ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NGUYỄN MỸ ANH - 21002187

BÁO CÁO BÀI TẬP LỚN CUỐI KỲ
MÔN HỌC THỰC TẬP TIN HỌC ỨNG DỤNG

Kỹ thuật điện tử và tin học

HÀ NỘI – 2024

Overview

Báo cáo này cung cấp hướng dẫn chi tiết về cách thiết lập Docker container để chạy Apache Spark và PostgreSQL, kết nối SparkSQL với cơ sở dữ liệu PostgreSQL, và thực hiện các thao tác SQL bằng script Python.

Quy trình cài đặt

Bước 1: Chuẩn bị các tập tin

Đảm bảo các tập tin sau đây có trong cùng một thư mục:

1. Dockerfile: Xây dựng image Docker.
2. entrypoint.sh: Script khởi tạo PostgreSQL, tạo cơ sở dữ liệu, bảng và thiết lập Spark.
3. spark_sql.py: Script Python để tương tác với PostgreSQL bằng SparkSQL.
4. data.csv: Tập dữ liệu mẫu có 1 triệu dòng để nhập vào PostgreSQL.

	A	B	C
1	Region	Country	Item Type
2	Sub-Saharan Africa	South Africa	Fruits
3	Middle East and North Africa	Morocco	Clothes
4	Australia and Oceania	Papua New Guinea	Meat
5	Sub-Saharan Africa	Djibouti	Clothes
6	Europe	Slovakia	Beverages
7	Asia	Sri Lanka	Fruits
8	Sub-Saharan Africa	Seychelles	Beverages
9	Sub-Saharan Africa	Tanzania	Beverages
10	Sub-Saharan Africa	Ghana	Office Supplies
11	Sub-Saharan Africa	Tanzania	Cosmetics
12	Asia	Taiwan	Fruits
13	Middle East and North Africa	Algeria	Cosmetics
14	Asia	Singapore	Snacks
15	Australia and Oceania	Papua New Guinea	Clothes
16	Asia	Vietnam	Personal Care
17	Sub-Saharan Africa	Uganda	Personal Care
18	Sub-Saharan Africa	Zimbabwe	Office Supplies

Tập dữ liệu data.csv

Bước 2: Xây dựng Docker image

Mở terminal trong thư mục chứa các tập tin trên và chạy lệnh sau:

docker build -t spark .

