ETL Process using AWS

Objective:

- To leverage ETL tools available in the Amazon cloud with help of employee migration data.
- To evaluate and compare Glue and EMR (ETL AWS Technologies) costs and computational effort.

TechStack:

1. AWS Glue Studio, Cloud 9 Console.
2. AWS EMR - EC2
3. S3
4. PySpark
5. Git

Business Scenario:

Motivation behind choosing this use case is, in scenarios when the company becomes a subsidiary of a parent company, employee data needs to be migrated.

For Example: MLReply has become an independent company from Data Reply since August. Leveraging ETL process in the cloud environment for such cases when the job runs daily, two factors need to be monitored:

- What would be the costs at daily rate.
- How much development time does it take.

Data Description:

The employee migration data contains 2 tables. The main data table contains the information of employees with 9 columns and one million records. The second data table contains the information of employee names that has been changed in cases where some employees have new names.

Following are the fields in main table:

Employee ID, Name, Gender, Address, Country, Email, Unit, Experience, Domain.

Implementation:

AWS GLUE:

- Serverless data integration service
- Easy to discover, prepare and combine data for analytics, ML, and App Development.

Steps:

1. Creating Cloud9 environment to get the files from Git.
2. Create S3 bucket inside cloud 9 IDE.
3. Create Glue Data Catalog - For a given data set, you can store its table definition, physical location, add business relevant attributes, as well as track how this data has changed over time.
4. Add database and crawlers to crawl the CSV and JSON folders.
5. Run the crawlers - The table schema will be automatically generated by the crawler based on the csv file (in this use-case).

6. **6.** Creating AWS environment i.e Glue Dev Endpoint - to iteratively develop and test the extract, transform, and load (ETL) scripts
7. **7.** Connect development notebooks like Sagemaker Notebooks to the Dev Endpoint to test the code snippets.
8. **8.** Deploy and Run the ETL Job.

Observations:

- Amazon Glue's pay-as-you-go rate of $0.44 per DPU (One DPU=4 vCPU and 16 GB of memory)
- It might seem reasonable at first, organizations commonly find themselves with bloated bills after prolonged use.
- The cost depends on the jobs run and dev-endpoint. In this scenario, before even deploying the jobs the cost was more for dev-endpoint.
- Below is the tabulation of the costs:

Time	**Cost**
per DPU-Hour	0.44$
1h	21$
4 DAYS	100$

Merits:

1. AWS Glue is capable of automatically generating ETL pipeline code in Scala or Python
2. Serverless architecture and job scheduling.
3. Increased data visibility - By acting as the metadata repository for information on your data sources and stores
4. Developer endpoints - For users who prefer to manually create and test their own custom ETL scripts

Demerits:

1. Only two languages - When it comes to customizing ETL codes, AWS Glue only supports Python and Scala
2. Limited integrations
3. Expensive when jobs has to run on daily basis.

Amazon Elastic MapReduce (EMR):

1. EMR is a big data platform designed to reduce the cost of processing and analyzing huge amounts of data.
2. A managed service where you configure your own cluster of EC2 instances.

