**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

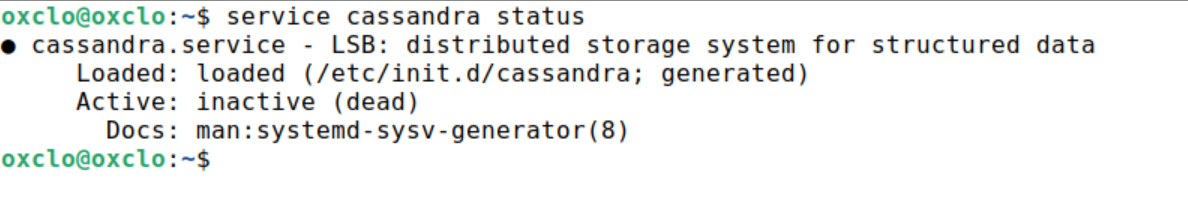
Load data from CSV into Cassandra using Spark Python

**Software Requirements**

(see separate document for installation of these)

* Apache Spark 3.1.2
* Python 3.9.x
* Apache Cassandra 3.x
* Nano text editor or other text editor

**Part A**

1. Make sure Cassandra is running
   1. In a Terminal window (Crtl-Alt-T) type:  
        
      service cassandra status
   2. You will probably see  
      
   3. Start Cassandra with:

sudo service cassandra start  
  
and then check the status again.

1. Now you can ask Cassandra about its own situation:  
   nodetool status  
     
   You should see something like:  
   

1. You can also try:  
     
   nodetool info  
     
   You should see something li
2. Now you can start the Cassandra Shell:  
   Type:

cqlsh

You should see something like:

/usr/bin/cqlsh.py:460: DeprecationWarning: Legacy execution parameters will be removed in 4.0. Consider using execution profiles.

/usr/bin/cqlsh.py:490: DeprecationWarning: Setting the consistency level at the session level will be removed in 4.0. Consider using execution profiles and setting the desired consitency level to the EXEC\_PROFILE\_DEFAULT profile.

Connected to Test Cluster at 127.0.0.1:9042

[cqlsh 6.0.0 | Cassandra 4.0 | CQL spec 3.4.5 | Native protocol v5]

Use HELP for help.

cqlsh>

1. You can ignore the warnings.
2. 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;

1. Now we need to select to use that keyspace:  
   use test;
2. The command prompt will change to:  
   cqlsh:test>
3. Let’s create a simple (key, value) table. Type:  
     
   create table kv ( key text, value text, primary key (key));
4. Now type  
     
   desc kv;
5. You should see:  
   
6. 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');

1. Now type:  
   select \* from kv;  
     
   You should see:  
    key | value

-----+-------

a | 1

c | 3

b | 2

(3 rows)

1. You can also do other simple SQL of course

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

key | value

-----+-------

a | 1

(1 rows)

1. Now exit the cqlsh:  
   exit
2. Congratulations! You have Cassandra running and working.

**PART B – Stress testing Cassandra**

1. 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

1. You should see:  
   
2. Now you can try a full test:  
     
   cassandra-stress mixed n=100000
3. At what thread count did you get the highest throughput? And the lowest latency?

**PART C – Loading data from CSV files into Cassandra**

1. 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)

);

1. 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.*
2. 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\_2.12:3.0.1

1. 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)
2. 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/\*')
3. 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)

1. 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")))

1. 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)

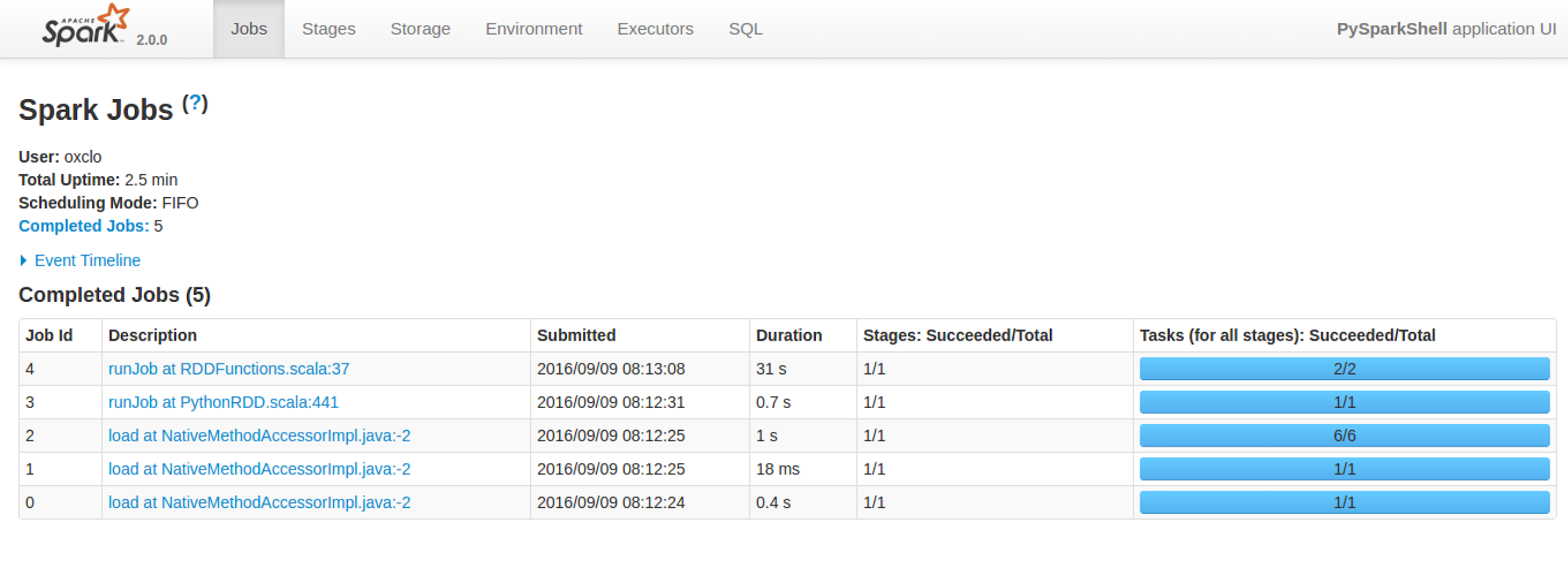
1. We need to map this function onto the data. We can convert RDD to/from DF in one line:  
     
   newDF = df.rdd.map(toRow).toDF()
2. Finally, we can do the work:

newDF.write \

.format("org.apache.spark.sql.cassandra") \   
 .mode('append') \

.options(table="winddata", keyspace="wind").save()

This will take a bit longer!

1. Browse to <http://localhost:4040>   
   It will look similar to:  
   
2. Click on the most recent job:   
   
3. You can also get more details by clicking on a stage in the DAG (Directed Acyclic Graph) picture:  
   
4. Check that the data has loaded. In your **cqlsh** window type:  
     
   select \* from wind.winddata limit 15;
5. You should see something like:

1. Congratulations, you have finished this lab.