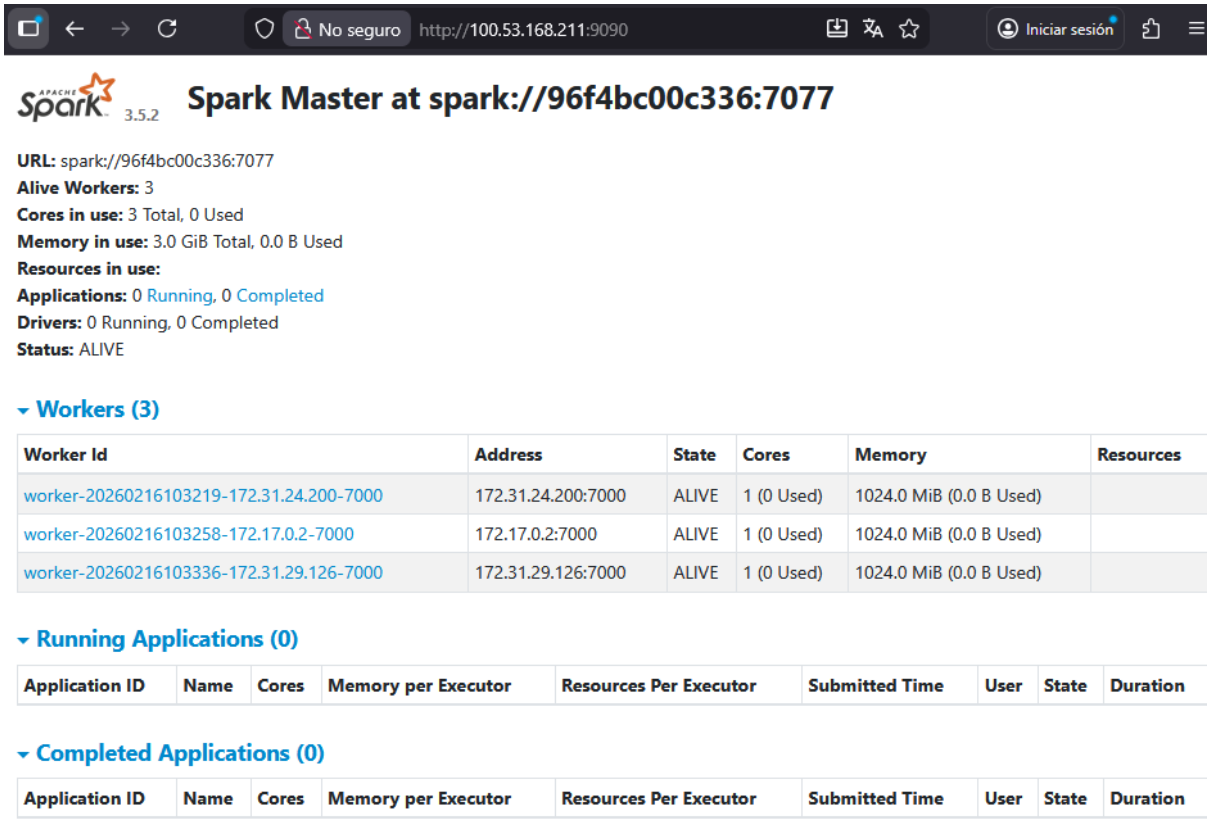
master



Workers



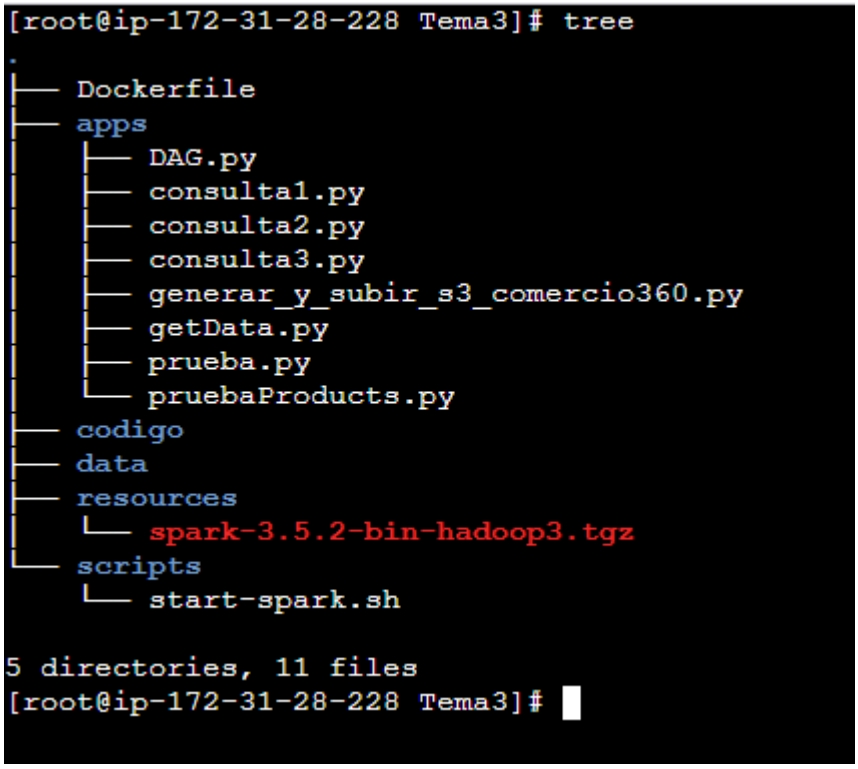
WEB UI



Al Docker se le crea una imagen en base a un Dockerfile, que partiendo del que se nos proporcionó, se le han cambiado y añadido cosas:

- Se le cambio la versión de Apache Spark de la 4.0.1 a la 3.5.2
- Se le cambio la versión del openjdk del 11 al 17. Ya que java11 tiene funciones ya deprecadas para Spark y la versión mínima más moderna que no da problemas de versión.
- Se ha añadido a las librerías de Python Boto3, para hacer conexiones al bucket de S3.
- La variable de entorno de SPARK_MASTER, se le ha añadido la ip privada del EC2 con el contenedor del nodo master.

Con el Dockerfile montado se ha procedido a montar en el EC2 este sistema de ficheros:



En la carpeta /apps guardaremos los programas o jobs a lanzar desde el nodo submit.

