**Exercise 10**

*Get started with Cassandra and import data via Spark*

**Prior Knowledge**

Unix Command Line Shell

HDFS

Simple Python

Spark Python  
Simple SQL syntax

**Learning Objectives**

Understand Cassandra’s CQL shell

Integrate Python, Cassandra and Spark

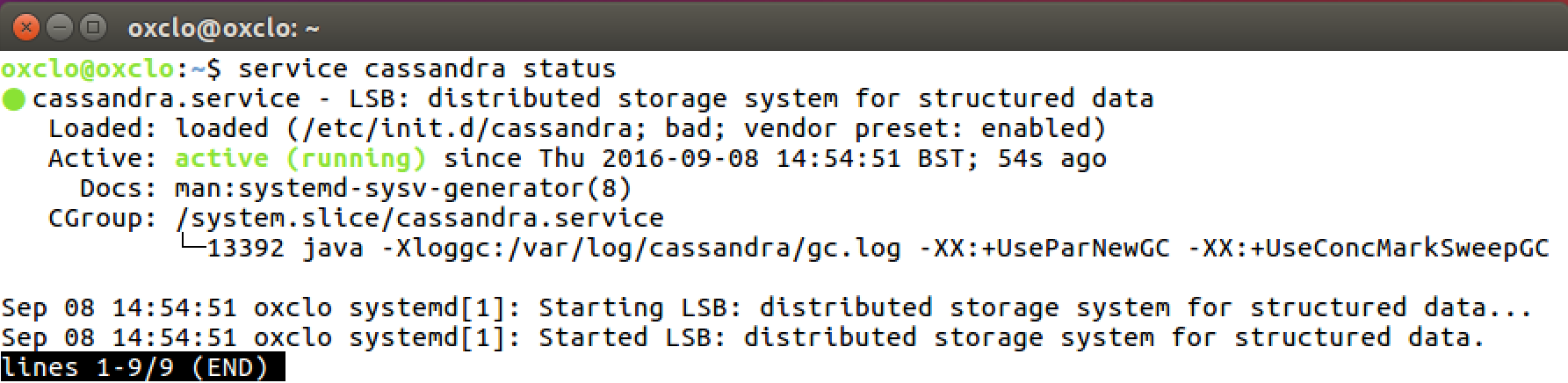
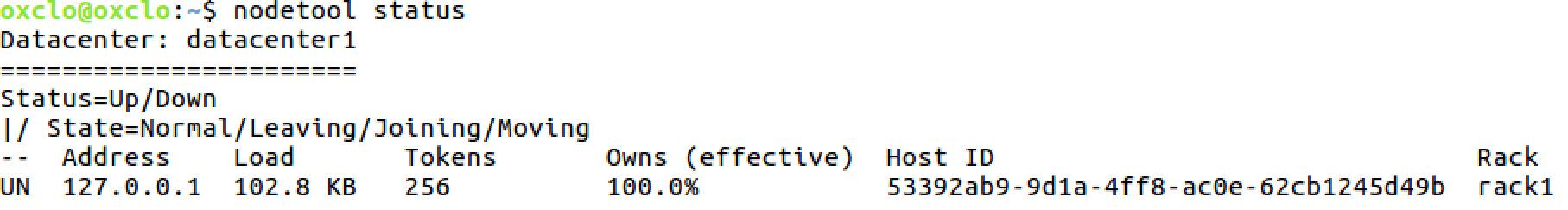
Load data from CSV into Cassandra using Spark Python

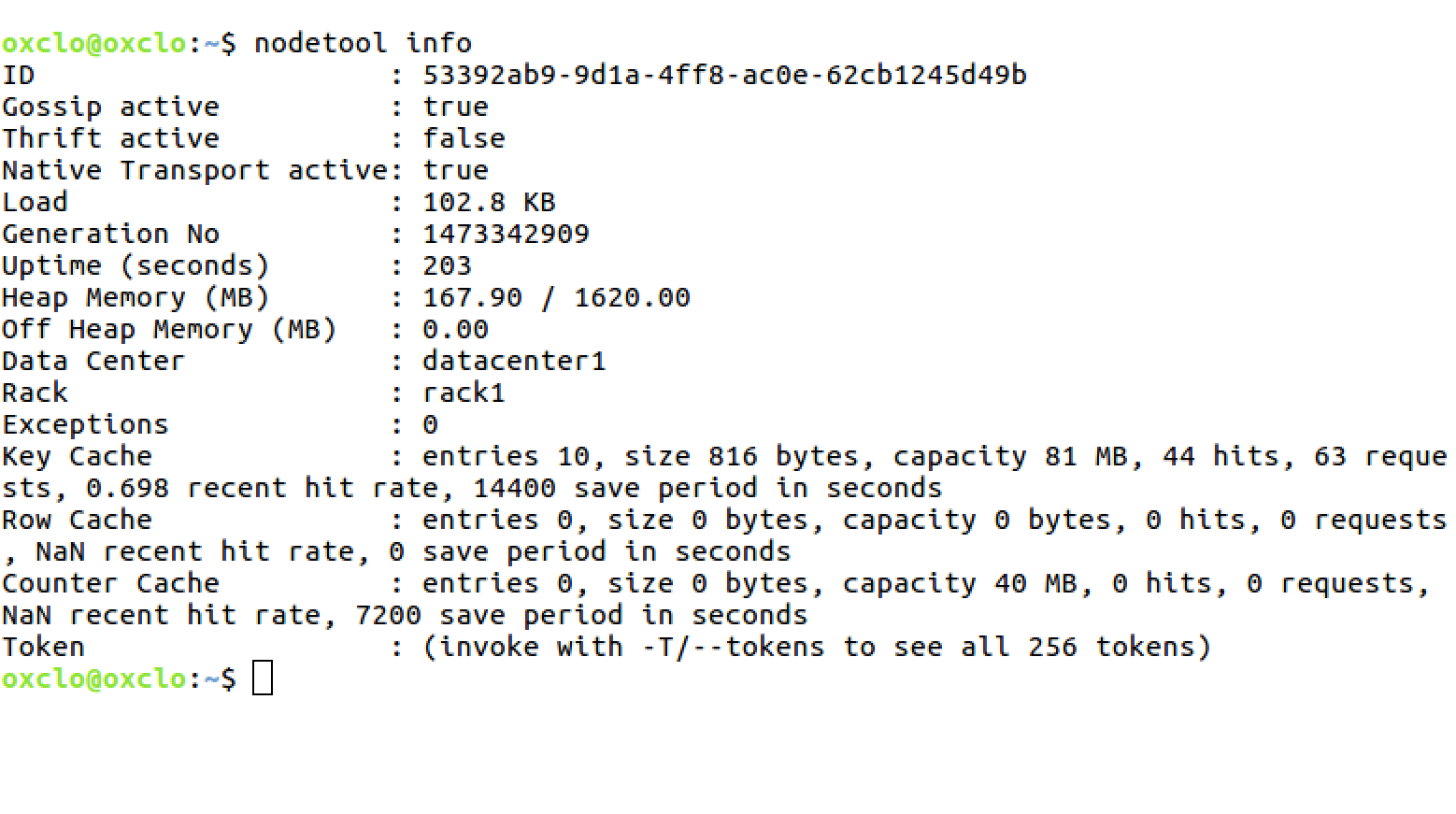
**Software Requirements**

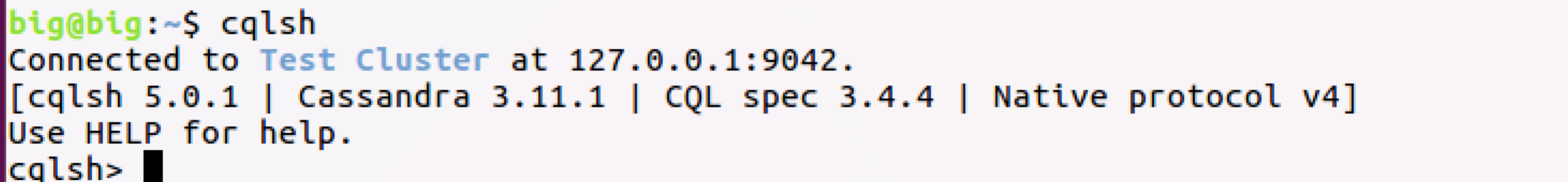
(see separate document for installation of these)

* Apache Spark 2.2.0
* Python 2.7.12
* Apache Cassandra 3.11.1
* 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 should see  
      
