Starting the Cluster

Dockerfile on the server

Docker

```dockerfile
FROM apache/spark:4.0.1
# switch to root to install packages
USER root
RUN pip install --no-cache-dir "pandas==2.3.2" "pyarrow==21.0.0"
# switch back to spark user
USER spark
```

compose.yaml on the server

YAML

```yaml
services:
  spark:
    build: .
    hostname: apache-spark
    ports:
      - "7077:7077"    # Spark master port
      - "8080:8080"    # Spark master web UI
      - "8081:8081"    # Spark worker web UI
      - "15002:15002"  # Spark Connect server port
      - "4040:4040"    # Spark Connect web UI
    command: >
      bash -c "/opt/spark/sbin/start-master.sh;
              /opt/spark/sbin/start-connect-server.sh;
              /opt/spark/sbin/start-worker.sh spark://192.168.1.7:7077;
              sleep infinity"
```

Shell

```shell
$ ls
compose.yaml  Dockerfile
$ docker compose up -d
[+] Running 1/1
 ✔ Container spark-docker-spark-1  Started
 0.5s
```

Task 1

Task1: Retail Store Insights

Scenario: You're a data analyst at a large retail store. The store sells a variety of products, including books and fruits. The management wants insights into sales patterns, customer preferences, product popularity, and potential promotions.

Objective: Analyze the provided dataset to extract meaningful insights and present them to the management.

Instructions:

1. **Initialization**:
   - Set up your environment by initializing a Spark session. Name this session "RetailStoreInsights".

In [1]:

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
  .remote("sc://192.168.1.7:15002") \
  .appName("RetailStoreInsights") \
  .getOrCreate()

# limit() shows a nice HTML table in Jupyter, while show() prints plain text
spark.conf.set('spark.sql.repl.eagerEval.enabled', True)

spark
```

Out[1]:

```
<pyspark.sql.connect.session.SparkSession at 0x14c54de0b50>
```

2. **Data Loading**:
   - Load the provided dataset into a DataFrame. This dataset will have O information about various products, including their type (e.g., fruit or book) and price. Create DataFrame based on bellow data:

```
...
```

3. **Data Exploration**:
   - Familiarize yourself with the dataset:
     - Display the first 10 rows to understand the structure and content.

In [2]:

```python
from pyspark.sql import Row

data = [
  ('Ulysses', 'Book', 23.17, 16),
  ('Apple', 'Fruit', 2.34, 8),
  ('Pineapple', 'Fruit', 2.57, 1),
  ('Apple', 'Fruit', 2.43, 6),
  ('To Kill a Mockingbird', 'Book', 24.14, 19),
  ('To Kill a Mockingbird', 'Book', 11.18, 11),
  ('Watermelon', 'Fruit', 3.35, 15),
  ('Pride and Prejudice', 'Book', 24.99, 3),
  ('To Kill a Mockingbird', 'Book', 21.82, 17),
  ('Moby Dick', 'Book', 14.83, 20),
  ('Pride and Prejudice', 'Book', 5.03, 16),
  ('Jane Eyre', 'Book', 20.40, 8),
  ('Moby Dick', 'Book', 5.55, 20),
  ('Don Quixote', 'Book', 19.75, 17),
  ('Watermelon', 'Fruit', 2.31 , 9),
  ('Hamlet', 'Book', 18.20, 12),
  ('Mango', 'Fruit', 4.10, 7),
  ('1984', 'Book', 16.75, 14),
  ('Strawberry', 'Fruit', 1.90, 25),
  ('War and Peace', 'Book', 22.50, 9),
  ('Orange', 'Fruit', 3.05, 13),
  ('The Great Gatsby', 'Book', 12.30, 10),
  ('Peach', 'Fruit', 2.80, 11),
  ('Grapes', 'Fruit', 2.60, 18),
  ('Pride and Prejudice', 'Book', 9.50, 5)
]

df = spark.createDataFrame([
  Row(product_name=row[0], category=row[1], price=row[2], quantity=row[3])
  for row in data
], schema = 'product_name STRING, category STRING, price FLOAT, quantity SHORT')
df.createOrReplaceTempView("retail_sales") # give it a name for sql
df.limit(10)
```

Out[2]:

```
+--------------------+--------+-----+--------+
|        product_name|category|price|quantity|
+--------------------+--------+-----+--------+
|             Ulysses|    Book|23.17|      16|
|               Apple|   Fruit| 2.34|       8|
|           Pineapple|   Fruit| 2.57|       1|
|               Apple|   Fruit| 2.43|       6|
|To Kill a Mocking...|    Book|24.14|      19|
|To Kill a Mocking...|    Book|11.18|      11|
|          Watermelon|   Fruit| 3.35|      15|
| Pride and Prejudice|    Book|24.99|       3|
|To Kill a Mocking...|    Book|21.82|      17|
|           Moby Dick|    Book|14.83|      20|
+--------------------+--------+-----+--------+
```

- Print the schema of the DataFrame to understand the data types and columns.

In [3]:

```python
df.printSchema()
```

```
root
 |-- product_name: string (nullable = true)
 |-- category: string (nullable = true)
 |-- price: float (nullable = true)
 |-- quantity: short (nullable = true)

```

3. **Data Analysis**:
   - Extract specific columns of interest using the select operation. For this task, focus on the product name and its price.
   - Identify and display products that are priced above $2.

In [4]:

```python
spark.sql("""
  select * from retail_sales
  where price > 2
  order by price
""")
```

Out[4]:

```
+--------------------+--------+-----+--------+
|        product_name|category|price|quantity|
+--------------------+--------+-----+--------+
|          Watermelon|   Fruit| 2.31|       9|
|               Apple|   Fruit| 2.34|       8|
|               Apple|   Fruit| 2.43|       6|
|           Pineapple|   Fruit| 2.57|       1|
|              Grapes|   Fruit|  2.6|      18|
|               Peach|   Fruit|  2.8|      11|
|              Orange|   Fruit| 3.05|      13|
|          Watermelon|   Fruit| 3.35|      15|
|               Mango|   Fruit|  4.1|       7|
| Pride and Prejudice|    Book| 5.03|      16|
|           Moby Dick|    Book| 5.55|      20|
| Pride and Prejudice|    Book|  9.5|       5|
|To Kill a Mocking...|    Book|11.18|      11|
|    The Great Gatsby|    Book| 12.3|      10|
|           Moby Dick|    Book|14.83|      20|
|                1984|    Book|16.75|      14|
|              Hamlet|    Book| 18.2|      12|
|          Don Quixote|    Book|19.75|      17|
|           Jane Eyre|    Book| 20.4|       8|
|To Kill a Mocking...|    Book|21.82|      17|
+--------------------+--------+-----+--------+
only showing top 20 rows
```

- Group the data by product type (e.g., fruit or book) and determine the count for each category.

In [5]:

```python
# https://spark.apache.org/docs/latest/sql-pipe-syntax.html
spark.sql("""
  from retail_sales
  |> aggregate count(*) as category_count
     group by category
""")
```

Out[5]:

```
+--------+--------------+
|category|category_count|
+--------+--------------+
|    Book|            15|
|   Fruit|            10|
+--------+--------------+
```

- Calculate the average price of all products in the dataset.

In [6]:

```python
spark.sql("""
  from retail_sales
  |> aggregate avg(price) as avg_price
     group by product_name
  |> set avg_price = round(avg_price, 2)
""")
```

Out[6]:

```
+--------------------+---------+
|        product_name|avg_price|
+--------------------+---------+
|           Pineapple|     2.57|
|To Kill a Mocking...|    19.05|
|             Ulysses|    23.17|
|               Apple|     2.38|
|           Jane Eyre|     20.4|
|           Moby Dick|    10.19|
|          Watermelon|     2.83|
|  Pride and Prejudice|    13.17|
|                1984|    16.75|
|               Mango|      4.1|
|         Don Quixote|    19.75|
|              Hamlet|     18.2|
|              Orange|     3.05|
|               Peach|      2.8|
|    The Great Gatsby|     12.3|
|              Grapes|      2.6|
|          Strawberry|      1.9|
|       War and Peace|     22.5|
+--------------------+---------+
```

- The store is considering a promotion where they offer a 10% discount on all products. Add a new column to the DataFrame that calculates the discounted price for each product.

In [7]:

```python
spark.sql("""
  from retail_sales
  |> extend price - (price * 0.1) as discounted_price
  |> set discounted_price = round(discounted_price, 2)
  |> select product_name, discounted_price, price as original_price
""")
```

Out[7]:

```
+--------------------+----------------+--------------+
|        product_name|discounted_price|original_price|
+--------------------+----------------+--------------+
|             Ulysses|           20.85|         23.17|
|               Apple|            2.11|          2.34|
|           Pineapple|            2.31|          2.57|
|               Apple|            2.19|          2.43|
|To Kill a Mocking...|           21.73|         24.14|
|To Kill a Mocking...|           10.06|         11.18|
|          Watermelon|            3.01|          3.35|
| Pride and Prejudice|           22.49|         24.99|
|To Kill a Mocking...|           19.64|         21.82|
|           Moby Dick|           13.35|         14.83|
| Pride and Prejudice|            4.53|          5.03|
|           Jane Eyre|           18.36|          20.4|
|           Moby Dick|             5.0|          5.55|
|          Don Quixote|           17.78|         19.75|
|          Watermelon|            2.08|          2.31|
|              Hamlet|           16.38|          18.2|
|               Mango|            3.69|           4.1|
|                1984|           15.08|         16.75|
|          Strawberry|            1.71|           1.9|
|       War and Peace|           20.25|          22.5|
+--------------------+----------------+--------------+
only showing top 20 rows
```

- Using SQL, perform the following operations:
  - Determine the total number of products sold (including duplicates).

note: this operations seemed to be a bit out of place, so I interpreted them as best as I could.

In [8]:

```python
spark.sql("""
  from retail_sales
  |> aggregate sum(quantity) as n_sold_total
""")
```

Out[8]:

```
+------------+
|n_sold_total|
+------------+
|         310|
+------------+
```

- Calculate the total sales (sum of prices) for each product category (e.g., fruits vs. books).

In [9]:

```python
spark.sql("""
  from retail_sales
  |> aggregate sum(quantity) as n_sold
     group by category
""")
```

Out[9]:

```
+--------+------+
|category|n_sold|
+--------+------+
|    Book|   197|
|   Fruit|   113|
+--------+------+
```

In [10]:

```python
spark.sql("""
  from retail_sales
  |> aggregate sum(price * quantity) as revenue
     group by category
""")
```

Out[10]:

```
+--------+-----------------+
|category|          revenue|
+--------+-----------------+
|    Book|3211.2000007629395|
|   Fruit| 300.3599935770035|
+--------+-----------------+
```

- (Optional) Identify the products based on the frequency of their occurrence in the dataset. (To find out which products appear most often in the dataset.)

In [11]:

```python
spark.sql("""
  from retail_sales
  |> aggregate sum(quantity) as n_sold
     group by category, product_name
  |> order by n_sold desc
""")
```

Out[11]:

```
+--------+-------------------+------+
|category|       product_name|n_sold|
+--------+-------------------+------+
|    Book|To Kill a Mocking...|    47|
|    Book|          Moby Dick|    40|
|   Fruit|         Strawberry|    25|
|   Fruit|         Watermelon|    24|
|    Book| Pride and Prejudice|    24|
|   Fruit|             Grapes|    18|
|    Book|         Don Quixote|    17|
|    Book|            Ulysses|    16|
|   Fruit|              Apple|    14|
|    Book|               1984|    14|
|   Fruit|             Orange|    13|
|    Book|              Hamlet|    12|
|   Fruit|               Peach|    11|
|    Book|    The Great Gatsby|    10|
|    Book|       War and Peace|     9|
|    Book|           Jane Eyre|     8|
|   Fruit|               Mango|     7|
|   Fruit|           Pineapple|     1|
+--------+--------------------+------+
```

In [12]:

```python
spark.stop()
```

Python

Task 2

Scenario: You're a data engineering intern at a tech company. As part of your training, you're given a basic PySpark script that demonstrates the use of Python UDFs and Pandas UDFs. Your task is to modify the script by implementing a new transformation function.

Instructions:

1. **Initialization:**
   - Start a Spark session named "UDFTransformation".

In [1]:

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
  .remote("sc://192.168.1.7:15002") \
  .appName("UDFTransformation") \
  .config("spark.sql.ansi.enabled", "false") \
  .config("spark.sql.execution.pythonUDF.arrow.enabled", "true") \
  .getOrCreate()

# limit() shows a nice HTML table in Jupyter, while show() prints plain text
spark.conf.set('spark.sql.repl.eagerEval.enabled', True)

spark
```

Out[1]:

```
<pyspark.sql.connect.session.SparkSession at 0x22751278d30>
```

2. **Using Python Native Function as UDF:**
   - Define a Python function that multiplies the input value by 3.
   - Convert this function into a Spark UDF.
   - Create a DataFrame with a series of numbers: [(4,), (5,), (6,)].

In [2]:

```python
from pyspark.sql.functions import udf

@udf(returnType='int')
def mult_by_3(s: int) -> int:
  return s * 3

df = spark.createDataFrame([(4, ), (5, ), (6, )], ['value'])
df
```

Out[2]:

```
+-----+
|value|
+-----+
|    4|
|    5|
|    6|
+-----+
```

- Apply the UDF to the DataFrame to triple each number.
- Display the updated DataFrame.

In [3]:

```python
dff = df.withColumn('value_x3', mult_by_3(df.value))
dff
```

Out[3]:

```
+-----+--------+
|value|value_x3|
+-----+--------+
|    4|      12|
|    5|      15|
|    6|      18|
+-----+--------+
```

3. **Using Pandas UDF:**
   - Define a Pandas UDF that subtracts 2 from each value in the input series.
   - Apply this UDF to the DataFrame from step 2.
   - Display the updated DataFrame with the subtracted values.

In [4]:

```python
import pandas as pd
import pyspark.pandas as ps
from pyspark.sql.functions import pandas_udf

@pandas_udf("int")
def sub_2(s: pd.Series) -> pd.Series:
  return s - 2

dffs = dff.withColumn('value_minus_2', sub_2(dff.value))
dffs
```

```
c:\Users\plancha\spark-lab1\.venv\lib\site-packages\pyspark\pandas\__init__.py:43: UserWarning: 'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is required to set this environment variable to '1' in both driver and executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will set it for you but it does not work if there is a Spark context already launched.
  warnings.warn(
```

Out[4]:

```
+-----+--------+-------------+
|value|value_x3|value_minus_2|
+-----+--------+-------------+
|    4|      12|            2|
|    5|      15|            3|
|    6|      18|            4|
+-----+--------+-------------+
```

4. **Cleanup:**

- Stop the Spark session.

In [5]:

```python
spark.stop()
```