Lab 4
SQL in Spark on AWS

Big Data

February 26, 2018

Amazon Web Services

In this lab we will learn how to use Amazon’s Elastic MapReduce to run MapReduce jobs. First, log in to the AWS Management Console here:

http://console.aws.amazon.com

Step 1: Create an S3 Bucket

Amazon S3 is a web service that provides storage. We will use S3 to store our code, input data, and output files for mapreduce jobs that we run using AWS.

1. In the AWS Management console, under Services/Storage, click S3 .
2. Click Create Bucket.
3. In the Bucket Name field, type emrbucket- followed by your initials and the date (for example, emrbucket-ecc022618). You don’t have to stick to this naming scheme, you could name your bucket whatever you want (but it has to be unique, lower case only, and can’t contain spaces, underscores, or periods).
4. Leave the default under Region and click Next.
5. Click Next again.
6. Under Manage public permissions, select: Grant public read access to this bucket
7. Click Next.
8. Click Create bucket.

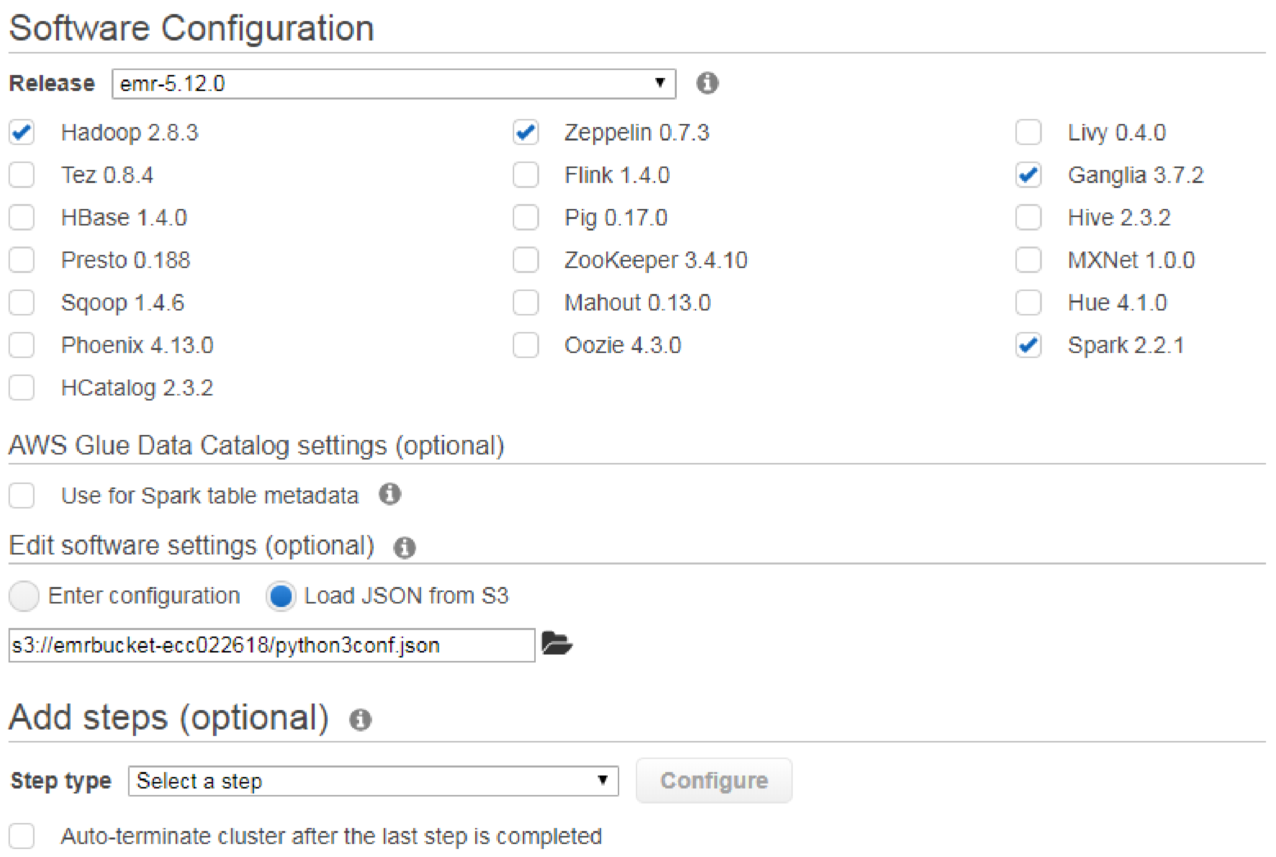
Step 2: Launch an EMR cluster

The next step is to create and launch the EMR cluster. EMR provisions Amazon EC2 instances (virtual servers) to run jobs. These EC2 instances are preloaded with Hadoop and other needed libraries.

