EMR Workshop Lab 3 – Spark-based ETL on EMR

(Updated 14-November-18)

This lab demonstrates submitting and monitoring Spark-based ETL work to an Amazon EMR cluster.

You can submit Spark job to your cluster interactively, or you can submit work as a EMR step using the console, CLI, or API. You can submit steps when the cluster is launched, or you can submit steps to a running cluster.

Job Description:

This sample ETL job does the following things:

- Read CSV data from Amazon S3
- Add current date to the dataset
- Write updated data back to Amazon S3 in Parquet format

Exercise 1: Running Spark job on Amazon EMR

- Create an S3 bucket with folders:
    - files
    - logs
    - input
    - output
- Get sample data from here (1.8MB file): https://s3.amazonaws.com/aws-data-analytics-blog/emrimmersionday/tripdata.csv
- Upload file to your "input" folder in your S3 bucket
- SSH to master node of your previously created cluster.
- Copy and paste the following script into spark-etl.py, make sure that you don't have invisible characters. Use vi on mac/Linux or Notepad on windows. Alternatively, you can download this script from here and edit it:

```
import sys
from datetime import datetime

from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import *

if __name__ == "__main__":

    print(len(sys.argv))
    if (len(sys.argv) != 3):
        print("Usage: spark-etl [input-folder] [output-folder]")
        sys.exit(0)

    spark = SparkSession\
        .builder\
        .appName("SparkETL")\
        .getOrCreate()

    nyTaxi = spark.read.option("inferSchema",
"true").option("header", "true").csv(sys.argv[1])

    updatedNYTaxi = nyTaxi.withColumn("current_date",
lit(datetime.now()))
    updatedNYTaxi.printSchema()
    print(updatedNYTaxi.show())
    print("Total number of records: " +
str(updatedNYTaxi.count())
    updatedNYTaxi.write.parquet(sys.argv[2])
```

- Submit that pySpark "spark-etl.py" job on the cluster. This Spark job will query the NY taxi data from input location, add a new column "current_date" and write transformed data in the output location in Parquet format.

```
>> spark-submit spark-etl.py s3://<YOUR-BUCKET>/input/
s3://<YOUR-BUCKET>/output/spark
```

- Check the "output/spark" in 3 minutes.

Exercise 2: Monitoring Spark job on Amazon EMR

There are several ways to monitor Spark job status, logs on Amazon EMR. Those are:

- Check Spark job logs on the command line
- Check YARN Application logs on Amazon EMR Console
- Check status and logs on Spark UI

Test 1: Check Spark job logs on the command line:

When Spark job submitted through spark-submit on the command line, it shows up logs on the console. You can pipe that output to a file and grep that file to troubleshoot or you can check the status, output, debug printout on the terminal as well. For example, this job prints out input schema on the terminal:

```
18/08/02 19:01:13 INFO DAGScheduler: Missing parents: List()
18/08/02 19:01:13 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[8] at csv at NativeMethodAccessorImpl.java:0), which has no missing parent
18/08/02 19:01:13 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 13.0 KB, free 413.8 MB)
18/08/02 19:01:13 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 7.0 KB, free 413.8 MB)
18/08/02 19:01:13 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on ip-10-0-0-15.ec2.internal:46143 (size: 7.0 KB, free: 414.4 MB)
18/08/02 19:01:13 INFO SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:1079
18/08/02 19:01:13 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[8] at csv at NativeMethodAccessorImpl.java:0) (first
 Vector(0))
18/08/02 19:01:13 INFO YarnScheduler: Adding task set 1.0 with 1 tasks
18/08/02 19:01:13 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, ip-10-0-0-46.ec2.internal, executor 1, partition 0, RACK_LOCAL, 8326 bytes)
18/08/02 19:01:13 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on ip-10-0-0-46.ec2.internal:46165 (size: 7.0 KB, free: 2.8 GB)
18/08/02 19:01:15 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on ip-10-0-0-46.ec2.internal:46165 (size: 26.6 KB, free: 2.8 GB)
18/08/02 19:01:15 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 1821 ms on ip-10-0-0-46.ec2.internal (executor 1) (1/1)
18/08/02 19:01:15 INFO YarnScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
18/08/02 19:01:15 INFO DAGScheduler: ResultStage 1 (csv at NativeMethodAccessorImpl.java:0) finished in 1.830 s
18/08/02 19:01:15 INFO DAGScheduler: Job 1 finished: csv at NativeMethodAccessorImpl.java:0, took 1.834821 s
root
 |-- VendorID: integer (nullable = true)
 |-- lpep_pickup_datetime: string (nullable = true)
 |-- lpep_dropoff_datetime: string (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- RatecodeID: integer (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- ehail_fee: string (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- payment_type: integer (nullable = true)
 |-- trip_type: integer (nullable = true)
 |-- current_date: timestamp (nullable = false)
```