En la carpeta /resources se guarda el .tar.gz de Spark-hadoop para montar la imagen de docker .

sudo wget https://archive.apache.org/dist/spark/spark-3.5.2/spark-3.5.2-bin-hadoop3.tgz

Este es el comando usado para traerse al EC2 el archivo con todo Spark-Hadoop

En la carpeta /scripts se guarda el script de bash start-spark.sh para iniciar el nodo de Spark.

Para crear la imagen de los contenedores docker se hace uso de este comando:

sudo docker build -t spark-master-image .

Mientras este comando se ejecute en la carpeta del Dockerfile, este ya se ocupará de toda la configuración de la imagen, que está preparada para funcionar tanto para los worker, como para el master y el submit.

Para la construcción de cada nodo se han hecho uso de comando distintos, ya que aunque la imagen te sirva para los tres tipos de nodos, necesita que le indiquemos que función cumplirá y algunas configuracione propias del nodo:

- NODO MASTER:

```
sudo docker run -d \
  --name spark-master \
  -p 9090:8080 \
  -p 7077:7077 \
  -v /home/ec2-user/Tema3/apps:/opt/spark-apps \
  -v /home/ec2-user/Tema3/data:/opt/spark-data \
  -e SPARK_LOCAL_IP=0.0.0.0 \
  -e SPARK_WORKLOAD=master \
  spark-master-image
```

(Le especificamos, su nombre, redirección de ciertos puestos y que se copie el contenido en el contenedor del las carpetas /apps y /data, a parte de de indicarle que es el nodo master)

- NODO WORKER:

```
sudo docker run -d \
  --name spark-worker \
  --net=host \
  -e SPARK_MASTER=spark://172.31.16.159:7077 \
  -e SPARK_WORKER_CORES=1 \
  -e SPARK_WORKER_MEMORY=1G \
  -e SPARK_DRIVER_MEMORY=1G \
  -e SPARK_EXECUTOR_MEMORY=1G \
  -e SPARK_WORKLOAD=worker \
  -e SPARK_WORKER_HOST=172.31.24.200 \
  -v /home/ec2-user/Tema3/apps:/opt/spark-apps \
  -v /home/ec2-user/Tema3/data:/opt/spark-data \
  spark-image
```

(Le indicamos la ip privada del nodo master, para que sea a cuál conectarse, a parte las prestaciones que tiene que tener cada contenedor worker.

IMPORTANTE, la línea --net=host, ya que como están en instancias de EC2 separadas, el contenedor master no sabe si le están enterando conexiones de máquinas distintas y acostumbrará a dar la ip 172.0.0.2 a todos los worker, dando problemas de ya que no puede haber dos worker logeados con la misma ip, con esa línea le fuerzas a que el nodo master les asigne la ip privada de la instancia EC2 donde está corriendo cada worker)

SUBMIT:

Para crear un docker que al lanzar el job desde una shell bash que me abra se elimine

sudo docker run -it --rm --net=host -v /home/ec2-user/Tema3/apps:/opt/spark-apps -v ~/.aws:/root/.aws -e AWS_DEFAULT_REGION=us-east-1 spark-image bash

Para lanzar los jobs dentro del contenedor

```
spark-submit \
 --deploy-mode client \
 --packages org.apache.hadoop:hadoop-aws:3.3.4,com.amazonaws:aws-java-sdk-bundle:1.12.262 \
 /opt/spark-apps/consulta1.py


 spark-submit \
 --deploy-mode client \
 /opt/spark-apps/DAG.py
```

(El nodo submit es dintinto, ya que no mantendrá conexión con el cluster sino que se conectará, lanzar el job, y una vez el cluster termine de ejecutarlo se cerrará el contenedor.

Por lo tanto, no necesita de mucha configuración, igualmente la configuración del submit como cuál es la ip del master se lo indicaremos por código en los jobs.

Dependiendo del job requerirá que le indiquemos más o menos parámetros. Por ejemplo, en los jobs de las consultas a realizar necesita que haya en los worker ciertos paquetes de java, por lo que se lo tenemos que indicar)

```
spark-submit \
 --master spark://172.31.16.159:7077 \
```

*--deploy-mode client *

*/opt/spark-apps/generar_y_preparar_s3_coomercio360.py *

*--bucket hugito-quesadilla *

*--prefix comercio360/JuanNestorFranco/raw *

*--seed 123 *

*--customers 500 *

*--products 150 *

*--stores 10 *

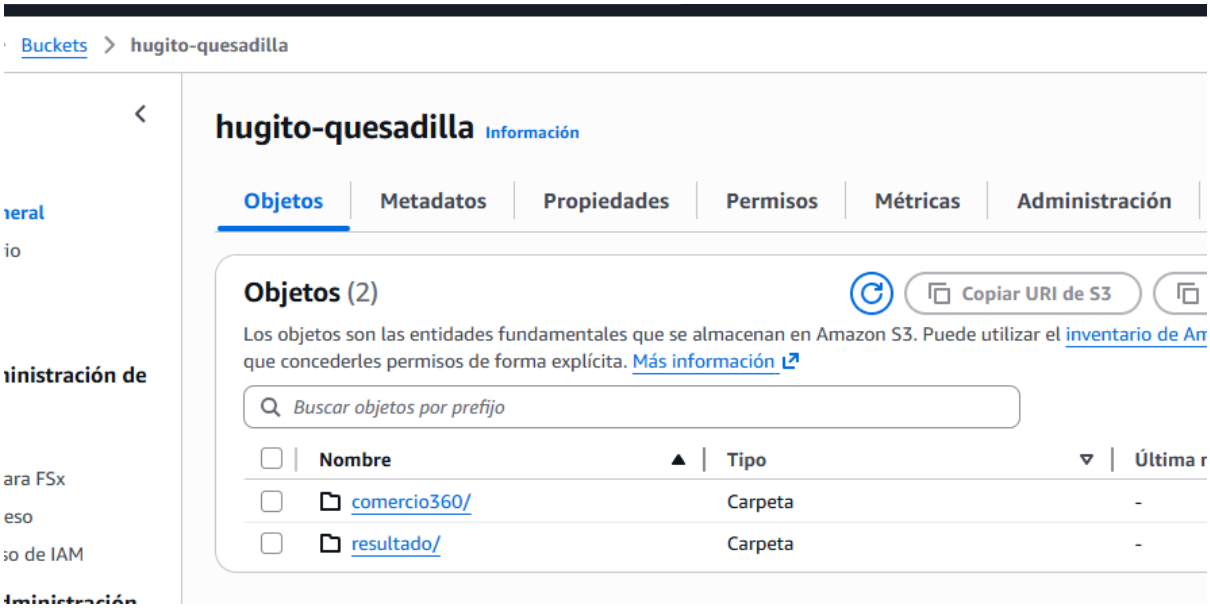
*--orders 4000 *

--max-items 6

(En este job que sirve para generar los datos de prueba requiere de que le especifiquemos otros parámetros por ejemplo)