1. Return to the AWS Management console.
2. Under Services/Analytics, click EMR .
3. Click Create Cluster.
4. At the top, click "Go to Advanced options"

Setting EMR options

- Select Hadoop, Zeppelin, Ganglia, and Spark under Software.
- Under "Edit software settings", select "Load JSON from S3" and enter "s3://emrbucket-ecc022618/python3conf.json" in the box.
- Click Next, and then Next again



Software Configuration

Release emr-5.12.0

✓ Hadoop 2.8.3	✓ Zeppelin 0.7.3	Livy 0.4.0
Tez 0.8.4	Flink 1.4.0	✓ Ganglia 3.7.2
HBase 1.4.0	Pig 0.17.0	Hive 2.3.2
Presto 0.188	ZooKeeper 3.4.10	MXNet 1.0.0
Sqoop 1.4.6	Mahout 0.13.0	Hue 4.1.0
Phoenix 4.13.0	Oozie 4.3.0	✓ Spark 2.2.1
HCatalog 2.3.2		

AWS Glue Data Catalog settings (optional)

Use for Spark table metadata

Edit software settings (optional)

Enter configuration [✓ Load JSON from S3]

s3://emrbucket-ecc022618/python3conf.json

Add steps (optional)

Step type Select a step

Auto-terminate cluster after the last step is completed

Setting EMR options

- Under General Options / Logging, enter in the S3 folder box:

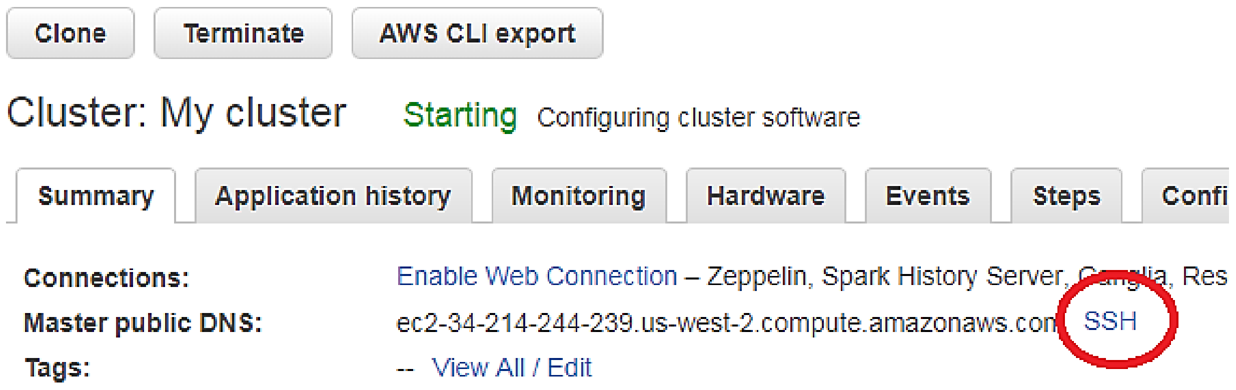
  s3://<your-bucket-name>/logs/

- Click Next
- Under Security Options / EC2 keypair, click the dropdown menu and select the key pair you created in Part 3 of Lab 1.
- Click Create Cluster

Your cluster is now being started. This will take around 10 minutes until the cluster is up and running and available for computation.

Step 3: SSH into the master node

- After a few minutes, an address will appear next to "Master public DNS" in the EMR console. Click the "SSH" to see a popup that gives you instructions on how to SSH to the master node.



Cluster: My cluster Starting Configuring cluster software

Connections: Enable Web Connection – Zeppelin, Spark History Server, Ganglia, Res
Master public DNS: ec2-34-214-244-239.us-west-2.compute.amazonaws.com SSH
Tags: -- View All / Edit

- Follow the instructions (note there are tabs for both Windows and Mac/Linux) to SSH to the master node.
- Wait until the cluster finishes starting up. You can test this by typing "which pyspark". If "/usr/bin/pyspark" is returned, you are ready to run pyspark.
- Type "pyspark" to start pyspark.

