New Data Lake/Streaming/Demonst...

Citi Bike Live Map Demonstration

This tutorial was built for BDCS-CE version 17.4.1 and OEHCS 0.10 as part of the New Data Lake User Journey: here (https://github.com/oracle/learning-library/tree/master/workshops/journey2-new-data-lake). Questions and feedback about the tutorial: david.bayard@oracle.com (mailto:david.bayard@oracle.com)

```
NOTE: Please ensure that you have run the "Working with Spark Interpreter" and "Working with OEHCS and Spark Streaming" Tutorials first.
```

Contents

As a reminder, the documentation for BDCS-CE can be found here (https://docs.oracle.com/cloud/latest/big-data-compute-cloud/index.html)

OEHCS Setup

This demonstration will use Oracle Event Hub Cloud Service (OEHCS) and Spark Streaming. Be sure that you have completed the OEHCS tutorial "Working with OEHCS and Spark Streaming" before running this demonstration. That tutorial will help you setup connectivity to OEHCS and create a Kafka topic. You will need to enter the OEHCS Connection Descriptor and OEHCS Topic in the paragraph below and run it.

Parameters (be sure to run this)

```
%spark
z.angularBind("BIND_ObjectStorage_Container", "journeyC")
z.angularBind("BIND_OEHCS_ConnectionDescriptor", z.input("OEHCS_ConnectionDescriptor","141.144.144.128:6667"))
z.angularBind("BIND_OEHCS_Topic", z.input("OEHCS_Topic","gse00010212-TutorialOEHCS"))


scala.tools.nsc.io.File("/var/lib/zeppelin/bikes_part3.sh").writeAll(
  "export ObjectStorage_Container=\""+z.angular("BIND_ObjectStorage_Container")+"\"\n" +
  "export OEHCS_ConnectionDescriptor=\""+z.angular("BIND_OEHCS_ConnectionDescriptor")+"\"\n" +
  "export OEHCS_Topic=\""+z.angular("BIND_OEHCS_Topic")+"\"\n"
)
println("done")
```

OEHCS_ConnectionDescriptor

140.86.32.89:6667

OEHCS_Topic

gse00002281-TutorialOEHCS

done

Preparing Bike Data for Streaming

We will take our bike trip data and wrangle it into a format that will be easier to use with streaming. Specifically, our current data provides 1 row of data which has both the start and end times of a bike trip. We will wrangle the data into a new file such that the start of the trip is its own row and the end of the trip is also its own row. And we will sort our new file by the appropriate event time for that row (start or end). This will make it easier for our Kafka producer program to stream the data for us. We will use Spark SQL to make the wrangling easy. Finally, we'll write our new data to a container in the Object Store.

Spark code to wrangle raw data into a streaming input dataset

```
%spark


//a previous tutorial placed the csv file into your Object Store citibike container
//notice the use of the swift://CONTAINER.defaut/ syntax

val df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("swift://"+z.angular("BIND_ObjectStorage_Container")+".default/citibike/raw/201612-citibike-tri

//cache the data frame for performance
df.cache()


println("Here is the schema detected from the CSV")
df.printSchema()
println("..")

println("# of rows: %s".format(
  df.count()
))
println("..")

df.createOrReplaceTempView("bike_trips_temp")


println("Wrangling the existing data into a new dataframe")
// create a new DataFrame that creates separate rows for start and end events
val df = sqlContext.sql(s"""select b.`Start Time` EventTime, "Pickup" EventType,
    case when b.gender=1 then 'Male' when b.gender=2 then 'Female' else 'unknown' end GenderStr, b.* from bike_trips_temp b
union all
select b.`Stop Time` EventTime, "Dropoff" EventType,
    case when b.gender=1 then 'Male' when b.gender=2 then 'Female' else 'unknown' end GenderStr, b.* from bike_trips_temp b
""")

println("Writing new data to Object Store.  This may take 5 minutes.. Please be patient.  If bored, you can explore the running status via the Spark UI.")
//write the new data frame out as a csv file
df.repartition(1).sortWithinPartitions("EventTime").write.format("com.databricks.spark.csv").mode("overwrite").option("header", "true").save("swift://"+z.angular("BIND_ObjectStorage_Con
    .default/citibike/scratch/bike_streaming_input")

println("done")
```

```
df: org.apache.spark.sql.DataFrame = [Trip Duration: string, Start Time: string ... 13 more fields]
res18: df.type = [Trip Duration: string, Start Time: string ... 13 more fields]
Here is the schema detected from the CSV
root
 |-- Trip Duration: string (nullable = true)
 |-- Start Time: string (nullable = true)
 |-- Stop Time: string (nullable = true)
 |-- Start Station ID: string (nullable = true)
 |-- Start Station Name: string (nullable = true)
 |-- Start Station Latitude: string (nullable = true)
 |-- Start Station Longitude: string (nullable = true)
 |-- End Station ID: string (nullable = true)
 |-- End Station Name: string (nullable = true)
 |-- End Station Latitude: string (nullable = true)
 |-- End Station Longitude: string (nullable = true)
 |-- Bike ID: string (nullable = true)
 |-- User Type: string (nullable = true)
 |-- Birth Year: string (nullable = true)
 |-- Gender: string (nullable = true)
..
# of rows: 812192
..
Wrangling the existing data into a new dataframe
df: org.apache.spark.sql.DataFrame = [EventTime: string, EventType: string ... 16 more fields]
Writing new data to Object Store.  This may take 5 minutes.. Please be patient.  If bored, you can explore the running status via the Spark UI.
done
```