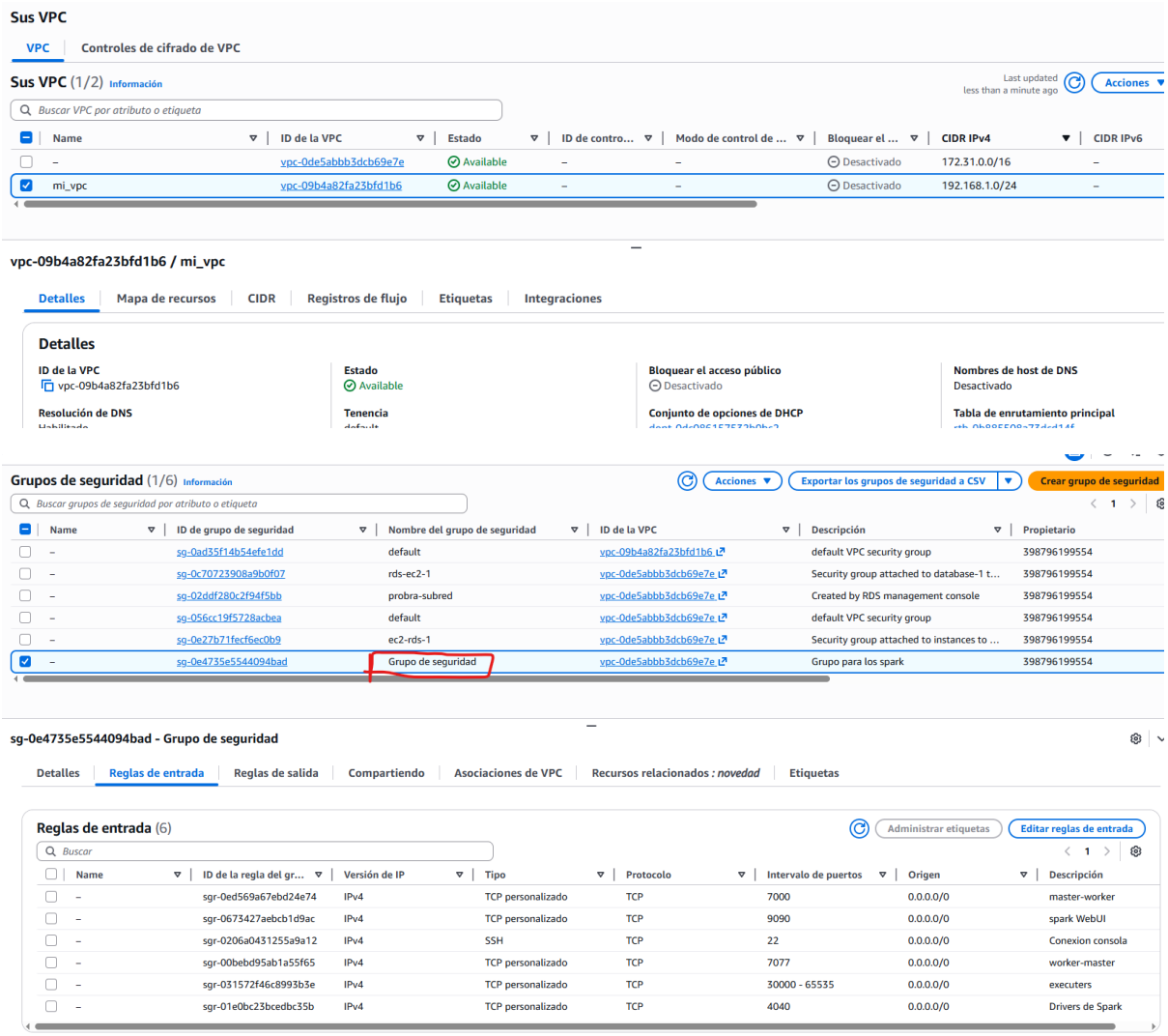
También tenemos el bucket que hemos generado para este ejercicio, que cuenta con los csv generados de prueba.

En mi caso el bucket se ha llamado hugito-quesadilla.

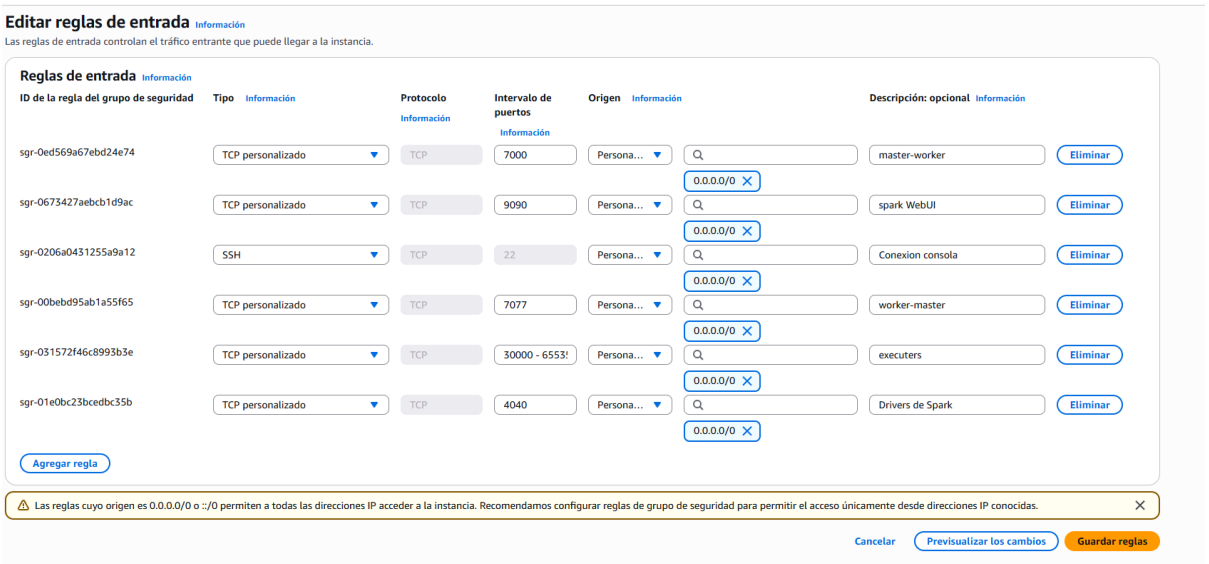


3. Configuración de red

Para la conexión entre las instancias EC2 se optó por hacer una vpc personalizada dándole el rango de IPs dentro de 192.168.1.0/24.



Esto también trajo la creación de un grupo de seguridad propio, ya que un despliegue profesional no sería seguro si se permitiera todo el tráfico de red.



Las reglas de entrada personalizadas que podemos ver en el grupo de seguridad son:

- Puerto 9090: ya que el docker redirige el tráfico web allí
- Puerto 7070: para la comunicación del nodo master con el nodo worker
- Puerto 7000: exactamente lo contrario, la comunicación worker con master
- Rango de puertos del 3000 - máx. : Se abrieron todos los puertos a partir del 3000, para que spark pudiera una vez lanzado el job, mandar drivers, transmitir el código al master, ...

El contenedor de los Buckets en S3 que hemos creado para el ejercicio no está dentro de la red VPC y por tanto no pertenece a ningún grupo de seguridad, por lo que nos conectaremos a él a través de su direccionamiento DNS.

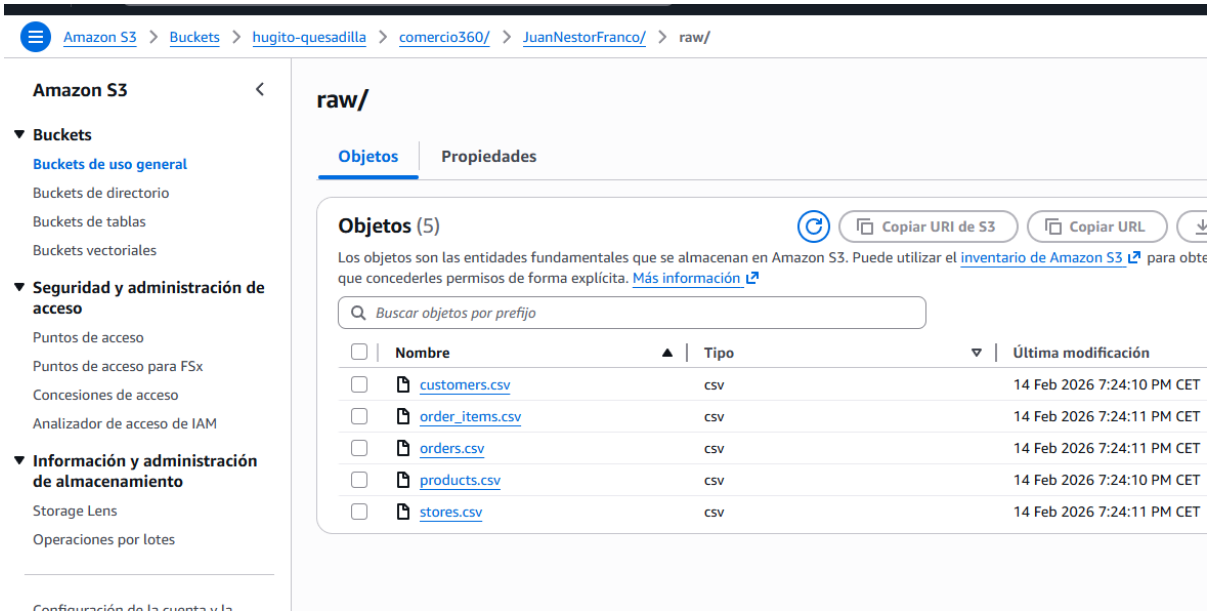
4. Pipeline de datos

Por lo tanto, después de todo lo explicado, sigamos el recorrido de un job de Spark en esta red, teniendo como ejemplo, el job que se nos proporcionó de generar_y _preparar_S3_Comercio360.py .

Primero se programaría el job, que en este ejercicio estarán todos programados en python con el uso de Pyspark. Para el ejemplo planteado, como requiere una conexión con el bucket para cargar los datos se ha hecho uso de la librería de python boto3, que añadimos al Dockerfile para tenerlo ya instalado. Todos los jobs han de estar guardados en una carpeta que especifiquemos, en este caso la carpeta /apps.

Luego a través del submit este job se envía al master, que lo desfragmente en trozos y lo manda a los worker.

Estos realizan todo el cómputo , en el caso de ejemplo se cargan los bucket en el S3.



El único inconveniente en mantener todas las librerías y versionados correctos en todos los worker, ya que si en uno solo se está usando algo que falta o que no está actualizado dará un error.

5. Consultas realizadas

Aunque aquí se deja una copia del código de cada Job, estos se pueden encontrar en el github adjunto.

Para las conexiones a S3, en vezs de usar Boto3 ya que me etaba dando problemas he optado por usar el paquete que ya trae Spark de s3a.

Consulta A – Top productos por facturación diaria

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window


spark = SparkSession.builder \
    .master("spark://172.31.16.159:7077") \
    .appName("Consulta A") \
    .getOrCreate()
```

```
orders = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/orders.csv")


order_items = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/order_items.csv")


order_items_enriched = order_items.withColumn(
    "importe_linea",
    F.col("quantity") * F.col("unit_price")
)

df = order_items_enriched.join(
    orders.select("order_id", "order_date"),
    "order_id"
)

ventas_diarias = df.groupBy("order_date", "product_id") \
    .agg(
        F.sum("quantity").alias("unidades"),
        F.sum("importe_linea").alias("importe_total")
    )
```

