UCB MIDS W205 Summer 2018 - Kevin Crook's agenda for Synchronous Session #10

Update docker images (before class)

Run these command in your droplet (but **NOT** in a docker container):

```
docker pull confluentinc/cp-zookeeper:latest
docker pull confluentinc/cp-kafka:latest
docker pull midsw205/spark-python:0.0.5
docker pull midsw205/base:0.1.8
```

Update the course-content repo in your docker container in your droplet (before class)

See instructions in previous synchronous sessions.

Activity - setup a web server running a simple web API service which will service web API calls by publishing them to a kafka topic, this time we will format them into simple json objects instead of just publishing text, using curl make web API calls to our web service to test, manually consume the kafka topic to verify our web service is working

Create a new directory for flask with kafka and spark:

```
mkdir ~/w205/flask-with-kafka-and-spark/
```

Move into the new directory:

```
cd ~/w205/flask-with-kafka-and-spark/
```

Copy the yml file from the course content repo. We will use this same yml file for both activity 1 and activity 2. Note that if you are in your droplet, you will need to change the directory to a fully qualified path name. If you are using Windows, you will also need to change it to a fully qualified Windows path:

```
cp ~/w205/course-content/10-Transforming-Streaming-Data/docker-compose.yml .
```

Startup the cluster:

```
docker-compose up -d
```

Create a kafka topic called events (same as we have done several times before):

```
docker-compose exec kafka \
  kafka-topics \
    --create \
    --topic events \
    --partitions 1 \
    --replication-factor 1 \
    --if-not-exists --zookeeper zookeeper:32181
```

Same command on 1 line for convenience:

```
docker-compose exec kafka kafka-topics --create --topic events --partitions 1 --replication-factor 1 --
```

Should see the following output:

```
Created topic "events".
```

Create a file game__api__with__json__events.py witht he following python code. This is very similar to what we did last week. We are using the python kafka module's class KafkaProducer. We are also using the flask module's class Flask. Recall that we previously installed the kafka and flask modules:

```
#!/usr/bin/env python
import json
from kafka import KafkaProducer
from flask import Flask

app = Flask(__name__)
producer = KafkaProducer(bootstrap_servers='kafka:29092')


def log_to_kafka(topic, event):
    producer.send(topic, json.dumps(event).encode())


@app.route("/")
def default_response():
    default_event = {'event_type': 'default'}
    log_to_kafka('events', default_event)
    return "\nThis is the default response!\n"


@app.route("/purchase_a_sword")
def purchase_a_sword():
    purchase_sword_event = {'event_type': 'purchase_sword'}
    log_to_kafka('events', purchase_sword_event)
    return "\nSword Purchased!\n"
```

Run the python flash script in the mids container of our docker cluster. This will run and print output to the command line each time we make a web API call. It will hold the command line until we exit it with a control-C. So you will need another command line prompt:

```
docker-compose exec mids \
  env FLASK_APP=/w205/flask-with-kafka-and-spark/game_api_with_json_events.py \
  flask run --host 0.0.0.0
```

Same command on 1 line for convenience:

```
docker-compose exec mids env FLASK_APP=/w205/flask-with-kafka-and-spark/game_api_with_json_events.py fla
```

Run the curl utility in the mids container of our docker cluster to make API calls. Try each command several times in random order.

```
docker-compose exec mids curl http://localhost:5000/
docker-compose exec mids curl http://localhost:5000/purchase_a_sword
```

Run the kafkacat utility in the mids container of our docker cluster to consume the topic:

```
docker-compose exec mids \
  kafkacat -C -b kafka:29092 -t events -o beginning -e
```

Same command on 1 line for convenience:

```
docker-compose exec mids kafkacat -C -b kafka:29092 -t events -o beginning -e
```

You should see similar to the following:

```
{"event_type": "default"}
{"event_type": "default"}
{"event_type": "default"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
...
```

Leave the cluster running for the next activity. We will also use the same kafka topic.

Activity - enhance the previous activity by adding more key value attributes to the json objects that we publish to kafka, use the pyspark python to spark interface to read the json objects into a data frame and process them

Use control-C to exit the flask web server.

