Setting up Spark + EMR Cluster

This document will be an introduction to Apache Spark and AWS EMR (Elastic MapReduce). We'll see how to submit a job, via Jupyter Notebook, to Spark running on an AWS EMR cluster and output the results to S3. We'll also see how to view the results of our data analysis directly on Jupyter. This tutorial is based on this EMR tutorial from the Amazon docs.

First, let's create a new EMR Cluster

To create an EMR Cluster, follow this guide. Basically, to make your EMR Cluster, run this command on your machine:

```
aws emr create-cluster --name "v5.29withHive" --release-label emr-5.29.0 --applications Na
me=Hadoop Name=Livy Name=Spark Name=Hive Name=Ganglia Name=Zeppelin \
 --ec2-attributes SubnetId=SUBNET_ID --instance-type m5.xlarge --instance-count 1 --use-de
fault-roles
```

There's a couple things to note here:

- Make sure you have AWS CLI
- Make sure to replace SUBNET_ID with the id of a default Subnet. You can find this in the AWS console by going to VPC > Subnets. When you click on a default subnet, the "Default Subnet" field should say "yes".
- We create the cluster using CLI (as opposed to using the EMR console) because you want to control which applications are downloaded on the cluster. Specifically, we want to download Spark, Livy, Hive, Ganglia, and Zeppelin. The two important ones here are Spark and Hive. You need Hive if you want to run .sql() on your Spark dataframe in Jupyter Notebook later.
- We're using an older EMR version, 5.29.0. This version has default a security group that allows all incoming traffic, including for the EMR notebook we'll be making next. This is no longer recommended, but easier. If you want to use a

more up to date version, follow the steps here to setup the necessary inbound rules for your security group.

Now, wait until the status of your cluster is "Waiting". Great! You've created an EMR Cluster with Spark downloaded on it.

Create your EMR Notebook

To create an EMR Notebook, follow this guide. Follow this guide exactly—for step 4, choose the cluster we just made.

Prepare Storage (S3) for Cluster Input and Output

Step 1 from Amazon's EMR tutorial shows you how to create an S3 bucket, and upload the CSV input data file. We'll be doing computation on the same food_establishment_data.csv

You do not have to complete the "To prepare the example PySpark script for EMR" section, since we'll be running a Python script manually in Jupyter notebook.

Run some PySpark on Jupyter Notebook to Output to S3

Go to the notebook details page of the EMR notebook we just created. Click "Open in Jupyter"

