Exercise 7

More Apache Spark and Python,
Running Spark in EC2 with Flintrock

Prior Knowledge
Unix Command Line Shell
Simple Python

Learning Objectives
Using Spark on EC2
Accessing S3 files on Spark
Reading CSV files 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. There is also a built in support for Spark on EC2 (Amazon EMR). Feel free to explore it later. However, for the moment we will use a tool called **flintrock** to instantiate our own Spark cluster in EC2.

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:

   ```
   code ~/.config/flintrock/config.yaml
   ```


It will look something like:

```
services:
  spark:
    version: 3.0.0
  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-0ea3405d2d2522162   # Amazon Linux 2, eu-west-1
    user: ec2-user
    instance-profile-name: ec2-access-s3

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

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

This is modified from the original 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 sure `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:

   ```
   Warning: Downloading Spark from an Apache mirror. Apache mirrors are often slow and
   unreliable, and typically only serve the most recent releases. We strongly recommend
   you specify a custom download source. For more background on this issue, please see:
   https://github.com/nchammas/flintrock/issues/238
   Launching 3 instances...
   [34.253.234.105] SSH online.
   [34.253.234.105] Configuring ephemeral storage...
   [52.51.185.103] SSH online.
   [52.212.199.209] SSH online.
   [52.51.185.103] Configuring ephemeral storage...
   [52.212.199.209] Configuring ephemeral storage...
   [34.253.234.105] Installing Java 1.8...
   [52.51.185.103] Installing Java 1.8...
   [52.212.199.209] Installing Java 1.8...
   [52.51.185.103] Installing Spark...
   [34.253.234.105] Installing Spark...
   [52.212.199.209] Installing Spark...
   [52.212.199.209] Configuring Spark master...
   Spark online.
   launch finished in 0:02:33.
   Cluster master: ec2-52-212-199-209.eu-west-1.compute.amazonaws.com
   Login with: flintrock login oxclo01-sc
   ```

   Ignore the Apache mirror warning.

   If you have issues you can try:

   ```
   flintrock --debug launch oxcloXX-sc
   ```

9. By default, the Amazon AMI that we are using has Python2 installed. We would prefer to stick to Python3[1], so let's install Python3 on all the cluster members: (all on one line and including single quotes)

   ```
   flintrock run-command oxcloXX-sc
         'sudo yum install -y python3'
   ```

10. Let's login to the master (all one line):

    ```
    flintrock login oxcloXX-sc
    ```

    You see something like:

    ```
    arning: Permanently added '34.248.141.115' (ECDSA) to the list of known hosts.
    Last login: Mon Jun 22 12:27:50 2020 from
    host86-163-185-226.range86-163.btcentralplus.com

           __|  __|_  )
           _|  (     /   Amazon Linux 2 AMI
          ___|\___|___|

    https://aws.amazon.com/amazon-linux-2/
    4 package(s) needed for security, out of 10 available
    Run "sudo yum update" to apply all updates.
    ```

[1] Actually we'd prefer to stick to Python2 because of the better syntax for lambda expressions that take tuples as a parameter! However, that isn't an option since Python2 is out of support.

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

12. Now we need to tell Spark to use Python 3:
```
export PYSPARK_PYTHON=/usr/bin/python3
```

13. Now we will 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
```

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

```
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   72  |   72  |   72  |   0   ||   72  |   72  |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-f12270ce-d330-44bc-a975-f95b9b86e2e2
        confs: [default]
        72 artifacts copied, 0 already retrieved (36764kB/144ms)
20/06/22 12:30:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
 using builtin-java classes where applicable
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).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Python version 3.7.6 (default, Feb 26 2020 20:54:15)
SparkSession available as 'spark'.
>>>
```

15. It is perfectly possible to get Jupyter to talk to Spark on our cluster, but it is slightly complex. We will do that later. We will just use the normal Python command-line for the moment.

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

17. 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)
```


18. 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.

19. 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). It is similar to the Pandas dataframe (and convertible into one: https://docs.databricks.com/spark/latest/spark-sql/spark-pandas.html)

20. 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!)

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

```
winds = df.rdd
```

22. 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))
```

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

```
maxes = mapped.reduceByKey(lambda a, b: a if (b==None or a>b) else b)
```

24. 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.

