Exercise 7

More Apache Spark and Python, EC2

Prior Knowledge
Unix Command Line Shell
Simple Python

Learning Objectives
Using Spark on EC2
Accessing S3 files on Spark
Reading CSV files in Spark
Seeing the differences between Spark and Hadoop by performing the Wind Analysis in Spark
Spark SQL

Software Requirements
(see separate document for installation of these)

- EC2 credentials
- Flintrock
- Livy

Part A. Starting Spark in EC2

1. Do you remember the Access Key and Secret Key from Exercise 1? You need those now. If you have lost them, ask for help.

2. In a terminal window type:
```
export AWS_ACCESS_KEY_ID=<your access key here>
export AWS_SECRET_ACCESS_KEY=<your secret key here>
```

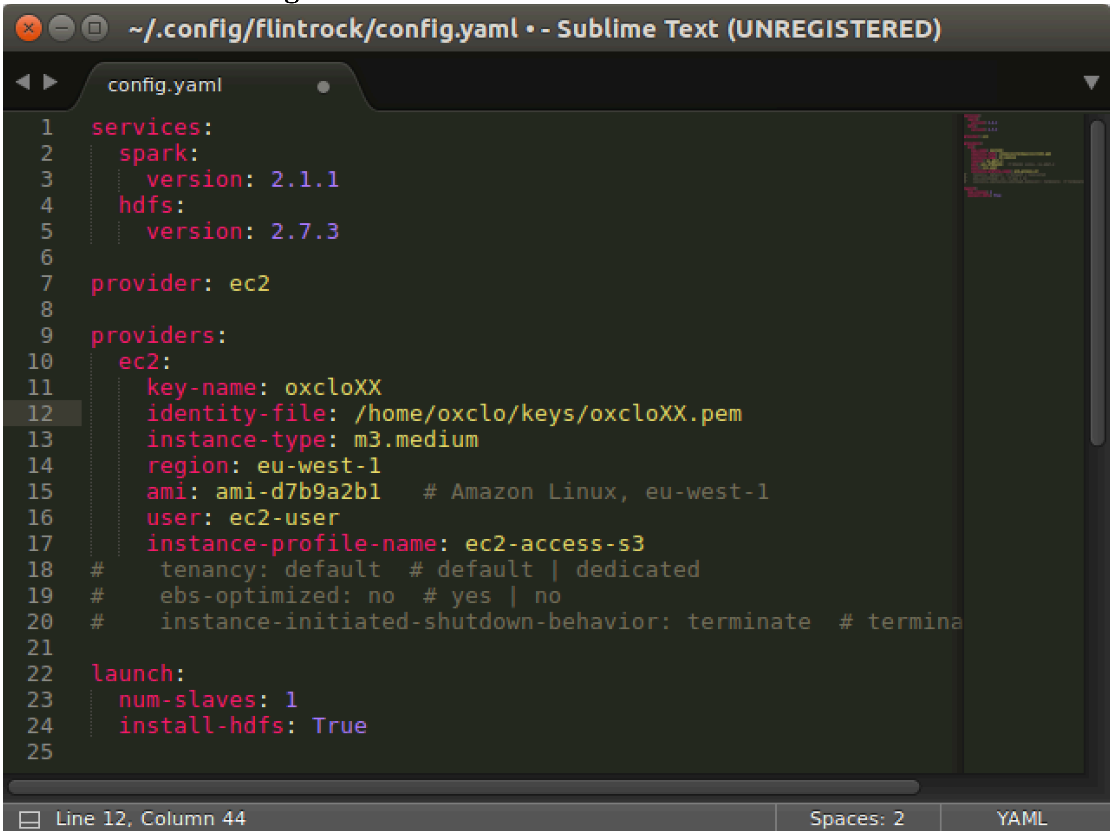
3. Further commands in this lab need to be in this same command window, because if you start a new window, these environment variables won't be there. In a more serious setup you would either add these to your profile or put them in a command-shell program so you could call them easily.

