Assignment 9.3

By Kurt Stoneburner

```
In [1]:   1  import os
          2  import shutil
          3  import json
          4  from pathlib import Path
          5
          6  import pandas as pd
          7
          8  from kafka import KafkaProducer, KafkaAdminClient
          9  from kafka.admin.new_topic import NewTopic
         10  from kafka.errors import TopicAlreadyExistsError
         11
         12  from pyspark.sql import SparkSession
         13  from pyspark.streaming import StreamingContext
         14  from pyspark import SparkConf
         15  from pyspark.sql.functions import window, from_json, col, expr, to_js
         16  from pyspark.sql.types import StringType, TimestampType, DoubleType,
         17  from pyspark.sql.functions import udf
         18
         19  current_dir = Path(os.getcwd()).absolute()
         20  checkpoint_dir = current_dir.joinpath('checkpoints')
         21  joined_checkpoint_dir = checkpoint_dir.joinpath('joined')
         22  locations_checkpoint_dir = checkpoint_dir.joinpath('locations')
         23  accelerations_checkpoint_dir = checkpoint_dir.joinpath('accelerations
         24
         25  if joined_checkpoint_dir.exists():
         26      shutil.rmtree(joined_checkpoint_dir)
         27
```

Configuration Parameters

> **TODO:** Change the configuration prameters to the appropriate values for your setup.

```
In [4]:   1  config = dict(
          2      bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
          3      first_name='Kurt',
          4      last_name='Stoneburner'
          5  )
          6
          7  config['client_id'] = '{}{}'.format(
          8      config['last_name'],
          9      config['first_name']
         10  )
         11  config['topic_prefix'] = '{}{}'.format(
```

```
12        config['last_name'],
13        config['first_name']
14  )
15
16  config['locations_topic'] = '{}-locations'.format(config['topic_prefi
17  config['accelerations_topic'] = '{}-accelerations'.format(config['top
18  config['joined_topic'] = '{}-joined'.format(config['topic_prefix'])
19
```

Out[4]: {'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Kurt',
 'last_name': 'Stoneburner',
 'client_id': 'StoneburnerKurt',
 'topic_prefix': 'StoneburnerKurt',
 'locations_topic': 'StoneburnerKurt-locations',
 'accelerations_topic': 'StoneburnerKurt-accelerations',
 'joined_topic': 'StoneburnerKurt-joined'}

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.

In [5]:
```python
1   def create_kafka_topic(topic_name, config=config, num_partitions=1, r
2       bootstrap_servers = config['bootstrap_servers']
3       client_id = config['client_id']
4       topic_prefix = config['topic_prefix']
5       name = '{}-{}'.format(topic_prefix, topic_name)
6
7       admin_client = KafkaAdminClient(
8           bootstrap_servers=bootstrap_servers,
9           client_id=client_id
10      )
11
12      topic = NewTopic(
13          name=name,
14          num_partitions=num_partitions,
15          replication_factor=replication_factor
16      )
17
18      topic_list = [topic]
19      try:
20          admin_client.create_topics(new_topics=topic_list)
21          print('Created topic "{}"'.format(name))
22      except TopicAlreadyExistsError as e:
23          print('Topic "{}" already exists'.format(name))
24
```

Topic "StoneburnerKurt-joined" already exists

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.

```
In [6]:   1  spark = SparkSession\
          2      .builder\
          3      .appName("Assignment09")\
          4      .getOrCreate()
          5
          6  df_locations = spark \
          7    .readStream \
          8    .format("kafka") \
          9    .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9
         10    .option("subscribe", config['locations_topic']) \
         11    .load()
         12
         13  ## TODO: Add code to create the df_accelerations dataframe
         14  df_accelerations = spark \
         15    .readStream \
         16    .format("kafka") \
         17    .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9
         18    .option("subscribe", config['accelerations_topic']) \
         19    .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.