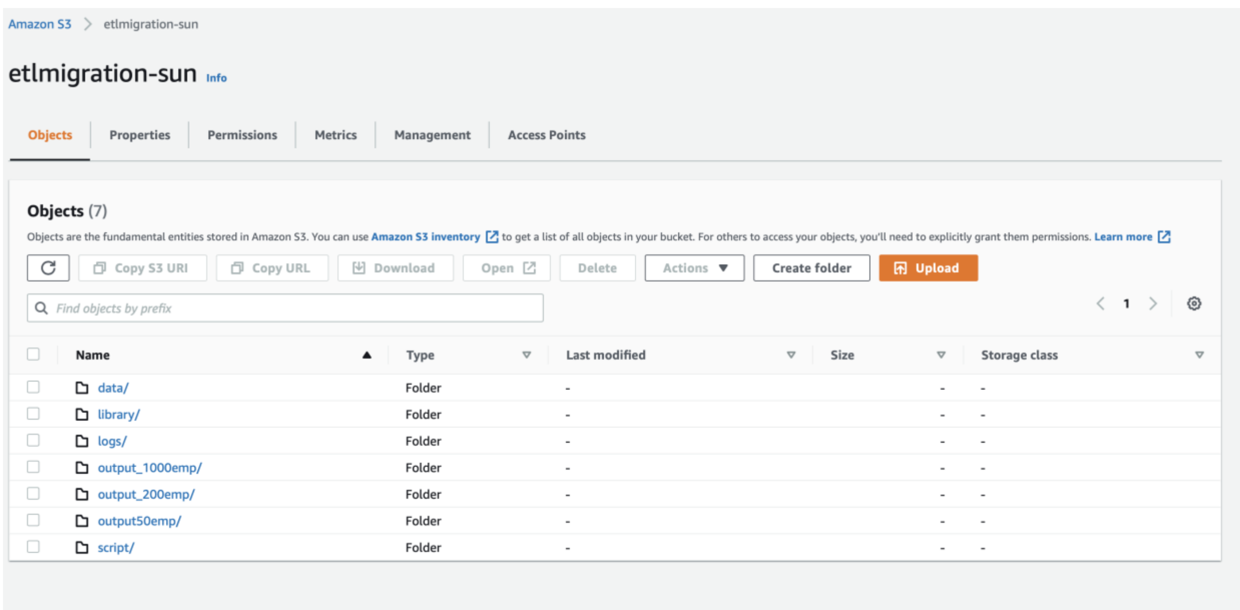
Steps:

1. Creating cluster using advanced options
2. Specify the software configuration and add step according to the application. In this case, it is Spark Application.
3. Choose the hardware, instance type for master and cluster node. Add the EBS root volume.
4. Add tags and general options according to the use case.
5. Check for EC2 security groups and generate EC2 Key pair.
6. Once the cluster moves from waiting to Ready state, log in to the EMR Cluster.

Snapshot1: Login to EMR Cluster

```
(base) s.radhakrishnan@C02F43GSMD6T Downloads % ssh -i "ETLSunitha.pem" ec2-user@ec2-13-212-162-142.ap-southeast-1.compute.amazonaws.com
Last login: Wed Sep 15 16:43:39 2021 from aftr-62-216-214-139.dynamic.mnet-online.de

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

https://aws.amazon.com/amazon-linux-2/
63 package(s) needed for security, out of 106 available
Run "sudo yum update" to apply all updates.
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[ec2-user@ip-172-31-2-36 ~]$
```

Snapshot2: Files stored in S3 bucket. Use the location of S3 for the script that you want to deploy it in your cluster.



Snapshot3: Copied files from AWS S3 to my EMR cluster.

```
[ec2-user@ip-172-31-2-36 emr]$ ls -lrt
total 28
-rw-rw-r-- 1 ec2-user ec2-user 2713 Sep  7 16:45 ETL2.py
-rw-rw-r-- 1 ec2-user ec2-user 2165 Sep  9 17:12 ETL.py
-rw-rw-r-- 1 ec2-user ec2-user 2764 Sep 13 14:05 data-migration.py
-rw-rw-r-- 1 ec2-user ec2-user 2765 Sep 13 14:49 data-migration_1000emp.py
-rw-rw-r-- 1 ec2-user ec2-user 2757 Sep 15 16:30 ETL_50emp.py
-rw-rw-r-- 1 ec2-user ec2-user 2762 Sep 15 16:43 ETL_200.py
-rw-rw-r-- 1 ec2-user ec2-user 2756 Sep 15 16:54 ETL_1000emp.py
[ec2-user@ip-172-31-2-36 emr]$
```