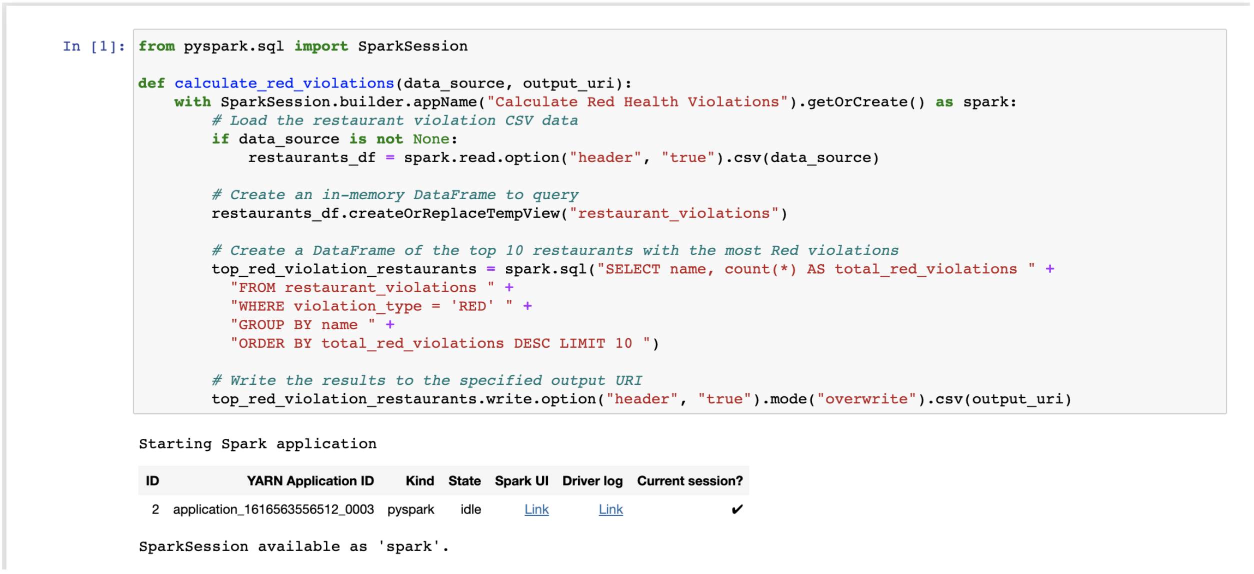
Click on the .ipynb file

Click on the Kernel tab > Change kernel > PySpark

Copy/Paste the PySpark script provided by the EMR tutorial. This script reads the CSV we uploaded to S3, then gets the top 10 restaurants with the most RED violation types:

```
from pyspark.sql import SparkSession

def calculate_red_violations(data_source, output_uri):
    with SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate() as
 spark:
        # Load the restaurant violation CSV data
        if data_source is not None:
            restaurants_df = spark.read.option("header", "true").csv(data_source)

        # Create an in-memory DataFrame to query
        restaurants_df.createOrReplaceTempView("restaurant_violations")

        # Create a DataFrame of the top 10 restaurants with the most Red violations
        top_red_violation_restaurants = spark.sql("SELECT name, count(*) AS total_red_viol
ations " +
          "FROM restaurant_violations " +
          "WHERE violation_type = 'RED' " +
          "GROUP BY name " +
          "ORDER BY total_red_violations DESC LIMIT 10 ")

        # Write the results to the specified output URI
        top_red_violation_restaurants.write.option("header", "true").mode("overwrite").csv
(output_uri)
```

```
# Call manually

calculate_red_violations('s3://BUCKET_NAME/food_establishment_data.csv', 's3://BUCKET_NAM
E/myOutputFolder')

# The first argument points to the CSV file you just uploaded.
# The second points to a folder "myOutputFolder" that will be created in your bucket.
# The output of your Spark job will be stored there.
```

