Lecture 3: Stream processing with Spark

Spark Structured Streaming is an extension of the Spark API that enables scalable stream processing.

It's a well-documented API link, which is however different from Spark Streaming, which has recently become deprecated since Spark version 3.4.0 (~last year)

The main difference between Spark `Streaming` and `Structured Streaming` is that the former is based on the low-level RDD API, while the latter is based on DataFrames.

Both APIs have however been designed to help with the continuous processing of streaming applications.

The continuous stream of input data can be ingested from many data sources such as **Kafka**, **Amazon S3**, or **TCP sockets**.

Processed data can be exported to an external database and used to make live dashboards or offline analyses, stored in files, or used in a further stage of a Kafka pipeline.

Overall, the practice of reading data from a set of sources, pre-processing it, and then storing it in a different format for later analysis is extremely common and has its own name: **real-time ETL pipelines**.

- **E**xtract
- **T**ransform
- **L**oad

Structured Streaming

The key idea of this stream processing model is to treat the continuous stream as a table that is continuously appended to.

This allows users to view the continuously incoming data as a DataFrame with new records being new rows to be included. It further expresses stream processing as a standard batch-like query on a static table, similar to what we have done in the Spark DataFrame lecture.

Internally, the stream is divided into micro-batches produced by a "trigger," which could represent any given condition (e.g., every 1 second):

- The input stream is a table (DataFrame) or the new rows to be appended to the previous DataFrame.
- Every operation/query will produce a result table (DataFrame).
- When the output table is updated, it can be written somewhere thanks to an output module/type.

NB: Although the input table can be viewed as an always-growing DataFrame, **Spark does not actually materialize the entire table**.

- Only the latest data are processed (latest batch) and then discarded.
- Conversely, the result table can be updated to keep track of the results from previous batches.

There are three output modes:

- ***Complete Mode*** where the entire output table is written to the Sink.
- ***Append Mode*** where only the new rows appended in the Result Table since the last trigger will be processed by the Sink.
- ***Update Mode*** where only the rows that were updated in the result table since the last trigger will be written to the Sink.

In this notebook, the *Update Mode* and *Console* output sink will be used.

- Results (i.e., only the updated records) will be displayed on the screen (i.e., on the "terminal").
- However, the Sink could be a database, a file, Kafka, or more, depending on the needs and applications.

Create and Start a Spark Session

```
# import the python libraries to create/connect to a Spark Session
from pyspark.sql import SparkSession

# build a SparkSession
#   connect to the master node on the port where the master node is
listening (7077)
#   declare the app name
#   configure the executor memory to 512 MB
#   either *connect* or *create* a new Spark Context
spark = SparkSession.builder \
    .master("spark://spark-master:7077")\
    .appName("My streaming spark application")\
    .config("spark.executor.memory", "512m")\
    .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
    .config("spark.sql.execution.arrow.pyspark.fallback.enabled",
"false")\
    .config("spark.sql.streaming.forceDeleteTempCheckpointLocation",
"true")\
    .config("spark.sql.adaptive.enabled", "false")\
    .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
24/04/08 13:57:11 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where
applicable
```

```
spark
```

```
<pyspark.sql.session.SparkSession at 0x7dbf7014a010>
```

TCP Socket Source

For this example, Spark will read data from a TCP socket using Spark Structured Streaming.

A TCP socket is a communication endpoint used to establish a connection between two devices over a network. You can think of it as a telephone connection: two endpoints have to establish a connection; once the connection is established, communication can occur, with data transfer; as soon as one of the two ends interrupts the connection, the whole communication is lost.

We will generate a dummy data stream representing fake credit card transactions.

A simple Python program will be used to create this data stream. You will be able to find it in `utils/producer.py`. When executed, the producer will try to establish a TCP connection and send data on port `5555` of a given `host` (`spark-master` in our case).

Before executing the producer program, take a moment to review the `producer.py` code to understand how it works. It's important to understand the logic of the program before using it to generate the streaming data.

```
! cat utils/producer.py
```

```
import socket
import json
import time
import random
import argparse

# Define some lists of first and last names to use for generating
random messages
first_names=('John','Andy','Joe','Alice','Jill')
last_names=('Johnson','Smith','Jones', 'Millers','Darby')

# Define a function for sending messages over the socket
def send_messages(client_socket):
    try:
        while 1:
            # Generate a random message with a random name, surname,
amount, delta_t, and flag
            msg = {
                'name': random.choice(first_names),
                'surname': random.choice(last_names),
                'amount':
float('{:.2f}'.format(random.random()*1000)),
                'delta_t': float('{:.2f}'.format(random.random()*10)),
                'flag': int(random.choices([0,1], weights=[0.8, 0.2])
[0])
```

```
            }
            try:
              # Encode the message as JSON and send it over the socket
              client_socket.send((json.dumps(msg)+"\n").encode('utf-
8'))
              # Sleep for a short amount of time to avoid overwhelming
the network
              time.sleep(0.2)
            except socket.error:
              exit()

    except KeyboardInterrupt:
        # If the user presses Ctrl+C, exit gracefully
        exit()

if __name__ == "__main__":

    # Parse command-line arguments to determine the hostname to use
    parser = argparse.ArgumentParser()
    parser.add_argument('--hostname', type=str, required=True)
    args = parser.parse_args()
    print('Using hostname:', args.hostname)

    # Create a new socket and bind it to the specified hostname and
port
    new_skt = socket.socket()
    host = args.hostname
    port = 5555
    new_skt.bind((host, port))
    print("Now listening on port: %s" % str(port))

    # Wait for a client to connect to the socket
    new_skt.listen(5) #  waiting for client connection.
    c, addr = new_skt.accept()
    print("Received request from: " + str(addr))
    # connection established, send messaged
    send_messages(c)
```

The producer will generate new records in the form of a random combination of:

- `name`
- `surname`
- `amount`: amount of the credit card transaction
- `delta_t`: time between transactions
- `flag`: random flag to indicate if potentially fraudulent or not

This information will be formatted into a `.json` data format

Creating the streaming DataFrame from a TCP socket source

To inform Spark that the data source will be a TCP socket located at a specific `hostname` and `port`, we can use the options `host` and `port` methods.

- When declaring this source a message appears, warning that the TCP source is only meant for testing purposes.

Refer to the documentation for additional available options.

Notice that the syntax is equivalent to the one used for example to read a set of files from a disk.

```
# the hostname and port number
hostname = "spark-master"
portnumber = 5555

rawMessagesDf = (
    spark
    .readStream
    .format("socket")
    .option("host", hostname)
    .option("port", portnumber)
    .load()
)
```

```
24/04/08 14:07:10 WARN TextSocketSourceProvider: The socket source
should not be used for production applications! It does not support
recovery.
```

Start the python producer.py script

From a terminal/WSL, connect to the `spark-master` Docker container using the command

```
docker exec -it spark-master bash
```

From inside the docker container, move to the `/mapd-workspace` folder and execute the python script with the option `--hostname spark-master`:

```
python /mapd-workspace/notebooks/utils/producer.py --hostname spark-
master
```

The producer application will be automatically closed when the streaming application terminates.

Running the first streaming application

The first streaming query will be a simple `show` of the DataFrame

- No processing between `rawMessagesDf` and output sink (`writeStream`)

Output mode is set to `update`, hence every new message received by the TCP source will be processed by the output sink.

- Output format `console` indicates that the output will be displayed on the screen before being discarded