```
window = Window.partitionBy("order_date").orderBy(F.col("importe_total").desc())

ventas_top = ventas_diarias.withColumn(
    "ranking",
    F.row_number().over(window)
).filter(F.col("ranking") <= 10)

resultado = ventas_top.select(
    F.col("order_date").alias("fecha"),
    "product_id",
    "unidades",
    "importe_total",
    "ranking"
).orderBy("fecha", "ranking")

resultado.show(20)

resultado.write \
    .mode("overwrite") \
    .option("header", "true") \
    .csv("s3a://hugito-quesadilla/resultado/consultaA_csv/")

resultado.write \
    .mode("overwrite") \
    .parquet("s3a://hugito-quesadilla/resultado/consultaA_parquet/")
```

Consulta B – Ticket medio y clientes únicos por categoría (mensual)

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

spark = SparkSession.builder \
        .master("spark://172.31.16.159:7077") \
        .appName("Consulta B") \
        .getOrCreate()

order_items = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/order_items.csv")

orders = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/orders.csv")

products = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/products.csv")

order_items_enriched = order_items.withColumn(
    "importe_linea",
    F.col("quantity") * F.col("unit_price")
)
```

```
df = order_items_enriched \
    .join(orders, "order_id") \
    .join(products, "product_id")


df = df.withColumn(
    "year_month",
    F.date_format("order_date", "yyyy-MM")
)

resultado = df.groupBy("year_month", "category") \
    .agg(
        F.countDistinct("customer_id").alias("clientes_unicos"),
        F.countDistinct("order_id").alias("num_pedidos"),
        F.sum("importe_linea").alias("importe_total")
    ) \
    .withColumn(
        "ticket_media",
        F.col("importe_total") / F.col("num_pedidos")
    ) \
    .orderBy("year_month", "category")

resultado.show()

resultado.write \
    .mode("overwrite") \
    .option("header", "true") \
```

```
    .csv("s3a://hugito-quesadilla/resultado/consultaB_csv/")

resultado.write \
    .mode("overwrite") \
    .parquet("s3a://hugito-quesadilla/resultado/consultaB_parquet/")
```

Consulta C – Detección de outliers de ventas por tienda

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window

spark = SparkSession.builder \
        .master("spark://172.31.16.159:7077") \
        .appName("Consulta C") \
        .getOrCreate()

order_items = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/order_items.csv")

orders = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .csv("s3a://hugito-quesadilla/comercio360/JuanNestorFranco/raw/orders.csv")

order_items_enriched = order_items.withColumn(
```

```
    "importe_linea",
    F.col("quantity") * F.col("unit_price")
)

df = order_items_enriched.join(
    orders.select("order_id", "store_id", "order_date"),
    "order_id"
)

ventas_diarias = df.groupBy("store_id", "order_date") \
    .agg(
        F.sum("importe_linea").alias("ventas_dia")
    )

ventas_diarias = ventas_diarias.withColumn(
    "order_ts",
    F.col("order_date").cast("timestamp")
)

window_30d = Window.partitionBy("store_id") \
    .orderBy(F.col("order_ts").cast("long")) \
    .rangeBetween(-2592000, 0)

ventas_stats = ventas_diarias.withColumn(
    "media_30d",
    F.avg("ventas_dia").over(window_30d)
).withColumn(
    "desviacion_30d",
```

```
    F.stddev("ventas_dia").over(window_30d)
)

resultado = ventas_stats.withColumn(
    "is_outlier",
    F.when(
        F.col("ventas_dia") >
        F.col("media_30d") + 2 * F.col("desviacion_30d"),
        True
    ).otherwise(False)
)

resultado_final = resultado.select(
    "store_id",
    "order_date",
    "ventas_dia",
    "media_30d",
    "desviacion_30d",
    "is_outlier"
).orderBy("store_id", "order_date")

resultado_final.show()

resultado.write \
    .mode("overwrite") \
    .option("header", "true") \
    .csv("s3a://hugito-quesadilla/resultado/consultaC_csv/")
```

```
resultado.write \
   .mode("overwrite") \
   .parquet("s3a://hugito-quesadilla/resultado/consultaC_parquet/")
```

DAG:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
   .master("spark://172.31.16.159:7077") \
   .appName("Consulta A") \
   .getOrCreate()

df = spark.range(5)
df.show()
df.explain()
```

6. Evidencias de ejecución

Consulta A – Top productos por facturación diaria

```
26/02/16 10:43:31 INFO CodeGenerator: Code generated in 18.354124 ms
26/02/16 10:43:31 INFO CodeGenerator: Code generated in 15.590477 ms
+----------+----------+--------+------------------+-------+
|     fecha|product_id|unidades|     importe_total|ranking|
+----------+----------+--------+------------------+-------+
|2025-10-17|       123|       7|            947.28|      1|
|2025-10-17|        60|      11|            601.12|      2|
|2025-10-17|       126|       4|            583.16|      3|
|2025-10-17|        74|       6|            544.95|      4|
|2025-10-17|        86|       3|            286.62|      5|
|2025-10-17|        21|       3|282.09000000000003|      6|
|2025-10-17|        65|       9|             247.1|      7|
|2025-10-17|       127|       4|            236.96|      8|
|2025-10-17|       124|       5|            178.63|      9|
|2025-10-17|        93|       5|            176.22|     10|
|2025-10-18|        21|       8| 695.0999999999999|      1|
|2025-10-18|       144|       9|            561.49|      2|
|2025-10-18|       114|       5|            498.61|      3|
|2025-10-18|        55|       5|            433.23|      4|
|2025-10-18|         4|       3|411.90000000000003|      5|
|2025-10-18|       119|       4|             392.0|      6|
|2025-10-18|        32|       3|            352.07|      7|
|2025-10-18|         1|       3|            350.43|      8|
|2025-10-18|       147|       6|            286.32|      9|
|2025-10-18|       143|       6|            256.34|     10|
+----------+----------+--------+------------------+-------+
only showing top 20 rows
```

Spark Master at spark://96f4bc00c336:7077

3.5.2

URL: spark://96f4bc00c336:7077
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20260216103219-172.31.24.200-7000	172.31.24.200:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103258-172.17.0.2-7000	172.17.0.2:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103336-172.31.29.126-7000	172.31.29.126:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

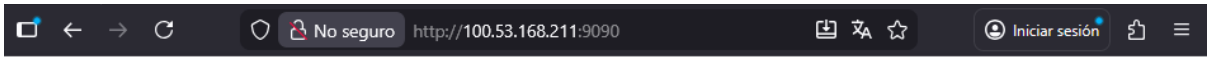
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duratio
app-20260216104302-0000	Consulta A	3	1024.0 MiB		2026/02/16 10:43:02	root	FINISHED	35 s

