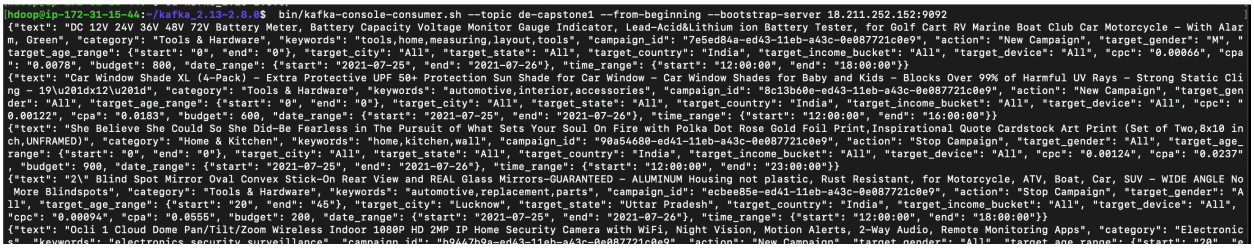
Code Logic: Online Advertising

First, I inspected the data using kafka console consumer by running this command -

bin/kafka-console-consumer.sh --topic de-capstone1 --from-beginning --bootstrap-server 18.211.252.152:9092



ad_manager.py

We are getting the data from kafka sink via initialising pykafka consumer then dumping the data with necessary modification to our mysql server using mysql python connector.

Our core functionality of ad_manager.py lies in process_row function where we compute cpm, current_slot_budget and status value as per provided formulas.

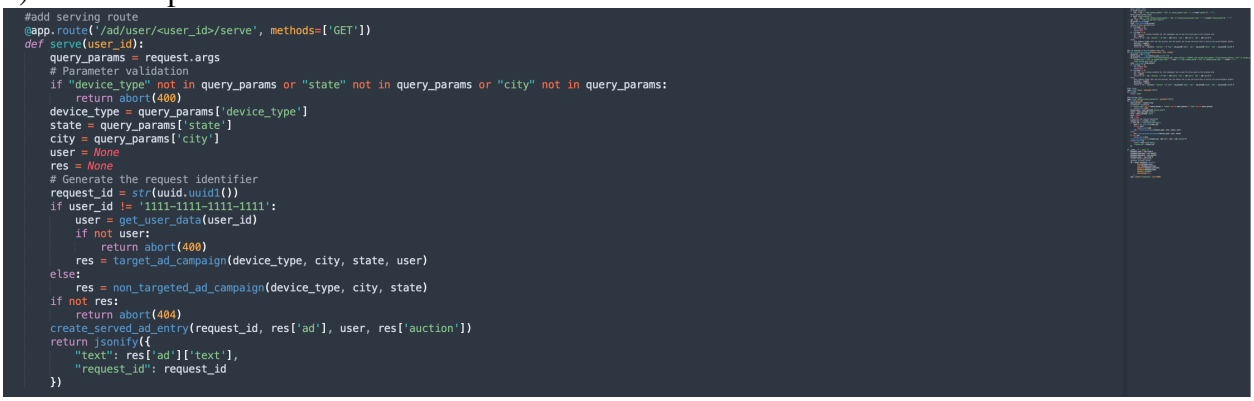
```
def calc_date_time_diff_in_minutes(self,msg):
    start_date = msg.get('date_range').get('start').split('-')
    end_date = msg.get('date_range').get('end').split('-')
    start_time = msg.get('time_range').get('start').split(':')
    end_time = msg.get('time_range').get('end').split(':')
    datetime_start = datetime(int(start_date[0]),int(start_date[1]),int(start_date[2]),int(start_time[0]),int(start_time[1]),int(start_time[2]))
    datetime_end = datetime(int(end_date[0]),int(end_date[1]),int(end_date[2]),int(end_time[0]),int(end_time[1]),int(end_time[2]))
    datetime_diff = datetime_end - datetime_start
    datetime_diff_minutes = datetime_diff.total_seconds() / 60
    return datetime_diff_minutes

# Process single row
def process_row(self, msg):
    # Get the db cursor
    db_cursor = self.db.cursor()

    #calculate CPM by provided formula
    cpm = (0.0075 * float(msg.get('cpc'))) + (0.0005 * float(msg.get('cpa')))

    #calculate current slot budget as per 10 minute interval
    datetime_diff_minutes = self.calc_date_time_diff_in_minutes(msg)
    slot_time_size = 10
    number_of_slots = datetime_diff_minutes/slot_time_size
    current_slot_budget = msg.get('budget')/number_of_slots

    #status of campaign
    status = 'INACTIVE 'if msg.get('action') == 'Stop Campaign' else 'ACTIVE'

    # DB query for supporting UPSERT operation
    sql = "REPLACE INTO ads(text,category,keywords,campaign_id,status,target_gender,target_age_start,target_age_end,target_city,target_state \
        ,target_country,target_income_bucket,target_device,cpc,cpa,cpm,budget,current_slot_budget,date_range_start,date_range_end,time_range_start,time_range_end) \
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,%s, %s, %s, %s, %s, %s,%s, %s, %s)"
    #values from kafka msg
    val = (msg.get('text'),msg.get('category'), msg.get('keywords'), msg.get('campaign_id'),status, msg.get('target_gender'), msg.get('target_age_range').get('start'), m
        , msg.get('target_city'), msg.get('target_state'), msg.get('target_country'), msg.get('target_income_bucket'), msg.get('target_device'), msg.get('cpc'), msg.get(
        , current_slot_budget, msg.get('date_range').get('start'), msg.get('date_range').get('end'), msg.get('time_range').get('start'), msg.get('time_range').get('end')
    db_cursor.execute(sql, val)
    # Commit the operation, so that it reflects globally
    self.db.commit()
    #output to console
    print(msg.get('campaign_id') + ' | ' + msg.get('action') + ' | ' + status)
```