   3. Type q to get back to the command line
   4. If not, try   
      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:  
   

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

cqlsh  
  
You should see:  
  
Let’s create a new database (Keyspace):

* 1. Type (all on a single line)  
       
     CREATE KEYSPACE TEST WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };
  2. Check it worked:  
     Type:  
       
     desc keyspace test;
  3. 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
   1. Type:  
      create table kv ( key text, value text, primary key (key));
   2. Now type  
      desc kv;
   3. 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.SizeTieredCompactionStrategy'}

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';

* 1. 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)

**PART B – 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 (available here: <https://freo.me/winddata-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. Type **exit** to leave the cqlsh command line.
2. 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.*
3. To use these, we need to start pyspark with the correct command line. Since we are starting pyspark via Jupyter, we need to pass this via an environment variable.
4. Start a terminal window and type (all on one line)  
     
   export PYSPARK\_SUBMIT\_ARGS=”--packages datastax:spark-cassandra-connector:2.0.3-s\_2.11 pyspark-shell”

It is important that you start jupyter from this window now. If you close the window, this environment will be lost.

1. Start Jupyter as before. Create a new Python2 notebook and copy the following code into the cell (you can get this from here: )
2. import time  
   from datetime import datetime  
   from pyspark.sql import SQLContext, Row  
   sqlContext = SQLContext(sc)
3. 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/\*')
4. 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)
5. 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: \

