Exercise 10

Get started with Cassandra and import data

Prior Knowledge
Unix Command Line Shell
Spark Python
Simple SQL syntax

Learning Objectives
Understand Cassandra's CQL shell
Integrate Python, Cassandra and Spark
Load data from CSV into Cassandra using Spark Python

Software Requirements
(see separate document for installation of these)

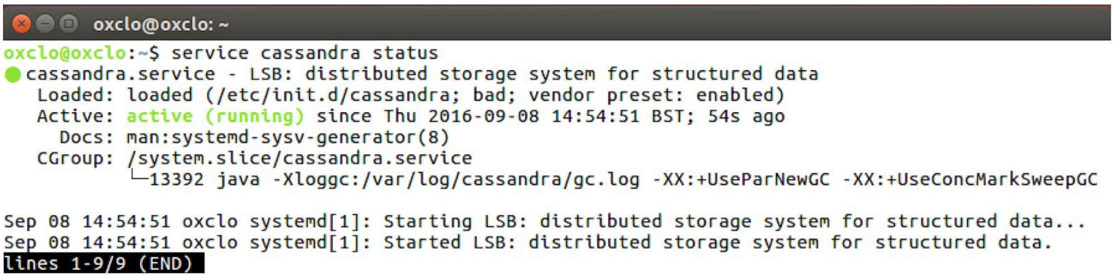
- Apache Spark 3.0.0
- Python 3.8
- Apache Cassandra 3.x
- Nano text editor or other text editor

Part A

1. Make sure Cassandra is running
    a. In a Terminal window (Crtl-Alt-T) type:

    ```
    service cassandra status
    ```

    b. You should see

    Please note, that if there is a problem then it may say
    `active(exited)`

    c. Type q to get back to the command line

e. If it is not working, try

```
sudo service cassandra stop
sudo service cassandra start
```

and then check the status again.

2. Now you can ask Cassandra about its own situation:

```
nodetool status
```

You should see something like:

```
oxclo@oxclo:~$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load      Tokens      Owns (effective)  Host ID                               Rack
UN  127.0.0.1  102.8 KB  256         100.0%            53392ab9-9d1a-4ff8-ac0e-62cb1245d49b  rack1
```

3. You can also try:

```
nodetool info
```

You should see something li

```
oxclo@oxclo:~$ nodetool info
ID                     : 53392ab9-9d1a-4ff8-ac0e-62cb1245d49b
Gossip active          : true
Thrift active          : false
Native Transport active: true
Load                   : 102.8 KB
Generation No          : 1473342909
Uptime (seconds)       : 203
Heap Memory (MB)       : 167.90 / 1620.00
Off Heap Memory (MB)   : 0.00
Data Center            : datacenter1
Rack                   : rack1
Exceptions             : 0
Key Cache              : entries 10, size 816 bytes, capacity 81 MB, 44 hits, 63 reque
sts, 0.698 recent hit rate, 14400 save period in seconds
Row Cache              : entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0 requests
, NaN recent hit rate, 0 save period in seconds
Counter Cache          : entries 0, size 0 bytes, capacity 40 MB, 0 hits, 0 requests,
NaN recent hit rate, 7200 save period in seconds
Token                  : (invoke with -T/--tokens to see all 256 tokens)
oxclo@oxclo:~$
```

4. Now you can start the Cassandra Shell:
Type:

```
cqlsh
```

You should see something like:

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native
      protocol v4]
Use HELP for help.
cqlsh>
```

5. Let's create a new database (Keyspace). Type (all on a single line)

   ```
   CREATE KEYSPACE TEST WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
   ```

   Check it worked. Type:

   ```
   desc keyspace test;
   ```

   You should see:

   ```
   CREATE KEYSPACE test WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}  AND durable_writes = true;
   ```

6. Now we need to select to use that keyspace:
   ```
   use test;
   ```

7. The command prompt will change to:
   ```
   cqlsh:test>
   ```

8. Let's create a simple (key, value) table. Type:

   ```
   create table kv ( key text, value text, primary key (key));
   ```

9. Now type

   ```
   desc kv;
   ```

10. You should see:

```
cqlsh:test> desc kv;

CREATE TABLE test.kv (
    key text PRIMARY KEY,
    value text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL",
"rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStra
tegy'}
    AND compression = {'sstable_compression':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

11. Add some simple values:

```
insert into kv (key, value) values ('a','1');
insert into kv (key, value) values ('b','2');
insert into kv (key, value) values ('c','3');
```

12. Now type:

```
select * from kv;
```

You should see:

```
 key | value
     -----+-------
         a |      1
         c |      3
         b |      2

     (3 rows)
```

13. You can also do other simple SQL of course

```
cqlsh:test> select * from kv where key='a' ;

 key | value
-----+-------
   a |     1

(1 rows)
```

14. Now exit the cqlsh:
    ```
    exit
    ```

15. Congratulations! You have Cassandra running and working.

PART B – Stress testing Cassandra

16. Now let's run a performance test on Cassandra. We will use the cassandra-stress tool which is part of the Cassandra distribution.

    First we need to write some data into Cassandra using the tool

    ```
    cassandra-stress write n=100000
    ```

17. You should see:

```
Connected to cluster: Test Cluster, max pending requests per connection 128, max
connections per host 8
Datatacenter: datacenter1; Host: localhost/127.0.0.1; Rack: rack1
Created keyspaces. Sleeping 1s for propagation.
Sleeping 2s...
Warming up WRITE with 25000 iterations...
Running WRITE with 200 threads for 100000 iteration
type       total ops,    op/s,    pk/s,   row/s,    mean,     med,     .95,     .99,
.999,     max,   time,   stderr, errors,  gc: #,  max ms,  sum ms,  sdv ms,      mb
total,          5528,    5536,    5536,    5536,    26.2,    19.5,    80.8,   126.3,
160.8,   176.3,    1.0,  0.00000,      0,       0,       0,       0,       0,       0
total,         14266,    5488,    5488,    5488,    41.7,    22.0,    67.0,   843.2,
906.7,   966.9,    2.6,  0.00309,      0,       1,      85,      85,       0,     149
total,         25042,    9973,    9973,    9973,    20.1,    18.0,    39.6,    58.4,
88.4,   113.1,    3.7,  0.17349,      0,       0,       0,       0,       0,       0
total,         34623,    9166,    9166,    9166,    21.7,    19.6,    39.6,    95.7,
108.8,   134.4,    4.7,  0.14206,      0,       1,      67,      67,       0,     154
total,         41240,    7783,    7783,    7783,    25.7,    18.3,    73.5,   122.0,
172.8,   173.4,    5.6,  0.11341,      0,       0,       0,       0,       0,       0
total,         48536,    5775,    5775,    5775,    34.7,    17.7,   223.8,   246.2,
271.4,   358.3,    6.8,  0.10642,      0,       1,     219,     219,       0,     146
total,         58315,   10209,   10209,   10209,    19.2,    15.5,    47.1,    81.3,
134.9,   166.3,    7.8,  0.09884,      0,       1,     100,     100,       0,     146
total,         68423,    8514,    8514,    8514,    20.3,    13.6,    58.4,   150.9,
205.2,   480.6,    9.0,  0.08616,      0,       0,       0,       0,       0,       0
total,         80563,    7964,    7964,    7964,    27.4,    13.8,    66.5,   466.9,
735.6,  1128.5,   10.5,  0.07651,      0,       1,      58,      58,       0,     152
total,         87174,    6223,    6223,    6223,    31.4,    15.7,    87.3,   426.4,
623.0,   703.2,   11.6,  0.07327,      0,       0,       0,       0,       0,       0
total,         99424,   11794,   11794,   11794,    16.9,    13.8,    31.4,   132.7,
148.4,   160.2,   12.6,  0.08576,      0,       1,     118,     118,       0,     140
total,        100000,    9058,    9058,    9058,    22.1,    19.0,    49.8,    55.3,
55.7,    55.7,   12.7,  0.07836,      0,       0,       0,       0,       0,       0


Results:
Op rate                   :    7,896 op/s  [WRITE: 7,896 op/s]
Partition rate            :    7,896 pk/s  [WRITE: 7,896 pk/s]
Row rate                  :    7,896 row/s [WRITE: 7,896 row/s]
Latency mean              :   25.1 ms [WRITE: 25.1 ms]
Latency median            :   16.4 ms [WRITE: 16.4 ms]
Latency 95th percentile   :   55.1 ms [WRITE: 55.1 ms]
Latency 99th percentile   :  167.1 ms [WRITE: 167.1 ms]
Latency 99.9th percentile :  766.6 ms [WRITE: 766.6 ms]
Latency max               : 1128.5 ms [WRITE: 1,128.5 ms]
Total partitions          :    100,000 [WRITE: 100,000]
Total errors              :          0 [WRITE: 0]
Total GC count            : 6
Total GC memory           : 887.073 MiB
Total GC time             :    0.6 seconds
Avg GC time               :  107.8 ms
StdDev GC time            :   53.5 ms
Total operation time      : 00:00:12

END
```

18. Now you can try a full test:

```
cassandra-stress mixed n=100000
```

19. At what thread count did you get the highest throughput? And the lowest latency?

PART C – Loading data from CSV files into Cassandra

20. Firstly, we need to create a database and a table in which to store our data. Start up the **cqlsh** again and type the following commands (https://freo.me/oxclo-cass-ddl)

```
CREATE KEYSPACE wind
WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '1'};

USE wind;

CREATE TABLE winddata (
    stationid text,
    time timestamp,
    direction float,
    temp float,
    velocity float,
    PRIMARY KEY (stationid, time)
);
```

21. In order to load the CSV files into Cassandra, we are going to use a Spark packages to help us: the Cassandra plugin for Spark.

