ACIT 3855 – Lab 6B – Messaging with Kafka

Instructor	Mike Mulder (mmulder10@bcit.ca) – Sets A and B Rafi Mohammad (rafi_mohammad@bcit.ca) – Set C
Total Marks	10
Due Dates	Demo by end of next class: • Oct. 17th for Set A • Oct. 19th for Sets B and C

Purpose

- Used the Asynchronous Message Broker (Apache Kafka) setup last week to produce event messages
- Consume those messages in an existing service (Data Storage Service) and a new service (Audit Service)

Start up your VM where you deployed Zookeeper, Kafka and MySQL. Make sure all three containers are running (i.e., run docker-compose up –d)

Part 1 – Event Receiver (Lab 2) - Produce Event Messages

In your Event Receiver service (Lab 2) project, install the pykafka module. Here is the main documentation for pykafka: https://pykafka.readthedocs.io/en/latest/

Add the following to your app_conf.yml file:

```
events:
  hostname: <Your VM's Hostname>
  port: 9092
  topic: events
```

You will need to import the following into app.py:

- datetime
- json
- KafkaClient from pykafka (you will need to install pykafka in you PyCharm or with Pip)

Replace the requests.post calls in each of your two functions with the following:

Notes:

- The values in <> should now come from your configuration file.
- In the msg dictionary, the *type* value gets replaced with your event type and *reading* object gets replaced with your event object.

```
client = KafkaClient(hosts='<kafka-server>:<kafka-port>')
topic = client.topics[str.encode(<topic>)]
producer = topic.get_sync_producer()
```

```
msg = { "type": "bp",
        "datetime" :
           datetime.datetime.now().strftime(
             "%Y-%m-%dT%H:%M:%S"),
        "payload": reading }
msg_str = json.dumps(msg)
producer.produce(msg_str.encode('utf-8'))
```

You will need to hard-code your status code to 201 since you will no longer get it from the response of the requests.post call.

This will now produce the events, wrapped as the payload in the message, on the topic events.

Part 2 – Data Storage Service (Lab 3) - Consume Event Messages

In your Data Storage service (Lab 3) project, install the pykafka module.

Add the following to your app_conf.yml file:

```
events:
  hostname: <Your VM's Hostname>
  port: 9092
  topic: events
```

You will need to import the following into app.py:

- json
- KafkaClient from pykafka
- OffsetType from pykafka.common
- Thread from threading

Now you will add a thread to your existing code so that messages are consumed in parallel with the RESTful API.

In your "if __name__ == "__main__", create a thread that calls the process messages function

```
t1 = Thread(target=process_messages)
t1.setDaemon(True)
t1.start()
```

(Note: make sure to leave the existing app.run call).

Create the process_messages function and update as necessary:

```
def process_messages():
    """ Process event messages """
    hostname = "%s:%d" % (app_config["events"]["hostname"],
                          app_config["events"]["port"])
    client = KafkaClient(hosts=hostname)
    topic = client.topics[str.encode(app_config["events"]["topic"])]

    # Create a consume on a consumer group, that only reads new messages
    # (uncommitted messages) when the service re-starts (i.e., it doesn't
    # read all the old messages from the history in the message queue).
    consumer = topic.get_simple_consumer(consumer_group=b'event_group',
                                         reset_offset_on_start=False,
                                         auto_offset_reset=OffsetType.LATEST)

    # This is blocking - it will wait for a new message
    for msg in consumer:
        msg_str = msg.value.decode('utf-8')
        msg = json.loads(msg_str)
        logger.info("Message: %s" % msg)

        payload = msg["payload"]

        if msg["type"] == "event1": # Change this to your event type
            # Store the event1 (i.e., the payload) to the DB
        elif msg["type"] == "event2": # Change this to your event type
            # Store the event2 (i.e., the payload) to the DB

        # Commit the new message as being read
        consumer.commit_offsets()
```

In the for loop, check the message type, and store it to the correct database table. Also be sure to add some logging for troubleshooting.

Remove the previous POST API endpoints on the Data Storage service (Lab 3) both in the openapi.yml file and the app.py file as new events will now be received through messages from Kafka.

Test with Postman against the Receiver Service (Lab 2) to make sure the events are now received and stored to the MySQL database by the Data Storage Service (Lab 3) from messages received from the Kafka topic.

Verify that if you stop and restart the Data Storage Service (Lab 3) application, the consumer offset is retained (i.e., it doesn't re-read all the messages on the topic again). If it does, consult the documentation for how to address this.

If all works, test again with your Apache jMeter load test and make sure that everything still works as before.

Part 3 – Consume Messages at New Audit Service as Event Store

Create a new Lab 7 project called **Audit** in PyCharm (or whatever IDE you are using). Install the same Python modules as the other services (i.e., connexion, swagger-ui-bundle) plus the pykafka module.

Make sure it has applic ation configuration and logging setup.

Create an openapi.yml with two GET API endpoints, one for each event type. Each will have an index as a parameter and return the event at the index relative to the beginning of the message queue.