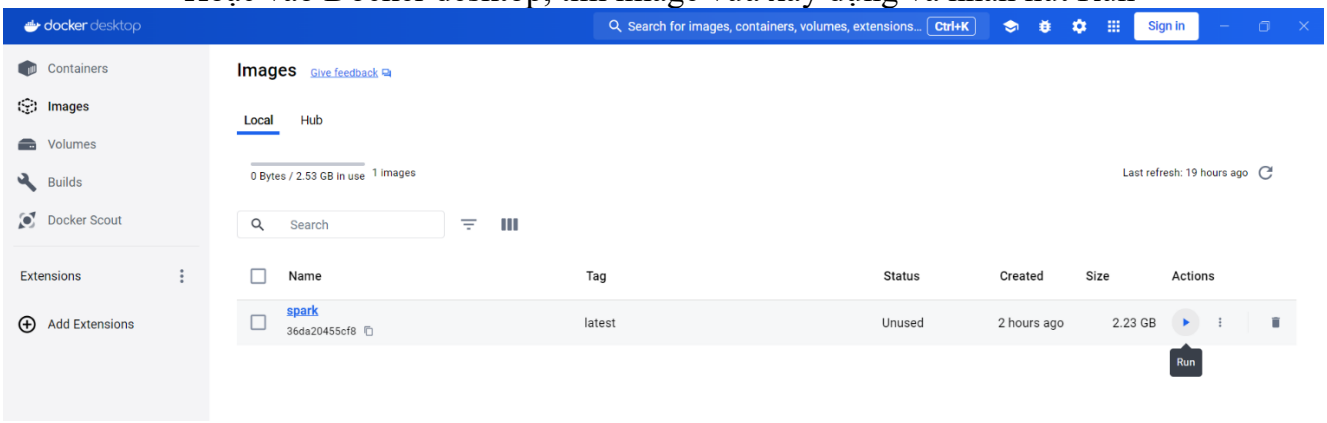
```
PS C:\Users\Admin\Documents\TTTHUD\test> docker build -t spark .
[+] Building 2217.5s (14/14) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                             0.0s
 => => transferring dockerfile: 1.72kB                                                                           0.0s
 => [internal] load metadata for docker.io/library/openjdk:8-jdk                                                 0.9s
 => [internal] load .dockerignore                                                                                0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [1/9] FROM docker.io/library/openjdk:8-jdk@sha256:86e863cc57215cfb181bd319736d0baf625fe8f150577f9eb58bd937f5452cb8   0.0s
 => [internal] load build context                                                                                0.0s
 => => transferring context: 97B                                                                                 0.0s
 => CACHED [2/9] RUN apt-get clean && apt-get update && apt-get install -y --no-install-recommends     python3     python3-pip     w  0.0s
 => [3/9] RUN set -eux;     wget -O /tmp/spark-3.2.0-bin-hadoop3.2.tgz https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.  1367.4s
 => [4/9] RUN pip3 install --upgrade pip setuptools                                                             15.8s
 => [5/9] RUN pip3 install pyspark psycopg2-binary                                                             823.7s
 => [6/9] COPY entrypoint.sh /usr/local/bin/                                                                     0.1s
 => [7/9] COPY data.csv /data/data.csv                                                                           0.3s
 => [8/9] COPY spark_sql.py /spark_sql.py                                                                        0.1s
 => [9/9] RUN chmod +x /usr/local/bin/entrypoint.sh                                                              1.0s
 => exporting to image                                                                                           7.8s
 => => exporting layers                                                                                          7.6s
 => => writing image sha256:36da20455cf842c4b32ba76e3202a3d6f499f47aa3408f9f2e71c9207e612e2d                     0.0s
 => => naming to docker.io/library/spark                                                                         0.0s

View build details: docker-desktop://dashboard/build/default/default/4u9wbnklchyy2o1whm4inx3r9

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Bước 3: Chạy Docker Container

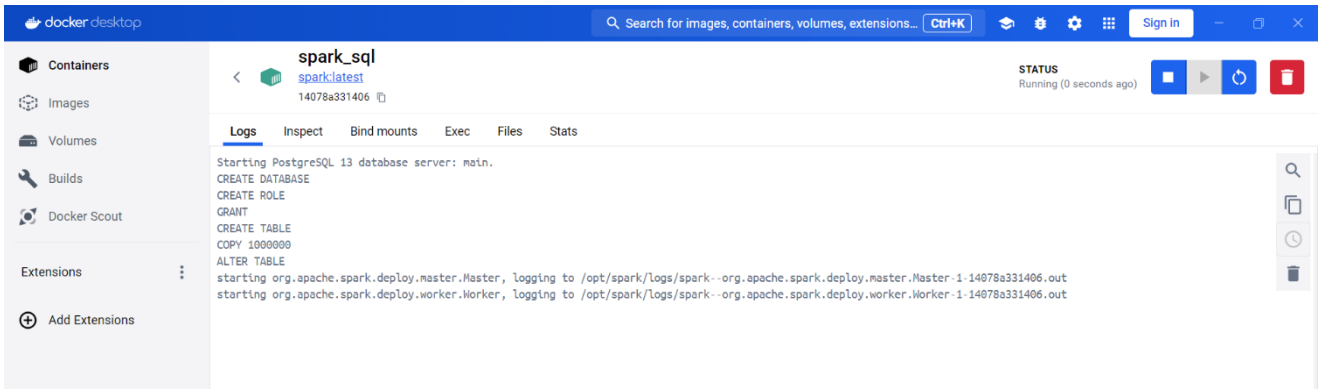
- Sau khi Docker image được xây dựng thành công, chạy container với lệnh sau:

docker run --name spark_sql -d spark

Hoặc vào Docker desktop, tìm image vừa xây dựng và nhấn nút Run



Container sẽ tự động thực thi tập lệnh entrypoint.sh, thiết lập PostgreSQL và khởi tạo Spark. Kiểm tra Logs xem có lỗi không:

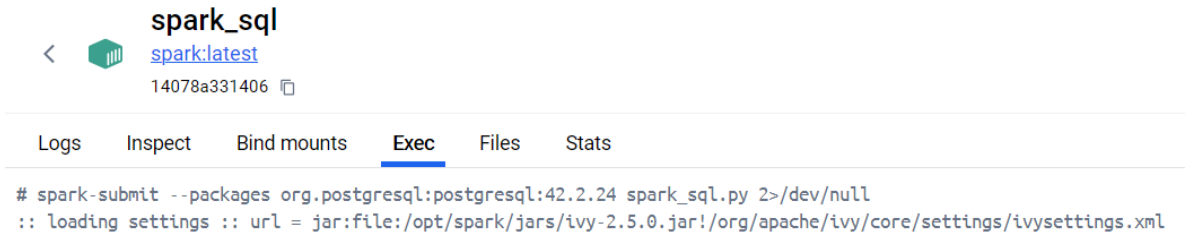


Bước 4: Kết nối SparkSQL với cơ sở dữ liệu

1. Chạy script Python

- Để chạy script spark_sql.py, chạy lệnh sau trong Exec của container vừa tạo:

spark-submit --packages org.postgresql:postgresql:42.2.24 spark_sql.py 2>/dev/null



- Bên trong container, SparkSQL đã được cấu hình sẵn để kết nối với cơ sở dữ liệu PostgreSQL sparkdb.

- Các thuộc tính kết nối được thiết lập trong spark_sql.py:

  JDBC URL: jdbc:postgresql://localhost:5432/sparkdb
  Tên người dùng: sparkuser
  Mật khẩu: sparkpassword

2. Kết quả:

Script sẽ thực hiện các thao tác CRUD (Create, Read, Update, Delete) và đo lường hiệu suất, hiển thị kết quả trên Exec.

- **Create**:
  Câu lệnh này chèn dữ liệu mới từ DataFrame new_data_df vào bảng sample_data trong cơ sở dữ liệu PostgreSQL. Sử dụng phương thức write.jdbc() của Spark DataFrame để ghi dữ liệu vào cơ sở dữ liệu, với chế độ "append" để thêm dữ liệu mới vào bảng.

```
# Creating a DataFrame to insert
data = [("South America", "Brazil", "Mango"),
            ("Africa", "Brazil", "Pineapple")]
columns = ["region", "country", "item"]
new_data_df = spark.createDataFrame(data, columns)

# Appending new data to PostgreSQL
new_data_df.write.jdbc(url=jdbc_url, table="sample_data", mode="append",
properties=properties)
```

Kết quả: Thêm 2 dòng dữ liệu mới, in ra tổng số dòng trước và sau khi thêm

```
----------------CREATE------------
Total records before insert: 1000000
New records inserted successfully.
Total records after insert: 1000002
```

- **Read:**
  Câu lệnh này sử dụng Spark SQL để chọn tất cả các cột và hàng từ bảng sample_data và hiển thị 10 hàng đầu tiên. Nó thực hiện thao tác đọc dữ liệu từ cơ sở dữ liệu PostgreSQL và trả về một DataFrame Spark, sau đó sử dụng phương thức show() để hiển thị dữ liệu.

```
spark.sql("SELECT * FROM sample_data LIMIT 10").show()
```

Kết quả:

```
----------------READ------------
+--------------------+----------------+---------------+
|              region|         country|           item|
+--------------------+----------------+---------------+
|  Sub-Saharan Africa|    South Africa|         Fruits|
|Middle East and N...|         Morocco|        Clothes|
|Australia and Oce...|Papua New Guinea|           Meat|
|  Sub-Saharan Africa|        Djibouti|        Clothes|
|              Europe|        Slovakia|      Beverages|
|                Asia|       Sri Lanka|         Fruits|
|  Sub-Saharan Africa|     Seychelles |      Beverages|
|  Sub-Saharan Africa|        Tanzania|      Beverages|
|  Sub-Saharan Africa|           Ghana|Office Supplies|
|  Sub-Saharan Africa|        Tanzania|      Cosmetics|
+--------------------+----------------+---------------+
```

- **Update:**
  Câu lệnh này cập nhật các dòng trong bảng sample_data trong cơ sở dữ liệu PostgreSQL. Nó sử dụng câu lệnh SQL UPDATE để thay đổi giá trị của cột item thành 'Papaya' cho tất cả các dòng có giá trị country là 'Brazil'.

