



Spark Data Streaming with MongoDB



30 DE MAYO DE 2024

Rim El abrouki

Spark Data Streaming with MongoDB

Tabla de contenido

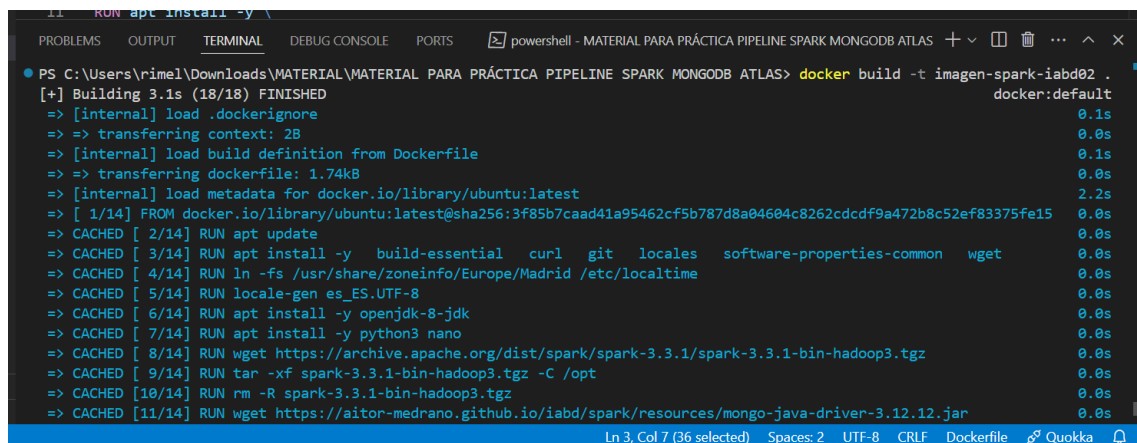
Introducción	2
Creación de Cluster en MongoAtlas y Conexión con MongoDB ATLAS	2
Crear Base de datos en MongoDB ATLAS	5
Alojamiento de los archivos *.csv.	7
Script para lanzar en Apache Spark (pyspark.sh o submit-spark.sh).....	9
Sesión de Spark.	9
Verificar si Data está Streaming.	11
Escribir datos en la consola.	11
Agregación en marco de datos de Spark	12
Consultas SQL sobre los datos de Streaming.....	12
Escribir consultas.	12
Escribiendo los datos de Streaming en MongoDB.....	13
Escribiendo Stream.	14
Conclusión	16

Introducción

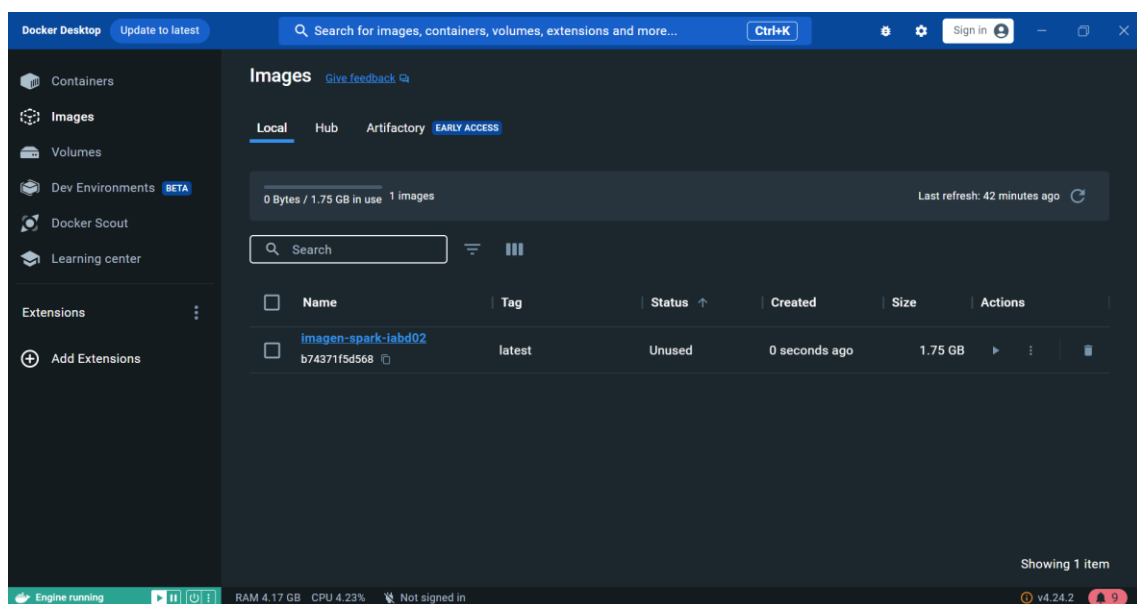
En este trabajo, exploraremos cómo usar Apache Spark para la transmisión de datos en tiempo real con Python, y cómo almacenar estos datos en MongoDB. Spark ofrece una API flexible y escalable para el streaming de datos, superando las limitaciones de latencia de Hadoop. Nuestro objetivo es demostrar cómo transmitir archivos CSV en tiempo real utilizando Spark SQL, ejecutar consultas y persistir los datos en MongoDB, ilustrando la configuración de un clúster de Spark con Docker y la integración con MongoDB Atlas.

Creación de Cluster en MongoAtlas y Conexión con MongoDB ATLAS

Creamos la imagen en el Docker:



```
PS C:\Users\rime1\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS> docker build -t imagen-spark-iabd02 .
[+] Building 3.1s (18/18) FINISHED
=> [internal] load .dockerignore                                docker:default 0.1s
=> => transferring context: 2B                                  0.0s
=> [internal] load build definition from Dockerfile             0.1s
=> => transferring dockerfile: 1.74kB                            0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 2.2s
=> [ 1/14] FROM docker.io/library/ubuntu:latest@sha256:3f85b7caad41a95462cf5b787d8a04604c8262cdcdf9a472b8c52ef83375fe15 0.0s
=> CACHED [ 2/14] RUN apt update                                0.0s
=> CACHED [ 3/14] RUN apt install -y build-essential curl git locales software-properties-common wget 0.0s
=> CACHED [ 4/14] RUN ln -fs /usr/share/zoneinfo/Europe/Madrid /etc/localtime 0.0s
=> CACHED [ 5/14] RUN locale-gen es_ES.UTF-8                    0.0s
=> CACHED [ 6/14] RUN apt install -y openjdk-8-jdk              0.0s
=> CACHED [ 7/14] RUN apt install -y python3 nano                0.0s
=> CACHED [ 8/14] RUN wget https://archive.apache.org/dist/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz 0.0s
=> CACHED [ 9/14] RUN tar -xf spark-3.3.1-bin-hadoop3.tgz -C /opt 0.0s
=> CACHED [10/14] RUN rm -R spark-3.3.1-bin-hadoop3.tgz         0.0s
=> CACHED [11/14] RUN wget https://aitor-madrano.github.io/iabd/spark/resources/mongo-java-driver-3.12.12.jar 0.0s
```