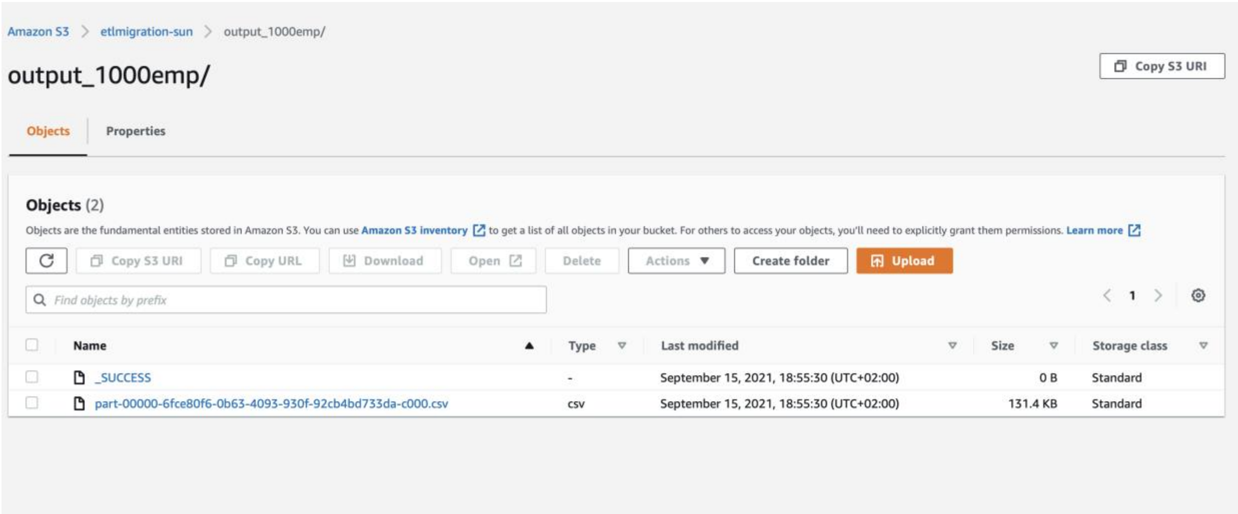
Snapshot4: Spark-Submit the python script using the spark application to run the application.

```
[ec2-user@ip-172-31-2-36 emr]$ sudo spark-submit ETL_50emp.py
21/09/15 16:34:32 INFO SparkContext: Running Spark version 2.4.7-amzn-1
21/09/15 16:34:32 INFO SparkContext: Submitted application: ETL_50emp.py
21/09/15 16:34:32 INFO SecurityManager: Changing view acls to: root
21/09/15 16:34:32 INFO SecurityManager: Changing modify acls to: root
21/09/15 16:34:32 INFO SecurityManager: Changing view acls groups to:
21/09/15 16:34:32 INFO SecurityManager: Changing modify acls groups to:
21/09/15 16:34:32 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); groups with view permissions: Set(); users  with modify permissic
et(root); groups with modify permissions: Set()
21/09/15 16:34:32 INFO Utils: Successfully started service 'sparkDriver' on port 33543.
21/09/15 16:34:32 INFO SparkEnv: Registering MapOutputTracker
21/09/15 16:34:32 INFO SparkEnv: Registering BlockManagerMaster
21/09/15 16:34:32 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
21/09/15 16:34:32 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
21/09/15 16:34:32 INFO DiskBlockManager: Created local directory at /mnt/tmp/blockmgr-d03de202-3426-43c9-85f8-1e1bcc2b1a5c
21/09/15 16:34:32 INFO MemoryStore: MemoryStore started with capacity 912.3 MB
21/09/15 16:34:32 INFO SparkEnv: Registering OutputCommitCoordinator
21/09/15 16:34:32 INFO Utils: Successfully started service 'SparkUI' on port 4040.
21/09/15 16:34:32 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://ip-172-31-2-36.ap-southeast-1.compute.internal:4040
21/09/15 16:34:33 INFO Utils: Using initial executors = 50, max of spark.dynamicAllocation.initialExecutors, spark.dynamicAllocation.minExecutors and spark.executor.instances
21/09/15 16:34:33 INFO RMProxy: Connecting to ResourceManager at ip-172-31-2-36.ap-southeast-1.compute.internal/172.31.2.36:8032
21/09/15 16:34:33 INFO Client: Requesting a new application from cluster with 2 NodeManagers
21/09/15 16:34:33 INFO Configuration: resource-types.xml not found
21/09/15 16:34:33 INFO ResourceUtils: Unable to find 'resource-types.xml'.
21/09/15 16:34:33 INFO ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/09/15 16:34:33 INFO ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/09/15 16:34:33 INFO Client: Verifying our application has not requested more than the maximum memory capability of the cluster (12288 MB per container)
21/09/15 16:34:33 INFO Client: Will allocate AM container, with 896 MB memory including 384 MB overhead
21/09/15 16:34:33 INFO Client: Setting up container launch context for our AM
21/09/15 16:34:33 INFO Client: Setting up the launch environment for our AM container
21/09/15 16:34:33 INFO Client: Preparing resources for our AM container