4. There is a project from the creators of Spark to run it in EC2, but it is not very good! Instead we will use a tool called **flintrock**

5. Before we can use flintrock, you need to modify the config file for flintrock so that it uses your own keys. Edit the flintrock config file:

   subl ~/.config/flintrock/config.yaml

It will look something like:



~/.config/flintrock/config.yaml • - Sublime Text (UNREGISTERED)
config.yaml
Line 12, Column 44
Spaces: 2
YAML

```
services:
  spark:
    version: 2.1.1
  hdfs:
    version: 2.7.3

provider: ec2

providers:
  ec2:
    key-name: oxcloXX
    identity-file: /home/oxclo/keys/oxcloXX.pem
    instance-type: m3.medium
    region: eu-west-1
    ami: ami-d7b9a2b1   # Amazon Linux, eu-west-1
    user: ec2-user
    instance-profile-name: ec2-access-s3
#    tenancy: default  # default | dedicated
#    ebs-optimized: no  # yes | no
#    instance-initiated-shutdown-behavior: terminate  # termina

launch:
  num-slaves: 1
  install-hdfs: True

```

The source for this is here:
https://freo.me/flintrock-conf

This is modified in a couple of ways. Firstly, it gives the Ireland region and AMI files. Secondly, there is an “instance-profile-name”. This is a AWS feature that gives the running VM access to other APIs - in this case S3.

6. Change the key name and identity file to match your key name and identity file. Save the file.

7. You should now be able to launch a cluster in Amazon:

```
cd flintrock
./flintrock launch oxcloXX-sc

(using your XX)
```

8. Now you should see something like:

```
Launching 2 instances...
[54.154.17.100] SSH online.
[54.154.17.100] Configuring ephemeral storage...
[54.154.17.100] Installing Java 1.8...
[34.253.201.139] SSH online.
[34.253.201.139] Configuring ephemeral storage...
[34.253.201.139] Installing Java 1.8...
[54.154.17.100] Installing HDFS...
[34.253.201.139] Installing HDFS...
[54.154.17.100] Installing Spark...
[34.253.201.139] Installing Spark...
[34.253.201.139] Configuring HDFS master...
[34.253.201.139] Configuring Spark master...
HDFS online.
Spark Health Report:
  * Master: ALIVE
  * Workers: 1
  * Cores: 1
  * Memory: 2.7 GB
launch finished in 0:03:49.
```

9. Let’s login to the master (all one line):

```
./flintrock login oxcloXX-sc
```

You see something like:

```
Warning: Permanently added '34.253.201.139' (ECDSA) to the list
of known hosts.
Last login: Mon Jul 10 18:55:35 2017 from host109-156-251-
208.range109-156.btcentralplus.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
1 package(s) needed for security, out of 1 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-6-32 ~]$
```

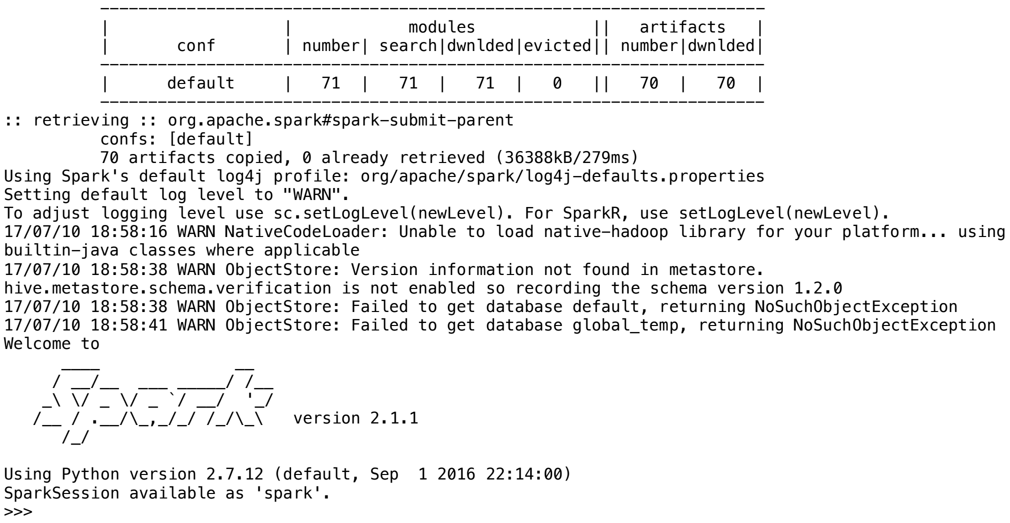
10. This basically just SSH’s you into the master. You could do the same from the EC2 console as before.