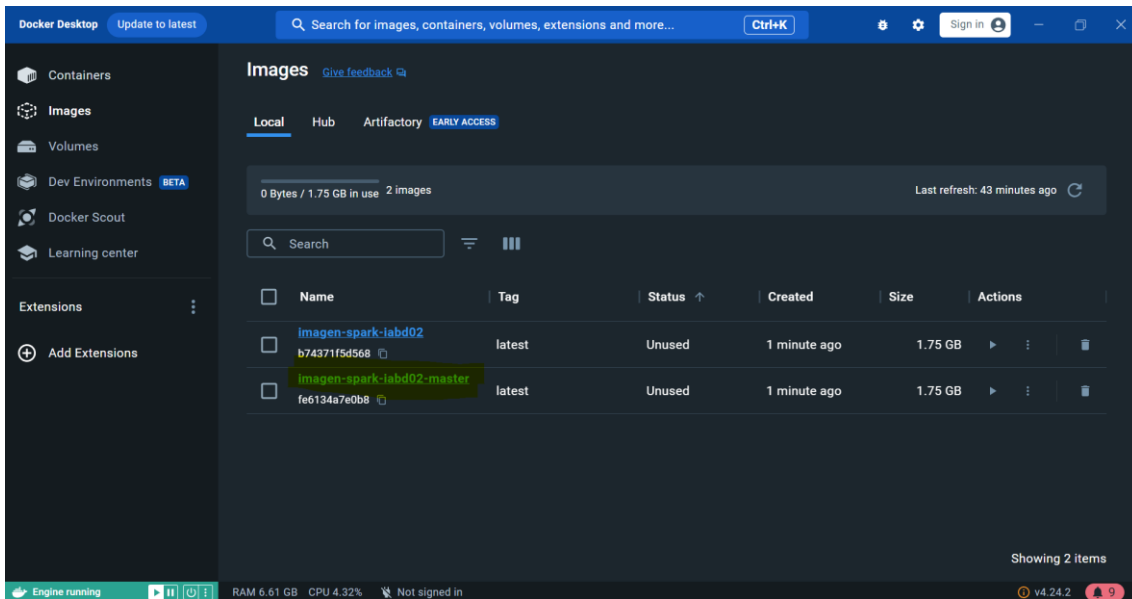


Creamos la imagen del Master:

```
11 # Esto depende de cómo lo hayas configurado en tu Dockerfile base.

PS C:\Users\rimel\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS\imagen_master> docker build -t imagen-spark-iabd02-master .
[+] Building 0.3s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 641B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/imagen-spark-iabd02:latest
=> [1/1] FROM docker.io/library/imagen-spark-iabd02:latest
=> exporting to image
=> => exporting layers
=> => writing image sha256:fe6134a7e0b8b4db8b15d56d9ea13a067918fb2b3ca72419ff6dca014e4bfb43
=> => naming to docker.io/library/imagen-spark-iabd02-master

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\rimel\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS\imagen_master>
```

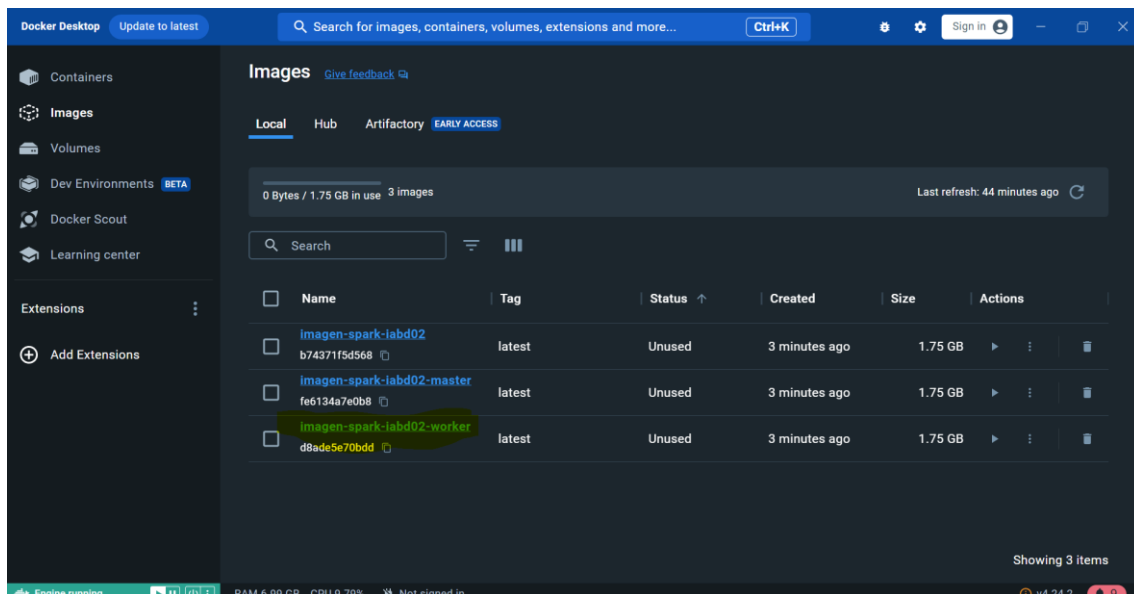


Creamos las imágenes de los Workers:

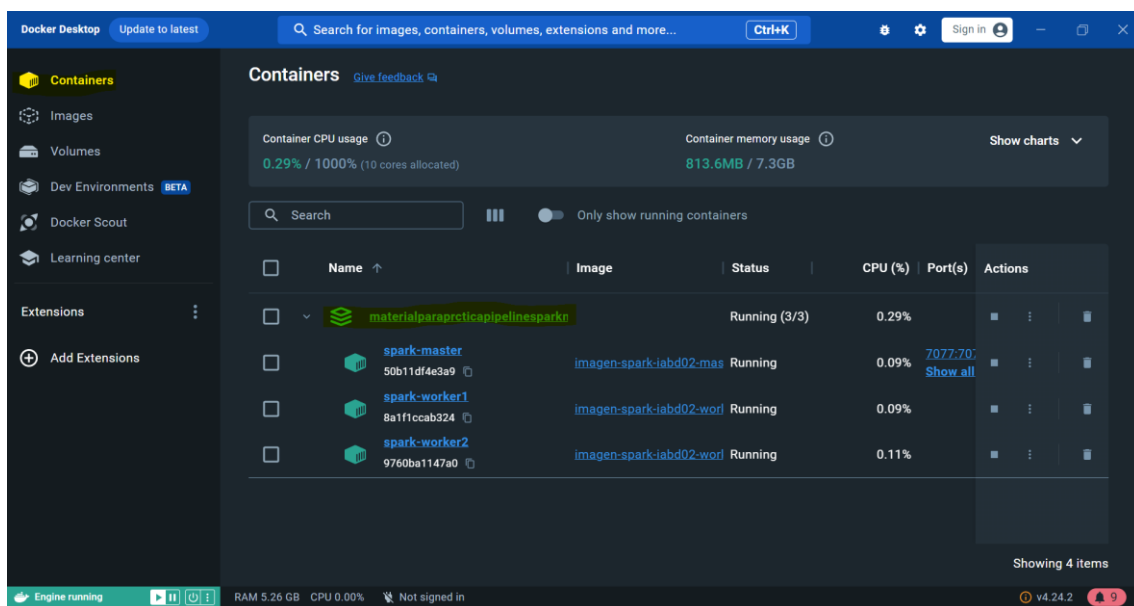
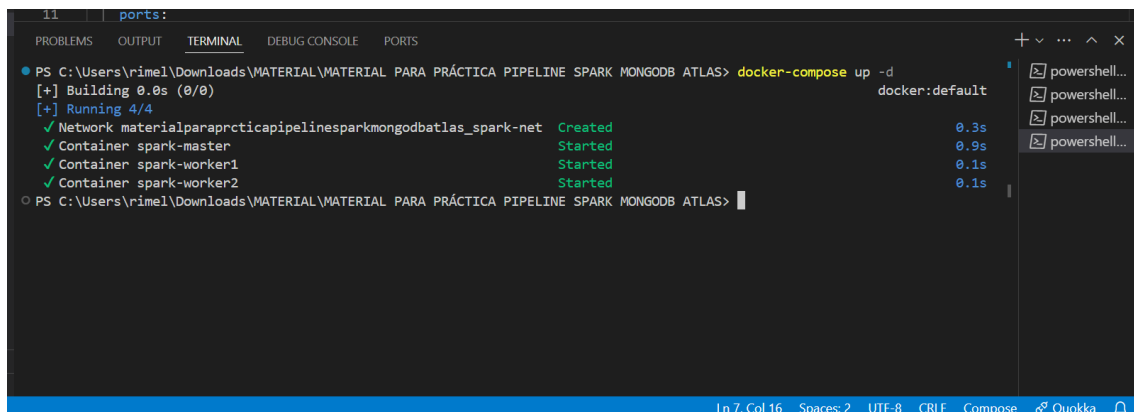
```
11

PS C:\Users\rimel\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS\imagen_worker> docker build -t imagen-spark-iabd02-worker .
[+] Building 0.2s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 760B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/imagen-spark-iabd02:latest
=> [1/1] FROM docker.io/library/imagen-spark-iabd02:latest
=> exporting to image
=> => exporting layers
=> => writing image sha256:d8ade5e70bdd783b6b4486c42d0c3dcf191f5953d90aa21b78034f8bed0aaf71
=> => naming to docker.io/library/imagen-spark-iabd02-worker

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\rimel\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS\imagen_worker>
```



