USING AMAZON EMR WITH APACHE AIRFLOW: HOW & WHY TO DO IT



In this introductory article, I explore Amazon EMR and how it works with Apache Airflow.

What is Amazon EMR?

Amazon EMR is an orchestration tool to create a Spark or Hadoop big data cluster and run it on Amazon virtual machines.

While Airflow provides DAG-based scheduling for EMR, organizations running EMR in enterprise production environments can benefit from the Control-M for Amazon EMR integration — adding SLA management, cross-system dependency handling, and unified operational visibility alongside other enterprise workloads.

What is Apache Airflow?

Apache Airflow is a tool for defining and running jobs—i.e., a big data pipeline—on:

- Apache Hadoop
- Snowflake (data warehouse charting)
- Amazon products including EMR, Redshift (data warehouse), S3 (file storage), and Glacier (long term data archival)
- Many other products

Airflow can also start and takedown Amazon EMR clusters. That's important because your EMR

clusters could get quite expensive if you leave them running when they are not in use.

Benefits of Airflow

For most use cases, there's two main advantages of Airflow running on an Apache Hadoop and Spark environment:

- Cost management
- Optional surge capacity (which, of course, is one element of cost)

Plus, it's simpler to use Airflow and its companion product Genie (developed by Netflix) to do things like run jobs using **spark-submit** or **Hadoop queues**, which have a lot of configuration options and an understanding of things like Yarn, a resource manager. In this setup, Airflow:

- Lets you bundle jar files, Python code, and configuration data into metadata
- Provides a feedback loop in case any of that goes wrong

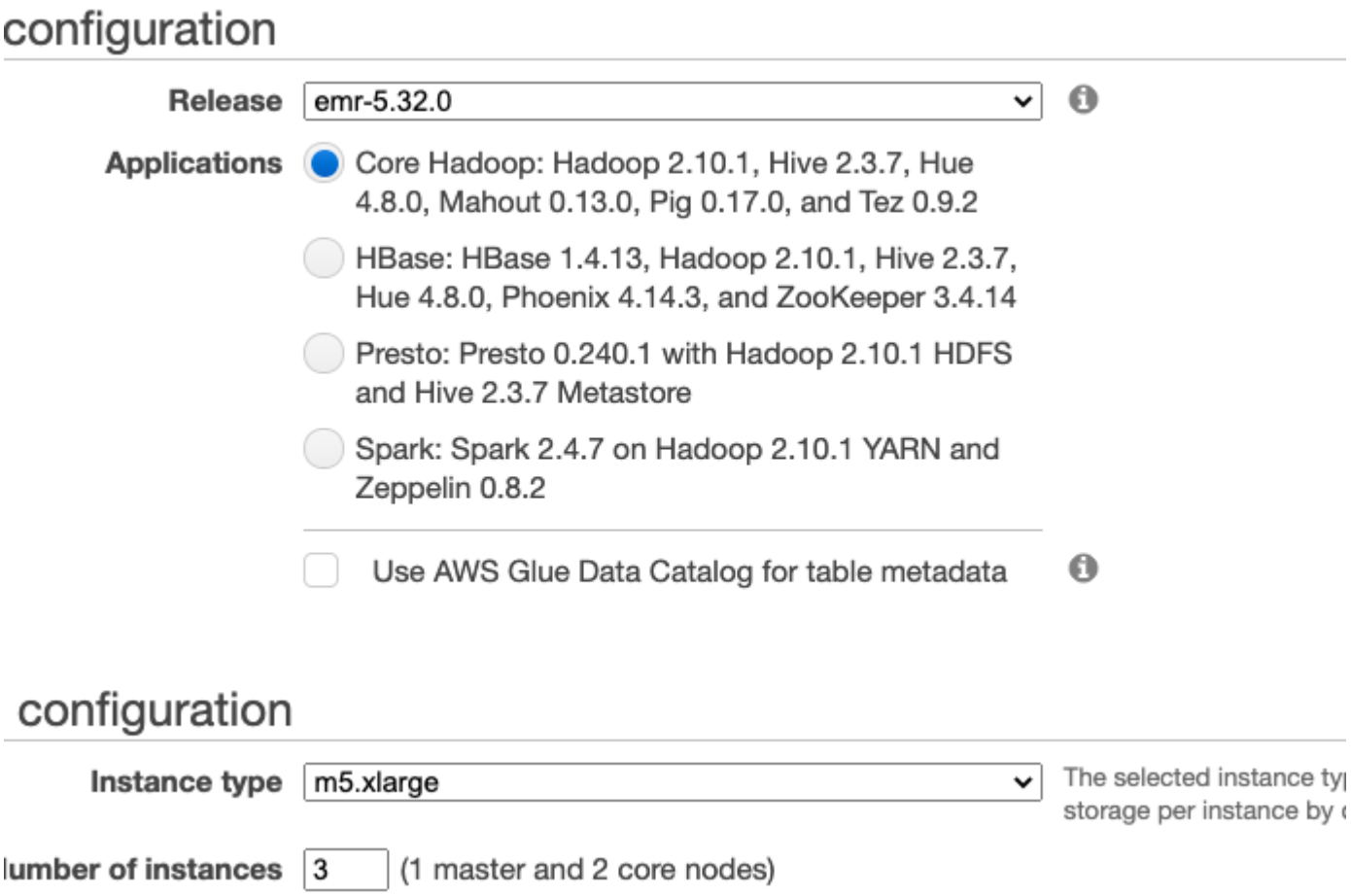
So, it might be better to use Airflow than the alternative: typing **spark-submit** into the command line and hoping for the best.