we run our ad_manager.py script via below command -

```
^CKeyboardInterrupt, exiting...
(base) sahiljain@Sahils-MacBook-Pro Capstone % python ad_manager.py 18.211.252.152:9092 de-capstone1 localhost root root1234 advertisement
8e91093a-ed76-11eb-a43c-0e087721c0e9 | New Campaign | ACTIVE
9dfa233e-ed76-11eb-a43c-0e087721c0e9 | New Campaign | ACTIVE
```

Then, we write down our ad_server.py script to serve ads from database based on Second-Price Auction strategy

1). Main endpoint which will serve ads



```
#add serving route
@app.route('/ad/user/<user_id>/serve', methods=['GET'])
def serve(user_id):
    query_params = request.args
    # Parameter validation
    if "device_type" not in query_params or "state" not in query_params or "city" not in query_params:
        return abort(400)
    device_type = query_params['device_type']
    state = query_params['state']
    city = query_params['city']
    user = None
    res = None
    # Generate the request identifier
    request_id = str(uuid.uuid1())
    if user_id != '1111-1111-1111-1111':
        user = get_user_data(user_id)
        if not user:
            return abort(400)
        res = target_ad_campaign(device_type, city, state, user)
    else:
        res = non_targeted_ad_campaign(device_type, city, state)
    if not res:
        return abort(404)
    create_served_ad_entry(request_id, res['ad'], user, res['auction'])
    return jsonify({
        "text": res['ad']['text'],
        "request_id": request_id
    })
```

If the user_id is 1111-1111-1111-1111 then we will run non targeted campaign whereas we can get user from database then we will show them ads according to their age, income bucket, and gender. So I have create two functions target_ad_campaign and not_targeted_ad_campaign where we use Second-Price Auction strategy. Screenshots are added below for respective functions.