Y lanzamos los contenedores desde Docker-compose:

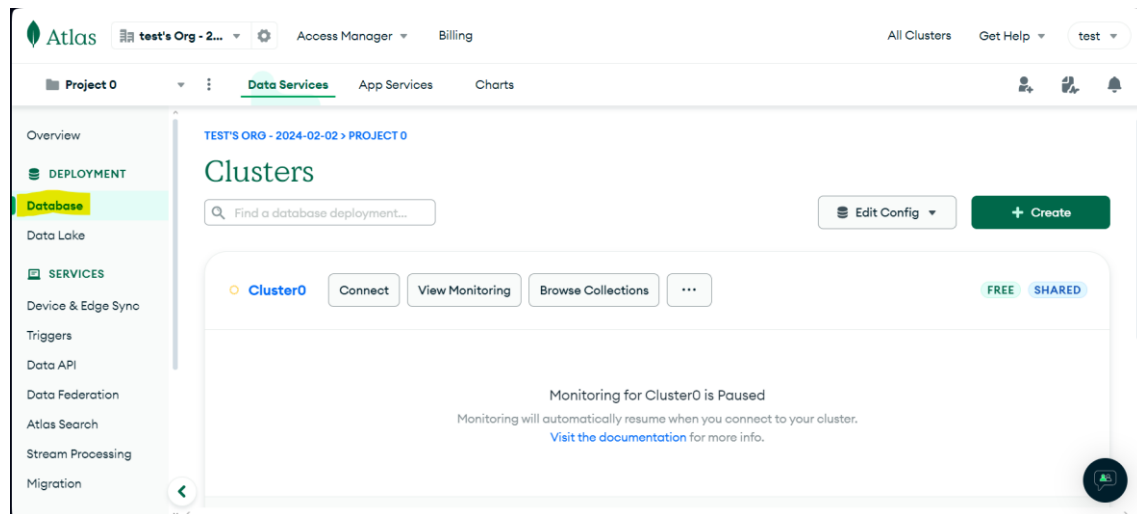


Aquí tenemos los containers del Master y dos worker.

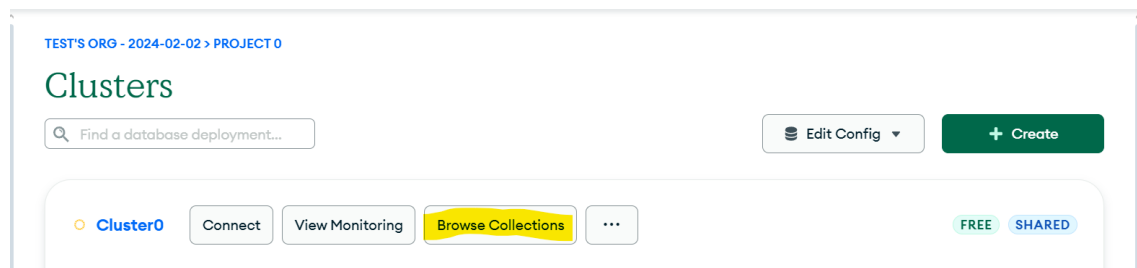
Crear Base de datos en MongoDB ATLAS

En mongodb Atlas, creamos una Base de Datos:

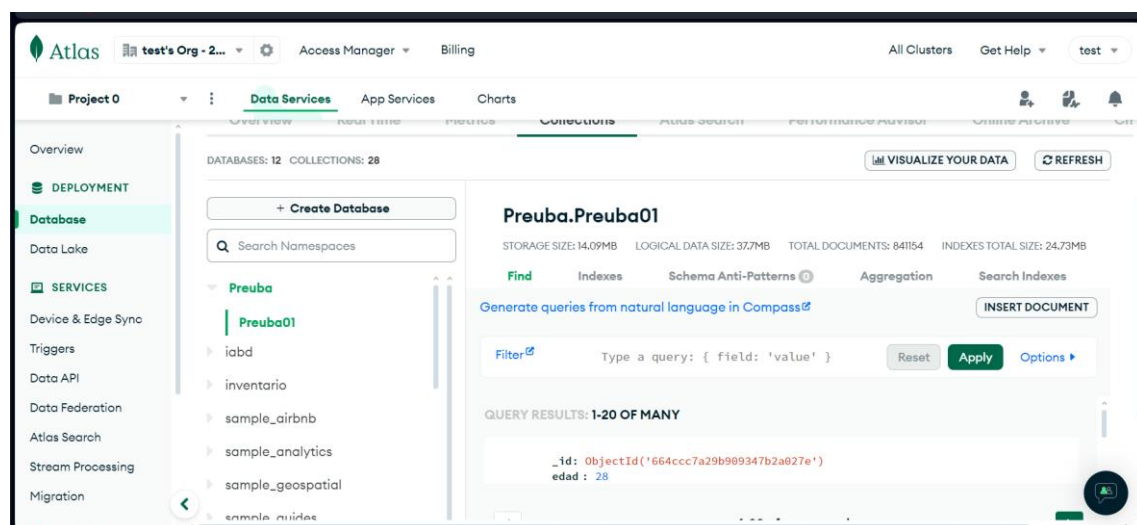
Entramos en el DataBase:



Y seleccionamos el Brows Collections:

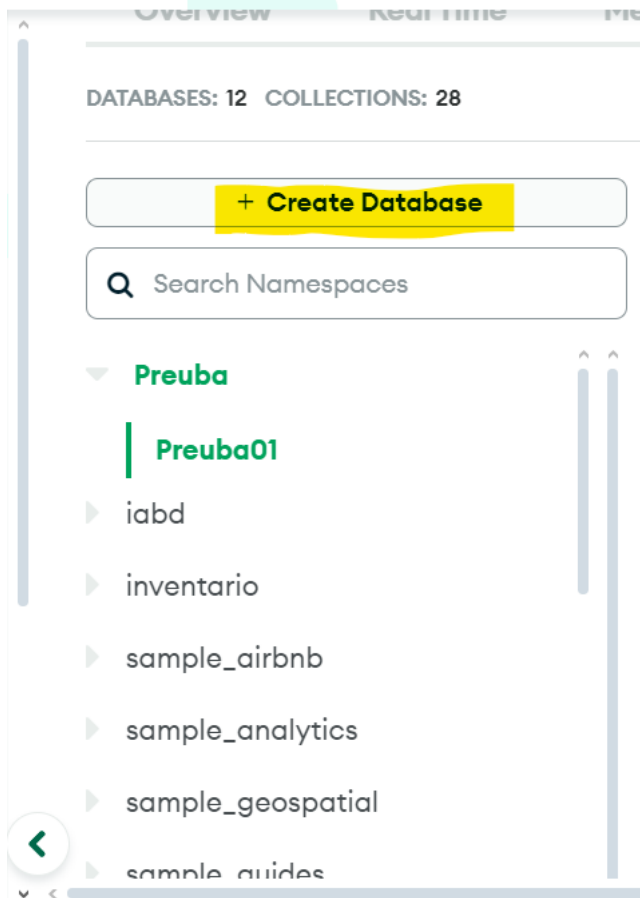


Pues aquí tenemos todas las bases de datos y sus colecciones:

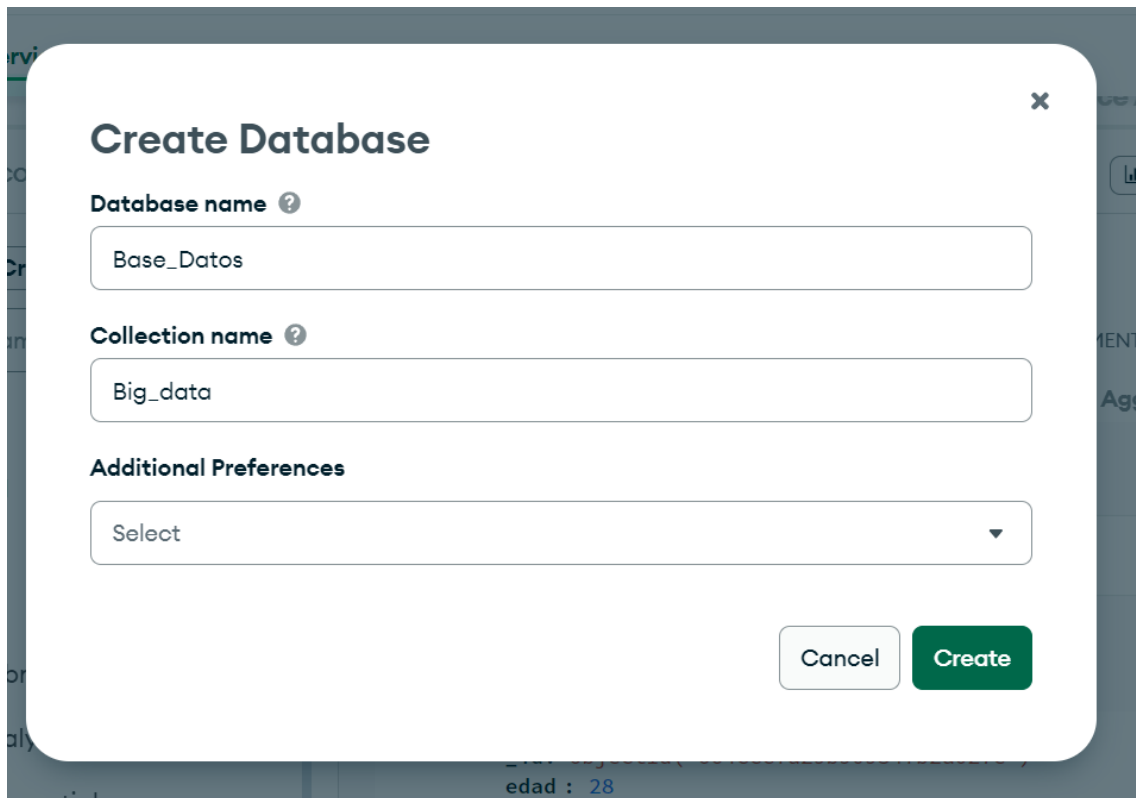


Creamos una nueva:

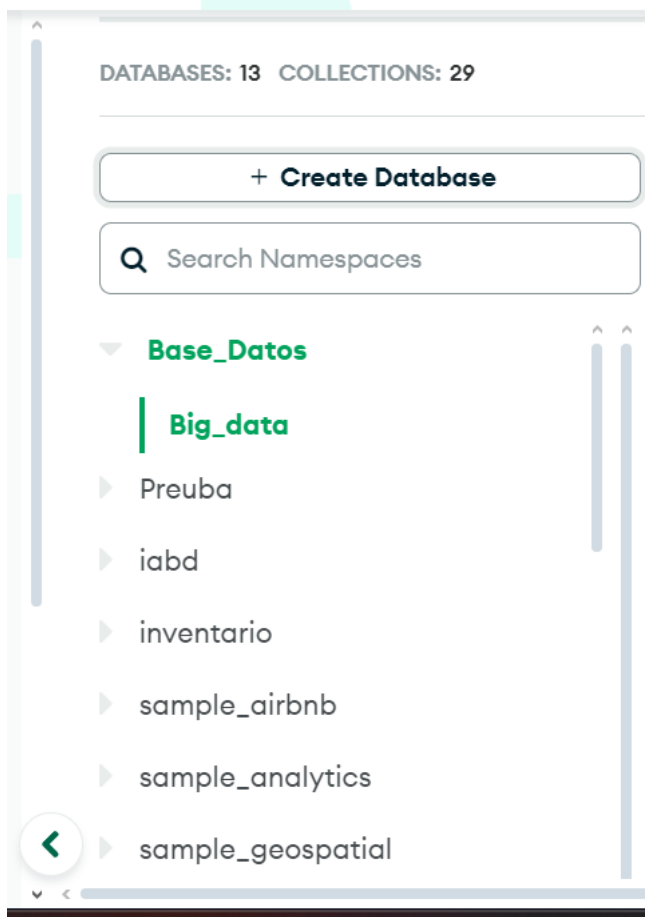
Pulsamos el Ceate DataBase:



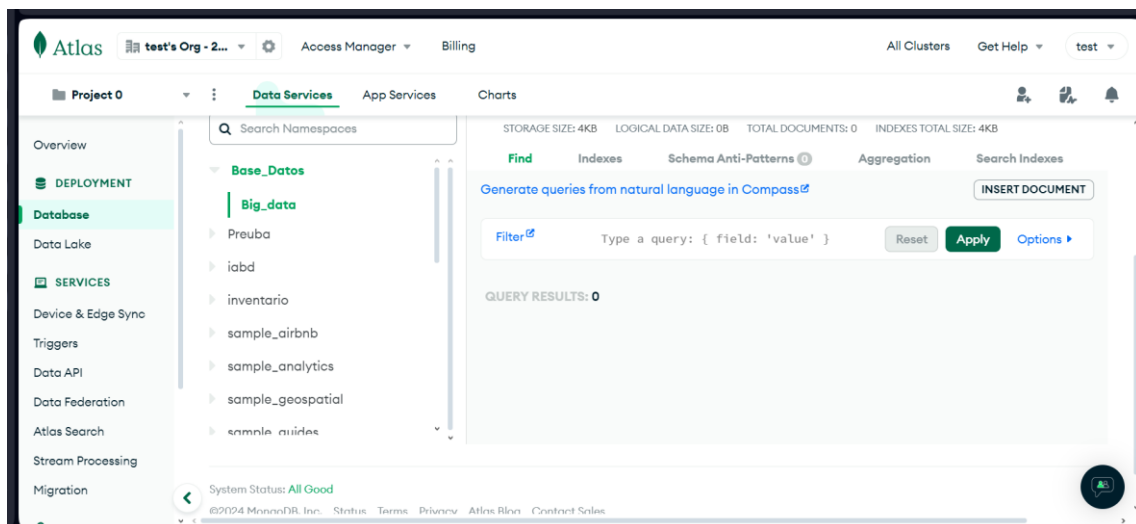
Y ponemos el nombre de la Base de Datos mas y nombre de la colección;



