

Ejercicios de logs NASA

Trabajando sobre los ficheros access_log_Jul95.csv y access_log_Aug95.csv

La Spark Session la he inicializado en un script de prueba anteriormente:

```
%python
from pyspark.sql import SparkSession

# Crear o obtener una SparkSession
spark = SparkSession.builder.appName("NASA").getOrCreate()
```

A continuación, trato los datos como se pide en el ejercicio:

```
from pyspark.sql.functions import col, to_timestamp

# Leer los archivos de texto en un RDD
rdd_jul = spark.sparkContext.textFile("/FileStore/tables/Formacion_Binaia/Practica_DataFrames/access_log_Jul95.csv")
rdd_aug = spark.sparkContext.textFile("/FileStore/tables/Formacion_Binaia/Practica_DataFrames/access_log_Aug95.csv")

rdd = rdd_jul.union(rdd_aug)

# Definir las columnas
columns = ["Host", "Date", "RequestMethod", "Resource", "Protocol", "HTTPStatusCode", "Size"]

# Definir la expresión regular para parsear las líneas del log. + -> 1 o más, * -> 0 o más
log_pattern = r"^(^(\S+\.)(\S+\.)(\S+)) - - \[(\d{2}/\d{3}/\d{4}):(\d{2}):(\d{2}):(\d{2})\] [^\s]+\s+(\d{4})\s+\"(\S+)\" (\S+)\"(\S+/\d{1,3}[\d{2-4}])\" (\d{3}) (\d+|-)$"

# Función para procesar cada línea del log y convertirla en una lista, ignorando los campos no necesarios
def parse_log_line(line):
    import re # Soporte para expresiones regulares
    if isinstance(line, str):
        match = re.match(log_pattern, line) # Intentar hacer coincidir la línea (line) con el patrón definido anteriormente (log_pattern). Si coincide match, sino None.
        if match:
            host = match.group(1)
            date = match.group(2) + " " + match.group(3)
            request_method = match.group(4)
            resource = match.group(5)
            protocol = match.group(6)
            http_status_code = match.group(7)
            size = match.group(8)
            if size == '-':
                size = None
            return (host, date, request_method, resource, protocol, http_status_code, size)
        return None

# Aplica parse_log_line a cada línea de los csv
rdd_parsed = rdd.map(parse_log_line)

# veo las filas nulas y filtro una si existe
num_none = rdd_parsed.filter(lambda x : x is None).count()
print(f"Total de filas nulas : {num_none}")

if num_none > 0:
    one_none = rdd.zip(rdd_parsed).filter(lambda x: x[1] is None).map(lambda x: x[0])
    # rdd.zip(rdd_parsed) combina los dos RDDs en un nuevo RDD de pares. Cada línea del nuevo es una tupla (rdd_linea - rdd_parsed_linea), filtra los pares donde rdd_parsed es nula y en map mapea (transforma) cada par (de nones, ya que la entrada son los nones de rdd_parsed, los mapea todos ellos a su elemento original, es decir, a la línea original)
    one_none_line = one_none.first()
    print(f"Una fila nula: {one_none_line}")

# Filtro las filas no nulas
rdd_parsed = rdd_parsed.filter(lambda x : x is not None)

# Convertir el RDD en DataFrame. toDF transforma la lista u objeto en un df asignando nombres de columnas
df = rdd_parsed.toDF(columns)

# Convertir la columna de fecha al formato adecuado y cambiar el tipo de datos de otras columnas
df = df.withColumn("Date", to_timestamp(col("Date"), "dd/MM/yyyy HH:mm:ss")) \
    .withColumn("Size", col("Size").cast("int")) \
    .withColumn("HTTPStatusCode", col("HTTPStatusCode").cast("int"))

# Mostrar algunas filas del DataFrame resultante
df.show(5, truncate=False)

# Guardar el DataFrame en formato Parquet
df.write.parquet("/FileStore/tables/Formacion_Binaia/Practica_DataFrames/nasa_logs.parquet")
```

Muestro el número de líneas “None”, y muestro una línea None de ejemplo. Este primer caso que muestro se da porque el HOST no cumple el patrón específico. Además, muestro las 5 primeras líneas del “parquet” válido.

```
Total de filas nulas : 16245
Una fila nula: pipe6.nyc.pipeline.com - - [01/Jul/1995:00:22:43 -0400] "GET /shuttle/missions/sts-71/movies/sts-71-mir-dock.mpg" 200 946425
```

Host	Date	RequestMethod	Resource	Protocol	HTTPstatuscode	Size
199.72.81.55	1995-07-01 00:00:01	GET	/history/apollo/	HTTP/1.0	200	6245
unicomp6.unicomp.net	1995-07-01 00:00:06	GET	/shuttle/countdown/	HTTP/1.0	200	3985
199.120.110.21	1995-07-01 00:00:09	GET	/shuttle/missions/sts-73/mission-sts-73.html	HTTP/1.0	200	4085
burger.letters.com	1995-07-01 00:00:11	GET	/shuttle/countdown/liftoff.html	HTTP/1.0	304	0
199.120.110.21	1995-07-01 00:00:11	GET	/shuttle/missions/sts-73/sts-73-patch-small.gif	HTTP/1.0	200	4179

only showing top 5 rows

- ¿Cuáles son los distintos protocolos web utilizados? Agrúpalos.

agg sirve para realizar consultas como sum, avg, max, min, count...

```
%python
from pyspark.sql.functions import count

protocol_counts = df.groupBy("Protocol").agg(count("*").alias("Count"))
protocol_counts.show()
```

▶ (2) Spark Jobs

▶ protocol_counts: pyspark.sql.dataframe.DataFrame = [Protocol: string, Count: long]

