

Assignment 9.3

May 15, 2023

0.1 Assignment 9.3

```
[ ]: 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, expr, to_json,
    struct, when
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')
joined_checkpoint_dir = checkpoint_dir.joinpath('joined')

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

joined_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.

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

```

        last_name='Avinaz'
    )

    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['joined_topic'] = '{}-joined'.format(config['topic_prefix'])

    config

```

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.

```

[ ]: 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('joined')

```

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.

```

[ ]: 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.

```

[ ]: 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)

```

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

```

[ ]: locationsWithWatermark = df_locations \
    .select(
        col('timestamp').alias('location_timestamp'),
        udf_parse_location(df_locations['value']).alias('json_value')
    ) \
    .select(
        col('location_timestamp'),
        col('json_value.ride_id').alias('location_ride_id'),
        col('json_value.speed').alias('speed'),
        col('json_value.latitude').alias('latitude'),
        col('json_value.longitude').alias('longitude'),
        col('json_value.geohash').alias('geohash'),
        col('json_value.accuracy').alias('accuracy')
    ) \
    .withWatermark('location_timestamp', "2 seconds")

accelerationsWithWatermark = df_accelerations \
    .select(
        col('timestamp').alias('acceleration_timestamp'),
        udf_parse_acceleration(df_accelerations['value']).alias('json_value')
    ) \
    .select(
        col('location_timestamp'),
        col('json_value.ride_id').alias('acceleration_ride_id'),

```

```

    col('json_value.x').alias('x'),
    col('json_value.y').alias('y'),
    col('json_value.z').alias('z'),
  ) \
  .withWatermark('acceleration_timestamp', "2 seconds")

```

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> for additional information.

```

[ ]: df_joined = locationsWithWatermark.join(
    accelerationsWithWatermark,
    expr("""
location_timestamp = acceleration_timestamp
"""),
    joinType = "leftOuter")
df_joined

```

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.

```

[ ]: ds_joined = df_joined \
    .withColumn(
        'value',
        to_json(
            struct(
                'ride_id', 'location_timestamp', 'speed',
                'latitude', 'longitude', 'geohash', 'accuracy',
                'acceleration_timestamp', 'x', 'y', 'z'
            )
        )
    ).withColumn(
        'key', col('ride_id')
    ) \
    .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['joined_topic']) \
    .option("checkpointLocation", str(joined_checkpoint_dir)) \
    .start()

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

```