

Assignment 9.2_BasitAbdul

October 28, 2022

0.1 Assignment 9.2

```
[1]: import os
import shutil
import json
from pathlib import Path

import pandas as pd

from kafka import KafkaProducer, KafkaAdminClient
from kafka.admin.new_topic import NewTopic
from kafka.errors import TopicAlreadyExistsError

from pyspark.sql import SparkSession
from pyspark.streaming import StreamingContext
from pyspark import SparkConf
from pyspark.sql.functions import window, from_json, col
from pyspark.sql.types import StringType, TimestampType, DoubleType, \
    StructField, StructType
from pyspark.sql.functions import udf

current_dir = Path(os.getcwd()).absolute()
checkpoint_dir = current_dir.joinpath('checkpoints')
locations_windowed_checkpoint_dir = checkpoint_dir.
    joinpath('locations-windowed')

if locations_windowed_checkpoint_dir.exists():
    shutil.rmtree(locations_windowed_checkpoint_dir)

locations_windowed_checkpoint_dir.mkdir(parents=True, exist_ok=True)
```

0.1.1 Configuration Parameters

TODO: Change the configuration parameters to the appropriate values for your setup.

```
[2]: config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Abdul',
```

```

        last_name='Basit'
    )

    config['client_id'] = '{}{}'.format(
        config['last_name'],
        config['first_name']
    )
    config['topic_prefix'] = '{}{}'.format(
        config['last_name'],
        config['first_name']
    )

    config['locations_topic'] = '{}-locations'.format(config['topic_prefix'])
    config['accelerations_topic'] = '{}-accelerations'.
    ↪format(config['topic_prefix'])
    config['windowed_topic'] = '{}-windowed'.format(config['topic_prefix'])

    config

```

```

[2]: {'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
      'first_name': 'Abdul',
      'last_name': 'Basit',
      'client_id': 'BasitAbdul',
      'topic_prefix': 'BasitAbdul',
      'locations_topic': 'BasitAbdul-locations',
      'accelerations_topic': 'BasitAbdul-accelerations',
      'windowed_topic': 'BasitAbdul-windowed'}

```

0.1.2 Create Topic Utility Function

The `create_kafka_topic` helps create a Kafka topic based on your configuration settings. For instance, if your first name is *John* and your last name is *Doe*, `create_kafka_topic('locations')` will create a topic with the name `DoeJohn-locations`. The function will not create the topic if it already exists.

```

[3]: def create_kafka_topic(topic_name, config=config, num_partitions=1,
    ↪replication_factor=1):
    bootstrap_servers = config['bootstrap_servers']
    client_id = config['client_id']
    topic_prefix = config['topic_prefix']
    name = '{}-{}'.format(topic_prefix, topic_name)

    admin_client = KafkaAdminClient(
        bootstrap_servers=bootstrap_servers,
        client_id=client_id
    )

```

```

topic = NewTopic(
    name=name,
    num_partitions=num_partitions,
    replication_factor=replication_factor
)

topic_list = [topic]
try:
    admin_client.create_topics(new_topics=topic_list)
    print('Created topic "{}"'.format(name))
except TopicAlreadyExistsError as e:
    print('Topic "{}" already exists'.format(name))

create_kafka_topic('windowed')

```

Created topic "BasitAbdul-windowed"

TODO: This code is identical to the code used in 9.1 to publish acceleration and location data to the LastnameFirstname-simple topic. You will need to add in the code you used to create the df_accelerations dataframe. In order to read data from this topic, make sure that you are running the notebook you created in assignment 8 that publishes acceleration and location data to the LastnameFirstname-simple topic.

```

[4]: spark = SparkSession\
    .builder\
    .appName("Assignment09")\
    .getOrCreate()

df_locations = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
    .option("subscribe", config['locations_topic']) \
    .load()

## TODO: Add code to create the df_accelerations dataframe
df_accelerations = spark\
    .readStream.format("kafka")\
    .option("kafka.bootstrap.servers", config['bootstrap_servers'][0])\
    .option("subscribe", config['accelerations_topic'])\
    .load()

```

The following code defines a Spark schema for location and acceleration data as well as a user-defined function (UDF) for parsing the location and acceleration JSON data.

```

[5]: location_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
    StructField('id', StringType(), nullable=True),

```

```

    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('course', DoubleType(), nullable=True),
    StructField('latitude', DoubleType(), nullable=True),
    StructField('longitude', DoubleType(), nullable=True),
    StructField('geohash', StringType(), nullable=True),
    StructField('speed', DoubleType(), nullable=True),
    StructField('accuracy', DoubleType(), nullable=True),
  ])

  acceleration_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
    StructField('id', StringType(), nullable=True),
    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('x', DoubleType(), nullable=True),
    StructField('y', DoubleType(), nullable=True),
    StructField('z', DoubleType(), nullable=True),
  ])

  udf_parse_acceleration = udf(lambda x: json.loads(x.decode('utf-8')),
    ↳ acceleration_schema)
  udf_parse_location = udf(lambda x: json.loads(x.decode('utf-8')),
    ↳ location_schema)

```

See <http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time> for details on how to implement windowed operations.

The following code selects the `timestamp` column from the `df_locations` dataframe that reads from the `LastnameFirstname-locations` topic and parses the binary value using the `udf_parse_location` UDF and defines the result to the `json_value` column.

```

df_locations \
  .select(
    col('timestamp'),
    udf_parse_location(df_locations['value']).alias('json_value')
  )

```

From here, you can select data from the `json_value` column using the `select` method. For instance, if you saved the results of the previous code snippet to `df_locations_parsed` you could select columns from the `json_value` field and assign them aliases using the following code.

```

df_locations_parsed.select(
  col('timestamp'),
  col('json_value.ride_id').alias('ride_id'),
  col('json_value.uuid').alias('uuid'),
  col('json_value.speed').alias('speed')
)

```

Next, you will want to add a watermark and group by `ride_id` and `speed` using a window duration

of *30 seconds* and a slide duration of *15 seconds*. Use the `withWatermark` method in conjunction with the `groupBy` method. The [Spark streaming documentation](#) should provide examples of how to do this.

Next use the mean aggregation method to compute the average values and rename the column `avg(speed)` to `value` and the column `ride_id` to `key`. The reason you are renaming these values is that the PySpark Kafka API expects `key` and `value` as inputs. In a production example, you would setup serialization that would handle these details for you.

When you are finished, you should have a streaming query with `key` and `value` as columns.

```
[6]: df_locations_parsed = df_locations \
      .select(
        col('timestamp'),
        udf_parse_location(df_locations['value']).alias('json_value')
      )

df_locations_parsed.select(
  col('timestamp'),
  col('json_value.ride_id').alias('ride_id'),
  col('json_value.uuid').alias('uuid'),
  col('json_value.speed').alias('speed')
)

df_locations_parsed.printSchema()
```

```
root
|-- timestamp: timestamp (nullable = true)
|-- json_value: struct (nullable = true)
|   |-- offset: double (nullable = true)
|   |-- id: string (nullable = true)
|   |-- ride_id: string (nullable = true)
|   |-- uuid: string (nullable = true)
|   |-- course: double (nullable = true)
|   |-- latitude: double (nullable = true)
|   |-- longitude: double (nullable = true)
|   |-- geohash: string (nullable = true)
|   |-- speed: double (nullable = true)
|   |-- accuracy: double (nullable = true)
```

```
[7]: windowedSpeeds = df_locations_parsed \
      .withWatermark("timestamp", "10 seconds") \
      .groupBy("json_value.ride_id", window("timestamp", "10 seconds", "5_
      ↪seconds"), 'json_value.speed') \
      .mean('json_value.speed') \
      .withColumnRenamed("ride_id", "key") \
      .withColumnRenamed("avg(json_value.speed AS `speed`)", "value") \
      .select(col('key'), col('value'))
```

```
windowedSpeeds.printSchema()
```

```
root
|-- key: string (nullable = true)
|-- value: double (nullable = true)
```

In the previous Jupyter cells, you should have created the `windowedSpeeds` streaming query. Next, you will need to write that to the `LastNameFirstname-windowed` topic. If you created the `windowsSpeeds` streaming query correctly, the following should publish the results to the `LastNameFirstname-windowed` topic.

```
[8]: ds_locations_windowed = windowedSpeeds \
      .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
      .writeStream \
      .format("kafka") \
      .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
      .option("topic", config['windowed_topic']) \
      .option("checkpointLocation", str(locations_windowed_checkpoint_dir)) \
      .start()

try:
    ds_locations_windowed.awaitTermination()
except KeyboardInterrupt:
    print("STOPPING STREAMING DATA")
```

STOPPING STREAMING DATA

```
[ ]:
```