For user_ids and user in database using below

```
#get ad campaign to be run
def target_ad_campaign(device_type, city, state, user):
    db_cursor = db.cursor()
    device_type = ' '.join(device_type.split('-'))
    #get eligible ad
    sql = "select * from advertisement.ads  where status = 'ACTIVE' and current_slot_budget > 0 and (target_device = 'All' or target_device like '" + device_type + "') and
        (target_city = 'All' or target_city like '" + city + "') and (target_state = 'All' or target_state like '" + state + "')"
    #user age check
    if user['age'] :
        sql  = sql + " and (target_age_start <= " + str(user['age']) + " and " + str(user['age']) +" <= target_age_end)"
    #user gender check
    if user['gender'] :
        sql  = sql + " and (target_gender = 'All' or target_gender like '" + str(user['gender'])  + "')"
    #user income bucket check
    if user['income_bucket'] :
        sql  = sql + " and (target_income_bucket = 'All' or target_income_bucket like '" + str(user['income_bucket'])  + "')"
    sql = sql + " order by cpm desc limit 2"
    db_cursor.execute(sql)
    rows = get_results(db_cursor)
    #handle 0 rows returned here?
    if len(rows) == 0:
        #no eligble ads;
        return None
    if len(rows) == 1:
        #If there is a single eligible Ad, the campaigner has to pay the price same as the original bid.
        ad = rows[0]
        return { 'ad' : ad, 'auction' : { "cpm" : ad['cpm'], 'cpc' : ad['cpc'], 'cpa' : ad['cpa']} }
    else:
        #The highest bidder will win the auction, but the winner has to pay the price that is bid by the second-highest bidder.
        ad_first = rows[0]
        ad_second = rows[1]
        return { 'ad' : ad_first, 'auction' : { "cpm" : ad_second['cpm'], 'cpc' : ad_second['cpc'], 'cpa' : ad_second['cpa']} }
```

For user_id as 1111-1111-1111-1111

```
#get ad campaign to be run without user info
def non_targeted_ad_campaign(device_type, city, state):
    db_cursor = db.cursor()
    device_type = ' '.join(device_type.split('-'))
    db_cursor.execute("select * from advertisement.ads  where status = 'ACTIVE' and current_slot_budget > 0 and (target_device = 'All' or target_device like '" + device_
        (target_city = 'All' or target_city like '" + city + "') and (target_state = 'All' or target_state like '" + state + "') \
        order by cpm desc limit 2")
    rows = get_results(db_cursor)
    if len(rows) == 0:
        #no eligble ads;
        return None
    if len(rows) == 1:
        #If there is a single eligible Ad, the campaigner has to pay the price same as the original bid.
        ad = rows[0]
        return { 'ad' : ad, 'auction' : { "cpm" : ad['cpm'], 'cpc' : ad['cpc'], 'cpa' : ad['cpa']} }
    else:
        #The highest bidder will win the auction, but the winner has to pay the price that is bid by the second-highest bidder.
        ad_first = rows[0]
        ad_second = rows[1]
        return { 'ad' : ad_first, 'auction' : { "cpm" : ad_second['cpm'], 'cpc' : ad_second['cpc'], 'cpa' : ad_second['cpa']} }
```

Then we create entry of served ad in served_ads table for record and then so to recalculate budget further after user feedback.

```
    return results

#save ad served to user in database;
def create_served_ad_entry(request_id, ad, user, auction):
    db_cursor = db.cursor()
    user_id = user['id'] if user else None;
    campaign_start_time = ad['date_range_start'] + ' ' + ad['time_range_start']
    campaign_end_time = ad['date_range_end'] + ' ' + ad['time_range_end']
    target_location =  ad['target_city'] + ', '  + ad['target_state'] + ', '  + ad['target_country']
    target_age_range = str(ad['target_age_start']) + '-' + str( ad['target_age_end'])
    ts = time()
    timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    sql = 'insert into served_ads(request_id,campaign_id,user_id,auction_cpm,auction_cpc,auction_cpa, \
        target_age_range,target_location,target_gender,target_income_bucket,target_device_type,campaign_start_time, \
        campaign_end_time,timestamp) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'
    val = (request_id, ad['campaign_id'], user_id, auction['cpm'], auction['cpc'], auction['cpa'], target_age_range , target_location, ad['target_gender'], \
        ad['target_income_bucket'], ad['target_device'], campaign_start_time, campaign_end_time, timestamp)
    db_cursor.execute(sql,val)
    db.commit()

#get ad campaign to be run
```