Continuing Last Week's Lab

To create a basic `SparkSession`, use `SparkSession.builder`:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Review: 3 tables; sailors, boats, reserves

Table 1: boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Table 2: reserves

sid	bid	day
22	101	10-OCT-18
22	102	10-OCT-18
22	103	10-AUG-18
22	104	10-JUL-18
31	102	10-NOV-18
64	101	05-SEP-18
64	102	08-SEP-18
74	103	08-SEP-18
31	103	06-NOV-18
31	104	12-NOV-18

Table 3: sailors

sid	sname	rating	age
22	dusting	7	45
29	brutus	1	33
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35
64	horatio	7	16
71	zorba	10	35
74	horatio	9	25.5
85	art	3	25.5
95	bob	3	63.5

Loading the tables

- The JSON files for the tables are stored in *my* public S3 bucket. You will access them from there.

- Type:

```
sailors = spark.read.json("s3://emrbucket-ecc022618/sailors.json")
boats = spark.read.json("s3://emrbucket-ecc022618/boats.json")
reserves = spark.read.json("s3://emrbucket-ecc022618/reserves.json")
```

- Check that the tables are loaded. You can view the DataFrames by typing

```
  sailors.show()
  reserves.show()
  boats.show()
```

Creating SQL temp view from DF

- Register the DataFrames as SQL temporary views. Type:

```
sailors.createOrReplaceTempView("sailors")
reserves.createOrReplaceTempView("reserves")
boats.createOrReplaceTempView("boats")
```

- You can then use spark.sql to run SQL queries. For example, type:

```
spark.sql("SELECT * FROM sailors").show()
```

Review: Writing SQL Queries

- Find the names and ages of all sailors

Review: Writing SQL Queries

- Find the names and ages of all sailors

```
spark.sql("SELECT sname, age FROM sailors").show()
```

Continuing our SQL Queries

- We will now finish the SQL queries from last week's class

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

```
spark.sql("SELECT sid FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

```
spark.sql("SELECT sid FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

This contains duplicates. To remove duplicates, use DISTINCT keyword:

```
spark.sql("SELECT distinct(sid) FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one incorrect query:

```
spark.sql("SELECT sname FROM sailors S, reserves R, boats B \
   WHERE S.sid = R.sid \
    AND R.bid = B.bid \
    AND (color like '%red%' AND color like '%green%')").show()
```

What happens?

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one incorrect query:

```
spark.sql("SELECT sname FROM sailors S, reserves R, boats B \
   WHERE S.sid = R.sid \
    AND R.bid = B.bid \
    AND (color like '%red%' AND color like '%green%')").show()
```

What happens?

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Another mistake: a sailor named Horatio has reserved a red boat, and a *different* sailor named Horatio has reserved a green boat – make sure to write your query such that Horatio is not returned as a sailor that has reserved both a red and green boat!

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Another mistake: a sailor named Horatio has reserved a red boat, and a *different* sailor named Horatio has reserved a green boat – make sure to write your query such that Horatio is not returned as a sailor that has reserved both a red and green boat!

$\longrightarrow$ Need to use SIDs rather than name since these are the primary key in the sailors table (i.e., they are unique)

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one example of a correct query:

```
spark.sql("SELECT DISTINCT (S.sname) \
    FROM sailors S, boats B, reserves R \
    WHERE S.sid = R.sid AND R.bid = B.bid AND B.color like '%red%' \
    AND S.sid in \
    (SELECT S2.sid FROM sailors S2, boats B2, reserves R2 \
    WHERE S2.sid = R2.sid AND R2.bid = B2.bid and B2.color like
'%green%')").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have not reserved boat number 103.

Practice Writing SQL Queries

- Find the names of sailors who have not reserved boat number 103.

```
spark.sql("SELECT sname FROM sailors S \
       WHERE S.sid NOT IN \
       (SELECT sid FROM Reserves WHERE bid = 103)").show()
```

Practice Writing SQL Queries

- Find the names of sailors whose rating is better than some sailor called Horatio.

Practice Writing SQL Queries

- Find the names of sailors whose rating is better than some sailor called Horatio.

```
spark.sql("SELECT S1.sname FROM sailors S1 \
       WHERE S1.rating > (SELECT MIN(S2.rating) FROM sailors S2 \
              WHERE S2.sname like '%horatio%')").show()
```

Practice Writing SQL Queries

- Find the average age of sailors with a rating of 10.