Shell script to get a local copy of our new streaming input datafile

```
%sh
# This script copies the new dataset we created to the local file system, where our producer script is expecting it to reside.

. bikes_part3.sh
echo Object Store Container = $ObjectStorage_Container


hadoop fs -ls swift://$ObjectStorage_Container.default/citibike/scratch/bike_streaming_input

cd citibike
rm bike_streaming_input.csv

hadoop fs -get swift://$ObjectStorage_Container.default/citibike/scratch/bike_streaming_input/part-00000* bike_streaming_input.csv

ls -l bike_streaming_input.csv
head bike_streaming_input.csv

echo "done"
```

```
Object Store Container = journeyC
Found 2 items
drw-rw-rw-   -          0 2017-11-16 20:44 swift://journeyC.default/citibike/scratch/bike_streaming_input/_SUCCESS
-rw-rw-rw-   1  325437807 2017-11-16 20:44 swift://journeyC.default/citibike/scratch/bike_streaming_input/part-00000-b3c94cea-b98e-492c-bec2-e2c697e22823.csv
rm: cannot remove `bike_streaming_input.csv': No such file or directory
```

```
-rw-r--r-- 1 zeppelin zeppelin 325437807 Nov 16 20:45 bike_streaming_input.csv
EventTime,EventType,GenderStr,Trip Duration,Start Time,Stop Time,Start Station ID,Start Station Name,Start Station Latitude,Start Station Longitude,End Station ID,End Station Name,End St
ation Latitude,End Station Longitude,Bike ID,User Type,Birth Year,Gender
2016-12-01 00:00:04,Pickup,Male,528,2016-12-01 00:00:04,2016-12-01 00:08:52,499,Broadway & W 60 St,40.76915505,-73.98191841,228,E 48 St & 3 Ave,40.7546011026,-73.971878855,26931,Subscrib
er,1964,1
2016-12-01 00:00:28,Pickup,Male,218,2016-12-01 00:00:28,2016-12-01 00:04:06,3418,Plaza St West & Flatbush Ave,40.6750207,-73.97111473,3358,Garfield Pl & 8 Ave,40.6711978,-73.97484126,271
22,Subscriber,1955,1
2016-12-01 00:00:39,Pickup,Male,399,2016-12-01 00:00:39,2016-12-01 00:07:19,297,E 15 St & 3 Ave,40.734232,-73.986923,345,W 13 St & 6 Ave,40.73649403,-73.99704374,19352,Subscriber,1985,1
2016-12-01 00:00:44,Pickup,Male,254,2016-12-01 00:00:44,2016-12-01 00:04:59,405,Washington St & Gansevoort St,40.739323,-74.008119,358,Christopher St & Greenwich St,40.73291553,-74.00711
384,20015,Subscriber,1982,1
2016-12-01 00:00:54,Pickup,Male,1805,2016-12-01 00:00:54,2016-12-01 00:31:00,279,Peck Slip & Front St,40.707873,-74.00167,279,Peck Slip & Front St,40.707873,-74.00167,23148,Subscriber,19
89,1
2016-12-01 00:01:13,Pickup,Male,483,2016-12-01 00:01:13,2016-12-01 00:09:17,245,Myrtle Ave & St Edwards St,40.69327018,-73.97703874,372,Franklin Ave & Myrtle Ave,40.694528,-73.958089,161
40,Subscriber,1986,1
2016-12-01 00:01:37,Pickup,Male,1114,2016-12-01 00:01:37,2016-12-01 00:20:12,470,W 20 St & 8 Ave,40.74345335,-74.00004031,453,W 22 St & 8 Ave,40.74475148,-73.99915362,19997,Subscriber,19
64,1
2016-12-01 00:01:50,Pickup,unknown,2680,2016-12-01 00:01:50,2016-12-01 00:46:30,3312,1 Ave & E 94 St,40.7817212,-73.94594,3325,E 95 St & 3 Ave,40.7849032,-73.950503,26105,Subscriber,,0
```