11. Now start pyspark once again.

This time we are going to add in a Spark Package that supports accessing S3 data (Amazon object storage). Once again, all one line

```
pyspark --master spark://0.0.0.0:7077
 --packages com.amazonaws:aws-java-sdk-
pom:1.10.34,org.apache.hadoop:hadoop-aws:2.7.2
```

12. You should see a lot of logging, eventually ending with:



```
          ---------------------------------------------------------------------
          |                  |            modules            ||   artifacts   |
          |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
          ---------------------------------------------------------------------
          |      default     |   71  |   71  |   71  |   0   ||   70  |   70  |
          ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent
          confs: [default]
          70 artifacts copied, 0 already retrieved (36388kB/279ms)
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/07/10 18:58:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
17/07/10 18:58:38 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
17/07/10 18:58:38 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/07/10 18:58:41 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 2.7.12 (default, Sep  1 2016 22:14:00)
SparkSession available as 'spark'.
>>>
```

13. It is perfectly possible to get Jupyter to talk to Spark on our cluster, but it is slightly complex, so we will just use the normal Python command-line for the moment.

14. We are going to use Spark's SQL support which in turn uses Apache Hive.

15. 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. In the Python command-line type:

```
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
```

16. Now let's load the data into a DataFrame. (one line)

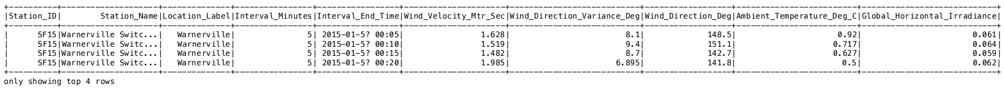
    ```
    df = sqlContext.read.format('com.databricks.spark.csv').
    options(header='true', inferschema='true').
    load('s3a://oxclo-wind/2015/*')
    ```

17. Spark should go away and think a bit, and also show some ephemeral log lines about the staging.

18. 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.

19. We can print a nice table showing the first few rows with:

    df.show(4)



```
+----------+--------------------+--------------+----------------+-----------------+---------------------+---------------------------+------------------+-------------------------+----------------------------+
|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
```

    (I shrunk this so you can see the table nicely!)

20. We can also convert the DataFrame into an RDD, allowing us to do functional programming on it (map/reduce/etc)

    ```
    winds = df.rdd
    ```

21. 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))
    ```

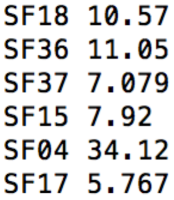
22. We can simply calculate the maximum values with this reducer:

    ```
    maxes = mapped.reduceByKey(lambda a, b: a if (a>b) else b)
    ```

23. And once again collect / print:

    ```
    for (k,v) in maxes.collect(): print k,v
    ```

24. You will see a bunch of log before the following appears:



```
SF18 10.57
SF36 11.05
SF37 7.079
SF15 7.92
SF04 34.12
SF17 5.767
```

25. You can also turn the response of a collect into a Python Map, which is handy. Try this:

```
maxes.collectAsMap()['SF04']
```

26. You can also try:

```
print maxes.collectAsMap()
```

PART B – Using SQL

27. There is an easier way to do all this if you are willing to write some SQL.

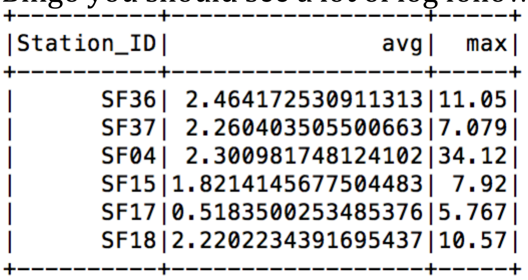
28. First we need to give our DataFrame a table name:

```
df.registerTempTable('wind')
```

29. 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()
```

30. Bingo you should see a lot of log followed by:



```
+----------+------------------+-----+
|Station_ID|               avg|  max|
+----------+------------------+-----+
|      SF36| 2.464172530911313|11.05|
|      SF37| 2.260403505500663|7.079|
|      SF04| 2.300981748124102|34.12|
|      SF15|1.8214145677504483| 7.92|
|      SF17|0.5183500253485376|5.767|
|      SF18|2.2202234391695437|10.57|
+----------+------------------+-----+
```

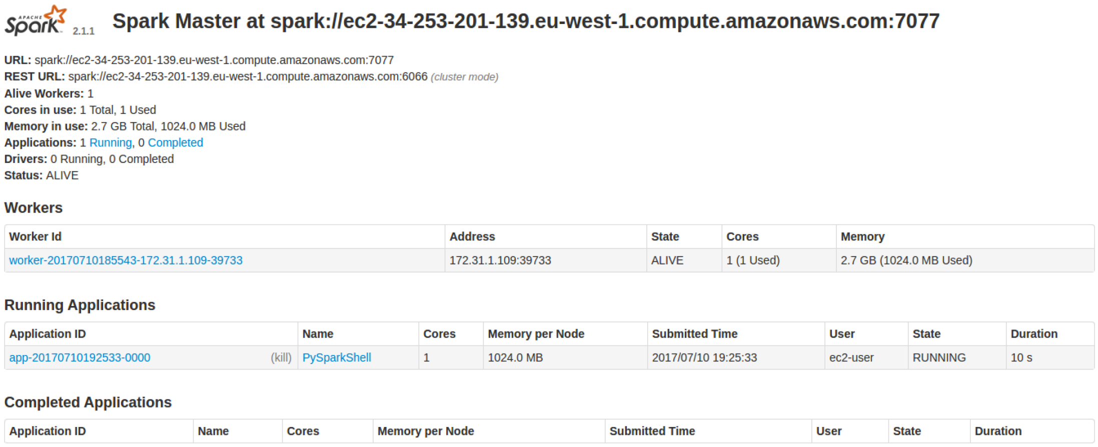
31. Recap. So fat we have:
    a. Started Spark in EC2
    b. Loaded data from S3
    c. Used SQL to read in CSV files
    d. Explored Map/Reduce on those CSV files
    e. Used SQL to query the data.

32. 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.

33. Go to http://xx.xx.xx.xx:8080 using the master's IP address.
    You should see something like:



Spark 2.1.1 Spark Master at spark://ec2-34-253-201-139.eu-west-1.compute.amazonaws.com:7077

URL: spark://ec2-34-253-201-139.eu-west-1.compute.amazonaws.com:7077
REST URL: spark://ec2-34-253-201-139.eu-west-1.compute.amazonaws.com:6066 *(cluster mode)*
Alive Workers: 1
Cores in use: 1 Total, 1 Used
Memory in use: 2.7 GB Total, 1024.0 MB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170710185543-172.31.1.109-39733	172.31.1.109:39733	ALIVE	1 (1 Used)	2.7 GB (1024.0 MB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20170710192533-0000 (kill)	PySparkShell	1	1024.0 MB	2017/07/10 19:25:33	ec2-user	RUNNING	10 s

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

34. Quit the pyspark shell:

    ```
    quit()
    ```

35. Exit the SSH session:

    ```
    exit
    ```

36. 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

37. 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.

38. 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.

39. Congratulations, this lab is complete.