Practice Writing SQL Queries

- Find the average age of sailors with a rating of 10.

```
spark.sql("SELECT AVG(age) FROM Sailors WHERE rating = 10").show()
```

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

What happens if we write the query:

```
spark.sql("SELECT sname, MAX(age) FROM sailors").show()
```

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

A correct query:

```
spark.sql("SELECT S.sname, S.age FROM sailors S \
       WHERE S.age = (SELECT MAX(S2.age) FROM sailors S2)").show();
```

Practice Writing SQL Queries

- Find the average age of sailors with each rating, ordered by increasing rating

Practice Writing SQL Queries

- Find the average age of sailors with each rating, ordered by increasing rating

```
spark.sql("SELECT rating, AVG(age) FROM sailors \
        GROUP BY rating ORDER BY rating").show()
```

Saving a table to S3

```
spark.sql("SELECT rating, AVG(age) FROM sailors \
    GROUP BY rating ORDER BY rating")\
    .coalesce(1)\
    .write.save("s3://<your-bucket-name>/averageage.csv",format="csv")
```

Saving a table to S3

```
spark.sql("SELECT rating, AVG(age) FROM sailors \
    GROUP BY rating ORDER BY rating")\
    .coalesce(1)\
    .write.save("s3://<your-bucket-name>/averageage.csv",format="csv")
```

coalesce(1) will write the output to a single file (instead of multiple parts).
If you expect your output will be large, remove this line - you DO NOT want to write to a single file.

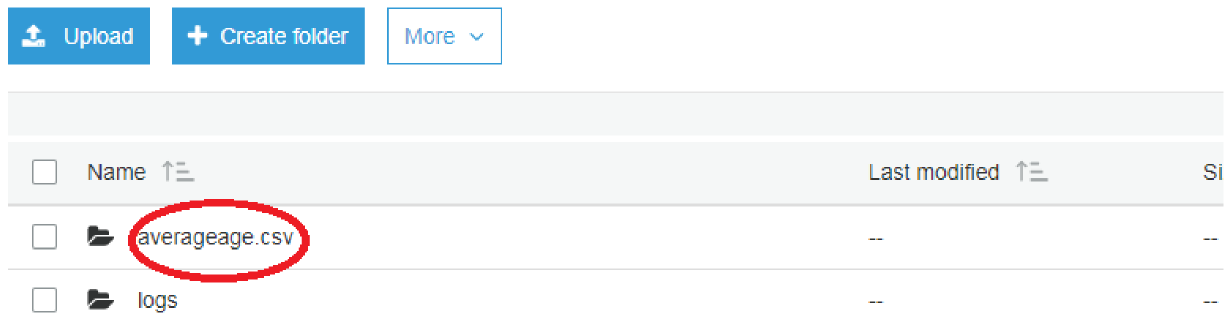
Saving a table to S3

```
spark.sql("SELECT rating, AVG(age) FROM sailors \
    GROUP BY rating ORDER BY rating")\
    .coalesce(1)\
    .write.save("s3://<your-bucket-name>/averageage.csv",format="csv")
```

coalesce(1) will write the output to a single file (instead of multiple parts).
If you expect your output will be large, remove this line - you DO NOT want to write to a single file.

Saving a table to S3

- Return to the S3 console in your browser: https://s3.console.aws.amazon.com/s3
- Click on the bucket you created for this lab. You should see the file "averageage.csv". Click on the name "averageage.csv"



Name	Last modified	Si
averageage.csv	--	--
logs	--	--

- Select the checkbox next to "part-00000...."
- In the popup window, click Download
- The file is now downloaded to your local machine.

Deliverable

(due Monday, March 5, 2018, 6pm):

- **Submit your "part-00000..." file generated during the lab to NYU Classes**

REMEMBER TO TERMINATE YOUR CLUSTER WHEN YOU ARE FINISHED! (see following slide)

TERMINATE CLUSTER!

It is very important that you terminate your cluster through the EMR console once you are finished working.

Otherwise, it will keep running and your account will be charged!

- Go to the EMR console.
- Select the box to the left of your cluster and click the Terminate button.
- Click "Change" next to Termination protection: On
- Click the "Off" button and then click the green checkmark
- Click the "Terminate" button