Y pulsamos a créate ya lo tenemos:



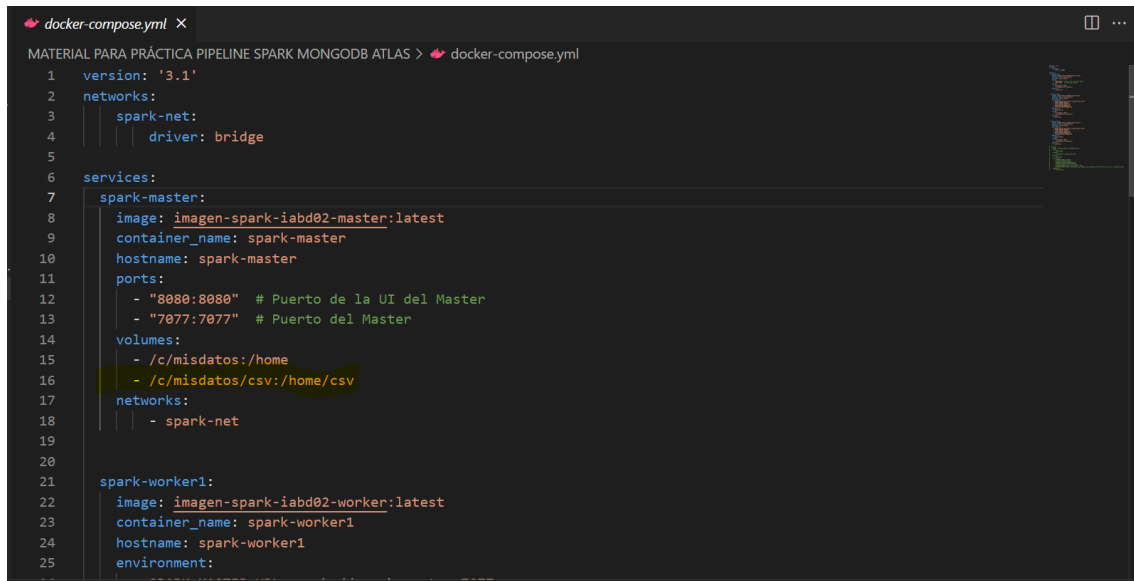
Ahora como nos vemos está vacía:



Alojamiento de los archivos *.csv.

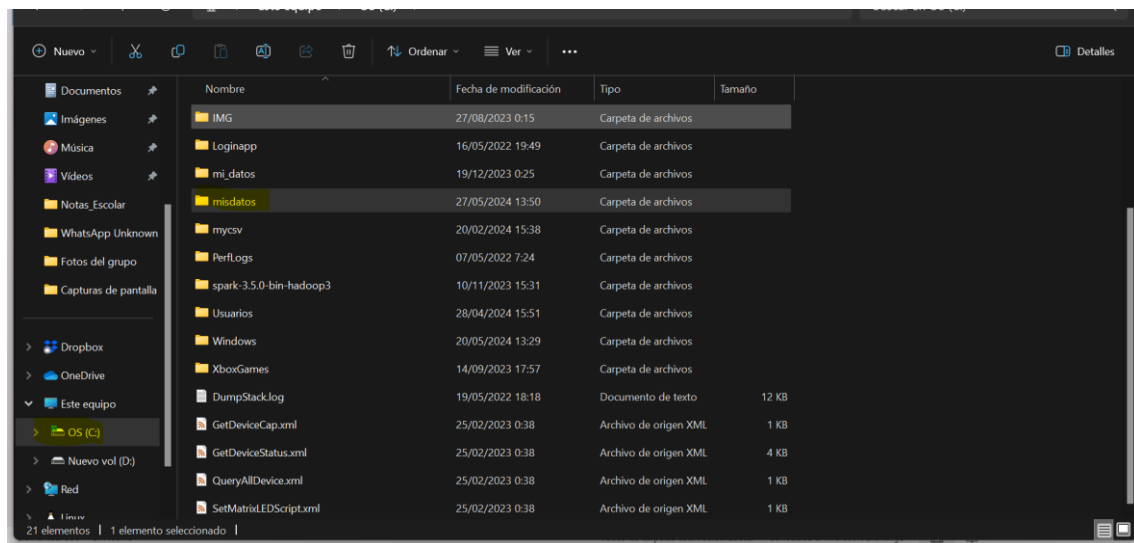
Utilizamos una carpeta alojada en nuestro disco duro del equipo físico: /C/misdatos/csv, para alojar los archivos *.csv que queremos se “sincronicen” con nuestra colección “Big_data” de nuestra base de datos: “Base_Datos”, que se encuentra alojada en nuestro MongoDB Atlas. Esta carpeta /C/misdatos/csv,

estará “linkada” al directorio /home/csv de nuestros nodos del cluster de Apache Spark (fijarse en el “docker-compose.yml”),

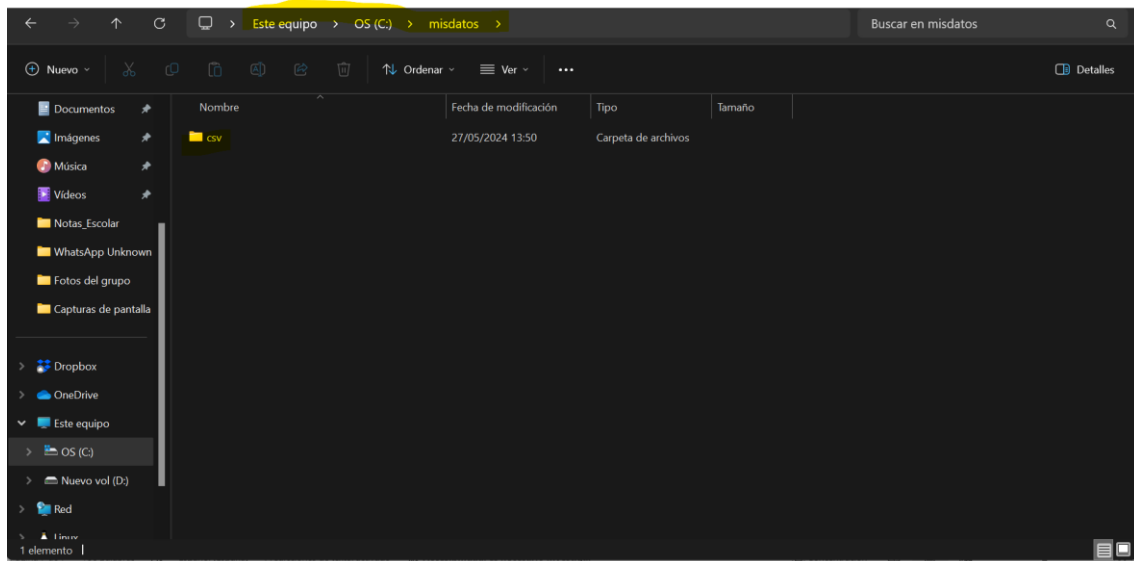


```
1 version: '3.1'
2 networks:
3   spark-net:
4     driver: bridge
5
6 services:
7   spark-master:
8     image: imagen-spark-iabd02-master:latest
9     container_name: spark-master
10    hostname: spark-master
11    ports:
12      - "8080:8080" # Puerto de la UI del Master
13      - "7077:7077" # Puerto del Master
14    volumes:
15      - /c/misdatos:/home
16      - /c/misdatos/csv:/home/csv
17    networks:
18      - spark-net
19
20
21   spark-worker1:
22     image: imagen-spark-iabd02-worker:latest
23     container_name: spark-worker1
24     hostname: spark-worker1
25     environment:
```

de tal manera que este directorio local /C/misdatos/csv, estará sincronizado con el directorio “/home/csv” de los nodos de nuestro cluster Spark.