Writing a Producer to stream data to OEHCS

For this demonstration, we will replay historical bike pickup and dropoff data back in "real-time". In other words, if we start the clock at 8am and our first bike is picked up at 8:00:14, our producer program will send the pickup event 14 seconds after it is ready to begin. If the next bike event is at 8:00:25, the producer will wait 11 more seconds before sending that event. We also have a "time acceleration" parameter we can use to speed up how fast our producer replays the historical data.

Our producer program is written in python. It will stream to OEHCS via the "kafka-python" library, as described here (https://github.com/dpkp/kafka-python)

Shell command to save our Python program to a script

```
%sh
. bikes_part3.sh

echo "
#!/usr/bin/env python

# standard libraries
import os
import sys
import csv
import json
from time import sleep
import datetime as dt

# Quality of Life Utils
import dateutil.parser

# kafka-python libraries
from kafka import KafkaProducer
from kafka.errors import KafkaError

if len(sys.argv) < 6:
    print 'usage: tutorial_kafka.py [inputfile:/path/to/file.csv] [acceleration-factor:integer] [recordcount:int-0 is infinite] [starttime:YYYY-MM-DD hh:mm:ss]'
    sys.exit(1)
```

```
-rw-r--r-- 1 zeppelin zeppelin 2059 Nov 16 20:45 citibike_kafka.py
done
```

Running the Live Map demonstration

Assuming you have run the above paragraphs, we are now ready to run our Live Map demonstration. This demonstration will show a live map showing the latest pickups and dropoffs. Pickups will be

shown with green markers with a green line indicating the direction where the bike will eventually be dropped off (since we are replaying historical data, we conviently know the final dropoff location!). Dropoff are shown in red with the line indicating where the bike came from. Longer lines indicate longer trips.

To do run the demo,

- **Run the Spark Streaming paragraph below**. This will start a Spark Streaming session with a 5 second window. As new data arrives, it will update the map (below). This session will run for a few minutes before stopping itself.
- Be sure to **also run the Producer paragraph below** otherwise there will be no data for the Spark Streaming session to see.
- With both the Spark Streaming and Producer running, **watch the output of those paragraphs as well as the map paragraph** to see changes.

The code editors for the next three paragraphs are hidden by default to make it simpler to run, but definitely show the code editors to see how the code is written.

Spark Streaming Live Map code

```
defined class BikeEvents
Creating new Streaming Context
topic:gse00002281-TutorialOEHCS
brokers:140.86.32.89:6667
Creating Kafka DStream
Setting up operations on DStream
Starting Streaming Context
Will now sleep for a few minutes, before stopping the StreamingContext
count = 0
count = 12
count = 27
count = 33
count = 33
count = 26
count = 24
count = 20
count = 30
```

