Assignment 9.2

In [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,
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)
```

Configuration Parameters

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

In [2]:
```
config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Kyle',
    last_name='Morris'
)

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
```

Out[2]:
```
{'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Kyle',
 'last_name': 'Morris',
 'client_id': 'MorrisKyle',
 'topic_prefix': 'MorrisKyle',
 'locations_topic': 'MorrisKyle-locations',
 'accelerations_topic': 'MorrisKyle-accelerations',
 'windowed_topic': 'MorrisKyle-windowed'}
```

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 [3]:
```
def create_kafka_topic(topic_name, config=config, num_partitions=1, replication_f
    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')
```

```
Topic "MorrisKyle-windowed" 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 [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", "kafka.kafka.svc.cluster.local:9092") \
  .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.

In [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')), acceleratic
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 (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 (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time) 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.

In [6]:
```
windowedSpeeds = ''


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

df_select = 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')
  )

windowedSpeeds = df_select \
    .withWatermark("timestamp", "30 seconds") \
    .groupBy(
        window(df_select.timestamp, "30 seconds", "15 seconds"),
        df_select.ride_id, df_select.speed) \
    .agg({'speed':'mean'})

windowedSpeeds = windowedSpeeds.withColumnRenamed("avg(speed)", "value") \
                 .withColumnRenamed("ride_id", "key")
```

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.

In [7]:
```
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")
```

```
---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
<ipython-input-7-3530311fe543> in <module>
----> 1 ds_locations_windowed = windowedSpeeds \
      2   .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
      3   .writeStream \
      4   .format("kafka") \
      5   .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:909
2") \

/usr/local/spark/python/pyspark/sql/streaming.py in start(self, path, format, o
utputMode, partitionBy, queryName, **options)
   1209             self.queryName(queryName)
   1210         if path is None:
-> 1211             return self._sq(self._jwrite.start())
   1212         else:
   1213             return self._sq(self._jwrite.start(path))

/usr/local/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in __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 shows a non-P
ythonic
    136                 # JVM exception message.
--> 137                 raise_from(converted)
    138             else:
    139                 raise

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

AnalysisException: Append output mode not supported when there are streaming ag
gregations on streaming DataFrames/DataSets without watermark;;
Project [cast(key#73 as string) AS key#78, cast(value#68 as string) AS value#7
9]
+- Project [window#56, ride_id#46 AS key#73, speed#48, value#68]
   +- Project [window#56, ride_id#46, speed#48, avg(speed)#61 AS value#68]
```

```
      +- Aggregate [window#63, ride_id#46, speed#48], [window#63 AS window#56,
 ride_id#46, speed#48, avg(speed#48) AS avg(speed)#61]
         +- Filter ((timestamp#12 >= window#63.start) AND (timestamp#12 < windo
w#63.end))
            +- Expand [ArrayBuffer(named_struct(start, precisetimestampconversi
on(((((CASE WHEN (cast(CEIL((cast((precisetimestampconversion(timestamp#12, Tim
estampType, LongType) - 0) as double) / cast(15000000 as double))) as double) =
 (cast((precisetimestampconversion(timestamp#12, TimestampType, LongType) - 0) a
s double) / cast(15000000 as double))) THEN (CEIL((cast((precisetimestampconver
sion(timestamp#12, TimestampType, LongType) - 0) as double) / cast(15000000 as
 double))) + cast(1 as bigint)) ELSE CEIL((cast((precisetimestampconversion(tim
estamp#12, TimestampType, LongType) - 0) as double) / cast(15000000 as doubl
e))) END + cast(0 as bigint)) - cast(2 as bigint)) * 15000000) + 0), LongType,
 TimestampType), end, precisetimestampconversion((((((CASE WHEN (cast(CEIL((cas
t((precisetimestampconversion(timestamp#12, TimestampType, LongType) - 0) as do
uble) / cast(15000000 as double))) as double) = (cast((precisetimestampconversi
on(timestamp#12, TimestampType, LongType) - 0) as double) / cast(15000000 as do
uble))) THEN (CEIL((cast((precisetimestampconversion(timestamp#12, TimestampTyp
e, LongType) - 0) as double) / cast(15000000 as double))) + cast(1 as bigint))
 ELSE CEIL((cast((precisetimestampconversion(timestamp#12, TimestampType, LongT
ype) - 0) as double) / cast(15000000 as double))) END + cast(0 as bigint)) - ca
st(2 as bigint)) * 15000000) + 0) + 30000000), LongType, TimestampType)), times
tamp#12-T30000ms, ride_id#46, uuid#47, speed#48), ArrayBuffer(named_struct(star
t, precisetimestampconversion(((((CASE WHEN (cast(CEIL((cast((precisetimestampc
onversion(timestamp#12, TimestampType, LongType) - 0) as double) / cast(1500000
0 as double))) as double) = (cast((precisetimestampconversion(timestamp#12, Tim
estampType, LongType) - 0) as double) / cast(15000000 as double))) THEN (CEIL
((cast((precisetimestampconversion(timestamp#12, TimestampType, LongType) - 0)
 as double) / cast(15000000 as double))) + cast(1 as bigint)) ELSE CEIL((cast
((precisetimestampconversion(timestamp#12, TimestampType, LongType) - 0) as dou
ble) / cast(15000000 as double))) END + cast(1 as bigint)) - cast(2 as bigint))
* 15000000) + 0), LongType, TimestampType), end, precisetimestampconversion
((((((CASE WHEN (cast(CEIL((cast((precisetimestampconversion(timestamp#12, Time
stampType, LongType) - 0) as double) / cast(15000000 as double))) as double) =
 (cast((precisetimestampconversion(timestamp#12, TimestampType, LongType) - 0)
 as double) / cast(15000000 as double))) THEN (CEIL((cast((precisetimestampconv
ersion(timestamp#12, TimestampType, LongType) - 0) as double) / cast(15000000 a
s double))) + cast(1 as bigint)) ELSE CEIL((cast((precisetimestampconversion(ti
mestamp#12, TimestampType, LongType) - 0) as double) / cast(15000000 as doubl
e))) END + cast(1 as bigint)) - cast(2 as bigint)) * 15000000) + 0) + 3000000
0), LongType, TimestampType)), timestamp#12-T30000ms, ride_id#46, uuid#47, spee
d#48)], [window#63, timestamp#12-T30000ms, ride_id#46, uuid#47, speed#48]
               +- EventTimeWatermark timestamp#12: timestamp, 30 seconds
                  +- Project [timestamp#12, json_value#43.ride_id AS ride_id#4
6, json_value#43.uuid AS uuid#47, json_value#43.speed AS speed#48]
                     +- Project [timestamp#12, <lambda>(value#8) AS json_value#
43]
                        +- StreamingRelationV2 org.apache.spark.sql.kafka010.Ka
fkaSourceProvider@61b7adb8, kafka, org.apache.spark.sql.kafka010.KafkaSourcePro
vider$KafkaTable@62e91a1e, org.apache.spark.sql.util.CaseInsensitiveStringMap@6
5672ce6, [key#7, value#8, topic#9, partition#10, offset#11L, timestamp#12, time
stampType#13], StreamingRelation DataSource(org.apache.spark.sql.SparkSession@1
5c7a07d,kafka,List(),None,List(),None,Map(subscribe -> MorrisKyle-locations, ka
fka.bootstrap.servers -> kafka.kafka.svc.cluster.local:9092),None), kafka, [key
#0, value#1, topic#2, partition#3, offset#4L, timestamp#5, timestampType#6]
```

In [ ]: