Exercise 8

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

Part A. Starting Spark in EC2

1. 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**

2. 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:

```
services:
  spark:
    version: 2.2.0
    download-source: "http://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-
hadoop2.7.tgz"
  hdfs:
    version: 2.7.3

provider: ec2

providers:
  ec2:
    key-name: oxclo01
    identity-file: /home/oxclo/keys/oxclo01.pem
    instance-type: m3.large
    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  # terminate | stop

launch:
  num-slaves: 2
  install-hdfs: False
```

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.

3. Change the key name and identity file to match your key name and identity file.

4. Make sure **`install-hdfs: False`**

5. Make num-slaves: **2**

6. Save the file

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

   ```
   flintrock launch oxcloXX-sc
   ```

   (using your XX)

8. Now you should see something like (except with more lines):

   ```
   Launching 3 instances...
   [34.240.42.233] SSH online.
   [34.245.14.42] SSH online.
   [52.214.61.215] SSH online.
   [34.245.14.42] Configuring ephemeral storage...
   [34.240.42.233] Configuring ephemeral storage...
   [52.214.61.215] Configuring ephemeral storage...
   [34.240.42.233] Installing Java 1.8...
   [52.214.61.215] Installing Java 1.8...
   [34.245.14.42] Installing Java 1.8...
   [34.245.14.42] Installing Spark...
   [52.214.61.215] Installing Spark...
   [34.240.42.233] Installing Spark...
   [52.214.61.215] Configuring Spark master...
   Spark online.
   launch finished in 0:04:15.
   Cluster master: ec2-52-214-61-215.eu-west-1.compute.amazonaws.com
   Login with: flintrock login oxclo01-sc
   ```

   If you have issues you can try:
   `flintrock --debug launch oxcloXX-sc`

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 but this time from the flintrock SSH session.

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 org.apache.hadoop:hadoop-aws:2.7.4
```

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

```
        org.xerial.snappy#snappy-java;1.0.4.1 from central in [default]
        xmlenc#xmlenc;0.52 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   71  |   71  |   71  |   0   ||   70  |   70  |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-4b5025a3-23bc-4a5f-9723-5c46c3556b4e
        confs: [default]
        70 artifacts copied, 0 already retrieved (36395kB/100ms)
2018-07-19 16:16:02 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

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
sqlc = SQLContext(sc)
```

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

```
df = sqlc.read.csv('s3a://oxclo-
wind/2015/*',header='true', inferSchema='true')
```

    Spark should go away and think a bit, and also show some ephemeral log lines about the staging. Ignore the warning:
    `ObjectStore:568 - Failed to get database default, returning NoSuchObjectException`

17. 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 as you will find out shortly)

18. 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!)

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

```
winds = df.rdd
```

20. 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))
```

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

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

22. And once again collect / print:

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

    Because python uses indentation, it can't tell if this is the end of the statement so you will see:
    ...

    Press Enter.

23. 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
```

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

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

25. You can also try:

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

PART B – Getting Jupyter running with Flintrock

26. Quit the pyspark REPL (Ctrl-D) and get back to the ec2 command line

27. Type the following commands to install and run jupyter into your master node:

```
sudo yum install gcc gcc-c++ -y
sudo pip install jupyter
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser'
#make sure the next part is on one line
pyspark --master spark://0.0.0.0:7077
     --packages  org.apache.hadoop:hadoop-aws:2.7.4
```

28. You will see something like:

