|  |
| --- |
| SAMRX Data Lake  Solution Design Approach |

**High Level Data Lake Creation Process**

1. Understanding / discussion on data - Key’s in various tables to use for partition and datetime field to be used for incremental Data Ingestion.
2. Creation of New Fields i.e. Year, Month and Day from datetime field for Incremental Data Ingestion.
3. Launching the EMR Cluster with one Master node and 2 core nodes.
4. Submitting One-Time Data Ingestion job on EMR manually.
5. Submitting Incremental Data Ingestion job on EMR through Step Function on scheduled basis.
6. Creation of Glue Crawler for creating schema for the Data Lake.
7. Query data through Athena from parquet in S3.

**Our Solution approach is multistep** –

1. Setup AWS Account
2. Setup AWS security
3. MFA Token for Root Account
4. Identity and Access management with defined roles and rights for AWS users and Services.
5. Virtual Private Cloud (a virtual private network) as per the required configuration.
6. Setup S3 Bucket for creating Data Lake.
7. Configure lambda to launch cluster and run Job Flow.
8. Create EMR Cluster for one-time Data Ingestion from Cassandra to S3.
9. Schedule Incremental Data Ingestion Job through State Machine.
10. Create Glue Crawler to create Schema for the respective data lake and schedule on Daily basis.
11. Configure Athena to query parquet data in S3.

**Learnings**

* Partitioning of data improved the data query performance.
* EMR Cluster calculation of nodes required for data preprocessing.
* Using parquet format gave many benefits like less space required for storing the data.

**Approach(s) Explored**

We researched on various options as follows

* **AWS DMS**
  + AWS Database Migration Service helps to migrate databases to AWS quickly and securely. The source database remains fully operational during the migration, minimizing downtime to applications that rely on the database. The AWS Database Migration Service can migrate your data to and from most widely used commercial and open-source databases.
  + Explored the AWS Data Migration service for the Incremental transfer of data. For this we have explored the AWS Schema Conversion Tool to setup on the data base side, but it can only perform CDC to AWS Dynamo DB, will not do incremental data transfer to AWS S3.

**Why this approach didn’t work**

Due to no direct support for Cassandra we had to drop this approach.

* **Debezium Server**
  + Debezium provides a ready-to-use application that streams change events from a source database to messaging infrastructure like Amazon Kinesis, Google Cloud Pub/Sub or Apache Pulsar.
  + Explored Debezium Server for the incremental transfer of data from Cassandra to S3 for creating data lake, we were able to work this out, from kinesis we have streamed data to s3 in data lake.

**Why this approach didn’t work**

As per the official documentation of Debezium Server, its currently in incubating state, i.e. exact semantics, configuration options etc. may change in future revisions that is why it is not recommend for the production purpose.

* **Debezium Connector for Cassandra**
  + The Cassandra connector can monitor a Cassandra cluster and record all row-level changes. The connector must be deployed locally on each node in the Cassandra cluster. The first time the connector connects to a Cassandra node, it performs a snapshot of all CDC-enabled tables in all keyspaces. The connector will also read the changes that are written to Cassandra commit logs and generates corresponding insert, update, and delete events. All events for each table are recorded in a separate kafka topic, where they can be consumed easily by applications and services.
  + As per the official documentation of Debezium Server, its currently in incubating state, i.e. exact semantics, configuration options etc. may change in future revisions that is why it is not recommend for the production purpose.

**Why this approach didn’t work**

As in this approach some setup also required from client side, so rejected due to some compliance.

* **AWS Data Pipeline**
* AWS Data Pipeline is a web service that helps you reliably process and move data between different AWS compute and storage services, as well as on-premises data sources, at specified intervals. With AWS Data Pipeline, you can regularly access your data where it’s stored, transform and process it at scale, and efficiently transfer the results to AWS services such as Amazon S3, Amazon RDS, Amazon DynamoDB, and Amazon EMR.
* Implemented the AWS Data Pipeline solution, which synchronize all the activities from launching EMR Cluster, running Job Flow for incremental data transfer, retry mechanism and sending the notification on successful run of the job or in case of some failure.