Dentro de la carpeta misdatos tenemos el archivo csv:



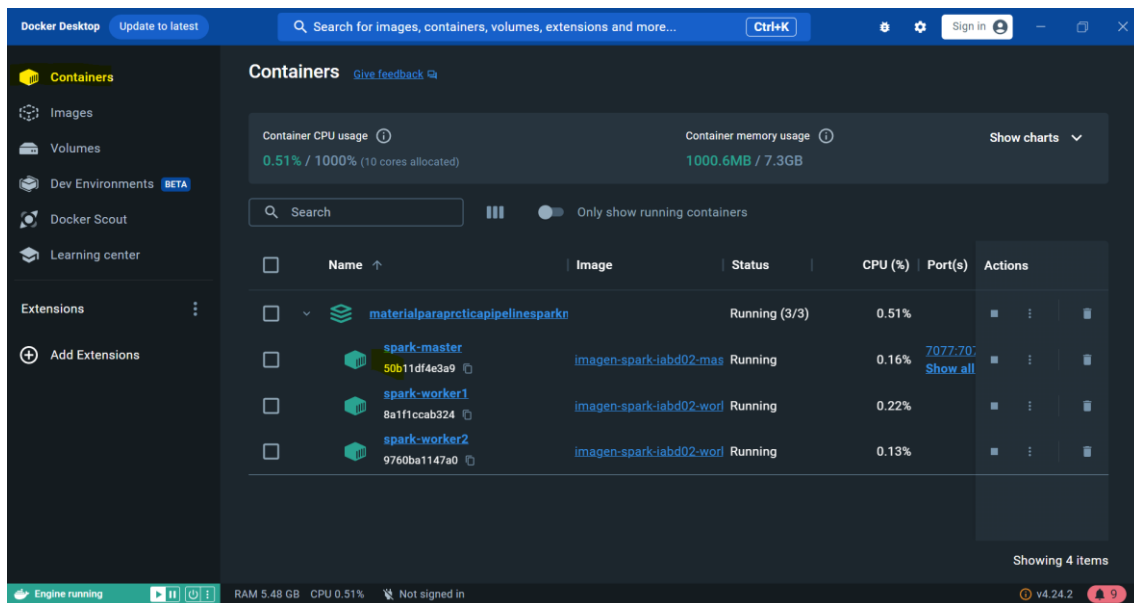
Script para lanzar en Apache Spark (pyspark.sh o submit-spark.sh)

Sesión de Spark.

lanzamos pyspark para ir probando los comandos que tenemos en nuestro script de python y así poder hacer nuestro pipeline entre /c/misdatos/csv y MongoDB Atlas a través de nuestro cluster Apache Spark en streaming.

```
PS C:\Users\rime1\Downloads\MATERIAL\MATERIAL PARA PRÁCTICA PIPELINE SPARK MONGODB ATLAS\imagen_master> docker exec -it 50b /bin/bash
root@spark-master:/opt/spark-3.3.1-bin-hadoop3/bin#
```

El 50b es los tres primeros dígitos del contenedor Master



Ejecutamos pyspark.sh para lanzar los comandos de nuestro script de Python

```

/bin/bash
root@spark-master:/opt/spark-3.3.1-bin-hadoop3/bin# pyspark --conf spark.executor.memory=2g --conf spark.executor.cores=2
Python 3.12.3 (main, Apr 10 2024, 05:33:47) [GCC 13.2.0] 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).
24/05/27 14:29:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java class
es where applicable
Welcome to

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

version 3.3.1

```

Primero se debe crear una sesión de Spark para trabajar con Spark. La sesión de Spark define dónde está corriendo nuestro nodo master de Spark, cuántos núcleos va a usar, etc.

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

Spark context Web UI available at http://spark-master:4040
Spark context available as 'sc' (master = local[*], app id = local-1716812956430).
SparkSession available as 'spark'.
>>>
>>>
>>> from pyspark.sql import SparkSession
>>>
>>> spark = SparkSession.builder \
...   .master("spark://spark-master:7077") \
...   .appName("PipelineApacheSpark") \
...   .getOrCreate()
24/05/27 14:31:24 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>>
>>>

```

Ahora tenemos que crear un esquema en Spark. Esto se puede hacer usando Spark SQL API.

```

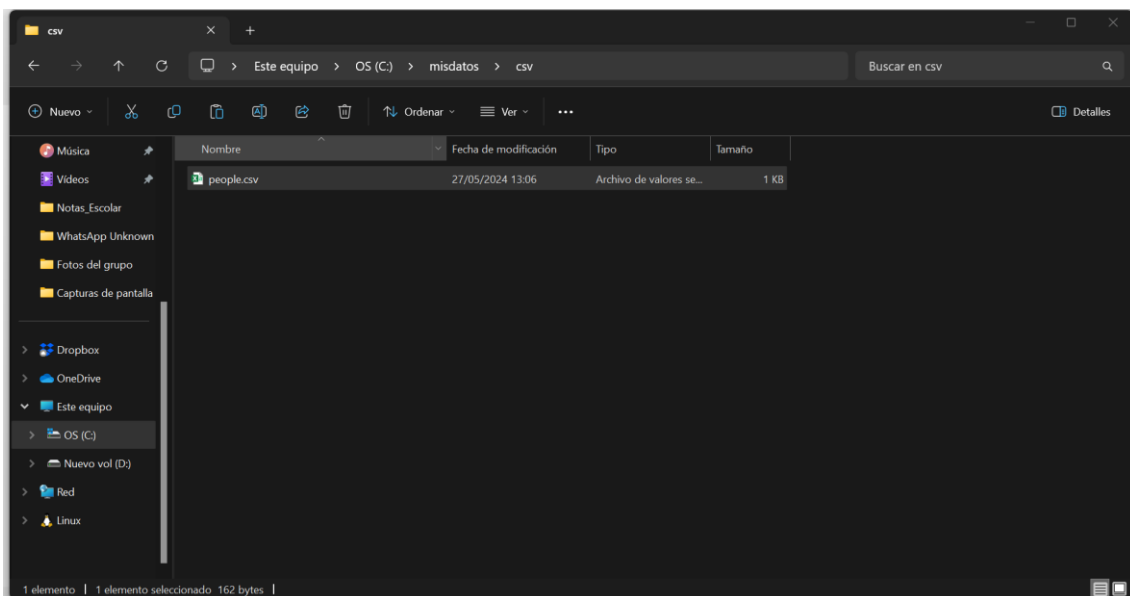
>>>
>>>
>>> from pyspark.sql.types import FloatType, StructField, StructType, StringType, IntegerType
>>>
>>> schemaIABD = StructType([StructField("edad", IntegerType(), True), StructField("nombre", StringType(), True), StructField("profesion", StringType(), True)])
>>>
>>> []

```

Ln 7, Col 16 Spaces: 2 UTF-8 CRLF Compose

Después de crear la sesión y el esquema podemos comenzar nuestra “pipeline/streaming” de lectura, para crear nuestro dataframe. Podemos transmitir todos los archivos en una carpeta o un solo archivo.

Mantendremos nuestro archivo people.csv en el directorio /c/misdatos/csv, para que así tengan acceso todos los nodos de mi cluster a ese archivo.



Verificar si Data está Streaming.

