UCB MIDS W205 Summer 2018 - Kevin Crook's agenda for Synchronous Session #11

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/cdh-minimal:latest
docker pull midsw205/spark-python:0.0.5
docker pull midsw205/base:latest
```

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

See instructions in previous synchronous sessions.

Activity - Continuing with what we did last week, we will add an hadoop container to our cluster, and our python spark code will subscribe to the kafka topic and write the results into parquet format in hdfs. We will also introduce the batch python spark interface called spark-submit to submit our python files instead of using pyspark. We will run a couple of variations. One will transform the events. Another will separate the events.

Create our directory and change to it:

```
mkdir ~/w205/spark-from-files/
cd ~/w205/spark-from-files
```

Copy our docker-compose.yml file into our local directory. Use vi to update the volume mounts if needed. We have had some issues with volume mounts. To be safe the docker support recommended putting double quotes around the mount points.

```
cp ~/w205/course-content/11-Storing-Data-III/docker-compose.yml .
```

Walk through the docker-compose.yml file. We are adding the cloudera hadoop container to what we have done so far. We are also exposing port 5000 so we can connect to our flask web API from both the droplet and from our desktop:

```
---
version: '2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 32181
      ZOOKEEPER_TICK_TIME: 2000
    expose:
      - "2181"
      - "2888"
      - "32181"
      - "3888"
```

```
    extra_hosts:
      - "moby:127.0.0.1"

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:32181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    expose:
      - "9092"
      - "29092"
    extra_hosts:
      - "moby:127.0.0.1"

  cloudera:
    image: midsw205/cdh-minimal:latest
    expose:
      - "8020" # nn
      - "50070" # nn http
      - "8888" # hue
    #ports:
    #- "8888:8888"
    extra_hosts:
      - "moby:127.0.0.1"

  spark:
    image: midsw205/spark-python:0.0.5
    stdin_open: true
    tty: true
    expose:
      - "8888"
    ports:
      - "8888:8888"
    volumes:
      - "~/w205:/w205"
    command: bash
    depends_on:
      - cloudera
    environment:
      HADOOP_NAMENODE: cloudera
    extra_hosts:
      - "moby:127.0.0.1"

  mids:
    image: midsw205/base:latest
    stdin_open: true
    tty: true
    expose:
      - "5000"
```

```
    ports:
      - "5000:5000"
    volumes:
      - "~/w205:/w205"
    extra_hosts:
      - "moby:127.0.0.1"
```

Copy the python files we will be using:

```
cp ~/w205/course-content/11-Storing-Data-III/*.py .
```

Starup the docker cluster:

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

Wait for the cluster to come up. Open a separate linux command line window for each of these. Cloudera hadoop may take a while to come up. You may want to also check the hadoop file system to see how eventual consistency works for the two directories for yarn and hive to both show up. You may also want to check kafka. Sometimes kafka has to reorg and it can take a while to come up. Remember to use control-C to exit processes with the -f option.

```
docker-compose logs -f cloudera
docker-compose exec cloudera hadoop fs -ls /tmp/
docker-compose logs -f kafka
```

Create a topic in kafka (same as we have been doing):

```
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 --
```

You should see:

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

Review the Python file game_api.py:

```
#!/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 flask with our game_api.py python code:

```
docker-compose exec mids \
  env FLASK_APP=/w205/spark-from-files/game_api.py \
  flask run --host 0.0.0.0
```

Same command on 1 line for convenience:

```
docker-compose exec mids env FLASK_APP=/w205/spark-from-files/game_api.py flask run --host 0.0.0.0
```

In another linux command line window, use curl to test our web API server. (Later we can try this from some other options)

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

Read the topic in kafka to see the generated events (same as we have done before):

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

Same command but on 1 line for convenience:

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

We Should see similar to this output:

```
{"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"
...
```

Review the following python spark file extract_events.py. Instead of using pyspark we will be using the spark-submit. (Later we can retry this with pyspark and with Jupyter Notebook):

```
#!/usr/bin/env python
"""Extract events from kafka and write them to hdfs
"""
import json
from pyspark.sql import SparkSession
```

```
def main():
    """main
    """
    spark = SparkSession \
        .builder \
        .appName("ExtractEventsJob") \
        .getOrCreate()

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

    events = raw_events.select(raw_events.value.cast('string'))
    extracted_events = events.rdd.map(lambda x: json.loads(x.value)).toDF()

    extracted_events \
        .write \
        .parquet("/tmp/extracted_events")


if __name__ == "__main__":
    main()
```

Submit our extract_events.py file to spark using spark-submit:

```
docker-compose exec spark \
  spark-submit \
    /w205/spark-from-files/extract_events.py
```

Same command on 1 line for convenience:

```
docker-compose exec spark spark-submit /w205/spark-from-files/extract_events.py
```