**Why this approach didn’t work**

As this service is not available in Mumbai region and the database is currently locate in Mumbai region, so it will incur us the inter-region data transfer cost. So, to eliminate this cost overhead of inter-region data transfer we have decide not to move forward with this approach, instead use some other services for the same purpose.

* **EMR Cluster through Lambda**
  + AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes.
  + Lambda is used for running the Job Flow on EMR for the incremental transfer of data from Cassandra to S3. Lambda is calling the EMR API to run the job flow on EMR.
  + But this API is asynchronous, so it basically starts the job flow and not wait for the results. And must check the status of the job manually. Due to which SNS notification can’t be implemented in this approach for the notification of Job failure or success.

To add more functionality like notification for the job success or failure, we have explored option for implementing Step Function

* **Step Function**
  + AWS Step Functions is a serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications. Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state. The output of one step acts as an input to the next. Each step in your application executes in order, as defined by your business logic.
  + A state machine is created which synchronizes the whole workflow from triggering the lambda for running the job flow on EMR, and then checks the status of the job after every 5 minutes followed by calling the SNS API for the notification about the status of Job.

Considering all aspects of use case, we finally selected this approach

**-Glue**

* + We were able to successfully run the same task in glue job i.e. incremental and one-time, still need to check the feasibility of the solution.

Currently not proceeding with this approach as need to check thoroughly and need to compare costing factor also.

IntelliJ IDEA :-

It’s an IDE written in java. The Spark Cassandra connector is compatible with Scala 2.11 and 2.12 hence Scala is preferred choice of language.

The algo is implemented in Scala in IntelliJ IDEA. It consists of two classes, one for one-time data transfer and the other for incremental data transfer. The code will require spark dependencies which could be added in sbt file.

Text

Description automatically generated

After building the project, create jar file of the project.

Build-> build artifacts ->build

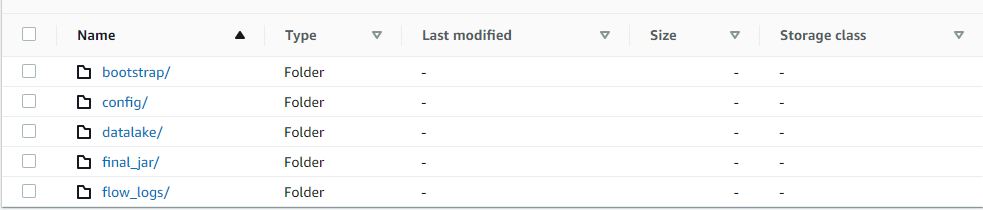
The jar file will be created in out folder. This jar file must be uploaded in final\_jar folder in AWS S3.

Services used: -

1. Amazon S3
2. AWS Step function
3. AWS Lambda
4. AWS EMR
5. AWS SNS
6. AWS GLUE

In S3 create 4 folders: -

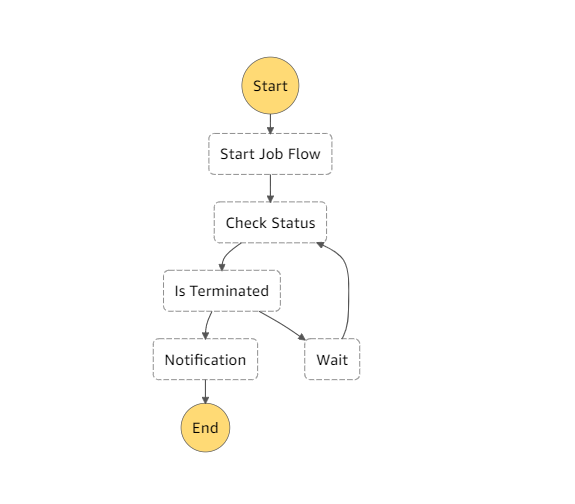
* Bootstrap – Contains bootstrap action that will run on EMR cluster before Amazon EMR installs the applications
* Flow\_logs – keeps track of all the logs
* Final\_jar – Keeps the jar file of the code that will migrate data
* Config - Contains csv file that has the required table with partition column that has to be migrated.



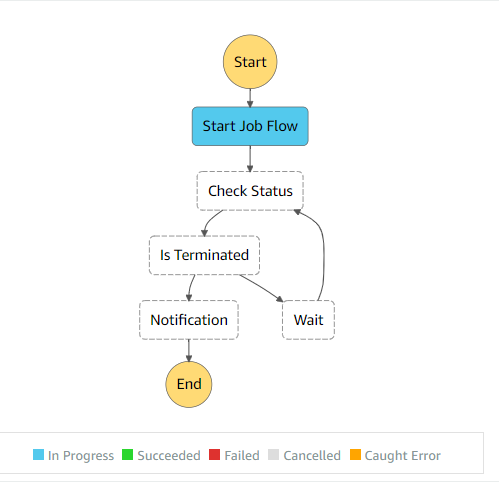
Step function: -

It is used to orchestrate the Lambda function and SNS topics. On execution the step function will trigger the lambda function.

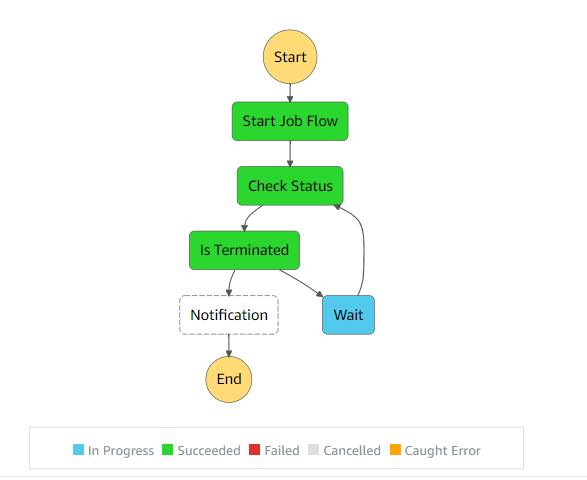
State Machine Flow

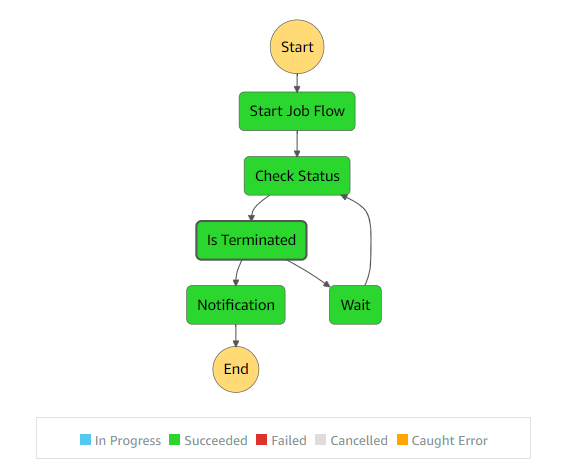


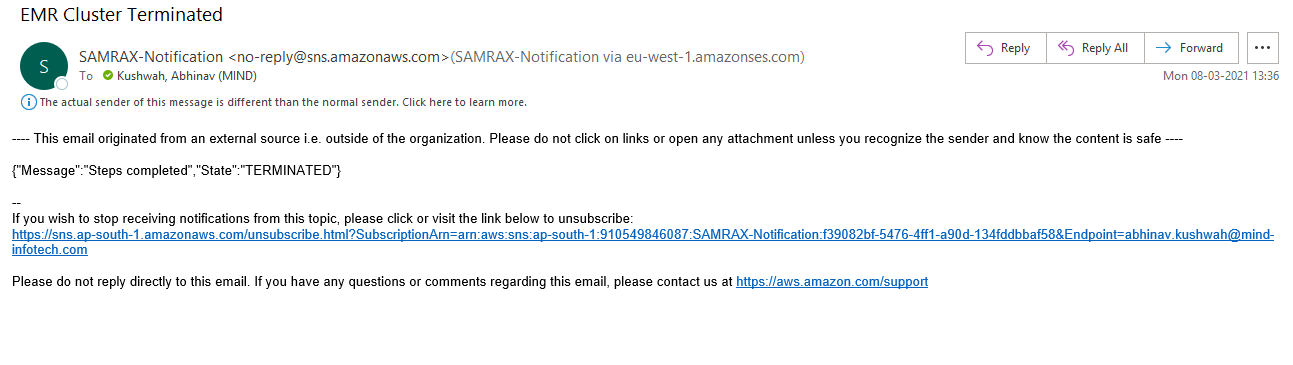
JobFlow in Progress



In Wait Status, will check status after every 1 minute.



State machine completed the execution

Email Notification on Job Completion

In Lambda: -

The Lambda function which is written in python will create transient cluster using Boto3 API and will run the spark job. In the function

1. Set the log path and config file path.
2. Use RunJobFlow API to launch transient cluster to run the spark job.
3. In the RunJobFlow API set the required instance configuration.
4. Set up application as Hadoop and spark.
5. Set file path for bootstrap action and save
6. Deploy the lambda function and test.

The function will set up transient emr cluster which will run spark jobs that will migrate the data and terminate after running all steps. The one-time class will run only once, and the incremental class will run on daily basis.

There is another lambda function that will check the status of emr.

The data migrated will be stored in bucket in datalake folder.

Graphical user interface, text, application

Description automatically generated

AWS Glue: -

In order to query on datalake the tables are created using glue crawler.

In AWS glue console add crawler.

Graphical user interface, text, application, email

Description automatically generated

1. Give crawler a name.
2. Choose **Crawler source type as** Data stores and **Repeat crawls of S3 data stores as** Crawl all folders and click next.
3. **Choose a data store** as S3 **and** give path of the S3 datalake.

Graphical user interface, text, application, email

Description automatically generated