Please note, there are lots of ways of loading CSV data into Cassandra, including a built-in Cassandra utility, which might be easier to use for small datasets.

This exercise is designed to demonstrate how to integrate Cassandra with Spark. For a really large dataset, if this was loaded from HDFS into Cassandra, this Spark-based approach would have the major benefit of parallelizing the operation.

22. To use these, we need to start Pyspark with the correct command line. Start a terminal window and start jupyter/spark with the right package:

```
pyspark --packages \
      com.datastax.spark:spark-cassandra-connector_2.12:3.0.0-alpha2
```

23. Now we need to set up our imports:
In your Jupyter notebook type (or cut and paste from http://freo.me/oxclo-spark-cass)

```
import time
from datetime import datetime
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)
```

24. Now lets load the CSV files into a SQL Dataframe:

```
df = sqlContext.read.format('com.databricks.spark.csv').\
  options(header='true', inferschema='true').\
  load('file:///home/oxclo/datafiles/wind/*')
```

25. Take a look at the data in df:

```
df.first()
```

After the log, you should see something like:

```
Row(Station_ID=u'SF04', Station_Name=u'Lincoln High School',
Location_Label=u'2162 24th Ave', Interval_Minutes=5,
Interval_End_Time=u'2015-01-5? 07:50',
Wind_Velocity_Mtr_Sec=0.979, Wind_Direction_Variance_Deg=40.31,
Wind_Direction_Deg=57.69, Ambient_Temperature_Deg_C=6.297,
Global_Horizontal_Irradiance=0.706)
```

26. We can take advantage of Python to do any kind of Map/Reduce finagling of the data. In our case, we are just going to sort the dates into something Python understands and also change the names of the columns to match the Cassandra table.

Firstly we want to map the `Interval_End_Time` into something we can put in Cassandra. Cassandra expects a Python `datetime.datetime` object.

This chunk of python will convert the string date/time into that:

```
convertTime = lambda t: \
datetime.fromtimestamp( \
time.mktime(time.strptime(t, "%Y-%m-%d? %H:%M")))
```

27. Secondly, we need to create a Python dictionary with the right names for our Cassandra Table. This function does that. I recommend you cut and paste!

```
toRow = lambda s: \
      Row(stationid=s.Station_ID, \
      time=convertTime(s.Interval_End_Time), \
      direction=s.Wind_Direction_Deg, \
      temp=s.Ambient_Temperature_Deg_C, \
      velocity=s.Wind_Velocity_Mtr_Sec)
```

28. We need to map this function onto the data. We can convert RDD to/from DF in one line:

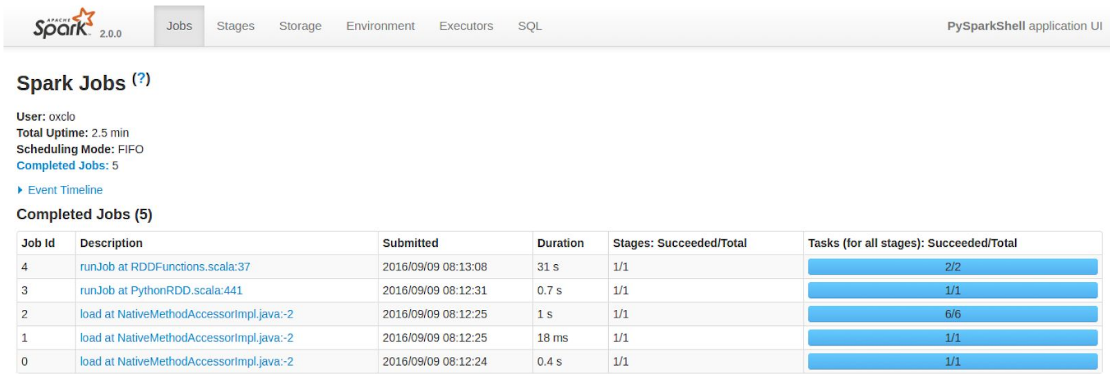
```
newDF = df.rdd.map(toRow).toDF()
```

29. Finally, we can do the work:

```
newDF.write \
      .format("org.apache.spark.sql.cassandra") \
      .mode('append') \
      .options(table="winddata", keyspace="wind").save()
```

