**Exercise 3**

*SparkSQL*

**Prior Knowledge**

Unix Command Line Shell

Simple Python

Apache Spark in Jupyter (from previous exercise)

**Learning Objectives**

SparkSQL

Reading CSV files in Spark

**Software Requirements**

(see separate document for installation of these)

* Apache Spark
* Jupyter

1. Let’s create a new directory for our work:  
     
   cd ~  
   mkdir sql  
   cd sql  
   jupyter notebook

1. We are going to use Spark’s SQL support which in turn uses Apache Hive.
2. This combined with the CSV package we saw earlier makes it very easy to work with data.   
   First let’s tell spark we are using SQL.   
   from pyspark.sql import SQLContext  
     
   sqlContext = SQLContext(sc)
3. Now let’s load the data into a DataFrame. (one line)  
     
   df = sqlContext.read.format('com.databricks.spark.csv').  
   options(header='true', inferschema='true').  
   load('s3a://oxclo-wind/2015/\*')
4. Spark should go away and think a bit, and also show some ephemeral log lines about the staging.
5. The df object we have is not an RDD, but instead a DataFrame. This is basically a SQL construct. But we can easily convert it into an RDD.
6. We can print a nice table showing the first few rows with:  
     
   df.show(4)  
   (I shrunk this so you can see the table nicely!)

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

|Station\_ID| Station\_Name|Location\_Label|Interval\_Minutes|Interval\_End\_Time|Wind\_Velocity\_Mtr\_Sec|Wind\_Direction\_Variance\_Deg|Wind\_Direction\_Deg|Ambient\_Temperature\_Deg\_C|Global\_Horizontal\_Irradiance|

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

| SF15|Warnerville Switc...| Warnerville| 5| 2015-01-5? 00:05| 1.628| 8.1| 148.5| 0.92| 0.061|

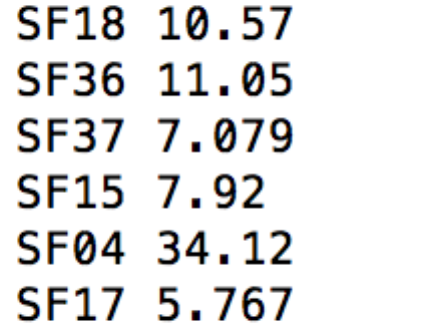
| SF15|Warnerville Switc...| Warnerville| 5| 2015-01-5? 00:10| 1.519| 9.4| 151.1| 0.717| 0.064|

| SF15|Warnerville Switc...| Warnerville| 5| 2015-01-5? 00:15| 1.482| 8.7| 142.7| 0.627| 0.059|

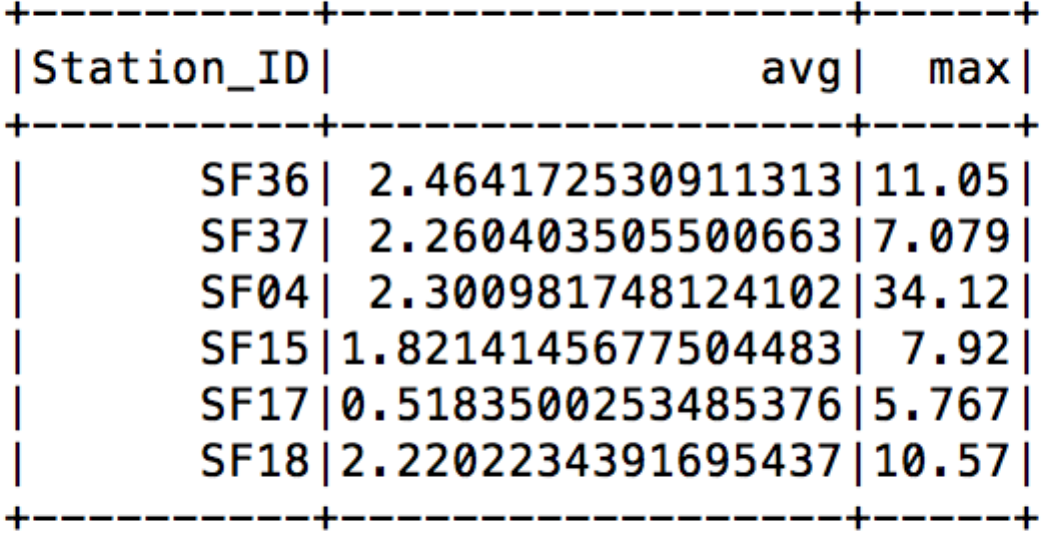
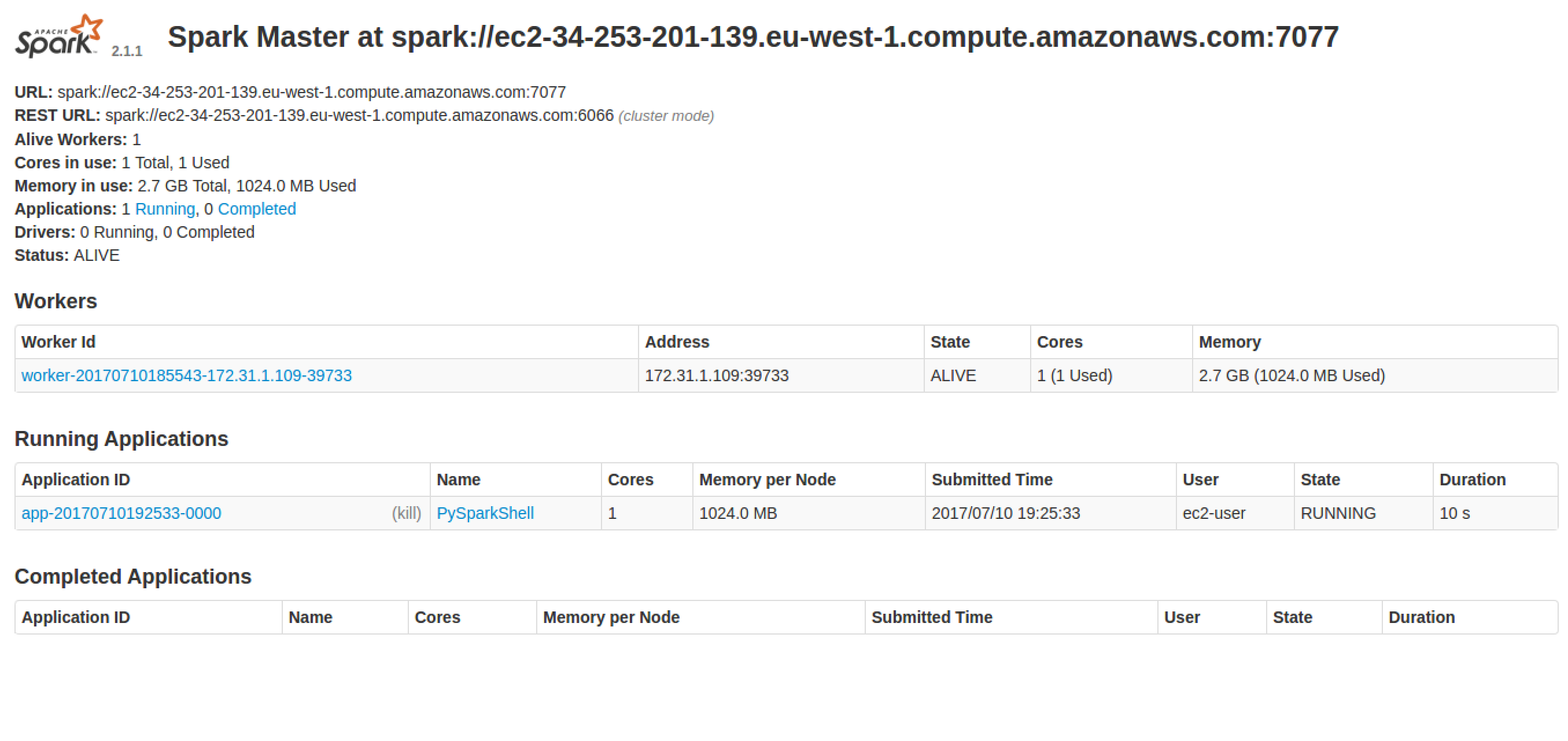
| SF15|Warnerville Switc...| Warnerville| 5| 2015-01-5? 00:20| 1.985| 6.895| 141.8| 0.5| 0.062|