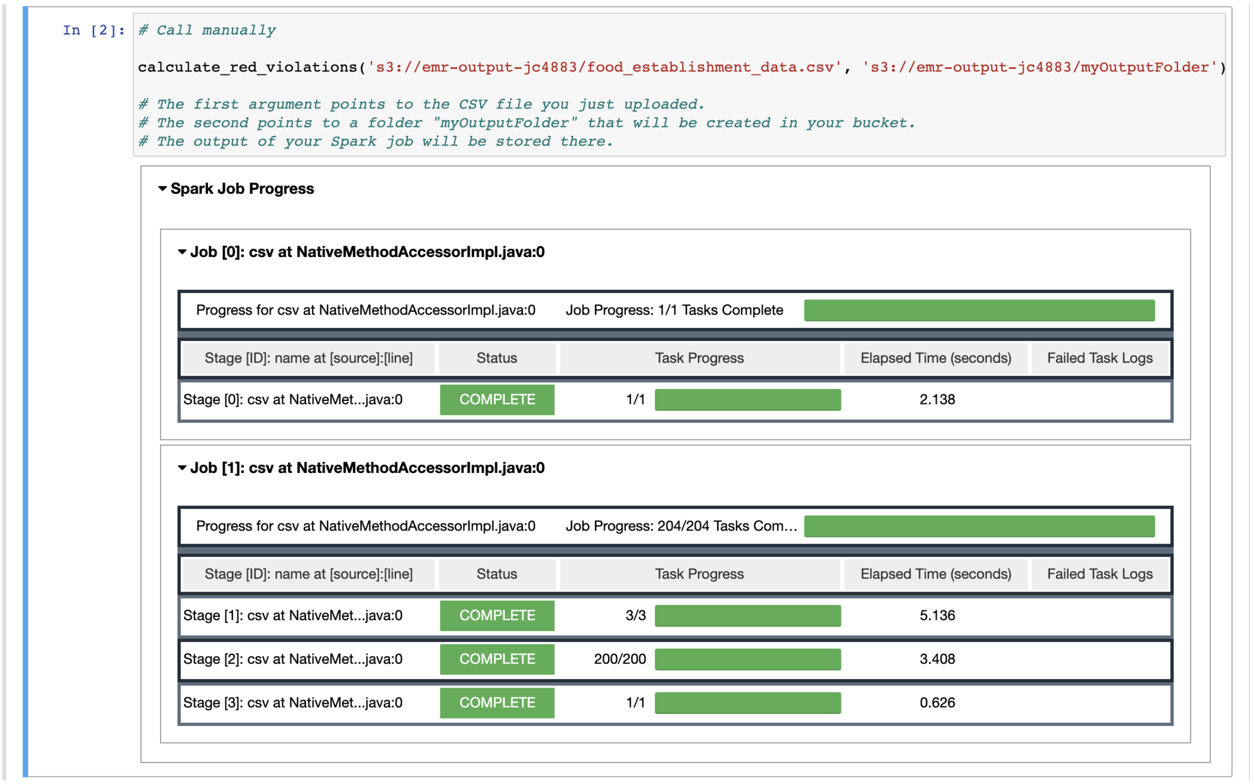


```
In [1]: from pyspark.sql import SparkSession

        def calculate_red_violations(data_source, output_uri):
            with SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate() as spark:
                # Load the restaurant violation CSV data
                if data_source is not None:
                    restaurants_df = spark.read.option("header", "true").csv(data_source)

                # Create an in-memory DataFrame to query
                restaurants_df.createOrReplaceTempView("restaurant_violations")

                # Create a DataFrame of the top 10 restaurants with the most Red violations
                top_red_violation_restaurants = spark.sql("SELECT name, count(*) AS total_red_violations " +
                  "FROM restaurant_violations " +
                  "WHERE violation_type = 'RED' " +
                  "GROUP BY name " +
                  "ORDER BY total_red_violations DESC LIMIT 10 ")

                # Write the results to the specified output URI
                top_red_violation_restaurants.write.option("header", "true").mode("overwrite").csv(output_uri)
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
2	application_1616563556512_0003	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.
```



```
In [2]: # Call manually

        calculate_red_violations('s3://emr-output-jc4883/food_establishment_data.csv', 's3://emr-output-jc4883/myOutputFolder')

        # The first argument points to the CSV file you just uploaded.
        # The second points to a folder "myOutputFolder" that will be created in your bucket.
        # The output of your Spark job will be stored there.
```

Spark Job Progress

Job [0]: csv at NativeMethodAccessorImpl.java:0

Progress for csv at NativeMethodAccessorImpl.java:0 Job Progress: 1/1 Tasks Complete

Stage [ID]: name at [source]:[line]	Status	Task Progress	Elapsed Time (seconds)	Failed Task Logs
Stage [0]: csv at NativeMet...java:0	COMPLETE	1/1	2.138	

Job [1]: csv at NativeMethodAccessorImpl.java:0

Progress for csv at NativeMethodAccessorImpl.java:0 Job Progress: 204/204 Tasks Com…

Stage [ID]: name at [source]:[line]	Status	Task Progress	Elapsed Time (seconds)	Failed Task Logs
Stage [1]: csv at NativeMet...java:0	COMPLETE	3/3	5.136	
Stage [2]: csv at NativeMet...java:0	COMPLETE	200/200	3.408	
Stage [3]: csv at NativeMet...java:0	COMPLETE	1/1	0.626	