1. Schedule as per requirement and configure output of the crawler to a database (new database can be created in this step).
2. Review all steps and finish. Then run the crawler.
3. Tables will be created as per the number of folders in datalake.

**Test Cases**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case No.** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Status (pass/fail)** |
| #1 | Cassandra Database is accessible | Account details and password | Able to connect to Cassandra DB | Able to connect to Cassandra DB | Pass |
| #2 | The table specified in the config file is migrated to S3 bucket | Cassandra tables | Mentioned table in config file is present in S3 | Mentioned table in config file is present in S3 | Pass |
| #3 | Partitioned specified in config file is created and store each partition in separate partition folder in same location | Cassandra tables with required partition column | Each partition in separate partition folder in same location | Each partition in separate partition folder in same location | Pass |
| #4 | Count of records of tables in intact | Cassandra tables | Count of record at Cassandra DB is equal to count of record in Athena | Count of record at Cassandra DB is equal to count of record in Athena | Pass |
| #5 | Daily update of tables in S3 | S3 folders | New folders created depending on partition and update | New folders created depending on partition and update | Pass |

Solution Summary - Training & Handover Notes

**About Challenge(s)**

Samvardhana Motherson Hamakyorex Engineered Logistics Limited (SAMRX) wants to get business insights from various data being ingested from many sensors like Door,Rollr,Driver,TPS etc. Currently 20 Vehicles are being used and this will increase to 50 vehicles and having target of 1500 vehicles by 2023. Considering huge volume and velocity of data, Data Lake need to be created.

As of now it involves ingestion of 7 GB of one-time data & 500 MB of incremental data from Cassandra DB to AWS cloud.