```

>>>
>>>
>>> df = spark.readStream.schema(schemaIABD).option("maxfilesperTrigger",1).csv("/home/csv", header = True)
>>> print(df.isStreaming)
True
>>> []

```

Escribir datos en la consola.

Antes de intentar “escribir” en nuestra colección de MongoDB Atlas, intentaremos escribir en la consola, es decir, vamos a aprobar que funciona correctamente el streaming, probando la salida por consola. El `writeStream` se utiliza para escribir stream, En este caso, escribimos nuestra secuencia en la consola usando el método de apéndice como modo de salida.

Y esto es lo que nos sale por consola:

```
True
>>> df.writeStream.format("console").outputMode("append").start().awaitTermination()
24/05/27 14:36:42 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query
didn't fail: /tmp/temporary-274aabd7-9aab-4f7d-b83f-60b7cac4db5b. If it's required to delete it under any circumstances, p
lease set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder
is best effort.
24/05/27 14:36:42 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets a
nd will be disabled.
-----
Batch: 0
-----
+-----+-----+
|edad|  nombre|  profesion|
+-----+-----+
| 28|    Juan| Ingeniero|
| 35|   Maria|electricidad|
| 42|   Pedro|  Profesor|
| 42|   Julian|  Profesor|
| 42|    Ana|  Profesor|
| 43| Jose Luis|  Profesor|
| 45|    Luis|  Profesor|
+-----+-----+
|
```

Cada vez que grabe un nuevo *.csv intentará leerlo para cargar el dataframe con el formato que le he especificado y me sacará por consola los registros, tal y como le he especificado en el comando.

Agregación en marco de datos de Spark

vamos a probar una agregación en Spark SQL. Nuestro objetivo es contar el número de “profesiones” distintas.

```
>>> dfc = df.groupBy("profesion").count()
>>> dfc.writeStream.outputMode("complete").format("console").start().awaitTermination()
24/05/27 14:39:26 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query
didn't fail: /tmp/temporary-3122187e-a9c6-4d1d-90b0-143856263f7f. If it's required to delete it under any circumstances, p
lease set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder
is best effort.
24/05/27 14:39:26 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets a
nd will be disabled.
-----
Batch: 0
-----
+-----+-----+
|  profesion|count|
+-----+-----+
| Ingeniero|    1|
|  Profesor|    5|
|electricidad|    1|
+-----+-----+
|
```

Consultas SQL sobre los datos de Streaming

Para realizar consultas SQL, primero necesitamos crear una vista temporal que actuará como un nombre de tabla.

Escribir consultas.

Seleccionar todos los nombres y edad de las personas que tienen de profesión “Profesor”.

Sacamos el resultado de ese dataframe (dfclean) por nuestra consola

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS

>>>
>>> df.createOrReplaceTempView("tempdf")
>>> dfclean = spark.sql("Select nombre,edad FROM tempdf where profesion == 'Profesor'")
>>> dfclean.writeStream.outputMode("append").format("console").start().awaitTermination()
24/05/27 14:40:55 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query
didn't fail: /tmp/temporary-19cf4ac8-c414-49ca-bb1d-d20d162d4b10. If it's required to delete it under any circumstances, p
lease set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder
is best effort.
24/05/27 14:40:55 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets a
nd will be disabled.
-----
Batch: 0
-----
+-----+-----+
| nombre|edad|
+-----+-----+
|  Pedro|  42|
|  Julian| 42|
|   Ana|  42|
|Jose Luis| 43|
|   Luis| 45|
+-----+-----+
|
```

Escribiendo los datos de Streaming en MongoDB.

La secuencia que estamos escribiendo en nuestra consola se puede escribir fácilmente en nuestro MongoDB Atlas.

- Para lanzar pyspark.sh tengo que especificar que la librería necesaria para para conectar con MongoDB. Con esto lo bajará del repositorio MAVEN.

```
5
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS

root@spark-master:/opt/spark-3.3.1-bin-hadoop3/bin# ./pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:10.1.1
Python 3.12.3 (main, Apr 10 2024, 05:33:47) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
:: loading settings :: url = jar:file:/opt/spark-3.3.1-bin-hadoop3/jars/ivy-2.5.0.jar!/org/apache/ivy/core/settings/ivysett
ings.xml
Ivy Default Cache set to: /root/.ivy2/cache
The jars for the packages stored in: /root/.ivy2/jars
org.mongodb.spark#mongo-spark-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-8c436664-3863-4626-ac3d-13f15a489fae;1.0
   confs: [default]
   found org.mongodb.spark#mongo-spark-connector_2.12;10.1.1 in central
   found org.mongodb#mongodb-driver-sync;4.8.2 in central
   [4.8.2] org.mongodb#mongodb-driver-sync;[4.8.1,4.8.99)
   found org.mongodb#bson;4.8.2 in central
   found org.mongodb#mongodb-driver-core;4.8.2 in central
   found org.mongodb#bson-record-codec;4.8.2 in central
downloading https://repo1.maven.org/maven2/org/mongodb/spark/mongo-spark-connector_2.12/10.1.1/mongo-spark-connector_2.12-1
0.1.1.jar ...
[SUCCESSFUL ] org.mongodb.spark#mongo-spark-connector_2.12;10.1.1!mongo-spark-connector_2.12.jar (71ms)
downloading https://repo1.maven.org/maven2/org/mongodb/mongodb-driver-sync/4.8.2/mongodb-driver-sync-4.8.2.jar ...
[SUCCESSFUL ] org.mongodb#mongodb-driver-sync;4.8.2!mongodb-driver-sync.jar (59ms)
downloading https://repo1.maven.org/maven2/org/mongodb/bson/4.8.2/bson-4.8.2.jar ...
[SUCCESSFUL ] org.mongodb#bson;4.8.2!bson.jar (79ms)
```

Los comandos iniciales son los mismos que para escribir en consola:

```
>> from pyspark.sql.types import FloatType, StructField, StructType, StringType, IntegerType
>>
>> schemaIABD = StructType([StructField("edad", IntegerType(), True), StructField("nombre", StringType(), True), StructField("profesion", StringType(), True)])
>>
>> # Guardamos nuestro archivo CSV en una carpeta llamada /c/misdatos/csv
>> # * maxfilesperTrigger → El número de archivos nuevos que se van a tener en cuenta # en cada microlote. Su valor predeterminado es 1000.
>> # * readStream → Se utiliza para leer datos de streaming.
>>
>> df = spark.readStream.schema(schemaIABD).option("maxfilesperTrigger",1).csv("/home/csv", header = True)
>>
>> # Verificar si los datos se están transmitiendo
>>
>> print(df.isStreaming)
true
>>
```

Escribiendo Stream.

“**write_row**” se llamará a cada lote de data y escribirá nuestros datos en el MongoDB.