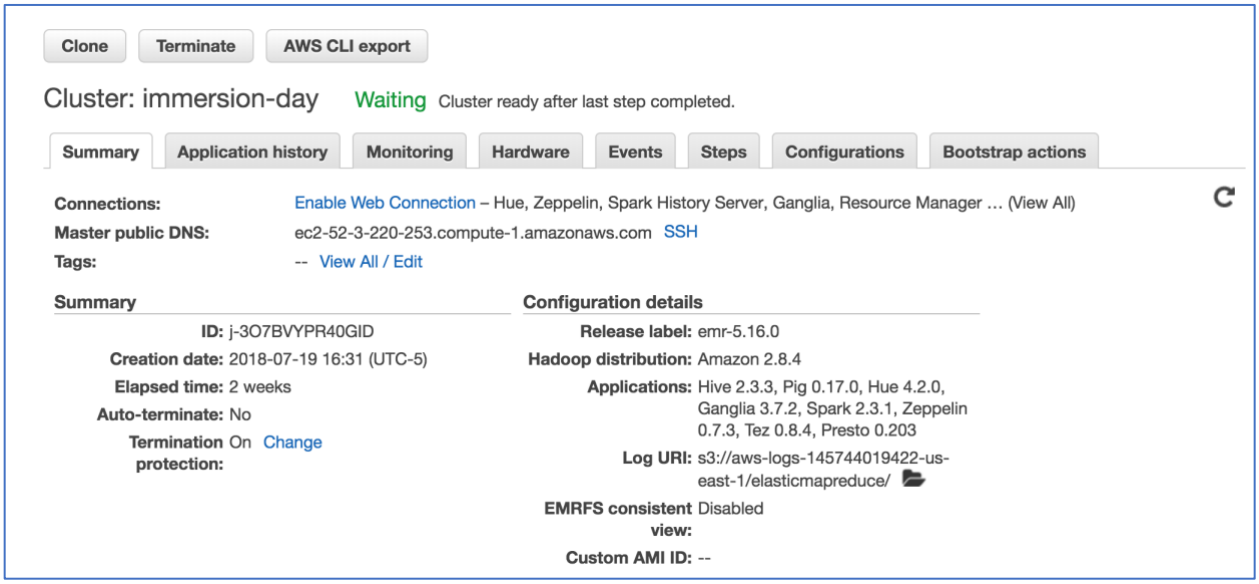
At the end of the job, it also prints out the total number of rows in NY Taxi data:

```
18/08/02 19:01:17 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on ip-10-0-0-15.ec2.internal:46143 (size: 3.8 KB, free: 414.4 MB)
18/08/02 19:01:17 INFO SparkContext: Created broadcast 8 from broadcast at DAGScheduler.scala:1079
18/08/02 19:01:17 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 4 (MapPartitionsRDD[18] at count at NativeMethodAccessorImpl.java:0) (first
ons Vector(0))
18/08/02 19:01:17 INFO YarnScheduler: Adding task set 4.0 with 1 tasks
18/08/02 19:01:17 INFO TaskSetManager: Starting task 0.0 in stage 4.0 (TID 4, ip-10-0-0-46.ec2.internal, executor 1, partition 0, NODE_LOCAL, 7765 bytes)
18/08/02 19:01:17 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on ip-10-0-0-46.ec2.internal:46165 (size: 3.8 KB, free: 2.8 GB)
18/08/02 19:01:17 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 10.0.0.46:57228
18/08/02 19:01:17 INFO TaskSetManager: Finished task 0.0 in stage 4.0 (TID 4) in 94 ms on ip-10-0-0-46.ec2.internal (executor 1) (1/1)
18/08/02 19:01:17 INFO YarnScheduler: Removed TaskSet 4.0, whose tasks have all completed, from pool
18/08/02 19:01:17 INFO DAGScheduler: ResultStage 4 (count at NativeMethodAccessorImpl.java:0) finished in 0.101 s
18/08/02 19:01:17 INFO DAGScheduler: Job 3 finished: count at NativeMethodAccessorImpl.java:0, took 0.434352 s
Total number of records: 20000
18/08/02 19:01:17 INFO FileSourceStrategy: Pruning directories with:
18/08/02 19:01:17 INFO FileSourceStrategy: Post-Scan Filters:
18/08/02 19:01:17 INFO FileSourceStrategy: Output Data Schema: struct<VendorID: int, lpep_pickup_datetime: string, lpep_dropoff_datetime: string, store_and_f
eID: int ... 17 more fields>
18/08/02 19:01:17 INFO FileSourceScanExec: Pushed Filters:
18/08/02 19:01:17 INFO ParquetFileFormat: Using default output committer for Parquet: org.apache.parquet.hadoop.ParquetOutputCommitter
18/08/02 19:01:17 INFO FileOutputCommitter: File Output Committer Algorithm version is 2
18/08/02 19:01:17 INFO FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: true
18/08/02 19:01:17 INFO SQLHadoopMapReduceCommitProtocol: Using user defined output committer class org.apache.parquet.hadoop.ParquetOutputCommitter
18/08/02 19:01:17 INFO FileOutputCommitter: File Output Committer Algorithm version is 2
18/08/02 19:01:17 INFO FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: true
18/08/02 19:01:17 INFO SQLHadoopMapReduceCommitProtocol: Using output committer class org.apache.parquet.hadoop.ParquetOutputCommitter
18/08/02 19:01:17 INFO CodeGenerator: Code generated in 28.320185 ms
18/08/02 19:01:17 INFO MemoryStore: Block broadcast_9 stored as values in memory (estimated size 311.5 KB, free 413.1 MB)
18/08/02 19:01:17 INFO ContextCleaner: Cleaned accumulator 131
18/08/02 19:01:17 INFO ContextCleaner: Cleaned accumulator 103
```

