Kafka Producer Tinto

August 2, 2021

[28]:
```
import json
import uuid

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

0.0.1 Configuration Parameters

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

[29]:
```
config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Tinto',
    last_name='Kurian'
)

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

config
```

[29]:
```
{'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Tinto',
 'last_name': 'Kurian',
 'client_id': 'KurianTinto',
 'topic_prefix': 'KurianTinto'}
```

0.0.2 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.

[30]:
```
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('locations')
create_kafka_topic('accelerations')
```

```
Topic "KurianTinto-locations" already exists
Topic "KurianTinto-accelerations" already exists
```

0.0.3 Kafka Producer

The following code creates a `KafkaProducer` object which you can use to send Python objects that are serialized as JSON.

Note: This producer serializes Python objects as JSON. This means that object must be JSON serializable. As an example, Python `DateTime` values are not JSON serializable and must be converted to a string (e.g. ISO 8601) or a numeric value (e.g. a Unix timestamp) before being sent.

[31]:
```
producer = KafkaProducer(
  bootstrap_servers=config['bootstrap_servers'],
  value_serializer=lambda x: json.dumps(x).encode('utf-8')
)
```

0.0.4 Send Data Function

The `send_data` function sends a Python object to a Kafka topic. This function adds the `topic_prefix` to the topic so `send_data('locations', data)` sends a JSON serialized message to `DoeJohn-locations`. The function also registers callbacks to let you know if the message has been sent or if an error has occured.

[32]:
```
def on_send_success(record_metadata):
    print('Message sent:\n    Topic: "{}"\n    Partition: {}\n    Offset: {}'.
 ↪format(
        record_metadata.topic,
        record_metadata.partition,
        record_metadata.offset
    ))

def on_send_error(excp):
    print('I am an errback', exc_info=excp)
    # handle exception

def send_data(topic, data, config=config, producer=producer, msg_key=None):
    topic_prefix = config['topic_prefix']
    topic_name = '{}-{}'.format(topic_prefix, topic)

    if msg_key is not None:
        key = msg_key
    else:
        key = uuid.uuid4().hex

    producer.send(
        topic_name,
        value=data,
        key=key.encode('utf-8')
    ).add_callback(on_send_success).add_errback(on_send_error)
```

[33]:
```
# example_data = dict(
#     key1='value1',
#     key2='value2'
# )

# send_data('accelerations', example_data)
```

[34]:
```
# import pyarrow.parquet as pq
# table2 = pq.read_table('/home/jovyan/dsc650/data/processed/bdd/accelerations/
 ↪')
```

[35]:
```
# table2.head()
```

[36]:
```
# import pandas as pd
# data_accelerations = pd.read_parquet('/home/jovyan/dsc650/data/processed/bdd/
 ↪accelerations/').to_json()
```

[37]:
```
# send_data('locations', data_accelerations)
```

[38]:
```
# send_data('accelerations', data_accelerations)
```

[39]:
```
import os
import json
import time
from collections import namedtuple
import heapq
import uuid
import pandas as pd
import s3fs
import pyarrow.parquet as pq

endpoint_url='https://storage.budsc.midwest-datascience.com'
s3 = s3fs.S3FileSystem(
    anon=True,
    client_kwargs={
        'endpoint_url': endpoint_url
    }
)

acceleration_columns = [
    'offset',
    'id',
    'ride_id',
    'uuid',
    'x',
    'y',
    'z',
#     't'
]
Acceleration = namedtuple('Acceleration', acceleration_columns)
def read_accelerations():
    df = pq.ParquetDataset(
        's3://data/processed/bdd/accelerations',
        filesystem=s3
    ).read_pandas().to_pandas()

    df = df[acceleration_columns].sort_values(by=['offset'])

    records = [Acceleration(*record) for record in df.to_records(index=False)]
```

```
    return records
accelerations = read_accelerations()

location_columns = [
    'offset',
    'id',
    'ride_id',
    'uuid',
    'course',
    'latitude',
    'longitude',
    'geohash',
    'speed',
    'accuracy',
#     't'
]
Location = namedtuple('Location', location_columns)
def read_locations():
    df = pq.ParquetDataset(
        's3://data/processed/bdd/locations',
        filesystem=s3
    ).read_pandas().to_pandas()

    df = df[location_columns].sort_values(by=['offset'])

    records = [Location(*record) for record in df.to_records(index=False)]

    return records

locations = read_locations()
```

[40]:
```
# heapq.heapify(accelerations)
# heapq.heapify(locations)

# heapq.heappop(accelerations)
```

[41]:
```
# send_data('accelerations', accelerations)
```

[42]:
```
# send_data('locations', locations)
```

[43]:
```
events = locations + accelerations
heapq.heapify(events)
start_time = time.time()
current_event = heapq.heappop(events)

# topic = 'locations'
```

```
# dict_data = current_event._asdict()
# dict_data['timestamp'] = (current_event.offset + start_time) * 1000.0
# send_data(topic, dict_data)
```

[44]:
```
def send_topic(topic):
    dict_data = current_event._asdict()
    dict_data['timestamp'] = (current_event.offset + start_time) * 1000.0
    while dict_data['timestamp'] == time.time():
        send_data(topic, dict_data)
```

[45]:
```
send_topic('locations')
```

[46]:
```
send_topic('accelerations')
```

[ ]:
```

```