The streaming query is triggered every 2 seconds

- refer to the documentation for additional trigger types.

```
query = (
    rawMessagesDf
    .writeStream
    .outputMode("update")
    .format("console")
    .trigger(processingTime='2 seconds')
    .option("truncate", False)
    .start()
)
```

```
24/04/08 14:09:20 WARN ResolveWriteToStream: Temporary checkpoint
location created which is deleted normally when the query didn't fail:
/tmp/temporary-34548789-0d0a-4d6e-96dc-dd63f24ca377. 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.


-------------------------------------------
Batch: 0
-------------------------------------------
+-----+
|value|
+-----+
+-----+


24/04/08 14:09:24 WARN ProcessingTimeExecutor: Current batch is
falling behind. The trigger interval is 2000 milliseconds, but spent
3764 milliseconds

-------------------------------------------
Batch: 1
-------------------------------------------
+--------------------------------------------------------------------------
----------------+
|value
|
+--------------------------------------------------------------------------
----------------+
|{"name": "Alice", "surname": "Johnson", "amount": 975.38, "delta_t":
0.12, "flag": 0}|
```

```
|{"name": "John", "surname": "Jones", "amount": 559.37, "delta_t":
8.81, "flag": 1}   |
|{"name": "Alice", "surname": "Darby", "amount": 948.03, "delta_t":
0.68, "flag": 0}  |
|{"name": "Andy", "surname": "Jones", "amount": 428.83, "delta_t":
6.05, "flag": 0}   |
|{"name": "Alice", "surname": "Jones", "amount": 814.96, "delta_t":
1.88, "flag": 0}  |
|{"name": "Andy", "surname": "Darby", "amount": 588.99, "delta_t":
7.39, "flag": 0}   |
|{"name": "Alice", "surname": "Jones", "amount": 871.28, "delta_t":
2.18, "flag": 0}  |
|{"name": "Joe", "surname": "Johnson", "amount": 481.53, "delta_t":
1.59, "flag": 0}  |
|{"name": "Joe", "surname": "Darby", "amount": 424.27, "delta_t":
6.55, "flag": 0}    |
|{"name": "Alice", "surname": "Jones", "amount": 878.46, "delta_t":
0.26, "flag": 0}  |
|{"name": "John", "surname": "Jones", "amount": 882.98, "delta_t":
1.43, "flag": 0}   |
|{"name": "Andy", "surname": "Darby", "amount": 342.53, "delta_t":
5.47, "flag": 0}   |
|{"name": "Alice", "surname": "Millers", "amount": 721.46, "delta_t":
1.17, "flag": 0}|
|{"name": "Joe", "surname": "Smith", "amount": 671.47, "delta_t": 1.7,
"flag": 0}     |
|{"name": "Alice", "surname": "Johnson", "amount": 851.12, "delta_t":
7.44, "flag": 0}|
|{"name": "John", "surname": "Millers", "amount": 510.92, "delta_t":
9.25, "flag": 0} |
|{"name": "Alice", "surname": "Smith", "amount": 298.08, "delta_t":
0.15, "flag": 0}  |
|{"name": "Jill", "surname": "Johnson", "amount": 580.68, "delta_t":
2.47, "flag": 0} |
+--------------------------------------------------------------------
----------------+


24/04/08 14:09:27 WARN ProcessingTimeExecutor: Current batch is
falling behind. The trigger interval is 2000 milliseconds, but spent
2512 milliseconds

-------------------------------------------
Batch: 2
-------------------------------------------
+--------------------------------------------------------------------
---------------+
|value
|
+--------------------------------------------------------------------
```

```
---------------+
|{"name": "Alice", "surname": "Darby", "amount": 719.57, "delta_t":
2.23, "flag": 0} |
|{"name": "Andy", "surname": "Jones", "amount": 646.64, "delta_t":
9.68, "flag": 0}  |
|{"name": "Andy", "surname": "Millers", "amount": 529.3, "delta_t":
1.23, "flag": 1} |
|{"name": "Jill", "surname": "Darby", "amount": 971.99, "delta_t":
3.2, "flag": 1}   |
|{"name": "Jill", "surname": "Millers", "amount": 497.09, "delta_t":
3.11, "flag": 0}|
|{"name": "Alice", "surname": "Jones", "amount": 162.58, "delta_t":
3.24, "flag": 0} |
|{"name": "Joe", "surname": "Jones", "amount": 748.83, "delta_t":
6.59, "flag": 0}   |
|{"name": "Andy", "surname": "Smith", "amount": 954.57, "delta_t":
6.73, "flag": 0}  |
|{"name": "Alice", "surname": "Smith", "amount": 916.08, "delta_t":
9.93, "flag": 1} |
|{"name": "Jill", "surname": "Darby", "amount": 358.87, "delta_t":
4.59, "flag": 0}  |
|{"name": "Jill", "surname": "Millers", "amount": 254.44, "delta_t":
3.38, "flag": 0}|
|{"name": "Joe", "surname": "Johnson", "amount": 609.45, "delta_t":
0.19, "flag": 0} |
+--------------------------------------------------------------------
---------------+

-------------------------------------------
Batch: 3
-------------------------------------------
+--------------------------------------------------------------------
--------------+
|value
|
+--------------------------------------------------------------------
--------------+
|{"name": "Joe", "surname": "Darby", "amount": 62.9, "delta_t": 1.53,
"flag": 1}    |
|{"name": "Joe", "surname": "Johnson", "amount": 517.35, "delta_t":
5.57, "flag": 0}|
|{"name": "John", "surname": "Smith", "amount": 588.87, "delta_t":
6.7, "flag": 0}  |
+--------------------------------------------------------------------
--------------+

-------------------------------------------
Batch: 4
-------------------------------------------
```

```
+--------------------------------------------------------------------
---------------+
|value
|
+--------------------------------------------------------------------
---------------+
|{"name": "Jill", "surname": "Millers", "amount": 271.17, "delta_t":
3.23, "flag": 0}|
|{"name": "John", "surname": "Millers", "amount": 479.89, "delta_t":
4.03, "flag": 0}|
|{"name": "Jill", "surname": "Jones", "amount": 499.63, "delta_t":
5.67, "flag": 1}  |
|{"name": "Jill", "surname": "Johnson", "amount": 649.46, "delta_t":