```
# Perform update
update_query = """
        UPDATE sample_data
        SET item = 'Papaya'
        WHERE country = 'Brazil'
"""
cursor.execute(update_query)
conn.commit()
```

Kết quả: Hiện thị các dòng có giá trị country là 'Brazil' trước và sau khi cập nhật

```
----------------UPDATE------------
Records before update:
+-------------+-------+---------+
|       region|country|     item|
+-------------+-------+---------+
|South America| Brazil|    Mango|
|       Africa| Brazil|Pineapple|
+-------------+-------+---------+


Records after update:
+-------------+-------+------+
|       region|country|  item|
+-------------+-------+------+
|South America| Brazil|Papaya|
|       Africa| Brazil|Papaya|
+-------------+-------+------+
```

- **Delete**: Câu lệnh này xóa các dòng từ bảng sample_data trong cơ sở dữ liệu PostgreSQL. Nó sử dụng câu lệnh SQL DELETE để xóa tất cả các bản ghi có giá trị country là 'Brazil' khỏi bảng. Ở đây xóa 2 dòng dữ liệu vừa mới thêm ở bước Create vì trong tập dữ liệu mẫu data.csv không có dòng nào có giá trị country là Brazil

```
delete_query = """
        DELETE FROM sample_data
        WHERE country = 'Brazil'
"""
cursor.execute(delete_query)
conn.commit()
```

Kết quả: Xóa và in ra tổng số dòng trước và sau khi xóa

```
----------------DELETE------------
Total records before delete: 1000002
Data deleted successfully.
Total records after delete: 1000000
```

- **Đánh giá hiệu suất**:
  Các câu lệnh này được sử dụng để đo thời gian xử lý của các truy vấn SQL trong Spark SQL. Câu lệnh thứ nhất trả về 5 hàng đầu tiên từ bảng sample_data, trong khi câu lệnh thứ hai chỉ trả về 5 hàng đầu tiên từ bảng sample_data mà country là 'France'. Việc đo thời gian này giúp đánh giá hiệu suất của các truy vấn có và không có điều kiện WHERE. Dùng hàm time trong python để tính khoảng thời gian thực thi của các truy vấn

```
# Measure time for query without WHERE clause
spark.sql("SELECT * FROM sample_data LIMIT 5").show()

# Measure time for query with WHERE clause
spark.sql("SELECT * FROM sample_data WHERE country = 'France' LIMIT 5").show()
```

Kết quả: Việc sử dụng điều kiện WHERE trong câu lệnh SQL thường mang lại hiệu suất cao hơn bởi vì nó giảm bớt khối lượng dữ liệu cần xử lý, tối ưu hóa quy trình xử lý dữ liệu và tận dụng các cơ chế tối ưu hóa của hệ thống cơ sở dữ liệu.

```
----------------Performance measurements------------
+--------------------+----------------+---------+
|              region|         country|     item|
+--------------------+----------------+---------+
|  Sub-Saharan Africa|    South Africa|   Fruits|
|Middle East and N...|         Morocco|  Clothes|
|Australia and Oce...|Papua New Guinea|     Meat|
|  Sub-Saharan Africa|        Djibouti|  Clothes|
|              Europe|        Slovakia|Beverages|
+--------------------+----------------+---------+

Time taken without WHERE clause: 0.8439431190490723 seconds

+------+-------+---------------+
|region|country|           item|
+------+-------+---------------+
|Europe| France|      Cosmetics|
|Europe| France|     Vegetables|
|Europe| France|Office Supplies|
|Europe| France|  Personal Care|
|Europe| France|      Beverages|
+------+-------+---------------+

Time taken with WHERE clause: 0.2763543128967285 seconds
```