Then, Simulator script sends the feedback request to localhost:5001 which handles feedback from user and calculate user_action and expenditure as per requirements.
We recalculate new_slot_budget and budget at runtime and deduct the used expenditure.
After handling the request, the feedback is sent to kafka queue using send_feedback_to_kafka function, kafka connection can be found in Line 19, in feedback_handler.py.

The code handling the feedback request can be found at Line 124.

Kafka queue messages are then handled by pyspark structured streaming, The code can be found in user_feedback_writer.py

Basically in user_feeback_writer what we do is create a source which is kafka queue in this case and read the json messages by defining the jsonSchema, then write down those messages to hdfs path.

1). Read from kafka

```
feedbackRaw = spark  \
        .readStream  \
        .format("kafka")  \
```

```
        .option("kafka.bootstrap.servers","localhost:9092")  \
        .option("subscribe","user-feedback")  \
        .load()
```

2). Define JSON schema

```
jsonSchema = StructType()  \
        .add('campaign_id', StringType())  \
        .add('user_id', StringType())  \
        .add('request_id', StringType())  \
        .add('click', IntegerType())  \
        .add('view', IntegerType()) \
        .add('acquisition', IntegerType()) \
        .add('auction_cpm', DoubleType())  \
        .add('auction_cpc', DoubleType()) \
        .add('auction_cpa', DoubleType()) \
        .add('target_age_range', StringType())  \
        .add('target_location',StringType()) \
        .add('target_gender',StringType()) \
        .add('target_income_bucket',StringType()) \
        .add('target_device_type', StringType()) \
        .add('campaign_start_time', StringType()) \
        .add('campaign_end_time', StringType()) \
        .add('user_action', StringType()) \
        .add('expenditure', DoubleType()) \
        .add('timestamp', TimestampType())
```

3). Write to HDFS

```
feedbackStream.writeStream \
  .format("csv") \
  .outputMode("append") \
  .option("truncate", "false") \
  .option("path", "/tmp/feedback") \
  .option("checkpointLocation", "/tmp/feedback") \
  .trigger(processingTime="1 minute") \
  .start() \
```

```
  .awaitTermination()
```

We use cron job to recalculate current slot budget every 10 minutes, we use the following command to do so -

The code in slot_budget_manager uses the same logic as in ad_manager.py to calculate and update current slot budget

*/10 * * * * python3 /home/hdoop/capstone/slot_budget_updater.py localhost root root1234 advertisement >> slot_budget_manager_logs.txt

After an hour, we stop the pipeline and get the required ad data using sqoop command and add data to hive database-

```
sqoop import \
--connect jdbc:mysql://localhost/advertisement \
--username hdoop \
--password root1234 \
--split-by id \
--table ads --m 1 \
--columns campaign_id, category, budget, cpm, cpc, cpa, target_device  \
--hive-import \
--hive-overwrite \
--hive-database advertisement \
--target-dir /user/hive/warehouse/advertisement/
```

Then we create hive external table over hdfs for user_feedback.

```
create external table if not exists user_feedback (campaign_id string, user_id string, request_id string, click integer, view integer, acquisition integer, auction_cpm double, auction_cpc double, auction_cpa double, target_age_range string, target_location string, target_gender string, target_income_bucket string, target_device_type string, campaign_start_time string, campaign_end_time string, user_action string, expenditure double, timestamp timestamp)
row format delimited
fields terminated by ‘,’
stored as textfile
location '/tmp/feedback'
```

Then we create our report via hive sql queries which can be found in report_queries.pdf and output can be found in folder named report_csvs.