Following were few technical challenges

* Selection of approach for ingesting data from Cassandra to s3
* Selection of key to be used for partition
* Finding the perfect connector to make a connection to Cassandra from EMR
* Submitting job to EMR.
* Selection of no of nodes and configuration for Core Nodes.

**Proposed Solution**

* Ingestion of data from Cassandra Database to S3.
* Transformed Cassandra Database keyspace data into parquet format and did partitioning.
* Running Amazon EMR on incremental data daily, partitioning and storing it in parquet format.
* Configured CloudWatch to start State Machine on daily basis.
* Created EMR Cluster for one-time Data Ingestion from Cassandra to S3.
* Created Glue Crawler to create schema for the respective data lake and scheduled on monthly basis.
* Configure Athena to query parquet data in S3.

**AWS Services used**

* AWS Lambda
* AWS EMR
* Amazon S3
* AWS Glue
* Amazon Athena

**Solution Outcome**

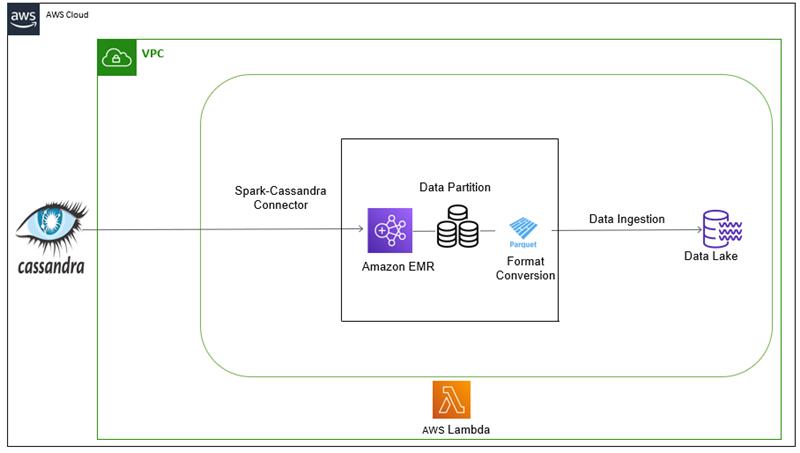
* Ingested 7 GB of data one-time and incremental from Cassandra to S3
* Created Data Lake in S3 followed by creating schema for Data Lake in Glue.
* Did analysis on data in S3 through running queries on Athena.

Increased profit due to fewer number of call backs rate for repair after sale

 Reduction in production time due to timely detection of defective parts

 Overall customer satisfaction increased which resulted in many intangible benefits e.g. helps to increase trust in brand which eventually leads to customer loyalty

**Architecture Diagram**



**How AWS services helped in creating Data Lake**

**AWS Lambda to handle the backend API calls**

It helped to initialize and validate the input and acted as the backend of the whole task. AWS Lambda lets us run code without provisioning or managing servers. Also, it helped to connect with various AWS API’s to acquire various insights from the inputs.

**Amazon EMR**

Amazon EMR is a managed cluster platform that simplifies running big data frameworks such as Apache Spark. It transforms and move large amounts of data into and out of other databases.

**Amazon S3 to store parquet files**

It is an object storage service that offers industry-leading scalability, data availability, security, and performance. Parquet is columnar storage format. It will allow Athena to only query and process the required columns and ignore the rest

**AWS Lake Formation**

It is a service that makes it easy to set up a secure data lake in days.

**AWS Glue to load partitioned data**

It is a fully managed extract, transform, and load (ETL) service that made it easy for us to prepare and load the processed, partitioned data for analytics. One can create and run an ETL job by simply pointing AWS Glue to the data stored in S3 bucket, which is then immediately searchable, query-able, and available for ETL.

**Amazon Athena to query the partitioned data for insights**

It is an interactive query service that made it easy to analyze the output parquet data stored in Amazon S3 using standard SQL. It is serverless, so there is no infrastructure to manage, simply point to your data in Amazon S3, define the schema, and start querying using standard SQL and the results were delivered within seconds.