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

only showing top 4 rows

1. We can also convert the DataFrame into an RDD, allowing us to do functional programming on it (map/reduce/etc)  
     
   winds = df.rdd
2. Let’s do the normal step of mapping the data into a simple <K,V> pair. Each column in the row can be accessed by the syntax e.g. row.Station\_ID  
     
   We can therefore map our RDD with the following:   
   mapped = winds.map(lambda s: (s.Station\_ID, s.Wind\_Velocity\_Mtr\_Sec))
3. We can simply calculate the maximum values with this reducer:  
     
   maxes = mapped.reduceByKey(lambda a, b: a if (a>b) else b)
4. And once again collect / print:  
     
   for (k,v) in maxes.collect(): print k,v
5. You will see a bunch of log before the following appears:
6. You can also turn the response of a collect into a Python Map, which is handy. Try this:  
     
   maxes.collectAsMap()['SF04']
7. You can also try:  
   print maxes.collectAsMap()

**PART B – Using SQL**

1. There is an easier way to do all this if you are willing to write some SQL.
2. First we need to give our DataFrame a table name:  
   df.registerTempTable('wind')
3. Now we can use a simple SQL statement against our data.   
   ALL ON ONE Line type:  
     
   sqlContext.sql("SELECT Station\_ID, avg(Wind\_Velocity\_Mtr\_Sec) as avg,max(Wind\_Velocity\_Mtr\_Sec) as max from wind group by Station\_ID").show()
4. Bingo you should see a lot of log followed by:  
   
5. Recap. So fat we have:
   1. Started Spark in EC2
   2. Loaded data from S3
   3. Used SQL to read in CSV files
   4. Explored Map/Reduce on those CSV files
   5. Used SQL to query the data.
6. Find the IP address of the Spark Master. There are two ways. Firstly, it showed up in the console when you first launched the flintrock cluster:  
   [34.253.201.139] Configuring Spark master...  
      
   Alternatively, you can find it as “oxcloXX-sc-master” in the EC2 instances.
7. Go to <http://xx.xx.xx.xx:8080> using the master’s IP address.  
   You should see something like:
8. Quit the pyspark shell:   
   quit()
9. Exit the SSH session:  
   exit
10. If you want you can try adding another slave and then rerun the analysis. You can see the extra core working in the Web UI

flintrock add-slave --num-slaves 1 oxcloXX-sc

To save you retyping all that spark code, look here: <https://freo.me/wind-sql>

1. If you are planning to do the **Jupyter on EC2 exercise** straight away, then you can start it now and use your existing flintrock/EC2 cluster. Otherwise please follow the next instruction to shut down the EC2 instances.
2. We must remember to stop our cluster as well (its costing money…)  
   From Ubuntu terminal where you started the Spark cluster  
     
   flintrock destroy oxcloXX-sc  
     
   Type y when prompted.
3. Congratulations, this lab is complete.