Protocol	Count
HTTP/1.0	3445368

- ¿Cuáles son los códigos de estado más comunes en la web? Agrúpalos y ordénalos para ver cuál es el más común.

```
%python
from pyspark.sql.functions import desc

# Códigos de estado más comunes
status_count = df.groupBy("HTTPstatuscode").count().orderBy(desc("count"))
status_count.show()
```

▶ (2) Spark Jobs

▶ status_count: pyspark.sql.dataframe.DataFrame = [HTTPstatuscode: integer, count: long]

HTTPstatuscode	count
200	3092457
304	266140
302	72585
404	20691
403	225
500	57
501	41
400	2

- ¿Y los métodos de petición (verbos) más utilizados?

```
%python
# Métodos de petición más utilizados
method_count = df.groupBy("RequestMethod").count().orderBy(desc("count"))
method_count.show()
```

▶ (2) Spark Jobs

```
▶ method_count: pyspark.sql.dataframe.DataFrame = [RequestMethod: string, count: long]
```

RequestMethod	count
GET	3444217
HEAD	7761
POST	217

- ¿Qué recurso tuvo la mayor transferencia de bytes de la página web?

```
%python
# Recurso con la mayor transferencia de bytes
resource_bytes = df.groupBy("Resource").sum("Size").orderBy(desc("sum(Size)")).limit(1)
resource_bytes.show()
```

▶ (2) Spark Jobs

```
▶ resource_bytes: pyspark.sql.dataframe.DataFrame = [Resource: string, sum(Size): long]
```

Resource	sum(Size)
/shuttle/missions...	3190800196

- Además, queremos saber que recurso de nuestra web es el que más tráfico recibe. Es decir, el recurso con más registros en nuestro log.

```
%python
# Recurso con más registros en el log
resource_count = df.groupBy("Resource").count().orderBy(desc("count")).limit(1)
resource_count.show()
```

▶ (2) Spark Jobs

```
▶ resource_count: pyspark.sql.dataframe.DataFrame = [Resource: string, count: long]
```

Resource	count
/images/NASA-logo...	208113

- ¿Qué días la web recibió más tráfico?

```
%python
from pyspark.sql.functions import date_format

# Días con más tráfico
traffic_by_day = df.withColumn("Day", date_format(col("Date"), "yyyy-MM-dd")) \
    .groupBy("Day").count().orderBy(desc("count"))
traffic_by_day.show()
```

▶ (2) Spark Jobs

```
▶ traffic_by_day: pyspark.sql.dataframe.DataFrame = [Day: string, count: long]
```

Day	count
1995-07-13	133487
1995-07-06	100710
1995-07-05	94329
1995-07-12	91804
1995-08-31	89673
1995-07-03	89440
1995-07-07	86964
1995-07-14	83923
1995-08-30	80165
1995-07-11	80019
1995-07-17	74924
1995-07-10	72609
1995-07-19	72489
1995-07-04	70439
1995-08-29	67704
1995-07-20	66432
1995-07-01	64706
1995-07-21	64456

- ¿Cuáles son los hosts son los más frecuentes?

```
%python
# Hosts más frecuentes
host_count = df.groupBy("Host").count().orderBy(desc("count")).limit(10)
host_count.show()
```

▶ (2) Spark Jobs

▶ host_count: pyspark.sql.dataframe.DataFrame = [Host: string, count: long]

```
+-----+-----+
|          Host|count|
+-----+-----+
|piweba3y.prodigy.com|21988|
|piweba4y.prodigy.com|16437|
|piweba1y.prodigy.com|12825|
|  edams.ksc.nasa.gov|11964|
|      163.206.89.4| 9697|
|      news.ti.com| 8161|
|www-d1.proxy.aol.com| 8047|
|  alyssa.prodigy.com| 8037|
| siltb10.orl.mmc.com| 7573|
|www-a2.proxy.aol.com| 7516|
+-----+-----+
```

- ¿A qué horas se produce el mayor número de tráfico en la web?

```
%python
# Horas con mayor tráfico
traffic_by_hour = df.withColumn("Hour", date_format(col("Date"), "HH")) \
    .groupBy("Hour").count().orderBy(desc("count"))
traffic_by_hour.show()
```

▶ (2) Spark Jobs

▶ traffic_by_hour: pyspark.sql.dataframe.DataFrame = [Hour: string, count: long]

```
+-----+-----+
|Hour| count|
+-----+-----+
| 15|229843|
| 12|226615|
| 13|224772|
| 14|223065|
| 16|216919|
| 11|210169|
| 10|192681|
| 17|178023|
| 09|177979|
| 08|148730|
| 18|145660|
| 22|131243|
| 19|130817|
| 21|129743|
| 20|129548|
| 23|123773|
| 00|110109|
| 07|101154|
```

- ¿Cuál es el número de errores 404 que ha habido cada día?

```
%python
# Número de errores 404 por día
errors_404 = df.filter(col("HTTPstatuscode") == 404) \
    .withColumn("Day", date_format(col("Date"), "yyyy-MM-dd")) \
    .groupBy("Day").count().orderBy(desc("Day"))

errors_404.show()
```

▶ (2) Spark Jobs

▶ errors_404: pyspark.sql.dataframe.DataFrame = [Day: string, count: long]

Day	count
1995-08-31	525
1995-08-30	567
1995-08-29	414
1995-08-28	409
1995-08-27	370
1995-08-26	366
1995-08-25	412
1995-08-24	420
1995-08-23	344
1995-08-22	288
1995-08-21	305
1995-08-20	312
1995-08-19	207
1995-08-18	255
1995-08-17	268
1995-08-16	257
1995-08-15	326
1995-08-14	287