21/09/15 16:34:33 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
21/09/15 16:34:35 INFO Client: Uploading resource file:/mnt/tmp/spark-db5027f5-cadb-4704-88a2-5ff774e88596/__spark_libs__668353376907698894.zip -> hdfs://ip-172-31-2-36.ap-southeast-1.compute.internal:8020/
root/.sparkStaging/application_1631197486247_0013/__spark_libs__668353376907698894.zip
21/09/15 16:34:36 INFO Client: Uploading resource file:/etc/spark/conf/hive-site.xml -> hdfs://ip-172-31-2-36.ap-southeast-1.compute.internal:8020/user/root/.sparkStaging/application_1631197486247_0013/hive
.xml
21/09/15 16:34:36 INFO Client: Uploading resource file:/usr/lib/spark/python/lib/pyspark.zip -> hdfs://ip-172-31-2-36.ap-southeast-1.compute.internal:8020/user/root/.sparkStaging/application_1631197486247_0
yspark.zip
21/09/15 16:34:36 INFO Client: Uploading resource file:/usr/lib/spark/python/lib/py4j-0.10.7-src.zip -> hdfs://ip-172-31-2-36.ap-southeast-1.compute.internal:8020/user/root/.sparkStaging/application_163119
7_0013/py4j-0.10.7-src.zip
21/09/15 16:34:36 INFO Client: Uploading resource file:/mnt/tmp/spark-db5027f5-cadb-4704-88a2-5ff774e88596/__spark_conf__4236270622893540232.zip -> hdfs://ip-172-31-2-36.ap-southeast-1.compute.internal:802
/root/.sparkStaging/application_1631197486247_0013/__spark_conf__.zip
21/09/15 16:34:36 INFO SecurityManager: Changing view acls to: root
21/09/15 16:34:36 INFO SecurityManager: Changing modify acls to: root
21/09/15 16:34:36 INFO SecurityManager: Changing view acls groups to:
21/09/15 16:34:36 INFO SecurityManager: Changing modify acls groups to:
21/09/15 16:34:36 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(root); groups with view permissions: Set(); users  with modify permissic
et(root); groups with modify permissions: Set()
21/09/15 16:34:38 INFO Client: Submitting application application_1631197486247_0013 to ResourceManager
21/09/15 16:34:38 INFO YarnClientImpl: Submitted application application_1631197486247_0013
21/09/15 16:34:38 INFO SchedulerExtensionServices: Starting Yarn extension services with app application_1631197486247_0013 and attemptId None
21/09/15 16:34:39 INFO Client: Application report for application_1631197486247_0013 (state: ACCEPTED)
21/09/15 16:34:39 INFO Client:
         client token: N/A
         diagnostics: AM container is launched, waiting for AM container to Register with RM
         ApplicationMaster host: N/A
         ApplicationMaster RPC port: -1
         queue: default
         start time: 1631723678040
         final status: UNDEFINED
         tracking URL: http://ip-172-31-2-36.ap-southeast-1.compute.internal:20888/proxy/application_1631197486247_0013/
         user: root
21/09/15 16:34:40 INFO Client: Application report for application_1631197486247_0013 (state: ACCEPTED)
```