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 – Getting Jupyter running with Flintrock

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

28. Type the following commands to install and run jupyter into your master node (available here: https://freo.me/flintrock-j)

```
sudo yum install gcc gcc-c++ -y
sudo pip3 install jupyter
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser'
pyspark --master spark://0.0.0.0:7077 \
     --packages  org.apache.hadoop:hadoop-aws:2.7.4
```

29. 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=71c8d14cbf639b2c047e1e456a331b6b0e1d64f986c80370
[I 21:20:38.934 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[W 21:20:38.934 NotebookApp] No web browser found: could not locate runnable browser.
[C 21:20:38.935 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=71c8d14cbf639b2c047e1e456a331b6b0e1d64f986c80370
```

30. 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.

31. 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
    ```

    Explanation

    ```
    -4  - use IPv4 only
    -fN - go into the background and don't execute any
          remote command
    -L xxxx:localhost:yyyy
                port forward from xxxx on the local
                server to yyyy on the remote server
    ```

32. 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.

33. *Note that any python code you save here will be stored on the AWS instance and deleted when you destroy the cluster!*

    PART C - SQL

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

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

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


36. Now we need to give our DataFrame a table name:

```
df.createOrReplaceTempView('wind')
```

37. Now we can use a simple SQL statement against our data.

```
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()
```

38. Bingo you should see:

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

39. There is a reference to SparkSQL syntax here:

    https://spark.apache.org/docs/3.0.0/sql-ref.html

40. We can also use a different approach to SQL that doesn't need us to give the table a name and use "SQL in Quotes"

```
from pyspark.sql.functions import max, mean, col

df.groupBy('Station_ID').\
      agg(mean(col('Wind_Velocity_Mtr_Sec')),\
      max(col('Wind_Velocity_Mtr_Sec'))).show()
```

41. I also use a lot of DF->RDD->DF all on one line like this:

```
cleanDF = df.rdd.map(lambda row: \
      Row(station = row.Station_ID, \
          wind = row.Wind_Velocity_Mtr_Sec)).toDF()
cleanDF.show()
```

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

42. Recap. So far 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.

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

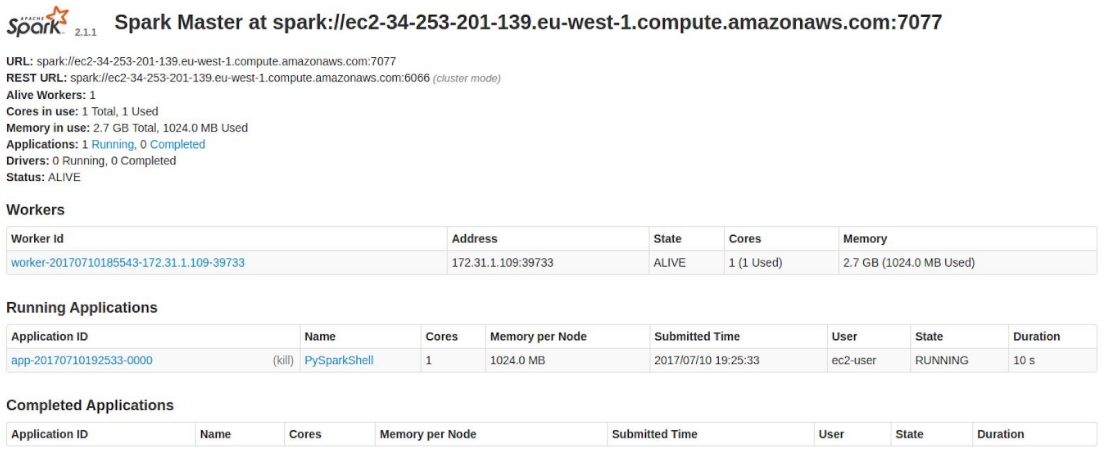
```
flintrock describe oxcloXX-sc
```

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
```

Go to the master's page:
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:



The same DNS name on port 4040 is also accessible - check it out

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

```
flintrock destroy oxcloXX-sc
```

Type y when prompted.

45. Congratulations, this lab is complete.