Consulta B – Ticket medio y clientes únicos por categoría (mensual)

```
26/02/16 10:54:34 INFO CodeGenerator: Code generated in 19.674682 ms
+----------+-----------+---------------+-----------+------------------+------------------+
|year_month|   category|clientes_unicos|num_pedidos|     importe_total|      ticket_media|
+----------+-----------+---------------+-----------+------------------+------------------+
|   2025-10|Alimentación|            217|        269| 5912.960000000005|21.981263940520464|
|   2025-10|     Belleza|             97|        111| 5757.849999999999|51.872522522522516|
|   2025-10|    Deporte|            127|        138|15781.640000000001|114.35971014492755|
|   2025-10|Electrónica|            122|        143|38355.329999999994| 268.2190909090909|
|   2025-10|      Hogar|            142|        169|17296.730000000007|102.34751479289945|
|   2025-10|   Juguetes|            107|        123| 7126.990000000002| 57.94300813008132|
|   2025-10|     Libros|            122|        136| 5889.839999999997| 43.30764705882351|
|   2025-10|      Moda|            185|        220|19826.029999999984| 90.11831818181811|
|   2025-11|Alimentación|            335|        558|11938.219999999994|21.394659498207876|
|   2025-11|     Belleza|            204|        248|13003.489999999994|52.433427419354814|
|   2025-11|    Deporte|            221|        293|          33844.93|115.51170648464164|
|   2025-11|Electrónica|            239|        308| 77674.11999999995|252.18870129870115|
|   2025-11|      Hogar|            246|        348| 36433.68000000001|104.69448275862071|
|   2025-11|   Juguetes|            187|        234|13007.039999999995|   55.585641025641|
|   2025-11|     Libros|            228|        306|12157.509999999997|39.730424836601294|
|   2025-11|      Moda|            303|        459|41268.909999999974| 89.91047930283219|
|   2025-12|Alimentación|            333|        562|12438.600000000006|22.132740213523142|
|   2025-12|     Belleza|            189|        231|12437.920000000006| 53.84380952380955|
|   2025-12|    Deporte|            227|        294|35945.420000000006|122.26333333333335|
|   2025-12|Electrónica|            219|        283| 68984.87000000004|243.76279151943476|
+----------+-----------+---------------+-----------+------------------+------------------+
only showing top 20 rows
```



Spark 3.5.2

Spark Master at spark://96f4bc00c336:7077

URL: spark://96f4bc00c336:7077
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 4 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20260216103219-172.31.24.200-7000	172.31.24.200:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103258-172.17.0.2-7000	172.17.0.2:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103336-172.31.29.126-7000	172.31.29.126:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

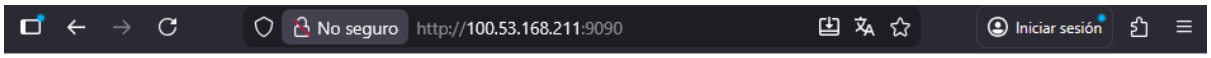
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (4)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260216105412-0003	Consulta B	3	1024.0 MiB		2026/02/16 10:54:12	root	FINISHED	32 s
app-20260216104920-0002	Consulta B	3	1024.0 MiB		2026/02/16 10:49:20	root	FINISHED	33 s
app-20260216104633-0001	Consulta B	3	1024.0 MiB		2026/02/16 10:46:33	root	FINISHED	2.1 min
app-20260216104302-0000	Consulta A	3	1024.0 MiB		2026/02/16 10:43:02	root	FINISHED	35 s

Consulta C – Detección de outliers de ventas por tienda

```
26/02/16 10:56:11 INFO CodeGenerator: Code generated in 43.536835 ms
26/02/16 10:56:11 INFO CodeGenerator: Code generated in 23.423579 ms
+--------+----------+------------------+------------------+------------------+----------+
|store_id|order_date|        ventas_dia|        media_30d|    desviacion_30d|is_outlier|
+--------+----------+------------------+------------------+------------------+----------+
|       1|2025-10-17|            715.27|            715.27|              NULL|     false|
|       1|2025-10-18|             115.5|           415.385| 424.1014341522556|     false|
|       1|2025-10-19|           1507.73|             779.5| 698.3338835399583|     false|
|       1|2025-10-20|            292.89|          657.8475| 619.9280589646404|     false|
|       1|2025-10-21| 794.5799999999999|           685.194| 540.3445697238013|     false|
|       1|2025-10-22|            920.74|724.4516666666667| 492.7725535951314|     false|
|       1|2025-10-23|387.84000000000003|676.3642857142858|467.48343795465604|     false|
|       1|2025-10-24|            421.45|             644.5|442.08971616953175|     false|
|       1|2025-10-25|1829.9199999999998|776.2133333333334| 571.9689870089112|     false|
|       1|2025-10-26|151.15999999999997|           713.708| 574.3412341650408|     false|
|       1|2025-10-27|115.47999999999999|659.3236363636363| 573.9471378319214|     false|
|       1|2025-10-28|            764.88|            668.12| 548.0848311911379|     false|
|       1|2025-10-29|1160.3400000000001|705.9830769230769| 542.2184652484736|     false|
|       1|2025-10-30|            693.36|705.0814285714285| 520.9575746177882|     false|
|       1|2025-10-31|            650.55|701.4459999999999|502.20469366583984|     false|
|       1|2025-11-01| 646.7600000000001|698.0281249999999|485.36841055317626|     false|
|       1|2025-11-02|            377.48|679.1723529411764|  476.343122621832|     false|
|       1|2025-11-03|             157.1|650.1683333333333| 478.2235308286652|     false|
|       1|2025-11-04|           1619.46|701.1836842105262| 515.2097336167576|     false|
|       1|2025-11-05| 591.3199999999999|695.6904999999999|502.06970477975625|     false|
+--------+----------+------------------+------------------+------------------+----------+
only showing top 20 rows
```



Spark Master at spark://96f4bc00c336:7077

URL: spark://96f4bc00c336:7077
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 5 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20260216103219-172.31.24.200-7000	172.31.24.200:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103258-172.17.0.2-7000	172.17.0.2:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103336-172.31.29.126-7000	172.31.29.126:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

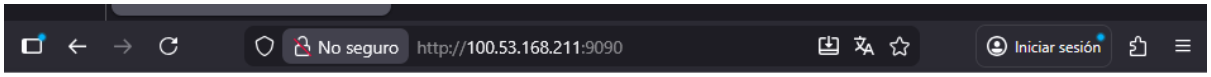
Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260216105550-0004	Consulta C	3	1024.0 MiB		2026/02/16 10:55:50	root	FINISHED	30 s
app-20260216105412-0003	Consulta B	3	1024.0 MiB		2026/02/16 10:54:12	root	FINISHED	32 s
app-20260216104920-0002	Consulta B	3	1024.0 MiB		2026/02/16 10:49:20	root	FINISHED	33 s
app-20260216104633-0001	Consulta B	3	1024.0 MiB		2026/02/16 10:46:33	root	FINISHED	2.1 min
app-20260216104302-0000	Consulta A	3	1024.0 MiB		2026/02/16 10:43:02	root	FINISHED	35 s