```
In [7]:   1  location_schema = StructType([
          2      StructField('offset', DoubleType(), nullable=True),
          3      StructField('id', StringType(), nullable=True),
          4      StructField('ride_id', StringType(), nullable=True),
          5      StructField('uuid', StringType(), nullable=True),
          6      StructField('course', DoubleType(), nullable=True),
          7      StructField('latitude', DoubleType(), nullable=True),
          8      StructField('longitude', DoubleType(), nullable=True),
          9      StructField('geohash', StringType(), nullable=True),
         10      StructField('speed', DoubleType(), nullable=True),
         11      StructField('accuracy', DoubleType(), nullable=True),
         12  ])
         13
         14  acceleration_schema = StructType([
         15      StructField('offset', DoubleType(), nullable=True),
         16      StructField('id', StringType(), nullable=True),
         17      StructField('ride_id', StringType(), nullable=True),
         18      StructField('uuid', StringType(), nullable=True),
         19      StructField('x', DoubleType(), nullable=True),
         20      StructField('y', DoubleType(), nullable=True),
         21      StructField('z', DoubleType(), nullable=True),
         22  ])
         23
         24  udf_parse_acceleration = udf(lambda x: json.loads(x.decode('utf-8')),
         25  udf_parse_location = udf(lambda x: json.loads(x.decode('utf-8')), loc
```

TODO:

- Complete the code to create the `accelerationsWithWatermark` dataframe.
  - Select the `timestamp` field with the alias `acceleration_timestamp`
  - Use the `udf_parse_acceleration` UDF to parse the JSON values
  - Select the `ride_id` as `acceleration_ride_id`
  - Select the `x`, `y`, and `z` columns
  - Use the same watermark timespan used in the `locationsWithWatermark` dataframe

```
In [8]:  1  locationsWithWatermark = df_locations \
         2    .select(
         3      col('timestamp').alias('location_timestamp'),
         4      udf_parse_location(df_locations['value']).alias('json_value')
         5    ) \
         6    .select(
         7      col('location_timestamp'),
         8      col('json_value.ride_id').alias('location_ride_id'),
         9      col('json_value.speed').alias('speed'),
        10      col('json_value.latitude').alias('latitude'),
        11      col('json_value.longitude').alias('longitude'),
        12      col('json_value.geohash').alias('geohash'),
        13      col('json_value.accuracy').alias('accuracy')
        14    ) \
        15    .withWatermark('location_timestamp', "2 seconds")
        16
        17
```

```
In [14]:  1  df_accelerations.printSchema()

root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

```
In [10]:  1  accelerationsWithWatermark = df_accelerations \
          2    .select(
          3      col('timestamp').alias('accelerations_timestamp'),
          4      udf_parse_acceleration(df_accelerations['value']).alias('json_val
          5    ) \
          6    .select(
          7        col('accelerations_timestamp'),
          8        col('json_value.ride_id').alias('acceleration_ride_id'),
          9        col('json_value.x').alias('x'),
         10        col('json_value.y').alias('y'),
         11        col('json_value.z').alias('z'),
         12  ) \
         13  .withWatermark('accelerations_timestamp', "2 seconds")
         14  accelerationsWithWatermark.printSchema()
```

```
root
 |-- accelerations_timestamp: timestamp (nullable = true)
 |-- acceleration_ride_id: string (nullable = true)
 |-- x: double (nullable = true)
 |-- y: double (nullable = true)
```

TODO:

- Complete the code to create the `df_joined` dataframe. See http://spark.apache.org /docs/latest/structured-streaming-programming-guide.html#stream-stream-joins (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#stream-stream-joins) for additional information.

```
In [11]:   1  df_joined = ''
           2  df_joined = locationsWithWatermark \
           3  .join(accelerationsWithWatermark,
           4        expr("""
           5        location_ride_id = acceleration_ride_id
           6        """
           7           )) \
           8  .select(
           9      col('location_ride_id').alias('ride_id'),
          10      col('location_timestamp'),
          11      col('speed'),
          12      col('latitude'),
          13      col('longitude'),
          14      col('geohash'),
          15      col('accuracy'),
          16      col('accelerations_timestamp'),
          17      col('x'),
          18      col('y'),
          19      col('z'),
          20
          21  )
          22
          23
          24
```

```
root
 |-- ride_id: string (nullable = true)
 |-- location_timestamp: timestamp (nullable = true)
 |-- speed: double (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- geohash: string (nullable = true)
 |-- accuracy: double (nullable = true)
 |-- accelerations_timestamp: timestamp (nullable = true)
 |-- x: double (nullable = true)
 |-- y: double (nullable = true)
 |-- z: double (nullable = true)
```

```
In [12]:   1  #df_joined = locationsWithWatermark \
           2  #.join(accelerationsWithWatermark)
```