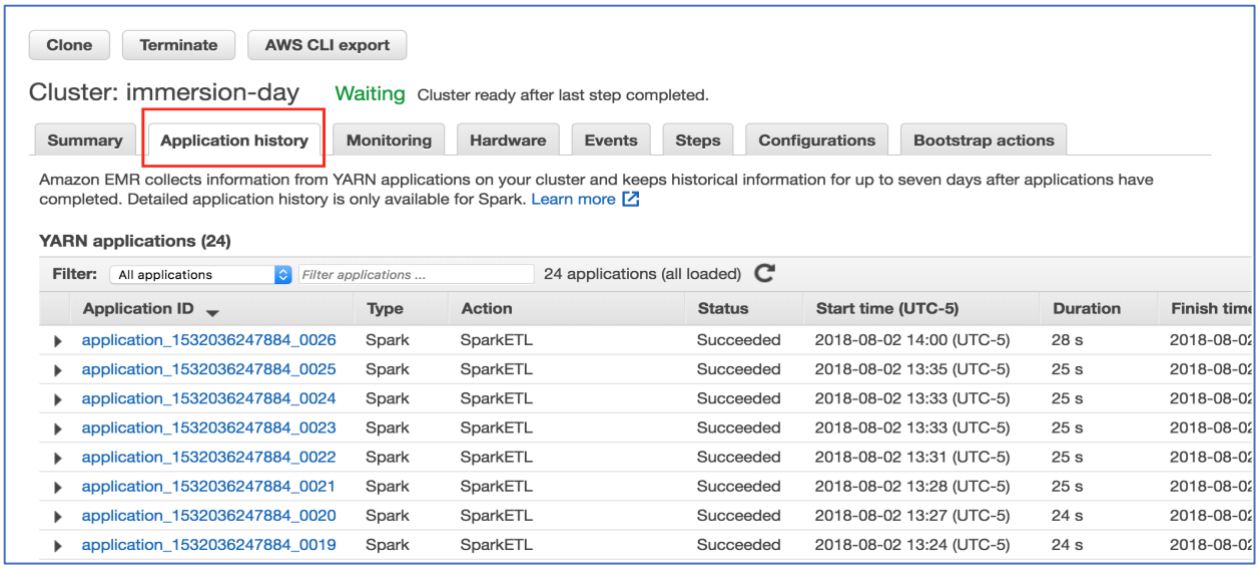
Test 2: Check YARN Application logs on Amazon EMR Console