1.19, "flag": 0}|
|{"name": "John", "surname": "Johnson", "amount": 32.09, "delta_t":
9.89, "flag": 0} |
|{"name": "Andy", "surname": "Millers", "amount": 693.94, "delta_t":
4.79, "flag": 0}|
|{"name": "Alice", "surname": "Darby", "amount": 552.06, "delta_t":
1.82, "flag": 0} |
|{"name": "Jill", "surname": "Millers", "amount": 268.27, "delta_t":
7.05, "flag": 1}|
|{"name": "Andy", "surname": "Millers", "amount": 643.94, "delta_t":
3.27, "flag": 0}|
|{"name": "Jill", "surname": "Jones", "amount": 602.2, "delta_t":
8.96, "flag": 0}   |
+--------------------------------------------------------------------
---------------+

-------------------------------------------
Batch: 5
-------------------------------------------
+--------------------------------------------------------------------
----------------+
|value
|
+--------------------------------------------------------------------
----------------+
|{"name": "Alice", "surname": "Jones", "amount": 668.65, "delta_t":
9.23, "flag": 0}  |
|{"name": "John", "surname": "Smith", "amount": 461.55, "delta_t":
4.5, "flag": 1}    |
|{"name": "Alice", "surname": "Johnson", "amount": 254.08, "delta_t":
8.53, "flag": 0}|
|{"name": "Joe", "surname": "Jones", "amount": 322.51, "delta_t":
8.43, "flag": 0}    |
|{"name": "Alice", "surname": "Darby", "amount": 799.45, "delta_t":
3.5, "flag": 0}   |
|{"name": "Jill", "surname": "Jones", "amount": 936.19, "delta_t":
```

```
3.61, "flag": 0}   |
|{"name": "Alice", "surname": "Millers", "amount": 54.63, "delta_t":
2.12, "flag": 0} |
|{"name": "Andy", "surname": "Jones", "amount": 183.57, "delta_t":
3.59, "flag": 0}   |
|{"name": "Alice", "surname": "Millers", "amount": 796.4, "delta_t":
1.79, "flag": 0} |
|{"name": "Andy", "surname": "Smith", "amount": 486.78, "delta_t":
8.86, "flag": 0}   |
+--------------------------------------------------------------------
----------------+

-------------------------------------------
Batch: 6
-------------------------------------------
+--------------------------------------------------------------------
---------------+
|value
|
+--------------------------------------------------------------------
---------------+
|{"name": "Jill", "surname": "Millers", "amount": 548.08, "delta_t":
2.79, "flag": 0}|
|{"name": "Alice", "surname": "Smith", "amount": 514.66, "delta_t":
8.18, "flag": 1} |
|{"name": "John", "surname": "Millers", "amount": 127.69, "delta_t":
3.38, "flag": 0}|
|{"name": "Jill", "surname": "Johnson", "amount": 64.42, "delta_t":
3.81, "flag": 0} |
|{"name": "John", "surname": "Smith", "amount": 172.0, "delta_t":
9.02, "flag": 0}   |
|{"name": "Alice", "surname": "Smith", "amount": 370.76, "delta_t":
8.49, "flag": 0} |
|{"name": "Jill", "surname": "Darby", "amount": 444.17, "delta_t":
7.15, "flag": 0}  |
|{"name": "Andy", "surname": "Jones", "amount": 973.06, "delta_t":
0.48, "flag": 0}  |
|{"name": "John", "surname": "Darby", "amount": 755.07, "delta_t":
2.23, "flag": 0}  |
|{"name": "John", "surname": "Millers", "amount": 656.43, "delta_t":
7.55, "flag": 1}|
+--------------------------------------------------------------------
---------------+

-------------------------------------------
Batch: 7
-------------------------------------------
+--------------------------------------------------------------------
---------------+
```

```
|value
|
+--------------------------------------------------------------------
---------------+
|{"name": "John", "surname": "Smith", "amount": 635.73, "delta_t":
4.86, "flag": 0}  |
|{"name": "Joe", "surname": "Smith", "amount": 379.65, "delta_t":
2.38, "flag": 0}   |
|{"name": "Joe", "surname": "Darby", "amount": 37.09, "delta_t": 3.94,
"flag": 0}    |
|{"name": "Joe", "surname": "Millers", "amount": 10.79, "delta_t":
9.19, "flag": 1}  |
|{"name": "Alice", "surname": "Darby", "amount": 953.93, "delta_t":
6.0, "flag": 0}  |
|{"name": "Joe", "surname": "Millers", "amount": 731.88, "delta_t":
6.49, "flag": 0} |
|{"name": "John", "surname": "Johnson", "amount": 459.51, "delta_t":
5.93, "flag": 0}|
|{"name": "Jill", "surname": "Jones", "amount": 237.68, "delta_t":
5.11, "flag": 0}  |
|{"name": "Joe", "surname": "Jones", "amount": 652.26, "delta_t":
3.95, "flag": 0}   |
|{"name": "Andy", "surname": "Jones", "amount": 444.47, "delta_t":
6.1, "flag": 0}   |
+--------------------------------------------------------------------
---------------+

-------------------------------------------
Batch: 8
-------------------------------------------
+--------------------------------------------------------------------
---------------+
|value
|
+--------------------------------------------------------------------
---------------+
|{"name": "Jill", "surname": "Jones", "amount": 785.62, "delta_t":
3.5, "flag": 0}   |
|{"name": "Jill", "surname": "Smith", "amount": 991.63, "delta_t":
6.23, "flag": 0}  |
|{"name": "John", "surname": "Darby", "amount": 239.91, "delta_t":
3.51, "flag": 0}  |
|{"name": "John", "surname": "Johnson", "amount": 386.0, "delta_t":
5.73, "flag": 0} |
|{"name": "John", "surname": "Jones", "amount": 794.23, "delta_t":
3.8, "flag": 0}   |
|{"name": "Joe", "surname": "Jones", "amount": 498.37, "delta_t":
9.05, "flag": 0}   |
|{"name": "Andy", "surname": "Millers", "amount": 260.55, "delta_t":
```

```
9.69, "flag": 1}|
|{"name": "Andy", "surname": "Millers", "amount": 623.01, "delta_t":
4.47, "flag": 0}|
|{"name": "Andy", "surname": "Darby", "amount": 118.91, "delta_t":
6.37, "flag": 0}  |
|{"name": "John", "surname": "Darby", "amount": 753.02, "delta_t":
8.96, "flag": 0}  |
+--------------------------------------------------------------------
---------------+
```