Create new python file game_api_with_extended_json_events.py with the following python code. This enhances the events even more:

```python
#!/usr/bin/env python
import json
from kafka import KafkaProducer
from flask import Flask, request

app = Flask(__name__)
producer = KafkaProducer(bootstrap_servers='kafka:29092')


def log_to_kafka(topic, event):
    event.update(request.headers)
    producer.send(topic, json.dumps(event).encode())


@app.route("/")
def default_response():
    default_event = {'event_type': 'default'}
    log_to_kafka('events', default_event)
    return "\nThis is the default response!\n"


@app.route("/purchase_a_sword")
def purchase_a_sword():
    purchase_sword_event = {'event_type': 'purchase_sword'}
    log_to_kafka('events', purchase_sword_event)
    return "\nSword Purchased!\n"
```

Run the python flash script in the mids container of our docker cluster. This will run and print output to the command line each time we make a web API call. It will hold the command line until we exit it with a control-C. So you will need another command line prompt:

```
docker-compose exec mids \
  env FLASK_APP=/w205/flask-with-kafka-and-spark/game_api_with_extended_json_events.py \
  flask run --host 0.0.0.0
```

Same command on 1 line for convenience:

```
docker-compose exec mids env FLASK_APP=/w205/flask-with-kafka-and-spark/game_api_with_extended_json_even
```

Run the curl utility in the mids container of our docker cluster to make API calls. Try each command several times in random order.

```
docker-compose exec mids curl http://localhost:5000/
docker-compose exec mids curl http://localhost:5000/purchase_a_sword
```

Run the kafkacat utility in the mids container of our docker cluster to consume the topic:

```
docker-compose exec mids \
  kafkacat -C -b kafka:29092 -t events -o beginning -e
```

Same command on 1 line for convenience:

```
docker-compose exec mids kafkacat -C -b kafka:29092 -t events -o beginning -e
```

You should see similar to the following. The first events with the old format that we wrote to the topic are still there for replay:

```
{"event_type": "default"}
{"event_type": "default"}
{"event_type": "default"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
{"event_type": "purchase_sword"}
...
{"Host": "localhost:5000", "event_type": "default", "Accept": "*/*", "User-Agent": "curl/7.47.0"}
{"Host": "localhost:5000", "event_type": "default", "Accept": "*/*", "User-Agent": "curl/7.47.0"}
{"Host": "localhost:5000", "event_type": "default", "Accept": "*/*", "User-Agent": "curl/7.47.0"}
{"Host": "localhost:5000", "event_type": "purchase_sword", "Accept": "*/*", "User-Agent": "curl/7.47.0"
{"Host": "localhost:5000", "event_type": "purchase_sword", "Accept": "*/*", "User-Agent": "curl/7.47.0"
{"Host": "localhost:5000", "event_type": "purchase_sword", "Accept": "*/*", "User-Agent": "curl/7.47.0"
{"Host": "localhost:5000", "event_type": "purchase_sword", "Accept": "*/*", "User-Agent": "curl/7.47.0"
...
```

Run a pyspark shell (may want another command line windows for this):

```
docker-compose exec spark pyspark
```

Last time, we used the method cache() to cache our data frames to prevent warning messages, which are very distracting if you are new to spark. Now that we are getting more familiar with spark, let's leave it off, so we will be getting warnings.

Using pyspark, consume the kafka topic:

```
raw_events = spark \
  .read \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka:29092") \
  .option("subscribe","events") \
  .option("startingOffsets", "earliest") \
  .option("endingOffsets", "latest") \
  .load()
```

Same command on 1 line for convenience:

```
raw_events = spark.read.format("kafka").option("kafka.bootstrap.servers", "kafka:29092").option("subscri
```

Cache our raw events:

```
raw_events.cache()
```

As we have done several times before, the value will be binary which is not easily human readable. We won't be using the other attributes. We will create a new data frame with just the value in string format.

```
events = raw_events.select(raw_events.value.cast('string'))
```

As we have done several times before, we will extract the values into individual json objects:

```
import json
extracted_events = events.rdd.map(lambda x: json.loads(x.value)).toDF()
```

Take a look at the extracted json values:

```
extracted_events.show()
```

Exit pyspark with:

```
exit()
```

Exit flask with:

```
control-C
```

Tear down the cluster with:

```
docker-compose down
```