DAG

```
26/02/16 10:57:31 INFO CodeGenerator: Code generated in 9.607868 ms
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
+---+
only showing top 5 rows

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- InMemoryTableScan [id#0L]
      +- InMemoryRelation [id#0L], StorageLevel(disk, memory, deserialized, 1 replicas)
            +- *(1) Range (0, 100000, step=1, splits=2)
```



Spark Master at spark://96f4bc00c336:7077

URL: spark://96f4bc00c336:7077
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 6 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20260216103219-172.31.24.200-7000	172.31.24.200:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103258-172.17.0.2-7000	172.17.0.2:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20260216103336-172.31.29.126-7000	172.31.29.126:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

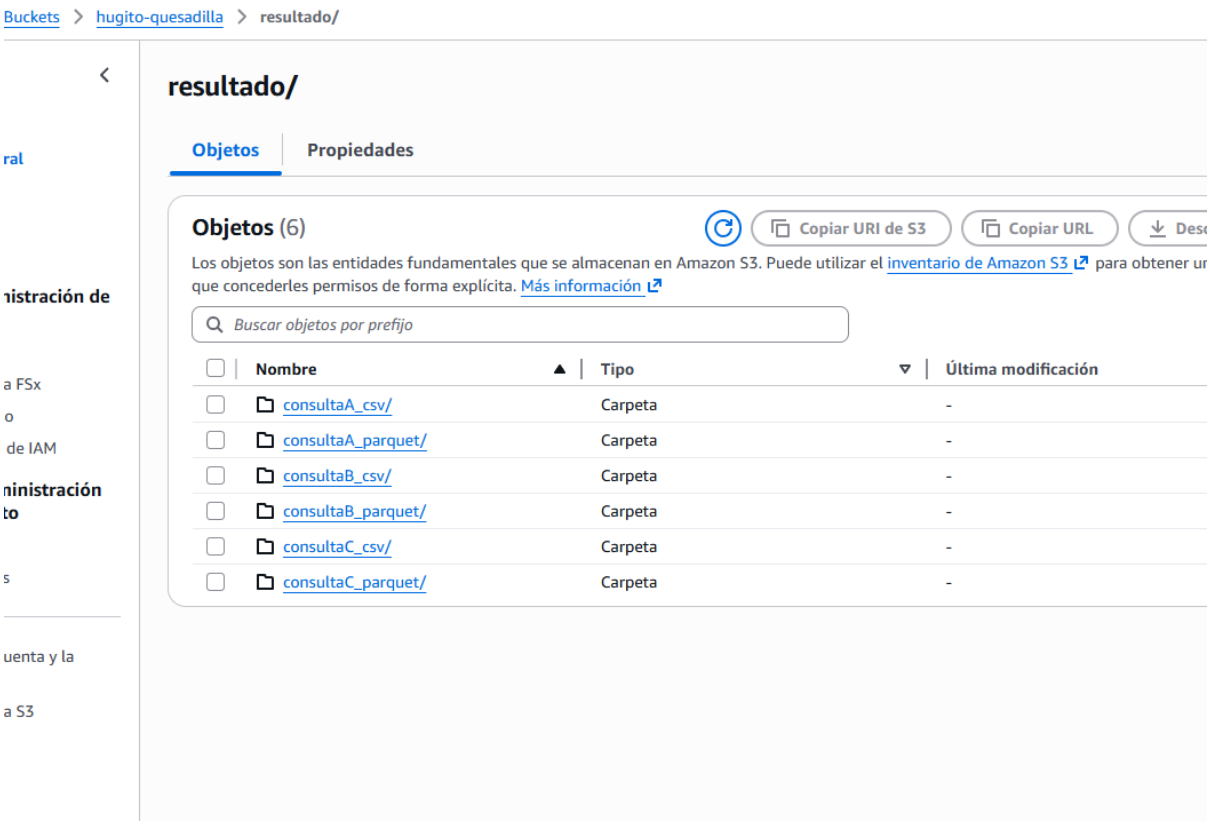
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