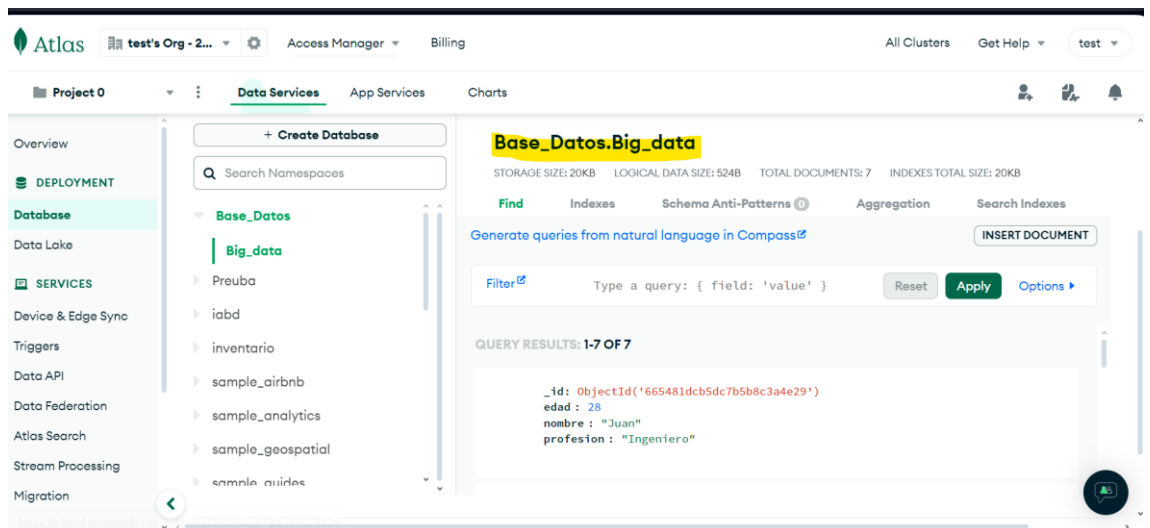
```
>>>
>>> def write_row(df, batch_id):
...     df.write.format("mongodb").mode("append") \
...         .option("connection.uri", "mongodb+srv://iabdb02:iabdiabd02@cluster0.giowyug.mongodb.net") \
...         .option("database", "Base_Datos").option("collection", "Big_data").save()
...     pass
... 
```

```
...
>>> df.writeStream.foreachBatch(write_row).start().awaitTermination()
24/05/27 14:51:37 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-67ee8b64-54e4-4a2f-85bf-6784107b7887. If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder is best effort.
24/05/27 14:51:37 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.

```

- La función que hemos creado se ejecutará cada “milisegundo” para realizar la conexión a nuestro MongoDB Atlas, buscando la base de datos “Base_Datos” y la colección “Big_data”, para grabar el dataframe “df”.
- La operación de escritura de los datos de streaming se puede realizar fila por fila. Para escribir nuestra fila de datos por fila, necesitamos crear una función de escritor.

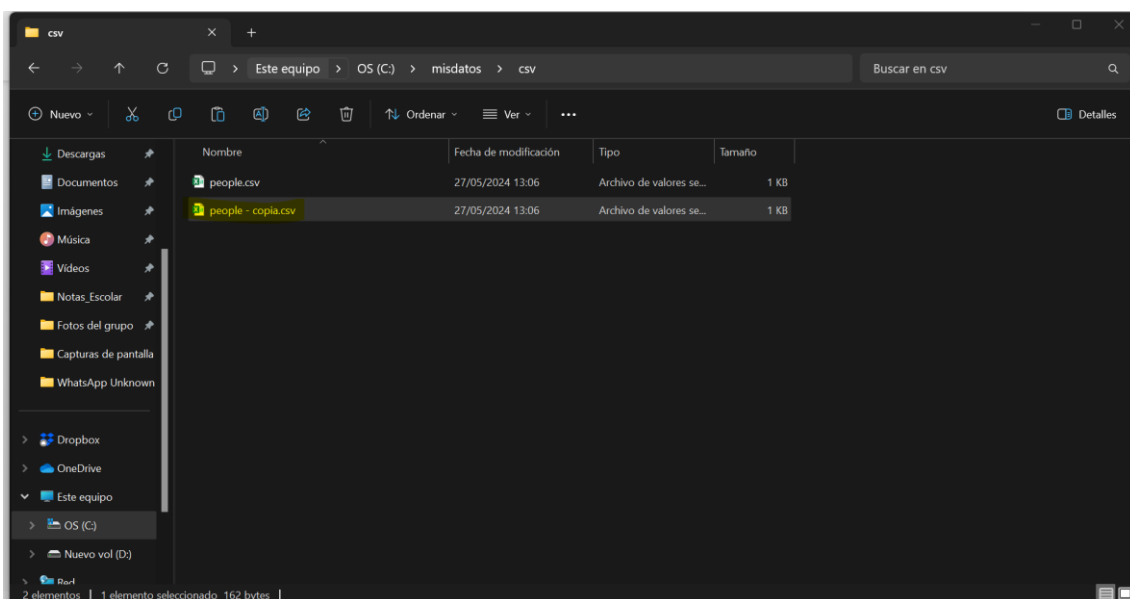
Así que tenemos un archivo people.csv en mi directorio /C/misdatos/CSV/, se tendría que guardar los registros de dicho people.csv, dentro de la colección “Big_data” de la base de datos “Base_Datos” de nuestro MongoDB Atlas.



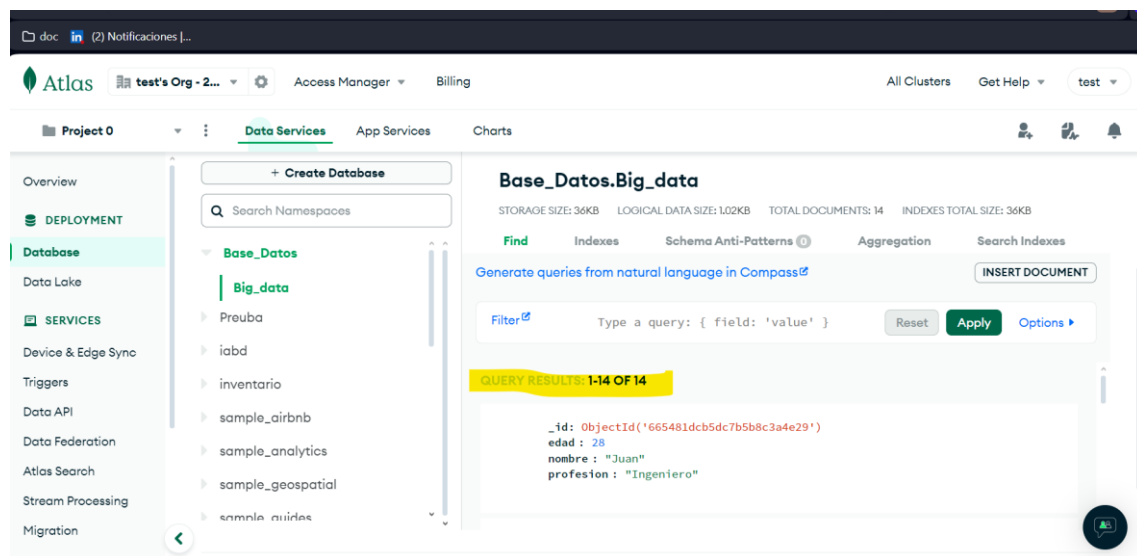
Podemos ver que en nuestra colección “Big_Data”, se ha actualizado con siete registros provenientes de nuestro people.csv en /C/misdatos/CSV



Añado otro archivo a *.csv en el directorio /C/misdatos/CSV, y vemos que se añaden los registros de este nuevo archivo *.csv



Resultados:



Conclusión

En este ejercicio, hemos construido un pipeline de streaming con Apache Spark en Python, configurando sesiones y transmitiendo datos en tiempo real. Destacamos cómo Spark puede transmitir datos desde archivos locales y escribirlos en MongoDB. También realizamos consultas SQL en datos transmitidos y demostramos la integración con MongoDB Atlas. Este trabajo ilustra la potencia de Spark para el procesamiento de datos en tiempo real, facilitando aplicaciones avanzadas de análisis de datos y aprendizaje automático en pipelines de transmisión de datos. Para más detalles, se recomienda consultar la documentación oficial de Spark Streaming.