For example, if you message queue for your topic looks like this:

1. Event1 Type Reading (Index 0)
2. Event2 Type Reading (Index 0)
3. Event1 Type Reading (Index 1)

4. Event1 Type Reading (Index 2)
5. Event1 Type Reading (Index 3)
6. Event2 Type Reading (Index 1)
7. Event2 Type Reading (Index 2)
8. Event1 Type Reading (Index 4)

/event1?index=3

Will return the event1 object for Index 3 (Element 5 in the numbered list above) and return code 200

/event2?index=2

Will return the event2 object for Index 2 (Element 7 in the numbered list above) and return code 200

/event1?index=10

Will return { “message”: “Not Found” } and return code 404

/event2?index=3

Will return { “message”: “Not Found” } and return code 404

There is an example of what the openapi.yml file might look like below (but you need to change it for your events).

```
openapi: 3.0.0
info:
  description: This API provides audit info
  version: "1.0.0"
  title: Audit API
  contact:
    email: mmulder10@bcit.ca

paths:
  /blood_pressure:
    get:
      summary: gets a blood pressure reading from history
      operationId: app.get_blood_pressure_reading
      description: Gets blood pressure readings from the event store
      parameters:
        - name: index
          in: query
          description: Gets the BP at the index in the event store
          schema:
            type: integer
            example: 100
      responses:
        '200':
          description: Successfully returned a blood pressure event
          content:
            application/json:
              schema:
                type: object
                items:
                  $ref: '#/components/schemas/BloodPressureReading'
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
        '404':
          description: Not Found
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string

  /heart_rate:
    get:
      summary: gets a heart rate reading from history
      operationId: app.get_heart_rate_reading
      description: Gets heart rate reading from the event store
      parameters:
        - name: index
          in: query
          description: Gets the HR at the index in the event store
          schema:
            type: integer
            example: 100
      responses:
        '200':
          description: Successfully returned a heart rate event
          content:
            application/json:
              schema:
                type: object
                items:
                  $ref: '#/components/schemas/HeartRateReading'
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
        '404':
          description: Not Found

components:
  schemas:
    BloodPressureReading:
      required:
      - patient_id
```

```yaml
      - device_id
      - blood_pressure
      - timestamp
      properties:
        patient_id:
          type: string
          format: uuid
          example: d290f1ee-6c54-4b01-90e6-d701748f0851
        device_id:
          type: string
          example: A12345
        blood_pressure:
          $ref: '#/components/schemas/BloodPressure'
        timestamp:
          type: string
          format: date-time
          example: 2016-08-29T09:12:33.001Z
      type: object

    HeartRateReading:
      required:
      - patient_id
      - device_id
      - heart_rate
      - timestamp
      properties:
        patient_id:
          type: string
          format: uuid
          example: d290f1ee-6c54-4b01-90e6-d701748f0851
        device_id:
          type: string
          example: A12345
        heart_rate:
          type: integer
          example: 85
        timestamp:
          type: string
          format: date-time
          example: 2016-08-29T09:12:33.001Z
      type: object

    BloodPressure:
      required:
      - systolic
      - diastolic
      properties:
        systolic:
          type: integer
          example: 120
        diastolic:
          type: integer
          example: 80
      type: object
```

Here is an outline of what one of the operation functions in app.py for the above API might look like:

```python
def get_blood_pressure_reading(index):
    """ Get BP Reading in History """
    hostname = "%s:%d" % (app_config["events"]["hostname"],
                          app_config["events"]["port"])
    client = KafkaClient(hosts=hostname)
    topic = client.topics[str.encode(app_config["events"]["topic"])]

    # Here we reset the offset on start so that we retrieve
    # messages at the beginning of the message queue.
    # To prevent the for loop from blocking, we set the timeout to
    # 100ms. There is a risk that this loop never stops if the
    # index is large and messages are constantly being received!
    consumer = topic.get_simple_consumer(reset_offset_on_start=True,
                                         consumer_timeout_ms=1000)

    logger.info("Retrieving BP at index %d" % index)
    try:
        for msg in consumer:
            msg_str = msg.value.decode('utf-8')
            msg = json.loads(msg_str)

            # Find the event at the index you want and
            # return code 200
            # i.e., return event, 200
    except:
        logger.error("No more messages found")

    logger.error("Could not find BP at index %d" % index)
    return { "message": "Not Found"}, 404
```

Simple Consumer documentation for more details on the consumer settings:
https://pykafka.readthedocs.io/en/latest/api/simpleconsumer.html

Grading and Submission

Submit the following to the Lab 6B Dropbox on D2L:

- A zipfile containing the code for your new Audit Log Service.

Demo the following to your instructor before the end of next class to receive your marks:

Receiver Service (Lab 2) Code: • Produces Events Storage Service (Lab 3) Code: • Consumes Events and Stores the Payload to the Database	4 marks
Demo of your Receiver, Storage and Processing Services against jMeter.	4 marks

The Receiver and Storage Services should be communicating via Kafka on your VM. The Storage Service should be writing to the MySQL database on your VM.	
Demo of your Audit Service. Show both a 200 response (with data) and a 404 response for each endpoint.	2 marks
Total	**10 marks**