If you correctly created the `df_joined` dataframe, you should be able to use the following code to create a streaming query that outputs results to the `LastnameFirstname-joined` topic.

```python
In [13]:
 1  ds_joined = df_joined \
 2    .withColumn(
 3      'value',
 4      to_json(
 5          struct(
 6              'ride_id', 'location_timestamp', 'speed',
 7              'latitude', 'longitude', 'geohash', 'accuracy',
 8              'accelerations_timestamp', 'x', 'y', 'z'
 9          )
10      )
11    ).withColumn(
12      'key', col('ride_id')
13    ) \
14    .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
15    .writeStream \
16    .format("kafka") \
17    .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9
18    .option("topic", config['joined_topic']) \
19    .option("checkpointLocation", str(joined_checkpoint_dir)) \
20    .start()
21
22  try:
23      ds_joined.awaitTermination()
24  except KeyboardInterrupt:
```

```
---------------------------------------------------------------------
-----
StreamingQueryException                   Traceback (most recent call
last)
<ipython-input-13-493b02e2e50e> in <module>
     21
     22 try:
---> 23     ds_joined.awaitTermination()
     24 except KeyboardInterrupt:
     25     print("STOPPING STREAMING DATA")

/usr/local/spark/python/pyspark/sql/streaming.py in awaitTermination(s
elf, timeout)
    101             return self._jsq.awaitTermination(int(timeout * 10
00))
    102         else:
--> 103             return self._jsq.awaitTermination()
    104
    105     @property

/usr/local/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py i
n __call__(self, *args)
   1302
   1303         answer = self.gateway_client.send_command(command)
-> 1304         return_value = get_return_value(
   1305             answer, self.gateway_client, self.target_id, self.
name)
   1306

/usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
    135                 # Hide where the exception came from that show
s a non-Pythonic
    136                 # JVM exception message.
--> 137                 raise_from(converted)
    138             else:
    139                 raise

/usr/local/spark/python/pyspark/sql/utils.py in raise_from(e)

StreamingQueryException: Writing job aborted.
=== Streaming Query ===
Identifier: [id = 46058cd5-ae85-417e-bebc-70a3138b6ea4, runId = d25b00
93-fa9f-40c3-8638-34efab131f8a]
Current Committed Offsets: {KafkaV2[Subscribe[StoneburnerKurt-location
s]]: {"StoneburnerKurt-locations":{"0":10705}},KafkaV2[Subscribe[Stone
burnerKurt-accelerations]]: {"StoneburnerKurt-accelerations":{"0":1060
2}}}
Current Available Offsets: {KafkaV2[Subscribe[StoneburnerKurt-location
s]]: {"StoneburnerKurt-locations":{"0":10706}},KafkaV2[Subscribe[Stone
burnerKurt-accelerations]]: {"StoneburnerKurt-accelerations":{"0":1060
2}}}

Current State: ACTIVE
Thread State: RUNNABLE

Logical Plan:
```

```
WriteToMicroBatchDataSource org.apache.spark.sql.kafka010.KafkaStreami
ngWrite@5f1eb19a
+- Project [cast(key#145 as string) AS key#159, cast(value#132 as stri
ng) AS value#160]
   +- Project [ride_id#120, location_timestamp#42-T2000ms, speed#48, l
atitude#49, longitude#50, geohash#51, accuracy#52, accelerations_times
tamp#66-T2000ms, x#72, y#73, z#74, value#132, ride_id#120 AS key#145]
      +- Project [ride_id#120, location_timestamp#42-T2000ms, speed#4
8, latitude#49, longitude#50, geohash#51, accuracy#52, accelerations_t
imestamp#66-T2000ms, x#72, y#73, z#74, to_json(struct(ride_id, ride_id
#120, location_timestamp, location_timestamp#42-T2000ms, speed, speed#
48, latitude, latitude#49, longitude, longitude#50, geohash, geohash#5
1, accuracy, accuracy#52, accelerations_timestamp, accelerations_times
tamp#66-T2000ms, x, x#72, y, y#73, z, z#74), Some(Etc/UTC)) AS value#1
32]
         +- Project [location_ride_id#47 AS ride_id#120, location_time
stamp#42-T2000ms, speed#48, latitude#49, longitude#50, geohash#51, acc
uracy#52, accelerations_timestamp#66-T2000ms, x#72, y#73, z#74]
            +- Join Inner, (location_ride_id#47 = acceleration_ride_id
#71)
```