Run this cell to stop the streaming query execution

```
query.stop()
```

Example of streaming query: data parsing

Data received from the TCP source is seen as a `string` by the spark application

We first must develop an application parsing the string and creating a column for each `json` field in order to start processing the dataset using the DataFrame API functionalities

It can be useful to start with a set of test data, to develop this query.

The same Spark code (transformations/actions) used for processing a "static" DataFrame can be used for the streaming context!

```
# dummy data for testing purposes
testData = [
    '{"name": "Jill", "surname": "Millers", "amount": 736.56,
"delta_t": 7.78, "flag": 0}',
    '{"name": "John", "surname": "Johnson", "amount": 986.47,
"delta_t": 3.9, "flag": 0}',
    '{"name": "John", "surname": "Jones", "amount": 249.9, "delta_t":
0.62, "flag": 1}',
    '{"name": "Andy", "surname": "Jones", "amount": 950.95, "delta_t":
6.02, "flag": 0}',
    '{"name": "Jill", "surname": "Millers", "amount": 724.32,
"delta_t": 9.19, "flag": 0}',
    '{"name": "John", "surname": "Johnson", "amount": 850.07,
"delta_t": 7.33, "flag": 1}',
    '{"name": "Andy", "surname": "Smith", "amount": 557.48, "delta_t":
9.64, "flag": 0}',
    '{"name": "Alice", "surname": "Darby", "amount": 424.75,
"delta_t": 7.76, "flag": 0}'
]
```

Create a Spark DataFrame by importing the `testData`.

It can be useful to investigate the `pyspark.sql.types` to check whether there is any helping function that could be used for this purpose.

```
from pyspark.sql.types import StringType

# create a spark dataframe from testdata
testDf = spark.createDataFrame(testData, StringType())

# show the dataframe content
testDf.show(n=5, truncate=False)
```

```
[Stage 9:>                                                          (0
+ 1) / 1]

+-----------------------------------------------------------------------
---------------+
|value
|
+-----------------------------------------------------------------------
---------------+
|{"name": "Jill", "surname": "Millers", "amount": 736.56, "delta_t":
7.78, "flag": 0}|
|{"name": "John", "surname": "Johnson", "amount": 986.47, "delta_t":
3.9, "flag": 0} |
|{"name": "John", "surname": "Jones", "amount": 249.9, "delta_t":
0.62, "flag": 1}   |
|{"name": "Andy", "surname": "Jones", "amount": 950.95, "delta_t":
6.02, "flag": 0}  |
|{"name": "Jill", "surname": "Millers", "amount": 724.32, "delta_t":
9.19, "flag": 0}|
+-----------------------------------------------------------------------
---------------+
only showing top 5 rows
```

Data are now in the same format as the one received by the socket data source.

The function `from_json` can be used to parse a `json` string with a given schema. Aa always, have a look at the documentation before using it.

Start by defining a schema for our data.

```
import pyspark.sql.functions as f
from pyspark.sql.types import StructField, StructType, StringType,
DoubleType, IntegerType

# create the schema
schema = StructType(
```

```
    [
        StructField("name", StringType()),
        StructField("surname", StringType()),
        StructField("amount", DoubleType()),
        StructField("delta_t", DoubleType()),
        StructField("flag", IntegerType())
    ]
)
```

And create a "parsed" DataFrame to verify if our schema is properly addressing the data types.

```
# parse the dataframe and show its structure
parsedTestDf = testDf.select(f.from_json('value', schema=schema))

parsedTestDf.printSchema()
```

```
root
 |-- from_json(value): struct (nullable = true)
 |    |-- name: string (nullable = true)
 |    |-- surname: string (nullable = true)
 |    |-- amount: double (nullable = true)
 |    |-- delta_t: double (nullable = true)
 |    |-- flag: integer (nullable = true)
```

The structure obtained using `from_json` is actually inherently nested... The `json` messages were correctly parsed, but a nested DataFrame is returned, which should be flattened before being able to use it as a "plain DataFrame".

As discussed in the DataFrame notebook, this can be easily solved in a variety of ways, for instance by selecting the columns we are interested in using from now on...

As an example, the `name` and `surname` columns could be selected.

```
# show values from the parsedTestDf
parsedTestDf.show(n=5, truncate=False)
```

```
[Stage 12:>                                                         (0
+ 1) / 1]

+--------------------------------+
|from_json(value)                |
+--------------------------------+
|{Jill, Millers, 736.56, 7.78, 0}|
|{John, Johnson, 986.47, 3.9, 0} |
|{John, Jones, 249.9, 0.62, 1}   |
|{Andy, Jones, 950.95, 6.02, 0}  |
|{Jill, Millers, 724.32, 9.19, 0}|
+--------------------------------+
```

```
only showing top 5 rows
```

```
# re-create the parsedTestDf by
# - aliasing the values produced by from_json as `data`
# - selecting the columns `data.name` and `data.surname`
parsedTestDf = (
    testDf
    .select(f.from_json('value', schema=schema).alias('data'))
    .select(f.col('data.name'), f.col('data.surname'))
)

# NB: data.* will select all the columns, as in plain SQL

# print the new schema
parsedTestDf.printSchema()
```

```
root
 |-- name: string (nullable = true)
 |-- surname: string (nullable = true)
```

```
# and show the new content
parsedTestDf.show(5)
```

```
+----+-------+
|name|surname|
+----+-------+
|Jill|Millers|
|John|Johnson|
|John|  Jones|
|Andy|  Jones|
|Jill|Millers|
+----+-------+
only showing top 5 rows
```

From static to streaming queries

Now that we have developed a way to extract and interpret the data using a static example, we can reuse the very same code for the streaming query, by simply "chaining" the application to the source and directing its results to the sink.