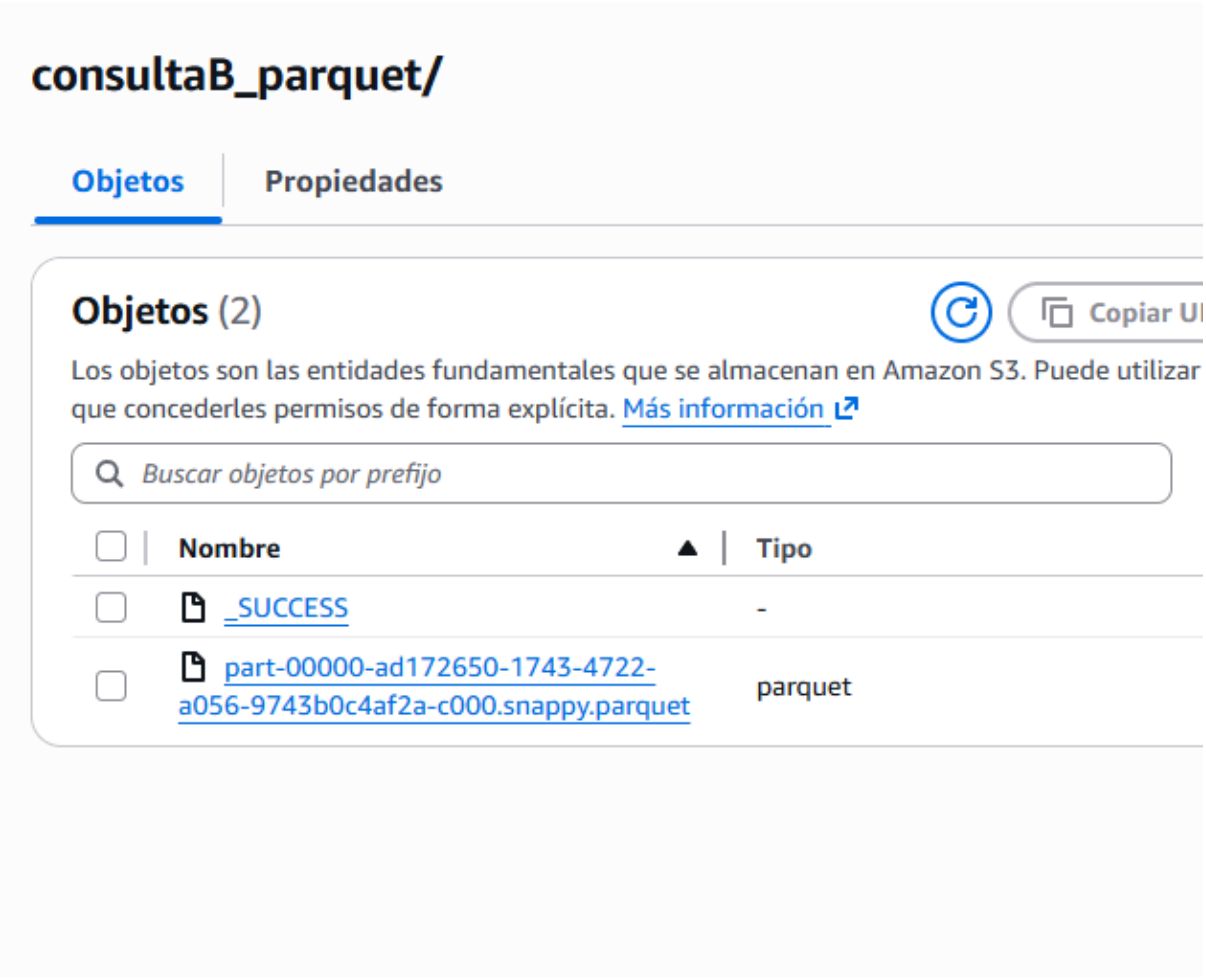
Completed Applications (6)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260216105720-0005	Consulta A	3	1024.0 MiB		2026/02/16 10:57:20	root	FINISHED	10 s
app-20260216105550-0004	Consulta C	3	1024.0 MiB		2026/02/16 10:55:50	root	FINISHED	30 s
app-20260216105412-0003	Consulta B	3	1024.0 MiB		2026/02/16 10:54:12	root	FINISHED	32 s
app-20260216104920-0002	Consulta B	3	1024.0 MiB		2026/02/16 10:49:20	root	FINISHED	33 s
app-20260216104633-0001	Consulta B	3	1024.0 MiB		2026/02/16 10:46:33	root	FINISHED	2.1 min
app-20260216104302-0000	Consulta A	3	1024.0 MiB		2026/02/16 10:43:02	root	FINISHED	35 s

No le cambie por descuedo el nombre a la consulta cuando hice el código de DAG.py

7. Resultados en S3





8. Problemas y soluciones

A lo largo del trabajo me he visto con bastantes problemas:

- Maquina EC2. El problema principal que me ha dado más problemas fue la configuraciónde Spark en una máquina EC2, ya que se trató siempre de que se instalará todo a través de un script que se lanzara con la creación del EC2, ya que configurándolo a mano haría que generar cada contenedor tomara mucho tiempo, aparte de que errores iniciales al principio de todo provocaría mucho atraso a la hora de irlos solucionando en cada instancia. Para solucionarlo se optó al final por instalar docker en todas las instancias y a través de ligeros cambios en el Dockerfile proporcionado configurar todas las instancias, ello haría que los cambios fueran fácilmente replicables.
- Sistema operativo de las instancias. Al principio se trató de instalar docker o Spark en una instancia con SO Ubuntu, pero la configuración de estos me dio muchos problemas. Para solucionarlo se optó al final por un Amazon Linux, ya que la instalación de docker en sistemas basados en Redhat son muy simples y fáciles.

- Boto3. Cuando todas las máquinas ya estaban funcionando y se lanzó el primer job, se cayó en la cuenta de que no había añadido la librería boto3 para las conexiones al bucket a los nodos worker. Esto me comió bastante tiempo entre que interpretaba trazabilidades y luego añadía la librería a los contenedores. Además de que su usó complicaba las cosas, ya que aunque le pasara los datos del bucket, con que credenciales de AWS tenía que usar, etc, no me las cogía bien, y tenía que estar constantemente pasándolas como parámetros. Al final la soluciones para boto3 que fui aplicando, solo lo hice porque el job dado para generar los datos así lo pedía, pero el resto de las consultas las he hecho con spark y el uso del paquete s3a.
- Credenciales de AWS. Todas las conexiones al bucket me eran denegadas por falta de credenciales, ya las añadiera como parámetros del job al lanzarlo o por código. Para solucionarlo use en la terminal de la máquina de EC2 el comando aws configure, que me permitía cargar en variables de entorno los id de aws y las credenciales, así como la región, y al cargar el contenedor que lanzo los añado dentro del contenedor, así dejó de darme problemas.
- Puertos drivers Spark. Spark a parte de todos los puertos que usa tambien requiere de puertos a partir del 3000 oara mandar los jobs al master, para gestionar drivers, etc, hasta encontrar los puertos que tenía que abrir me tire un rato largo.
- Paquete s3a. Este requería que todos los worker tuvieran unos paquetes de java específicos, como temía que los contenedores worker pesaran mucho y me diera problemas de espacio, se optó por añadirlos como parámetros al lanzar los jobs. Ahora tarda un poco más en lanzar cada job pero que quito d problemas de espacio.
- Ip privada en Spark. Como cada contenedor está en una instancia distinta, aunque me conectará un nodo, el master no sabe quién se está logeando, por lo que me asigna por defecto a todos los worker la misma ip (172.0.0.2), dándome errores de re-logeo, que no me dejaban añadir más de un worker, al finalo enncontro el parámetro a la hora de lanzar el contenedor –net=host, para que el master les asignara la ip privada de la instancia EC2.

9. Conclusiones

En retrospectiva no es un trabajo difícil una vez sabes lo que estás haciendo. Pero al principio no sabía ni por donde empezar, y difícilmente puede decir que arrancaba sabiendo lo que estaba haciendo.

Como dijiste este era un trabajo para machacar, que ha sido dificil, pero vamos mucho más preparados para montar un cluster de Spark de lo que hiría si me lo hubiera preparado para un examen.

Lo que más se me ha complicado es montar el cluster, aunque el lanzamiento de los jobs me ha tenido un buen rato leyendo trazabilidades de error, al menos ahí sabía que estaba fallando.