Producer for the Live Map

```
The producer will loop through the datafile until it finds the selected StartDate.  This ca
n take awhile depending on the date of the month you select.
As an example, starting on the 5th can take 90 seconds before the first data is sent.  Star
ting on the 10th can take 3 minutes. Starting on the 20th can take 6 minutes.  Starting on
the 30th can take 9 minutes.
..
..
Launching Producer for 2016-12-01 07:00:00 with a time acceleration factor of 5
0/400
1/400
2/400
3/400
4/400
5/400
6/400
7/400
8/400
9/400
10/400
11/400
12/400
13/400
14/400
15/400
16/400
17/400
18/400
19/400
20/400
21/400
22/400
23/400
24/400
25/400
26/400
27/400
28/400
29/400
30/400
31/400
32/400
```

```
33/400
34/400
35/400
36/400
37/400
38/400
39/400
40/400
41/400
42/400
43/400
44/400
45/400
46/400
47/400
48/400
49/400
50/400
51/400
52/400
53/400
54/400
55/400
56/400
57/400
58/400
59/400
60/400
61/400
62/400
63/400
64/400
65/400
66/400
67/400
68/400
69/400
70/400
71/400
72/400
73/400
74/400
75/400
76/400
77/400
78/400
79/400
80/400
81/400
82/400
83/400
84/400
85/400
86/400
87/400
```

```
88/400
89/400
90/400
91/400
92/400
93/400
94/400
95/400
96/400
97/400
98/400
99/400
100/400
101/400
102/400
103/400
104/400
105/400
106/400
107/400
108/400
109/400
110/400
111/400
112/400
113/400
114/400
115/400
116/400
117/400
118/400
119/400
120/400
121/400
122/400
123/400
124/400
125/400
126/400
127/400
128/400
129/400
130/400
131/400
132/400
133/400
134/400
135/400
136/400
137/400
138/400
139/400
140/400
141/400
142/400
```

143/400
144/400
145/400
146/400
147/400
148/400
149/400
150/400
151/400
152/400
153/400
154/400
155/400
156/400
157/400
158/400
159/400
160/400
161/400
162/400
163/400
164/400
165/400
166/400
167/400
168/400
169/400
170/400
171/400
172/400
173/400
174/400
175/400
176/400
177/400
178/400
179/400
180/400
181/400
182/400
183/400
184/400
185/400
186/400
187/400
188/400
189/400
190/400
191/400
192/400
193/400
194/400
195/400
196/400
197/400

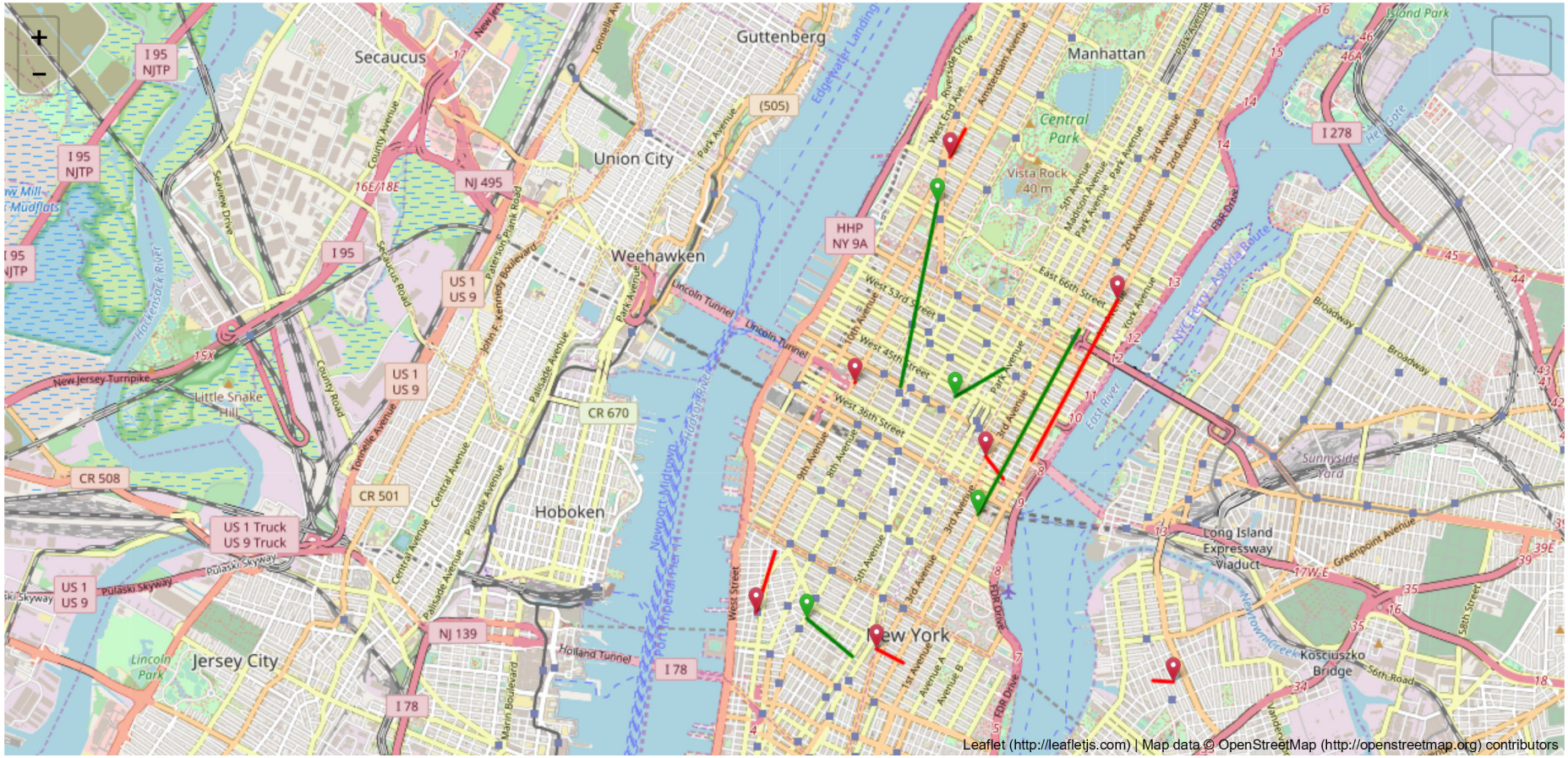
```
198/400
199/400
200/400
201/400
202/400
203/400
204/400
205/400
206/400
207/400
208/400
209/400
210/400
211/400
212/400
213/400
214/400
215/400
216/400
217/400
218/400
219/400
220/400
221/400
222/400
223/400
224/400
225/400
226/400
227/400
228/400
229/400
230/400
231/400
232/400
233/400
234/400
235/400
236/400
237/400
238/400
239/400
240/400
241/400
242/400
243/400
244/400
245/400
246/400
247/400
248/400
249/400
250/400
251/400
252/400
```