NB: Remember to restart the producer application before starting the queries.

Re-create the "raw" DataFrame connecting the Spark Structured Streaming Context to the input TCP socket.

Do not `start` the stream just yet!

```
# the hostname and port number
hostname = "spark-master"
portnumber = 5555

# recreate the streaming raw dataframe
rawMessagesDf = (
    spark
    .readStream
    .format("socket")
    .option("host", hostname)
    .option("port", portnumber)
    .load()
)
```

```
24/04/08 14:41:52 WARN TextSocketSourceProvider: The socket source
should not be used for production applications! It does not support
recovery.
```

Starting from the streaming DataFrame, issue the appropriate transformations to interpret its `json` format, and prepare a new parsed DataFrame including all (`*`) the columns.

```
# parse the json lines using the previous schema
# select all columns
parsedDf = (
    rawMessagesDf
    .select(
        f.from_json(f.col("value"), schema)
        .alias("data")
    )
    .select(f.col("data.*"))
)
```

Start the application as done previously, with:

- `outputMode=update`
- `format=console`

Choose the trigger as you prefer. A suggestion is to keep it simple, and use a time-based trigger of 2 seconds.

```
query = (
    parsedDf
    .writeStream
    .outputMode("update")
    .format("console")
    .trigger(processingTime='2 seconds')
    .option("truncate", False)
    .start()
)
```

```
24/04/08 14:41:59 WARN ResolveWriteToStream: Temporary checkpoint
location created which is deleted normally when the query didn't fail:
/tmp/temporary-30032f0d-6240-46c4-99c6-a43e0d95f9a4. 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.

-------------------------------------------
Batch: 0
-------------------------------------------
+----+-------+------+-------+----+
|name|surname|amount|delta_t|flag|
+----+-------+------+-------+----+
+----+-------+------+-------+----+

-------------------------------------------
Batch: 1
-------------------------------------------
+-----+-------+------+-------+----+
|name |surname|amount|delta_t|flag|
+-----+-------+------+-------+----+
|Joe  |Johnson|357.93|4.67   |0   |
|Joe  |Darby  |109.8 |1.92   |0   |
|Alice|Johnson|937.82|1.68   |0   |
|Jill |Smith  |31.87 |9.7    |0   |
+-----+-------+------+-------+----+

-------------------------------------------
Batch: 2
-------------------------------------------
+-----+-------+------+-------+----+
|name |surname|amount|delta_t|flag|
+-----+-------+------+-------+----+
|Jill |Smith  |561.21|6.31   |1   |
|Andy |Johnson|297.37|2.33   |0   |
|Alice|Millers|40.42 |4.51   |0   |
|Joe  |Darby  |231.14|5.86   |0   |
|Jill |Smith  |996.52|2.74   |0   |
|John |Johnson|118.78|8.12   |0   |
|Alice|Smith  |501.95|9.07   |0   |
|Andy |Millers|963.41|0.1    |1   |
|Jill |Smith  |231.23|8.29   |1   |
|Alice|Jones  |883.74|9.97   |0   |
+-----+-------+------+-------+----+

-------------------------------------------
Batch: 3
-------------------------------------------
+----+-------+------+-------+----+
|name|surname|amount|delta_t|flag|
```

```
+----+-------+------+-------+----+
|Joe |Darby  |420.05|2.2    |0   |
|Jill|Millers|41.17 |3.35   |0   |
|Jill|Darby  |490.93|6.83   |0   |
|Andy|Jones  |348.0 |1.45   |0   |
|John|Jones  |20.42 |0.46   |0   |
|John|Millers|990.26|0.24   |0   |
|Andy|Millers|308.16|0.8    |0   |
|Andy|Jones  |680.96|5.93   |0   |
|John|Smith  |935.45|1.58   |0   |
|Jill|Millers|470.14|7.66   |0   |
+----+-------+------+-------+----+

-------------------------------------------
Batch: 4
-------------------------------------------
+-----+-------+------+-------+----+
|name |surname|amount|delta_t|flag|
+-----+-------+------+-------+----+
|Joe  |Johnson|370.93|1.5    |1   |
|Alice|Smith  |884.63|7.89   |0   |
|Joe  |Darby  |278.22|5.05   |0   |
|Jill |Darby  |848.95|8.84   |1   |
|John |Smith  |637.93|4.04   |0   |
|Jill |Johnson|549.13|1.18   |0   |
|Alice|Darby  |943.03|1.26   |0   |
|Alice|Millers|523.53|2.83   |0   |
|Alice|Millers|149.92|4.64   |0   |
|Andy |Smith  |286.41|4.97   |1   |
+-----+-------+------+-------+----+


query.stop()
```