The point here is that you don't need Airflow. All the products it installs are open source. But it might cost less to have Airflow install all of that and then tear it down than the alternative: you leaving idle hardware running, especially if all of that is running at Amazon, plus perhaps paying a big data engineer to write and debug scripts to do all of this some other way.

EMR components

What goes into EMR? Here is the configuration wizard for EMR. You can see that it installs some of the products that normally you use with Spark and Hadoop, like:

- Yarn, a resource scheduler
- Pig
- Mahout for machine learning
- Zeppelin
- Jupyter



What does EMR do?

The name EMR is an amalgamation for Elastic and MapReduce. Elastic refers to Elastic Cluster, better known as EC2. Apache MapReduce is both a programming paradigm and a set of Java SDKs, in particular these two Java classes:

1. apache.hadoop.mapreduce.Mapper;
2. apache.hadoop.mapreduce.Reducer;

These run MapReduce operations and then optionally save the results to an Apache Hadoop Distributed File System (HDFS).

You might say that MapReduce is a little bit old-fashioned, since Apache Spark does the same thing as that Hadoop-centric approach, but in a more efficient way. That's probably why EMR has both products.

MapReduce explained

The concept of **map** is common to most programming languages. It means to run some function or some collection of data. **Reduce** means to count, sum, or otherwise create a subset of that now reduced data.

To illustrate what this means, the Hello World programming example for MapReduce is usually the WordCount program. (Hello World is the usually the first and simplest introduction to any programming language.)

The WordCount program does both **map** and **reduce**. The map step creates this tuple (wordX, 1) then sums the numbers 1. (Summing 1 is the same as counting, yes?) So, if a set of text contains wordX 10 times then the wrestling (wordX,10) counts the occurrence of that word.

To illustrate, let's say you have these sentences:

```
James hit the ball.
James makes the ball.
James lost the shoes.
```

MapReduce would then create this set of tuples (pairs):

```
(James, 1)
(hit, 1)
(the, 1)
…
(James, 1)
…
(the, 1)
```

The reduce step would then sum each of these tuples yielding these word counts:

```
(James, 3)
(hit, 1)
(ball, 2)
(the, 3)
…
```

If that does not seem too exciting, remember that Hadoop and Spark lets you run these operations on something large and messy, like records from your SAP transactional inventory system.

This idea and approach can scale without limit. That's because Hadoop and Spark can scale without limit—it can spread the load across servers. You could then feed the new reduced data set into a reporting system or a predictive model etc.

Other uses for EMR

Though EMR was developed primarily for the MapReduce and Hadoop use case, there are other areas where EMR can be useful:

- For example, Java code is very wordy. So, Amazon EMR typically deploys Apache Pig with EMR. This lets you use SQL, which is a lot shorter and simpler, to run MapReduce operations. Hive is similar.
- EMR also can host Zeppelin and Jupyter notebooks. These are web pages where you can write code. EMR supports graphics and many programming languages. For example, you can write Python code to run machine learning models against data you have stored in Hadoop or Spark.

How to install Airflow

Airflow is easy to install. EMR takes more steps, which is one reason why you might want to use Airflow. Beyond the initial setup, however, Amazon makes EMR cluster creation easier the second

time you use it by saving a script that you can run with the Amazon command line interface (CLI).

You basically source a Python environment (e.g., source py372/bin/activate, if using virtualenv) then run this to install Airflow, which is nothing more than a Python package:

```
export AIRFLOW_HOME=~/airflow
pip install apache-airflow
airflow db init
```

Then you create a user.

```
airflow users create \
--username admin \
--firstname walker \
--lastname walker \
--role Admin \
--email walker@walker.com
```

Then you start the web server interface, using any available port.

```
airflow webserver --port 7777
```

Airflow code example

Here is an Airflow code example from the Airflow GitHub, with excerpted code below. Basically, Airflow runs Python code on Spark to calculate the number Pi to 10 decimal places. This illustrates how Airflow is one way to package a Python program and run it on a Spark cluster.

Looking briefly at the code:

- **EmrCreateJobFlowOperator** creates the job.
- **EmrStepSensor** sets up monitoring via the web page.
- **EmrTerminateJobFlowOperator** removes the cluster.

```
cluster_creator = EmrCreateJobFlowOperator(
task_id='create_job_flow',
job_flow_overrides=JOB_FLOW_OVERRIDES,
aws_conn_id='aws_default',
emr_conn_id='emr_default',
)
step_adder = EmrAddStepsOperator(
task_id='add_steps',
job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}",
aws_conn_id='aws_default',
steps=SPARK_STEPS,
)
step_checker = EmrStepSensor(
task_id='watch_step',
job_flow_id="{{ task_instance.xcom_pull('create_job_flow',
key='return_value') }}",
```

```
step_id="{{ task_instance.xcom_pull(task_ids='add_steps', key='return_value')
}}",
aws_conn_id='aws_default',
)
cluster_remover = EmrTerminateJobFlowOperator(
task_id='remove_cluster',
job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}",
aws_conn_id='aws_default',
)
```

Related reading

- BMC Machine Learning & Big Data Guide
- BMC Guides, where we offer series of tutorials on a variety of tools, including Spark, Hadoop, AWS, Machine Learning, Snowflake, and more
- Top Machine Learning Frameworks To Use in 2021
- Data Storage Explained: Data Lake vs Warehouse vs Database