Spark job submitted on Amazon EMR cluster run as YARN application. you can view YARN application details using the Application history tab of a cluster's detail page in the console. Using Amazon EMR application history makes it easier for you to troubleshoot and analyze active jobs and job history. Instead of setting up and connecting to the master node to view open-source troubleshooting UIs or sift through log files, you can quickly view application metrics and access relevant log files.

Go to Amazon EMR console and select the cluster you created in the previous step to open up the cluster details window:



Click on the “Application history” tab



Application history tab shows a list of applications that are executed on the Amazon EMR cluster. Now click on the job you just submitted and check its log

Cluster: immersion-day Waiting Cluster ready after last step completed.

Summary | Application history | Monitoring | Hardware | Events | Steps | Configurations | Bootstrap actions

YARN applications > application_1532036247884_0026 (Spark)

Jobs | Stages | Executors

Jobs > Job 3

Status: Succeeded

Completed stages: 2

Event timeline

Stages (2)

Filter: Filter stages ... 2 stages (all loaded)

Stage ID	Status	Description	Submitted (UTC-5)	Duration	Tasks succeeded / total
4	Completed	count at NativeMethodAccessorImpl.java:0	2018-08-02 14:01 (UTC-5)	96 ms	1 / 1
3	Completed	count at NativeMethodAccessorImpl.java:0	2018-08-02 14:01 (UTC-5)	0.3 s	1 / 1

You can drill down further to retrieve “stdout” and “stderr” logs:

Amazon EMR

- Clusters
- Security configurations
- VPC subnets
- Events
- Help
- What's new

Shuffle read (size / records)	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1
Shuffle remote reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Task deserialization time	15 ms	15 ms	15 ms	15 ms	15 ms

Aggregated metrics by executor (1)

Filter: Filter executors ... 1 executors (all loaded)

Executor ID	Address	Task time	Total tasks	Failed tasks	Succeeded tasks	Shuffle read	Blacklisted
1	ip-10-0-0-46.ec2.internal:46165 stderr | stdout	94 ms	1	0	1	59.0 B	No

Tasks (1)

Filter: Filter tasks ... 1 tasks (all loaded)

ID	Attempt	Status	Locality level	Executor ID / Host	Launch time (UTC-5)	Duration	Task deserialization time	GC time
4	0	Succeeded	Node local	1 / ip-10-0-0-46.ec2.internal stderr | stdout	2018-08-02 14:01 (UTC-5)	94 ms	15 ms	3 ms