If you try this without having any kafka events, you will get the following error messages or similar:

```
Traceback (most recent call last):
  File "/w205/spark-from-files/extract_events.py", line 35, in <module>
    main()
  File "/w205/spark-from-files/extract_events.py", line 27, in main
    extracted_events = events.rdd.map(lambda x: json.loads(x.value)).toDF()
  File "/spark-2.2.0-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/sql/session.py", line 57, in toDF
  File "/spark-2.2.0-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/sql/session.py", line 535, in createDa
  File "/spark-2.2.0-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/sql/session.py", line 375, in _createI
  File "/spark-2.2.0-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/sql/session.py", line 346, in _inferS
  File "/spark-2.2.0-bin-hadoop2.6/python/lib/pyspark.zip/pyspark/rdd.py", line 1364, in first
ValueError: RDD is empty
```

Since our code wrote to the hadoop hdfs file system, let's check out and verify this:

```
docker-compose exec cloudera hadoop fs -ls /tmp/
docker-compose exec cloudera hadoop fs -ls /tmp/extracted_events/
```

Note that the following command that we used earlier:

```
docker-compose exec spark spark-submit filename.py
```

is short for this:

```
docker-compose exec spark \
  spark-submit \
    --master 'local[*]' \
    filename.py
```

We are running a spark "pseudo-distributed" cluster (aka not really a cluster with a "master" and "workers"). If we run a standalone cluster with a master node and worker nodes, we have to be more specific (this is just an example and won't work on our cluster):

```
docker-compose exec spark \
  spark-submit \
    --master spark://23.195.26.187:7077 \
    filename.py
```

If we were running our spark inside of a hadoop cluster, we would need to submit to yarn which is the resource manager for hadoop 2 (this is just an example and won't work on our cluster):

```
docker-compose exec spark \
  spark-submit \
    --master yarn \
    --deploy-mode cluster \
    filename.py
```

If we were running our spark inside of a mesos cluster, we would need to submit to mesos master (this is just an example and won't work on our cluster):

```
docker-compose exec spark \
  spark-submit \
    --master mesos://mesos-master:7077 \
    --deploy-mode cluster \
    filename.py
```

If we were running our spark inside of a kubernetes cluster, we would need to submit to kubernetes master (this is just an example and won't work on our cluster):

```
docker-compose exec spark \
  spark-submit \
    --master k8s://kubernetes-master:443 \
    --deploy-mode cluster \
    filename.py
```

Review the python file transform_events.py below. Note we change the event "Host" to "moe" and the event "Cache-Control" to "no-cache". We also do a show on our events. We also overwrite our parquet files.

```
#!/usr/bin/env python
"""Extract events from kafka, transform, and write to hdfs
"""
import json
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import udf


@udf('string')
def munge_event(event_as_json):
    event = json.loads(event_as_json)
    event['Host'] = "moe" # silly change to show it works
```

```
    event['Cache-Control'] = "no-cache"
    return json.dumps(event)


def main():
    """main
    """
    spark = SparkSession \
        .builder \
        .appName("ExtractEventsJob") \
        .getOrCreate()

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

    munged_events = raw_events \
        .select(raw_events.value.cast('string').alias('raw'),
                raw_events.timestamp.cast('string')) \
        .withColumn('munged', munge_event('raw'))
    munged_events.show()

    extracted_events = munged_events \
        .rdd \
        .map(lambda r: Row(timestamp=r.timestamp, **json.loads(r.munged))) \
        .toDF()
    extracted_events.show()

    extracted_events \
        .write \
        .mode("overwrite") \
        .parquet("/tmp/extracted_events")


if __name__ == "__main__":
    main()
```

Run transform_events.py and review the results.

```
docker-compose exec spark spark-submit /w205/spark-from-files/transform_events.py
```

Review the python file separate_events.py below. Note we now separate sword purchase events from default events.

```
#!/usr/bin/env python
"""Extract events from kafka and write them to hdfs
"""
import json
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import udf
```

```
@udf('string')
def munge_event(event_as_json):
    event = json.loads(event_as_json)
    event['Host'] = "moe" # silly change to show it works
    event['Cache-Control'] = "no-cache"
    return json.dumps(event)


def main():
    """main
    """
    spark = SparkSession \
        .builder \
        .appName("ExtractEventsJob") \
        .getOrCreate()

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

    munged_events = raw_events \
        .select(raw_events.value.cast('string').alias('raw'),
                raw_events.timestamp.cast('string')) \
        .withColumn('munged', munge_event('raw'))

    extracted_events = munged_events \
        .rdd \
        .map(lambda r: Row(timestamp=r.timestamp, **json.loads(r.munged))) \
        .toDF()

    sword_purchases = extracted_events \
        .filter(extracted_events.event_type == 'purchase_sword')
    sword_purchases.show()
    # sword_purchases \
        # .write \
        # .mode("overwrite") \
        # .parquet("/tmp/sword_purchases")

    default_hits = extracted_events \
        .filter(extracted_events.event_type == 'default')
    default_hits.show()
    # default_hits \
        # .write \
        # .mode("overwrite") \
        # .parquet("/tmp/default_hits")
```

```
if __name__ == "__main__":
    main()
```

Run separate_events.py and review the results.

```
docker-compose exec spark spark-submit /w205/spark-from-files/separate_events.py
```

Tear down the cluster:

```
docker-compose down
```