kafka_and_spark_streaming

June 1, 2025

1 Kafka and Spark Streaming Exercise

INFORMATION: This exercise is easier on the cluster!

Kafka is an excellent tool for data engineering projects due to its distributed, fault-tolerant, and scalable architecture, which facilitates real-time data streaming and processing. It serves as a highly reliable messaging system that efficiently handles large volumes of data streams from diverse sources. Kafka's ability to decouple data producers from consumers and its support for parallel data processing make it ideal for building robust and scalable data pipelines. Additionally, Kafka's durability and fault-tolerance ensure that data is safely persisted and replicated across nodes, minimizing the risk of data loss and ensuring continuous data availability for downstream applications and analytics.

Spark Streaming enables the real-time processing of data streams with high throughput and low latency. It seamlessly integrates with Apache Spark's core APIs, allowing developers to leverage Spark's powerful data processing capabilities for streaming data. Spark Streaming supports a wide range of data sources, including Kafka, Flume, and HDFS, and can process data in near real-time, making it ideal for applications that require instant insights and timely responses.

Use Python, `pyspark`, `pandas`, `confluent-kafka` and/or `kafka-python` to send messages to a Kafka topic and analyse them with Spark Streaming:

2 Kafka

2.1 Import Necessary Libraries

[9]:
```python
import pandas as pd
import json
import time
from kafka import KafkaProducer, KafkaConsumer
from kafka import KafkaAdminClient
from kafka.admin import KafkaAdminClient, NewTopic
```

[10]:
```python
admin = KafkaAdminClient(bootstrap_servers="172.29.16.101:9092")



# 4) Zum Abschluss die Liste aller Topics anzeigen
print("Aktuelle Topics:", admin.list_topics())
```

```
Aktuelle Topics: ['wikimedia-changes', 'big-data-test', 'music', 'g3-raw-html-
test', 'roulette', 'hello-world', 'g3-hello-world', 'flights',
'__consumer_offsets', 'stocks']
```

2.2 Load a dataset to stream

Select a suitable dataset from previous exercises and split it into individual JSON messages.

[12]:
```python
df = pd.read_csv("flight_prices_small.csv")
print(df.head())
print(f"Anzahl Datensätze: {len(df)}")
```

```
                              legId  searchDate  flightDate startingAirport  \
0  9ca0e81111c683bec1012473feefd28f  2022-04-16  2022-04-17             ATL
1  98685953630e772a098941b71906592b  2022-04-16  2022-04-17             ATL
2  98d90cbc32bfbb05c2fc32897c7c1087  2022-04-16  2022-04-17             ATL
3  969a269d38eae583f455486fa90877b4  2022-04-16  2022-04-17             ATL
4  980370cf27c89b40d2833a1d5afc9751  2022-04-16  2022-04-17             ATL

  destinationAirport fareBasisCode travelDuration  elapsedDays  \
0                BOS      LA0NX0MC        PT2H29M            0
1                BOS      LA0NX0MC        PT2H30M            0
2                BOS      LA0NX0MC        PT2H30M            0
3                BOS      LA0NX0MC        PT2H32M            0
4                BOS      LA0NX0MC        PT2H34M            0

   isBasicEconomy  isRefundable  …  segmentsArrivalTimeEpochSeconds  \
0           False         False  …                       1650223560
1           False         False  …                       1650200400
2           False         False  …                       1650218700
3           False         False  …                       1650227460
4           False         False  …                       1650213180

          segmentsArrivalTimeRaw  segmentsArrivalAirportCode  \
0  2022-04-17T15:26:00.000-04:00                         BOS
1  2022-04-17T09:00:00.000-04:00                         BOS
2  2022-04-17T14:05:00.000-04:00                         BOS
3  2022-04-17T16:31:00.000-04:00                         BOS
4  2022-04-17T12:33:00.000-04:00                         BOS

   segmentsDepartureAirportCode  segmentsAirlineName segmentsAirlineCode  \
0                           ATL                Delta                  DL
1                           ATL                Delta                  DL
2                           ATL                Delta                  DL
3                           ATL                Delta                  DL
4                           ATL                Delta                  DL

  segmentsEquipmentDescription segmentsDurationInSeconds segmentsDistance  \
0                   Airbus A321                      8940              947
1                   Airbus A321                      9000              947
2                Boeing 757-200                      9000              947
3                   Airbus A321                      9120              947
4                   Airbus A321                      9240              947

  segmentsCabinCode
0             coach
1             coach
2             coach
3             coach
4             coach

[5 rows x 27 columns]
Anzahl Datensätze: 10000
```

[13]:
```
records = df.to_dict(orient="records")
print("Beispiel-Record:", records[0])
```

```
Beispiel-Record: {'legId': '9ca0e81111c683bec1012473feefd28f', 'searchDate':
'2022-04-16', 'flightDate': '2022-04-17', 'startingAirport': 'ATL',
'destinationAirport': 'BOS', 'fareBasisCode': 'LA0NX0MC', 'travelDuration':
'PT2H29M', 'elapsedDays': 0, 'isBasicEconomy': False, 'isRefundable': False,
'isNonStop': True, 'baseFare': 217.67, 'totalFare': 248.6, 'seatsRemaining': 9,
'totalTravelDistance': 947.0, 'segmentsDepartureTimeEpochSeconds': '1650214620',
'segmentsDepartureTimeRaw': '2022-04-17T12:57:00.000-04:00',
'segmentsArrivalTimeEpochSeconds': '1650223560', 'segmentsArrivalTimeRaw':
'2022-04-17T15:26:00.000-04:00', 'segmentsArrivalAirportCode': 'BOS',
'segmentsDepartureAirportCode': 'ATL', 'segmentsAirlineName': 'Delta',
'segmentsAirlineCode': 'DL', 'segmentsEquipmentDescription': 'Airbus A321',
'segmentsDurationInSeconds': '8940', 'segmentsDistance': '947',
'segmentsCabinCode': 'coach'}
```

2.3 Create a producer and stream the messages

You need to use a Kafka producer to connect to a broker and send the messages to a topic.

[15]:
```
producer = KafkaProducer(
    bootstrap_servers="172.29.16.101:9092",
    value_serializer=lambda v: json.dumps(v).encode("utf-8")
)

topic = "flights"
for rec in records:
    producer.send(topic, rec)
    time.sleep(0.01)
producer.flush()
print(f"{len(records)} Nachrichten an Topic '{topic}' gesendet.")
```

```
10000 Nachrichten an Topic 'flights' gesendet.
```

2.4 Create a consumer and check if the messages can be read

A Kafka consumer can subscribe to one or more topics and process the messages. Display the messages from the previous step.

[16]:
```
consumer = KafkaConsumer(
    "flights",
    bootstrap_servers="172.29.16.101:9092",
    auto_offset_reset="earliest",
    value_deserializer=lambda v: json.loads(v.decode("utf-8"))
)
msg = next(iter(consumer))
print("Erste empfangene Nachricht:", msg.value)
```

```
Erste empfangene Nachricht: {'legId': '9ca0e81111c683bec1012473feefd28f',
'searchDate': '2022-04-16', 'flightDate': '2022-04-17', 'startingAirport':
'ATL', 'destinationAirport': 'BOS', 'fareBasisCode': 'LA0NX0MC',
'travelDuration': 'PT2H29M', 'elapsedDays': 0, 'isBasicEconomy': False,
'isRefundable': False, 'isNonStop': True, 'baseFare': 217.67, 'totalFare':
248.6, 'seatsRemaining': 9, 'totalTravelDistance': 947.0,
'segmentsDepartureTimeEpochSeconds': '1650214620', 'segmentsDepartureTimeRaw':
'2022-04-17T12:57:00.000-04:00', 'segmentsArrivalTimeEpochSeconds':
'1650223560', 'segmentsArrivalTimeRaw': '2022-04-17T15:26:00.000-04:00',
'segmentsArrivalAirportCode': 'BOS', 'segmentsDepartureAirportCode': 'ATL',
'segmentsAirlineName': 'Delta', 'segmentsAirlineCode': 'DL',
'segmentsEquipmentDescription': 'Airbus A321', 'segmentsDurationInSeconds':
'8940', 'segmentsDistance': '947', 'segmentsCabinCode': 'coach'}
```

3 Kafka and Spark Streaming

Spark can act as a Kafka consumer. This gives you the benefits of the Spark framework to process the Kafka messages.

3.1 Spark Context and Session

Initialize Spark Context and Spark Session

[ ]:
```
# TODO
```

3.2 Create a Spark DataFrame from a Kafka stream

[ ]:
```
# TODO
```

3.3 Convert the binary Kafka data to strings

```
[ ]: # TODO
```

3.4 Create a structured schema for the streamed data

Use objects like `StructType`, `StructField`, `IntegerType`, `BooleanType`, etc to create the schema. Afterwards apply the schema to the DataFrame.

```
[ ]: # TODO
```

3.5 Create a DataFrame grouped by a time window

E.g., the number of messages of the different types over the last minute.

```
[ ]: # TODO
```

3.6 Create a query stream of the DataFrame

Write the output of the DataFrame to a memory sink of your choice. Use the `start()` method to actually start the stream processing.

```
[ ]: # TODO
```

3.7 Export the processed data as a Pandas DataFrame and visualize it

```
[ ]: # TODO
```