Clicking on the “stdout” or “stderr” will open up the logs in a different window

```
2018-08-02T19:01:08.195+0000: [GC (Allocation Failure) 2018-08-02T19:01:08.195+0000: [ParNew: 68864K->7107K(77440K), 0.0102949 secs] 68864K->7107K(249472K),
0.0103694 secs] [Times: user=0.03 sys=0.00, real=0.01 secs]
2018-08-02T19:01:08.486+0000: [GC (Allocation Failure) 2018-08-02T19:01:08.486+0000: [ParNew: 75971K->4367K(77440K), 0.0155614 secs] 75971K->7021K(249472K),
0.0156083 secs] [Times: user=0.04 sys=0.01, real=0.01 secs]
2018-08-02T19:01:08.945+0000: [GC (Allocation Failure) 2018-08-02T19:01:08.945+0000: [ParNew: 73231K->3094K(77440K), 0.0059296 secs] 75885K->5748K(249472K),
0.0059828 secs] [Times: user=0.03 sys=0.00, real=0.01 secs]
2018-08-02T19:01:09.245+0000: [GC (Allocation Failure) 2018-08-02T19:01:09.245+0000: [ParNew: 71958K->5396K(77440K), 0.0051351 secs] 74612K->8050K(249472K),
0.0051857 secs] [Times: user=0.01 sys=0.01, real=0.01 secs]
2018-08-02T19:01:09.435+0000: [GC (Allocation Failure) 2018-08-02T19:01:09.435+0000: [ParNew: 58721K->3789K(77440K), 0.0144957 secs] 61375K->24236K(249472K),
0.0145426 secs] [Times: user=0.05 sys=0.01, real=0.02 secs]
2018-08-02T19:01:09.452+0000: [GC (CMS Initial Mark) [1 CMS-initial-mark: 20446K(172032K)] 40620K(249472K), 0.0011764 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
2018-08-02T19:01:09.453+0000: [CMS-concurrent-mark-start]
2018-08-02T19:01:09.497+0000: [CMS-concurrent-mark: 0.044/0.044 secs] [Times: user=0.09 sys=0.01, real=0.04 secs]
2018-08-02T19:01:09.498+0000: [CMS-concurrent-preclean-start]
2018-08-02T19:01:09.498+0000: [CMS-concurrent-preclean: 0.001/0.001 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
2018-08-02T19:01:09.498+0000: [CMS-concurrent-abortable-preclean-start]
2018-08-02T19:01:09.696+0000: [GC (Allocation Failure) 2018-08-02T19:01:09.697+0000: [ParNew: 72653K->3787K(77440K), 0.0135308 secs] 93100K->40618K(249472K),
0.0135801 secs] [Times: user=0.04 sys=0.01, real=0.02 secs]
2018-08-02T19:01:09.854+0000: [CMS-concurrent-abortable-preclean: 0.099/0.356 secs] [Times: user=0.75 sys=0.03, real=0.35 secs]
2018-08-02T19:01:09.854+0000: [GC (CMS Final Remark) [YG occupancy: 50883 K (77440 K)]2018-08-02T19:01:09.854+0000: [Rescan (parallel) , 0.0040955 secs]2018-08-
02T19:01:09.858+0000: [weak refs processing, 0.0000313 secs]2018-08-02T19:01:09.859+0000: [class unloading, 0.0027199 secs]2018-08-02T19:01:09.861+0000: [scrub
symbol table, 0.0030443 secs]2018-08-02T19:01:09.864+0000: [scrub string table, 0.0003214 secs][1 CMS-remark: 36830K(172032K)] 87714K(249472K), 0.0105317 secs]
[Times: user=0.02 sys=0.00, real=0.01 secs]
2018-08-02T19:01:09.865+0000: [CMS-concurrent-sweep-start]
2018-08-02T19:01:09.867+0000: [CMS-concurrent-sweep: 0.002/0.002 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]
2018-08-02T19:01:09.867+0000: [CMS-concurrent-reset-start]
2018-08-02T19:01:10.033+0000: [CMS-concurrent-reset: 0.166/0.166 secs] [Times: user=0.32 sys=0.14, real=0.16 secs]
2018-08-02T19:01:10.036+0000: [GC (Allocation Failure) 2018-08-02T19:01:10.036+0000: [ParNew: 72651K->5756K(77440K), 0.0050600 secs] 109465K->42571K(249472K),
0.0051023 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]
2018-08-02T19:01:10.304+0000: [GC (Allocation Failure) 2018-08-02T19:01:10.304+0000: [ParNew: 65746K->4463K(77440K), 0.0289622 secs] 102560K->59136K(249472K),
0.0290107 secs] [Times: user=0.10 sys=0.01, real=0.03 secs]
2018-08-02T19:01:10.752+0000: [GC (Allocation Failure) 2018-08-02T19:01:10.752+0000: [ParNew: 73327K->6154K(77440K), 0.0236802 secs] 128000K->77211K(249472K),
0.0237505 secs] [Times: user=0.08 sys=0.01, real=0.02 secs]
2018-08-02T19:01:11.206+0000: [GC (Allocation Failure) 2018-08-02T19:01:11.206+0000: [ParNew: 75018K->6796K(77440K), 0.0078448 secs] 146075K->79285K(249472K),
0.0078953 secs] [Times: user=0.02 sys=0.00, real=0.00 secs]
2018-08-02T19:01:11.521+0000: [GC (Allocation Failure) 2018-08-02T19:01:11.521+0000: [ParNew: 75660K->6330K(77440K), 0.0082200 secs] 148149K->80713K(249472K),
0.0082879 secs] [Times: user=0.03 sys=0.00, real=0.01 secs]
2018-08-02T19:01:11.848+0000: [GC (Allocation Failure) 2018-08-02T19:01:11.848+0000: [ParNew: 75194K->3885K(77440K), 0.0104556 secs] 149577K->79426K(249472K),
0.0105111 secs] [Times: user=0.04 sys=0.00, real=0.01 secs]
2018-08-02T19:01:12.296+0000: [GC (Allocation Failure) 2018-08-02T19:01:12.296+0000: [ParNew: 72749K->6635K(77440K), 0.0065865 secs] 148290K->82176K(249472K),
0.0066349 secs] [Times: user=0.02 sys=0.00, real=0.01 secs]
2018-08-02T19:01:12.303+0000: [GC (CMS Initial Mark) [1 CMS-initial-mark: 75541K(172032K)] 83416K(249472K), 0.0018176 secs] [Times: user=0.01 sys=0.00, real=0.00
secs]
```