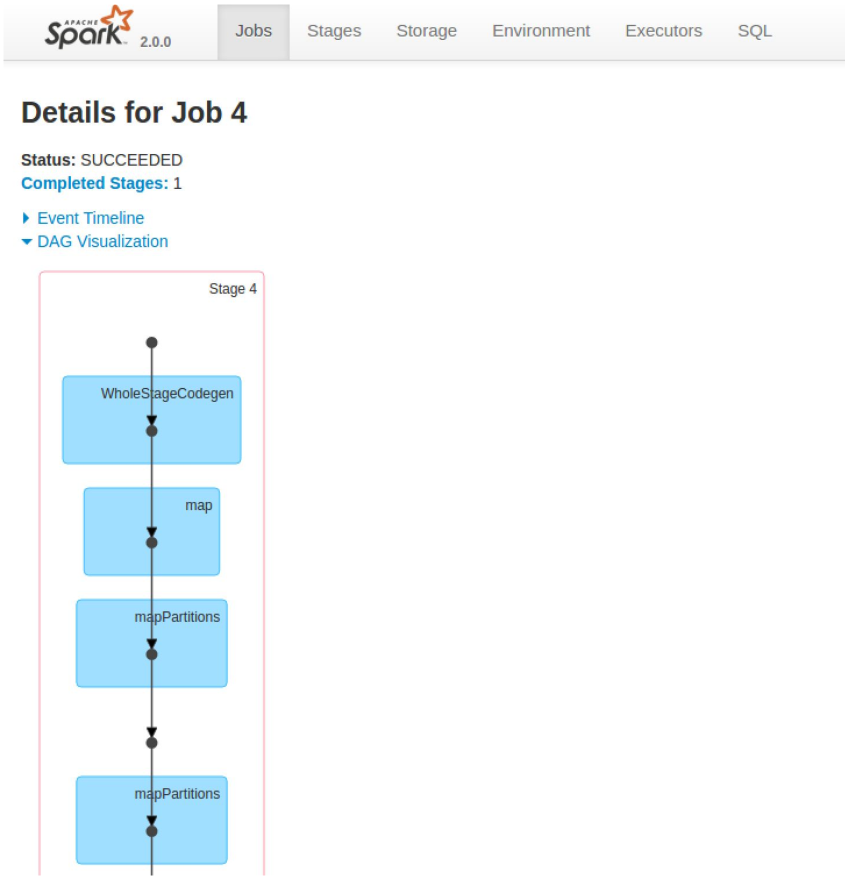
This will take a bit longer!

30. Browse to http://localhost:4040
It will look similar to:



Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	runJob at RDDFunctions.scala:37	2016/09/09 08:13:08	31 s	1/1	2/2
3	runJob at PythonRDD.scala:441	2016/09/09 08:12:31	0.7 s	1/1	1/1
2	load at NativeMethodAccessorImpl.java:-2	2016/09/09 08:12:25	1 s	1/1	6/6
1	load at NativeMethodAccessorImpl.java:-2	2016/09/09 08:12:25	18 ms	1/1	1/1
0	load at NativeMethodAccessorImpl.java:-2	2016/09/09 08:12:24	0.4 s	1/1	1/1

31. Click on the most recent job:



32. You can also get more details by clicking on a stage in the DAG (Directed Acyclic Graph) picture:

33. Check that the data has loaded. In your **cqlsh** window type:

```
select * from wind.winddata limit 15;
```

34. You should see something like:

```
 stationid | time                     | direction | temp  | velocity
-----------+--------------------------+-----------+-------+----------
      SF36 | 2015-01-01 00:00:00+0000 |     116.9 | 11.33 |    2.727
      SF36 | 2015-01-01 00:05:00+0000 |     108.5 | 11.25 |    1.814
      SF36 | 2015-01-01 00:10:00+0000 |     113.7 |  11.2 |    2.621
      SF36 | 2015-01-01 00:15:00+0000 |     117.8 | 11.11 |    3.678
      SF36 | 2015-01-01 00:20:00+0000 |     117.3 | 11.07 |    2.842
      SF36 | 2015-01-01 00:25:00+0000 |     117.3 | 11.07 |    2.629
      SF36 | 2015-01-01 00:30:00+0000 |     117.3 | 11.09 |    2.235
      SF36 | 2015-01-01 00:35:00+0000 |     117.2 | 11.09 |    2.043
      SF36 | 2015-01-01 00:40:00+0000 |     117.2 | 11.05 |    1.635
      SF36 | 2015-01-01 00:45:00+0000 |     117.3 | 10.93 |    2.224
      SF36 | 2015-01-01 00:50:00+0000 |     112.5 | 10.86 |    1.822
      SF36 | 2015-01-01 00:55:00+0000 |     108.7 |  10.8 |    0.866
      SF36 | 2015-01-01 01:00:00+0000 |     108.7 | 10.67 |    1.068
      SF36 | 2015-01-01 01:05:00+0000 |     108.6 | 10.54 |    1.393
      SF36 | 2015-01-01 01:10:00+0000 |     108.7 | 10.44 |    1.468

(15 rows)
```

35. Congratulations, you have finished this lab.