NOTE:

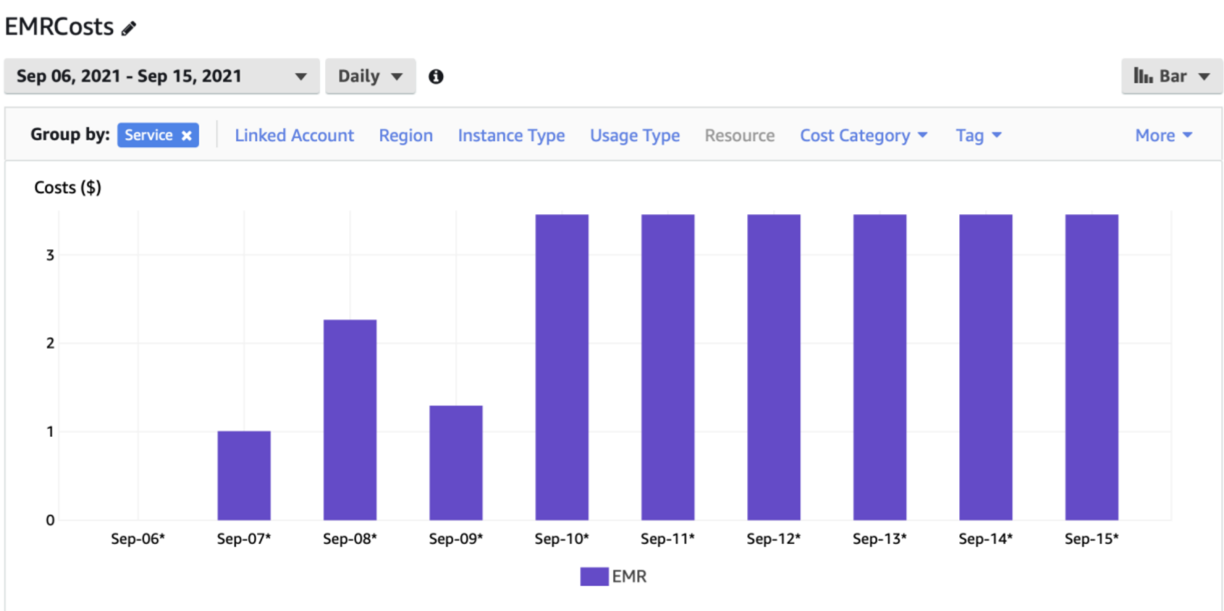
- **Spark submit can only be done as a root user.**
- **If the application fails, or it has some errors, it will be displayed in the CLI or can even check log files.**

Snapshot5: Output files generated and stored in S3 AWS.



Observations:

- Billing and Cost Management is one of the important perspectives that one should also take care of in cloud when working on any use-case.
- Following is the chart on the cost usage under the cost explorer:



Service	EMR($)	Total cost ($)
Service Total	39.274266672	39.274266672
2021-09-07	1.008	1.008
2021-09-08	2.266266672	2.266266672
2021-09-09	1.296	1.296
2021-09-10	3.456	3.456
2021-09-11	3.456	3.456
2021-09-12	3.456	3.456
2021-09-13	3.456	3.456
2021-09-14	3.456	3.456
2021-09-15	3.456	3.456
2021-09-16	3.456	3.456
2021-09-17	3.456	3.456
2021-09-18	3.456	3.456
2021-09-19	3.456	3.456
2021-09-20	144	144

Benchmarking was done regarding the development time:

EMR-Spark	Script Execution Time	EMR Cost	Cluster up and Run time
50 rows	2 seconds	5.72$	119.214 hrs
200 rows	5 seconds	14.94$	311.214 hrs
1000 rows	10 seconds	$23.29	485.214 Hrs

Merits:

1. As a cloud based computing service, Amazon EMR offers a data solution without the cost of maintaining an in-house computing infrastructure server.
2. Saves time in tasks like system administration.
3. In comparison with Glue, it saves a lot of costs for the company.

Demerits:

1. Synchronization of metadata from S3 tends to become inconsistent sometimes.
2. Management overhead would not be as simple as Glue.

Alternate/Similar technologies:

1. Amazon Apache Airflow – for scheduling ETL jobs.
2. AWS Data Pipeline
3. AWS Batch

Future Scope:

- Orchestrate Apache Spark applications using AWS Step functions and Apache Livy(REST API's).
- Emr - Apache Livy – Sagemaker.