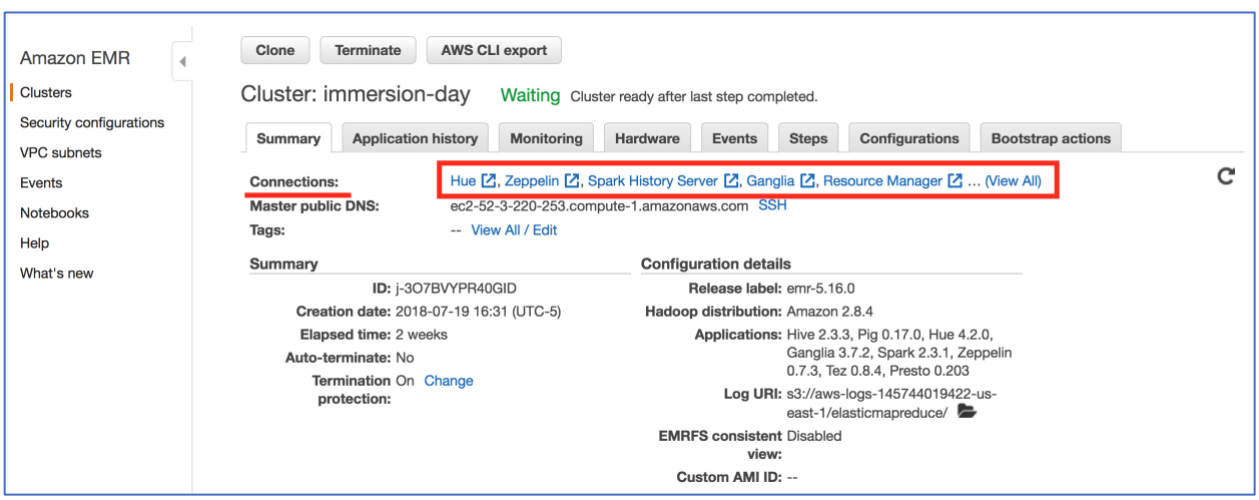
Test 3: Check status and logs on Spark UI

Spark UI displays useful information about the application you submit on Amazon EMR. You can view the Spark web UIs by following the procedures to create an SSH tunnel or create a proxy.