```
[I 21:20:38.933 NotebookApp] Serving notebooks from local directory:
/home/ec2-user
[I 21:20:38.934 NotebookApp] The Jupyter Notebook is running at:
[I 21:20:38.934 NotebookApp]
http://localhost:8888/?token=71c8d14cbf639b2c047e1e456a331b6b0e1d64f986c
80370
[I 21:20:38.934 NotebookApp] Use Control-C to stop this server and shut
```

29. Don't try to access that URL just yet. That is a URL that is only accessible from within the master node running on EC2 at the moment.

30. To allow us to access that URL, we need to setup an SSH tunnel to the master node.

    Start a new Ubuntu terminal window.

    Find the name of the master node once again:

    ```
    flintrock describe oxcloXX-sc
    ```

    ```
    state: running
    node-count: 3
    master: ec2-34-244-248-67.eu-west-1.compute.amazonaws.com
    slaves:
      - ec2-34-240-88-3.eu-west-1.compute.amazonaws.com
      - ec2-34-247-53-166.eu-west-1.compute.amazonaws.com
    ```

    Now start ssh thus (all one line, and replace the hostname)

    ```
    ssh -i ~/keys/oxcloXX.pem -4 -fN -L 8888:localhost:8888
    ec2-user@ec2-34-244-248-67.eu-west-1.compute.amazonaws.com
    ```

31. Now we can open that URL in the other window. You are now accessing the Jupyter server running in EC2. Now you can use the Jupyter model as before.

    PART C - SQL

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

33. We need to recreate the DataFrame first, so run this in a cell:

```
from pyspark.sql import SQLContext
sqlc = SQLContext(sc)
df = sqlc.read.csv('s3a://oxclo-wind/2015/*',header='true', inferSchema='true')
df.show(4)
```

34. Now we need to give our DataFrame a table name:
    `df.registerTempTable('wind')`

35. Now we can use a simple SQL statement against our data.
    ALL ON ONE Line type:

    ```
    sqlc.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()
    ```

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

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

37. 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.

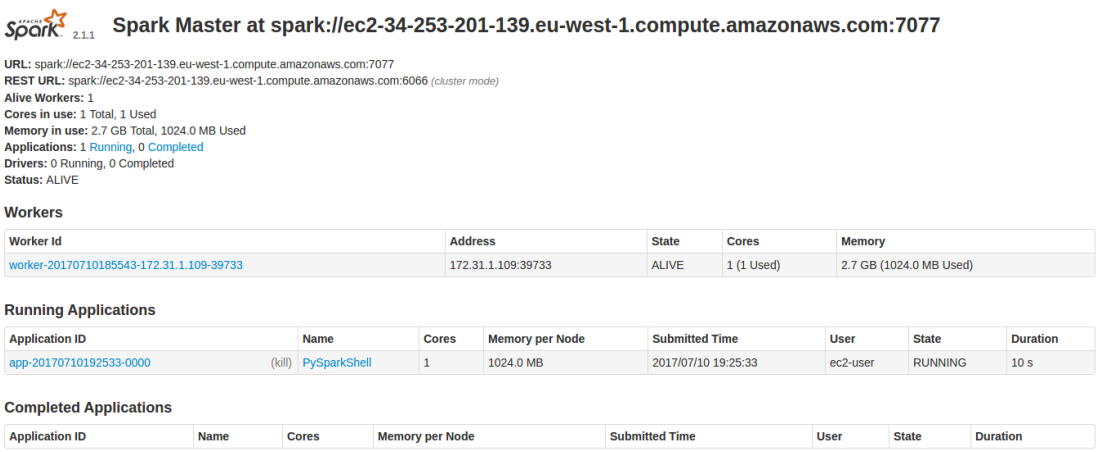
38. Find the IP address of the Spark Master: in your Ubuntu start a new terminal and type:

```
flintrock describe oxclo01-sc
```

You should see something like:

```
oxclo01-sc:
  state: running
  node-count: 3
  master: ec2-52-214-61-215.eu-west-1.compute.amazonaws.com
  slaves:
    - ec2-34-240-42-233.eu-west-1.compute.amazonaws.com
    - ec2-34-245-14-42.eu-west-1.compute.amazonaws.com
```

39. Go to e.g.
http://ec2-52-214-61-215.eu-west-1.compute.amazonaws.com :8080
using the master's DNS address (not the one in this text)
You should see something like:



40. 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-slaves --num-slaves 1 oxcloXX-sc
```

If you need it the code is here:
https://freo.me/wind-sql

41. We must remember to stop our cluster as well (its costing money…)
    From Ubuntu terminal

    ```
    flintrock destroy oxcloXX-sc
    ```

    Type y when prompted.

42. Congratulations, this lab is complete.