253/400
254/400
255/400
256/400
257/400
258/400
259/400
260/400
261/400
262/400
263/400
264/400
265/400
266/400
267/400
268/400
269/400
270/400
271/400
272/400
273/400
274/400
275/400
276/400
277/400
278/400
279/400
280/400
281/400
282/400
283/400
284/400
285/400
286/400
287/400
288/400
289/400
290/400
291/400
292/400
293/400
294/400
295/400
296/400
297/400
298/400
299/400
300/400
301/400
302/400
303/400
304/400
305/400
306/400
307/400

```
308/400
309/400
310/400
311/400
312/400
313/400
314/400
315/400
316/400
317/400
318/400
319/400
320/400
321/400
322/400
323/400
324/400
325/400
326/400
327/400
328/400
329/400
330/400
331/400
332/400
333/400
334/400
335/400
336/400
337/400
338/400
339/400
340/400
341/400
342/400
343/400
344/400
345/400
346/400
347/400
348/400
349/400
350/400
351/400
352/400
353/400
354/400
355/400
356/400
357/400
358/400
359/400
360/400
361/400
362/400
```

```
363/400
364/400
365/400
366/400
367/400
368/400
369/400
370/400
371/400
372/400
373/400
374/400
375/400
376/400
377/400
378/400
379/400
```

HTML to display the live map



Next Steps

This concludes our NYC Citi Bikes demonstration (for now).

You've seen how to:

- Load Citi Bike data to the Object Store
- Define a Spark SQL table on the data
- Run various SQL queries and display the output
- Show results on a Map
- Stream bike data and create a live map

Stay tuned for additional parts of this demonstration coming soon.

Change Log

November 16, 2017 - Fixed some minor deprecated Spark calls. Confirmed it works with 17.4.1
September 12, 2017 - Confirmed it works with BDCSCE 17.3.5. Fixed an issue with incorrectly generating the file used for streaming input for the producer.
August 23, 2017 - Changed to use scratch directory
August 13, 2017 - Confirmed it works with BDCSCE 17.3.3-20
August 11, 2017 - Journey v2. Confirmed it works with Spark 2.1
July 28, 2017 - Validated with BDCSCE 17.3.1 and OEHCS 0.10.2

```
%md
```