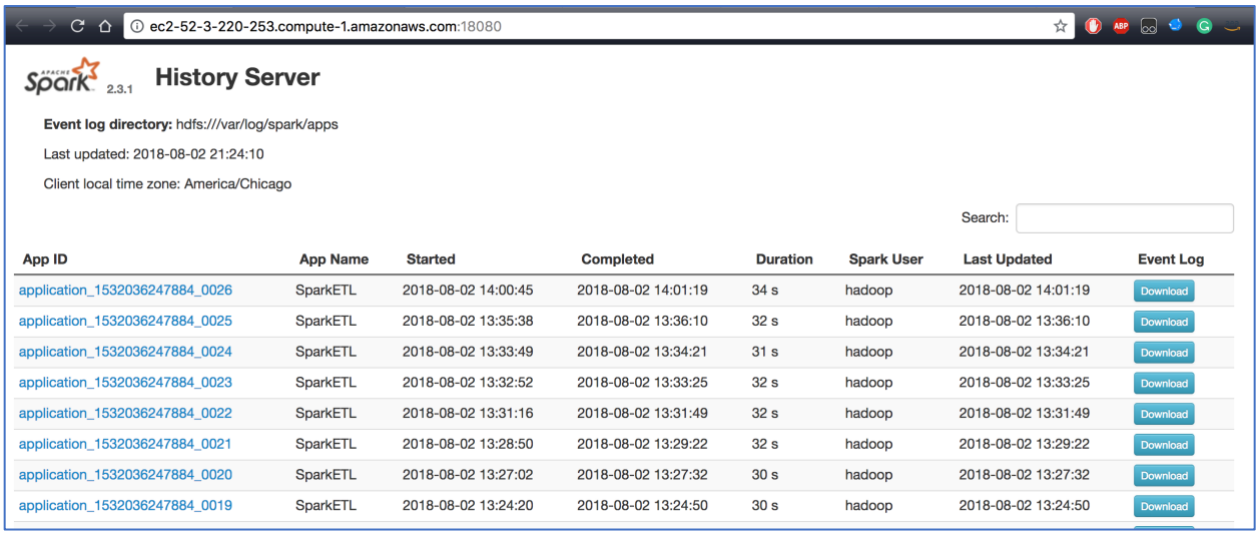
Follow this link to establish a connection to Web interfaces hosted on Amazon EMR cluster

https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-web-interfaces.html

Once the connection is established, you should be able to see UI links in the "Connections" section of your Amazon EMR cluster:

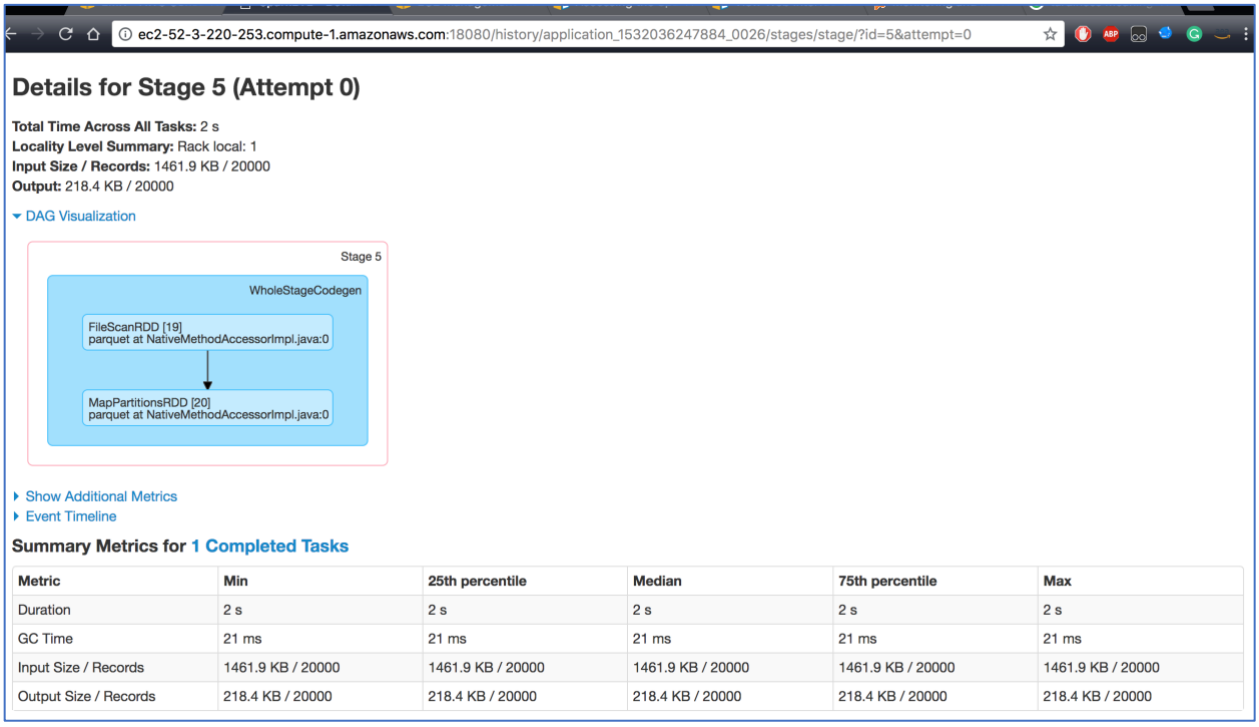


Clicking on the “Spark History Server” will bring up the Spark UI:



App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
application_1532036247884_0026	SparkETL	2018-08-02 14:00:45	2018-08-02 14:01:19	34 s	hadoop	2018-08-02 14:01:19	Download
application_1532036247884_0025	SparkETL	2018-08-02 13:35:38	2018-08-02 13:36:10	32 s	hadoop	2018-08-02 13:36:10	Download
application_1532036247884_0024	SparkETL	2018-08-02 13:33:49	2018-08-02 13:34:21	31 s	hadoop	2018-08-02 13:34:21	Download
application_1532036247884_0023	SparkETL	2018-08-02 13:32:52	2018-08-02 13:33:25	32 s	hadoop	2018-08-02 13:33:25	Download
application_1532036247884_0022	SparkETL	2018-08-02 13:31:16	2018-08-02 13:31:49	32 s	hadoop	2018-08-02 13:31:49	Download
application_1532036247884_0021	SparkETL	2018-08-02 13:28:50	2018-08-02 13:29:22	32 s	hadoop	2018-08-02 13:29:22	Download
application_1532036247884_0020	SparkETL	2018-08-02 13:27:02	2018-08-02 13:27:32	30 s	hadoop	2018-08-02 13:27:32	Download
application_1532036247884_0019	SparkETL	2018-08-02 13:24:20	2018-08-02 13:24:50	30 s	hadoop	2018-08-02 13:24:50	Download

Select application ID of your most recent Spark job and drill-down logs and check DAG of different stages:



Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	2 s	2 s	2 s	2 s
GC Time	21 ms	21 ms	21 ms	21 ms	21 ms
Input Size / Records	1461.9 KB / 20000	1461.9 KB / 20000	1461.9 KB / 20000	1461.9 KB / 20000	1461.9 KB / 20000
Output Size / Records	218.4 KB / 20000	218.4 KB / 20000	218.4 KB / 20000	218.4 KB / 20000	218.4 KB / 20000