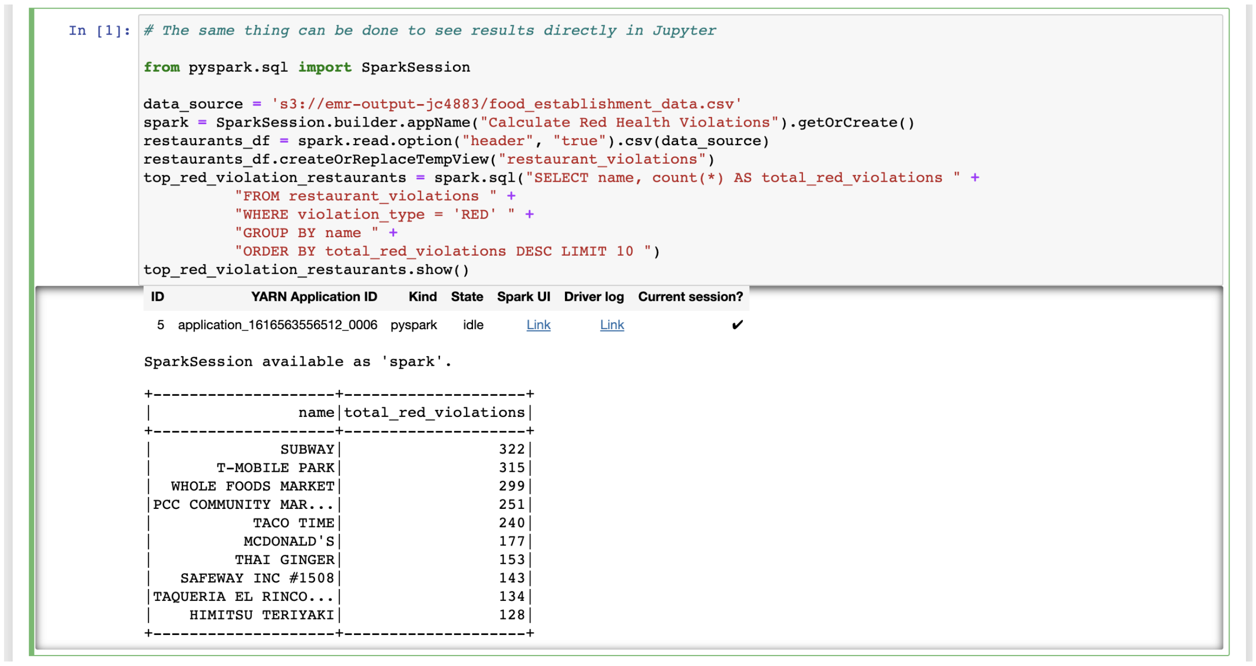
After you run these two cells, a file should be outputted into your S3 bucket with the result of the Spark job.

Congratulations! You have now used Spark to perform some computation on a large dataset.

See Results in Jupyter Notebook Directly

Also, you'll notice that you don't have to output a file to S3 at all—you can see results directly in Jupyter Notebook!

```
# The same thing can be done to see results directly in Jupyter

from pyspark.sql import SparkSession

data_source = 's3://emr-output-jc4883/food_establishment_data.csv'
spark = SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate()
restaurants_df = spark.read.option("header", "true").csv(data_source)
restaurants_df.createOrReplaceTempView("restaurant_violations")
top_red_violation_restaurants = spark.sql("SELECT name, count(*) AS total_red_violations " 
+
          "FROM restaurant_violations " +
          "WHERE violation_type = 'RED' " +
          "GROUP BY name " +
          "ORDER BY total_red_violations DESC LIMIT 10 ")
top_red_violation_restaurants.show()
```


In [1]:

```
# The same thing can be done to see results directly in Jupyter

from pyspark.sql import SparkSession

data_source = 's3://emr-output-jc4883/food_establishment_data.csv'
spark = SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate()
restaurants_df = spark.read.option("header", "true").csv(data_source)
restaurants_df.createOrReplaceTempView("restaurant_violations")
top_red_violation_restaurants = spark.sql("SELECT name, count(*) AS total_red_violations " +
          "FROM restaurant_violations " +
          "WHERE violation_type = 'RED' " +
          "GROUP BY name " +
          "ORDER BY total_red_violations DESC LIMIT 10 ")
top_red_violation_restaurants.show()
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
5	application_1616563556512_0006	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.

+--------------------+--------------------+
|                name|total_red_violations|
+--------------------+--------------------+
|              SUBWAY|                 322|
|         T-MOBILE PARK|                 315|
|  WHOLE FOODS MARKET|                 299|
|PCC COMMUNITY MAR...|                 251|
|           TACO TIME|                 240|
|          MCDONALD'S|                 177|
|         THAI GINGER|                 153|
|    SAFEWAY INC #1508|                 143|
|TAQUERIA EL RINCO...|                 134|
|     HIMITSU TERIYAKI|                 128|
+--------------------+--------------------+
```