Process each bach to identify possibly fraudolent transactions

1. compute the *number of flagged transactions per batch per user* (create a unique `userID` field as the combination of *FirstLastname* to idenfity individual users)
2. identify all the "suspicios" transactions per user: all users with more than one flagged transaction per batch will be assigned a `isFraud` boolean variable
3. format the resulting `userID` and `isFraud` information in a DataFrame to mimick a "live-report" of the suspicious transactions

```
# the hostname and port number
hostname = "spark-master"
portnumber = 5555

rawMessagesDf = (
    spark
    .readStream
```

```
    .format("socket")
    .option("host", hostname)
    .option("port", portnumber)
    .load()
)
```

```
24/04/08 14:43:18 WARN TextSocketSourceProvider: The socket source
should not be used for production applications! It does not support
recovery.
```

```
# parse json lines, use a schema
parsedDf = (
    rawMessagesDf
    .select(
        f.from_json(f.col("value"), schema)
        .alias("data")
    )
    .select(f.col("data.*"))
)

# find number of transactions for each user when flag = 1
#    declare a new column to create a unique user identifier
#    this can be easily done by concatenating first- and last-name
fields
#    check the concat function from pyspark.sql.functions
numTransactions = (
    parsedDf
    .where(f.col('flag')==1)
    .withColumn('id', f.concat(f.col('name'), f.col('surname')))
    .groupBy('id')
    .count()
)

# find suspicious transactions
#    filter only users with more than one transaction per batch
#    create a "fraud" column with a value of 1 for the selected users
(check the lit function)
#    from the dataframe, project the unique id, fraud flag and number
of transaction columns
susTransactions = (
    numTransactions
    .where(f.col('count')>1)
    .withColumn('fraud', f.lit(1))
    .select(f.col('id'), f.col('fraud'),
f.col('count').alias('nTransactions'))
)

# this line is a trick to force Spark to use a small number of
partitions (4 in this example)
spark.conf.set("spark.sql.shuffle.partitions", 4)
```

```
query = (
    susTransactions
    .writeStream
    .outputMode("update")
    .format("console")
    .trigger(processingTime='5 seconds')
    .option("truncate", False)
    .start()
)
```

```
24/04/08 14:44:00 WARN ResolveWriteToStream: Temporary checkpoint
location created which is deleted normally when the query didn't fail:
/tmp/temporary-63448311-cf99-4fd6-a2d5-8a1760678b0a. 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.
```

```
-------------------------------------------
Batch: 0
-------------------------------------------
+---+-----+-------------+
|id |fraud|nTransactions|
+---+-----+-------------+
+---+-----+-------------+
```

```
-------------------------------------------
Batch: 1
-------------------------------------------
+---------+-----+-------------+
|id       |fraud|nTransactions|
+---------+-----+-------------+
|JillDarby|1    |2            |
+---------+-----+-------------+
```

```
-------------------------------------------
Batch: 2
-------------------------------------------
+----------+-----+-------------+
|id        |fraud|nTransactions|
+----------+-----+-------------+
|JoeJohnson|1    |2            |
|JillDarby |1    |3            |
+----------+-----+-------------+
```

```
-------------------------------------------
Batch: 3
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AliceJones |1    |3            |
|JoeJohnson |1    |3            |
|AndyMillers|1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 4
-------------------------------------------
+----------+-----+-------------+
|id        |fraud|nTransactions|
+----------+-----+-------------+
|JoeJohnson|1    |4            |
|AndyJones |1    |2            |
+----------+-----+-------------+



-------------------------------------------
Batch: 5
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JoeJohnson |1    |6            |
|JillJohnson|1    |2            |
|JohnJohnson|1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 6
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JoeDarby   |1    |2            |
|JillJones  |1    |2            |
|JohnJohnson|1    |3            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 7
-------------------------------------------
+----------+-----+-------------+
```

```
|id        |fraud|nTransactions|
+----------+-----+-------------+
|JillJones |1    |3            |
|JohnSmith |1    |2            |
|AliceSmith|1    |3            |
+----------+-----+-------------+

-------------------------------------------
Batch: 8
-------------------------------------------
+----------+-----+-------------+
|id        |fraud|nTransactions|
+----------+-----+-------------+
|AliceJones|1    |4            |
|JillDarby |1    |4            |
+----------+-----+-------------+

-------------------------------------------
Batch: 9
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JillJones  |1    |4            |
|JohnSmith  |1    |4            |
|JoeDarby   |1    |4            |
|JillJohnson|1    |3            |
|AliceDarby |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 10
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AndyJohnson|1    |2            |
|JohnJohnson|1    |4            |
|AndySmith  |1    |2            |
|JillDarby  |1    |5            |
+-----------+-----+-------------+



-------------------------------------------
Batch: 11
-------------------------------------------
+----------+-----+-------------+
|id        |fraud|nTransactions|
```

```
+----------+-----+-------------+
|AndyDarby |1    |2            |
|JoeJones  |1    |2            |
|JoeMillers|1    |2            |
+----------+-----+-------------+

-------------------------------------------
Batch: 12
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AndyJohnson|1    |3            |
|JohnSmith  |1    |5            |
|JoeSmith   |1    |2            |
|JohnJones  |1    |2            |
|JoeMillers |1    |3            |
+-----------+-----+-------------+


query.stop()
```