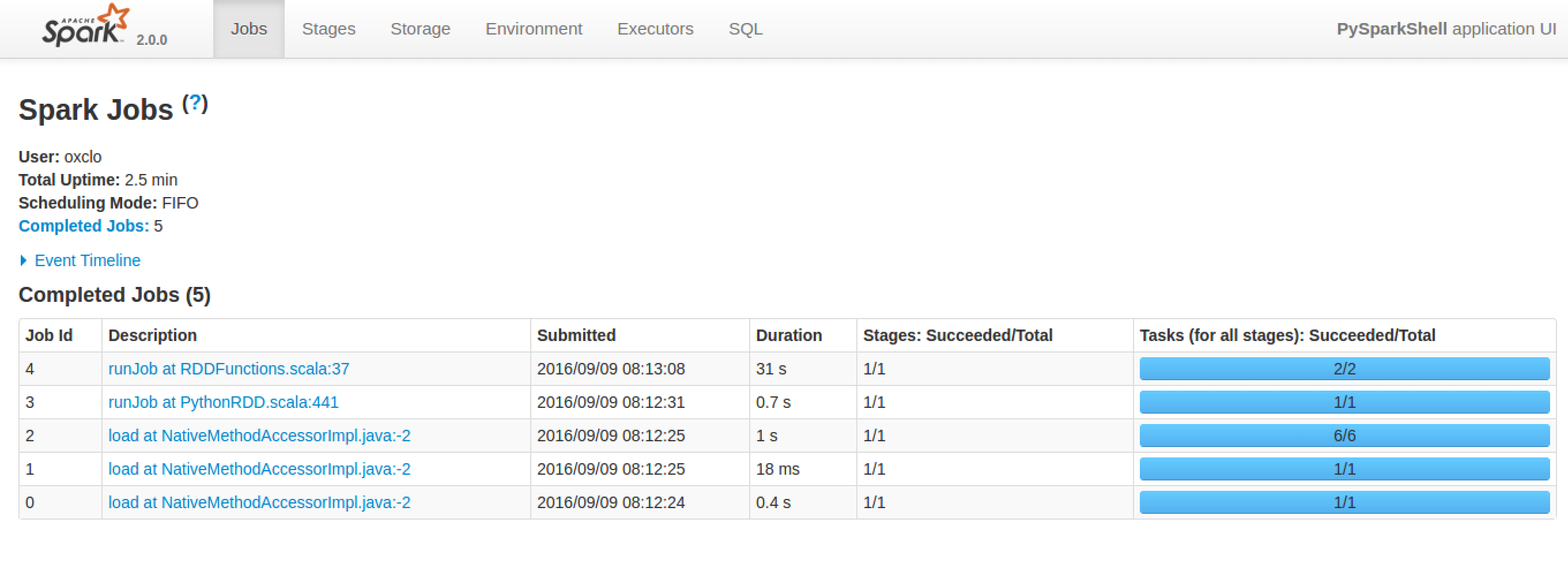
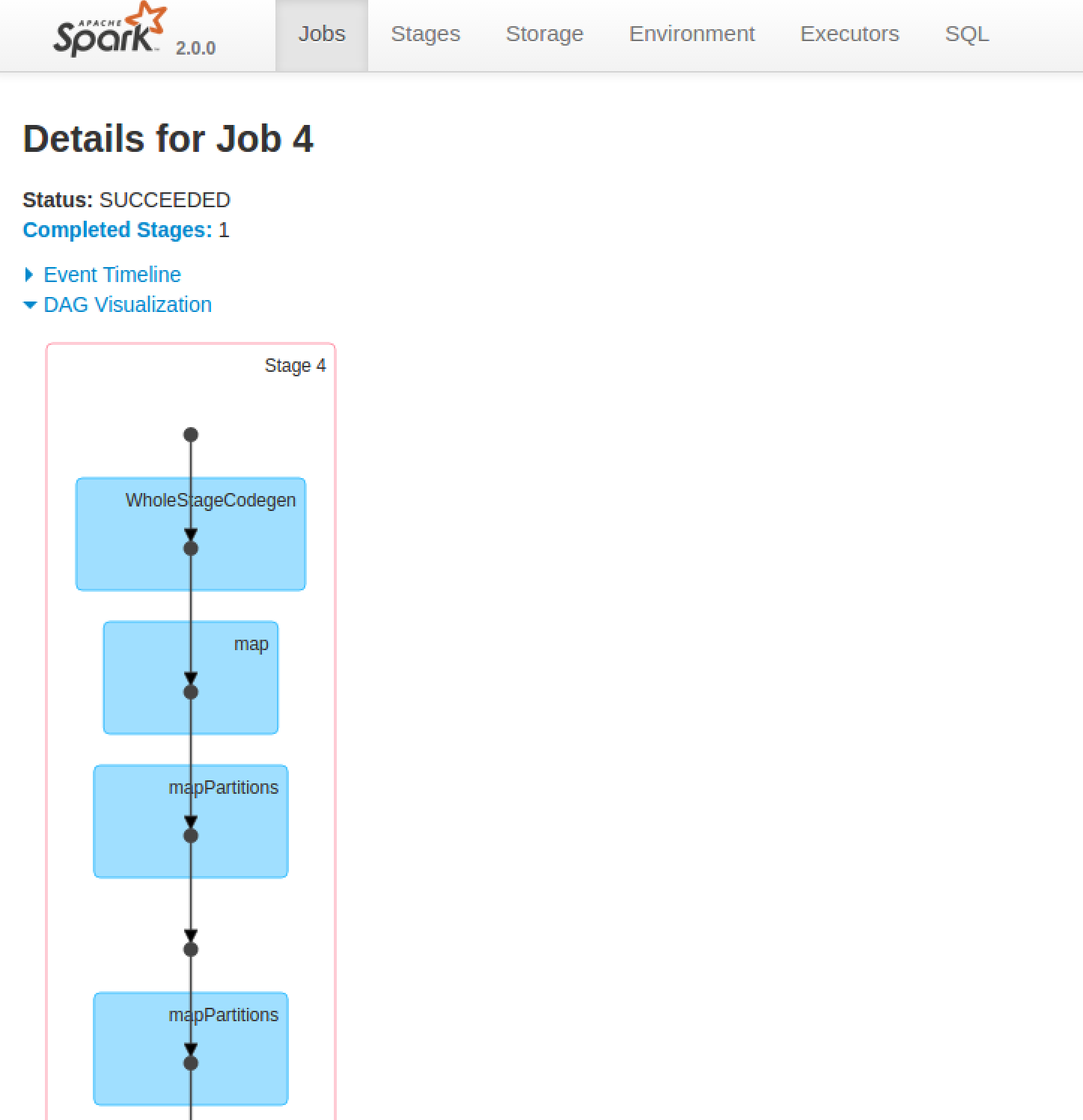
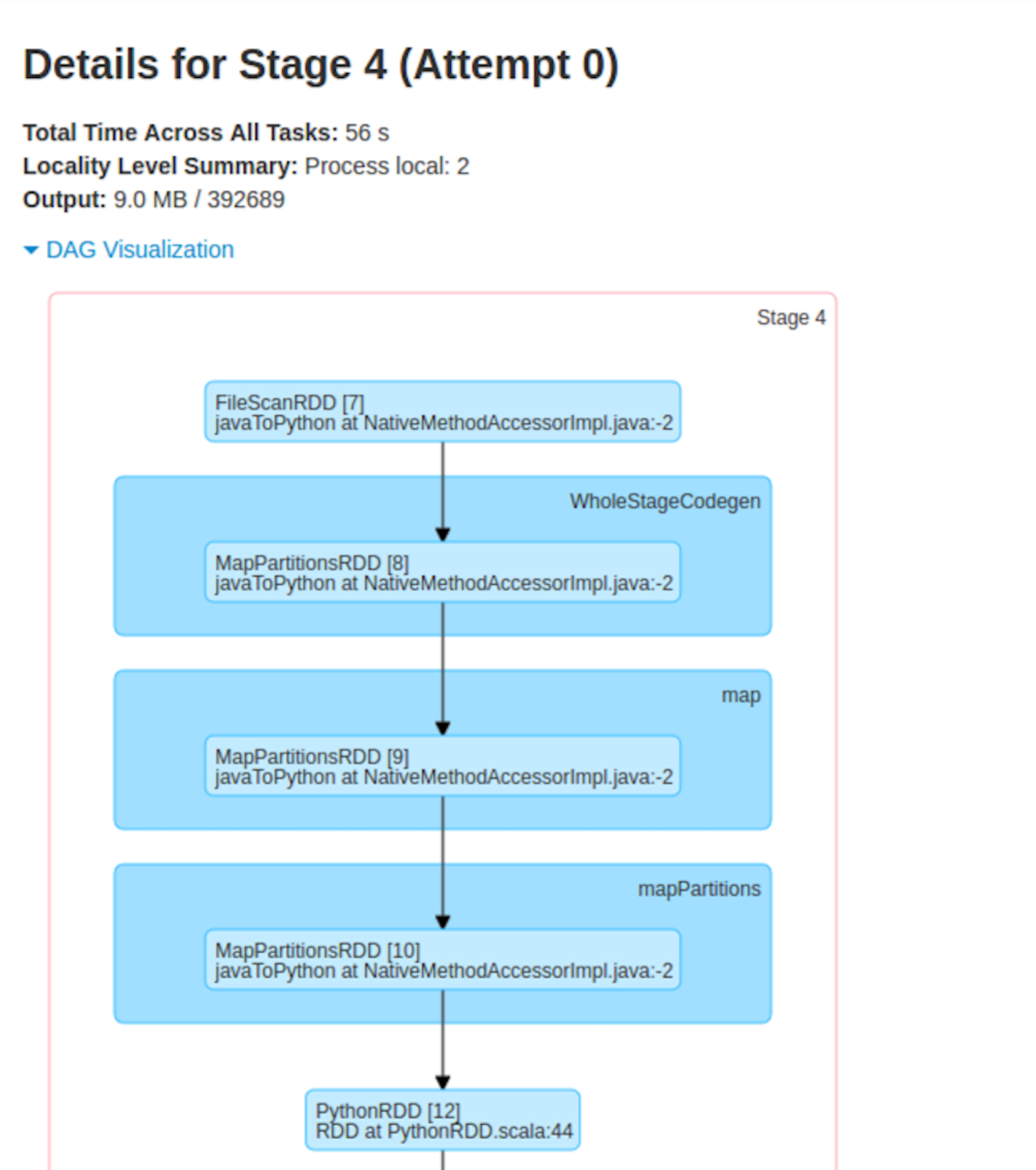
Row(stationid=s.Station\_ID, \

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:  
     
   newdDF.write\  
    .format("org.apache.spark.sql.cassandra")\  
    .mode('append')\  
    .options(table="winddata", keyspace="wind")\  
    .save()  
     
   This will take a bit longer!
3. Browse to <http://localhost:4040>   
   It will look similar to:  
   
4. Click on the most recent job:   
   
5. You can also get more details by clicking on a stage in the DAG (Directed Acyclic Graph) picture:  
   
6. Check that the data has loaded. In your **cqlsh** window type:  
     
   select \* from wind.winddata limit 15;
7. 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)

1. Congratulations, you have finished this lab.