Since this query contains aggregations, output mode can be changed to `complete`. In this case, the full set of results is displayed for every batch.

```
query = (
    susTransactions
    .writeStream
    .outputMode("complete")
    .format("console")
    .trigger(processingTime='5 seconds')
    .option("truncate", False)
    .start()
)

24/04/08 14:45:38 WARN ResolveWriteToStream: Temporary checkpoint
location created which is deleted normally when the query didn't fail:
/tmp/temporary-1b268e00-5b0d-4be8-bee3-b951a94f9d3c. 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.

-------------------------------------------
Batch: 0
-------------------------------------------
+---+-----+-------------+
|id |fraud|nTransactions|
+---+-----+-------------+
+---+-----+-------------+
```

```
-------------------------------------------
Batch: 1
-------------------------------------------
+---+-----+-------------+
|id |fraud|nTransactions|
+---+-----+-------------+
+---+-----+-------------+

-------------------------------------------
Batch: 2
-------------------------------------------
+---+-----+-------------+
|id |fraud|nTransactions|
+---+-----+-------------+
+---+-----+-------------+

-------------------------------------------
Batch: 3
-------------------------------------------
+----------+-----+-------------+
|id        |fraud|nTransactions|
+----------+-----+-------------+
|AndyDarby |1    |2            |
|JoeMillers|1    |2            |
+----------+-----+-------------+

-------------------------------------------
Batch: 4
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AndyDarby  |1    |2            |
|JoeMillers |1    |2            |
|AndyMillers|1    |3            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 5
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AliceSmith |1    |2            |
|AndyDarby  |1    |2            |
|AndyMillers|1    |3            |
|JoeMillers |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
```

```
Batch: 6
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|AndyJohnson|1    |2            |
|JohnMillers|1    |2            |
|AliceSmith |1    |3            |
|AndyDarby  |1    |2            |
|AndyMillers|1    |3            |
|JoeMillers |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 7
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |2            |
|AliceSmith |1    |3            |
|AndyDarby  |1    |4            |
|JillMillers|1    |2            |
|JohnJohnson|1    |2            |
|JillDarby  |1    |2            |
|AndyMillers|1    |3            |
|JoeMillers |1    |3            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 8
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |2            |
|JillJohnson|1    |2            |
|AliceSmith |1    |4            |
|AndyDarby  |1    |4            |
|JillMillers|1    |3            |
|JohnJohnson|1    |3            |
|JillDarby  |1    |2            |
|AndyMillers|1    |4            |
|JoeMillers |1    |3            |
+-----------+-----+-------------+
```

```
-------------------------------------------
Batch: 9
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JoeDarby   |1    |2            |
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |2            |
|JillJohnson|1    |3            |
|AliceSmith |1    |4            |
|AndyDarby  |1    |4            |
|JillMillers|1    |3            |
|JohnJohnson|1    |4            |
|JillDarby  |1    |2            |
|AndyMillers|1    |4            |
|JoeMillers |1    |3            |
|JoeJones   |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 10
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JoeDarby   |1    |2            |
|JillSmith  |1    |2            |
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |3            |
|JohnSmith  |1    |2            |
|JillJohnson|1    |3            |
|AliceSmith |1    |4            |
|AndyDarby  |1    |4            |
|JillMillers|1    |3            |
|JohnJohnson|1    |4            |
|JillDarby  |1    |2            |
|AndyMillers|1    |4            |
|JoeMillers |1    |3            |
|JoeJones   |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 11
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
```

```
+-----------+-----+-------------+
|JoeDarby   |1    |2            |
|JillSmith  |1    |2            |
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |3            |
|JohnSmith  |1    |2            |
|JillJohnson|1    |3            |
|AndyJones  |1    |2            |
|AliceSmith |1    |4            |
|AndyDarby  |1    |5            |
|JillMillers|1    |3            |
|JohnJohnson|1    |4            |
|JillDarby  |1    |2            |
|AndyMillers|1    |4            |
|JoeMillers |1    |3            |
|JoeJones   |1    |2            |
+-----------+-----+-------------+

-------------------------------------------
Batch: 12
-------------------------------------------
+-----------+-----+-------------+
|id         |fraud|nTransactions|
+-----------+-----+-------------+
|JoeDarby   |1    |3            |
|JillSmith  |1    |2            |
|JohnMillers|1    |3            |
|AliceJones |1    |2            |
|AndyJohnson|1    |3            |
|JohnSmith  |1    |2            |
|JillJohnson|1    |3            |
|AndyJones  |1    |3            |
|AliceSmith |1    |4            |
|AndyDarby  |1    |5            |
|JillMillers|1    |3            |
|JohnJohnson|1    |4            |
|JillDarby  |1    |2            |
|AndyMillers|1    |4            |
|JoeMillers |1    |3            |
|JoeJones   |1    |3            |
+-----------+-----+-------------+


query.stop()
```

Stop spark worker

```
spark.stop()
```