**AWS PROJECT**

**AUTOMATED DATA INGESTION TRANSFORMATION, AND ANALYTICS PIPELINE USING AWS SERVICES**

Abstract:

This project aims to automate the conversion of CSV files to Parquet format and facilitate seamless querying of data using AWS Glue, Lambda, S3, Athena, and SNS services. The project involves eight CSV files that need to be transformed and stored in an S3 bucket. The conversion process is carried out by an AWS Glue job, which utilizes a predefined database and crawler configuration. A Lambda function automates the conversion process, ensuring efficient and timely execution.

Initially, the eight CSV files are uploaded to the S3 bucket, where they are organized into subfolders for better management. A Glue job, pre-configured with a database and crawler, is triggered automatically when new files are added to the bucket. The Glue job seamlessly converts the CSV files to the highly optimized Parquet format, significantly improving data processing and storage efficiency.

To streamline the entire process, a Lambda function monitors the S3 bucket and initiates the Glue job whenever new files are detected. The Lambda function ensures the automation of the conversion process, eliminating the need for manual intervention and improving overall efficiency.

After the Glue job is completed, a notification is sent to subscribers using the Simple Notification Service (SNS). Subscribers receive timely updates regarding the status of the Glue job, ensuring effective monitoring and management of the conversion process.

The Glue job, upon successful completion, generates eight tables in Parquet format. These tables can be queried using Amazon Athena, a serverless interactive query service. Athena provides seamless integration with the Parquet files, allowing users to run complex SQL queries and perform data analysis without the need for infrastructure provisioning.

Furthermore, Athena also incorporates the log details and file names, ensuring comprehensive visibility into the conversion process. This enables users to track and monitor the entire data transformation workflow easily.

In conclusion, this project showcases an automated pipeline for converting CSV files to Parquet format, enabling efficient data processing and analysis. By leveraging the power of AWS Glue, Lambda, S3, Athena, and SNS services, users can streamline their data conversion process, automate notifications, and perform sophisticated querying and analysis using Parquet files with ease.

Introduction:

**Purpose of the Document:**

The purpose of this document is to provide an abstract or summary of a project that involves automating the conversion of CSV files to Parquet format using AWS services such as Glue, Lambda, S3, Athena, and SNS. The document aims to highlight the key components of the project, including its objectives, the technologies involved, and the benefits it offers.

**Project Overview:**

This project automates the conversion of eight CSV files to Parquet format using AWS Glue, Lambda, S3, Athena, and SNS. The CSV files are uploaded to an S3 bucket and transformed by a Glue job, improving data processing and storage efficiency. A Lambda function triggers the Glue job and notifies subscribers of the job's completion using SNS. The transformed data is crawled in the destination bucket, allowing seamless querying with Athena. This project streamlines the conversion process, automates notifications, and facilitates efficient data analysis with Parquet files.

**This project utilizes the following AWS services:**

**1. AWS Glue:** AWS Glue is a fully managed extract, transform, and load (ETL) service that simplifies the process of preparing and transforming data for analytics. It is used in this project to convert the CSV files to Parquet format and optimize the data for efficient processing and storage.

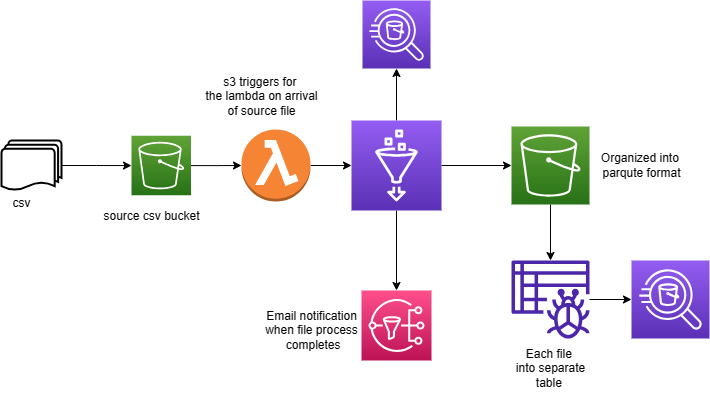
**2. AWS Lambda:** AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers. In this project, Lambda functions are used to automate the conversion process by monitoring the S3 bucket for new files and triggering the Glue job. Lambda functions are also utilized to crawl the destination bucket after the conversion is completed.

3. **Amazon S3:** Amazon Simple Storage Service (S3) is an object storage service that offers industry-leading scalability, durability, and security. It is used as the storage infrastructure for the CSV files, as well as the destination bucket for the transformed Parquet files.

**4. Amazon Athena:** Amazon Athena is an interactive query service that allows you to analyze data directly in Amazon S3 using standard SQL. In this project, Athena is used to query the Parquet files and perform data analysis without the need for infrastructure provisioning.

1. **Amazon SNS:** Amazon Simple Notification Service (SNS) is a fully managed messaging service that enables the sending of notifications to subscribers. In this project, SNS is used to send notifications to subscribers about the completion of the Glue job, providing timely updates on the status of the conversion process.

**Project Architecture:**

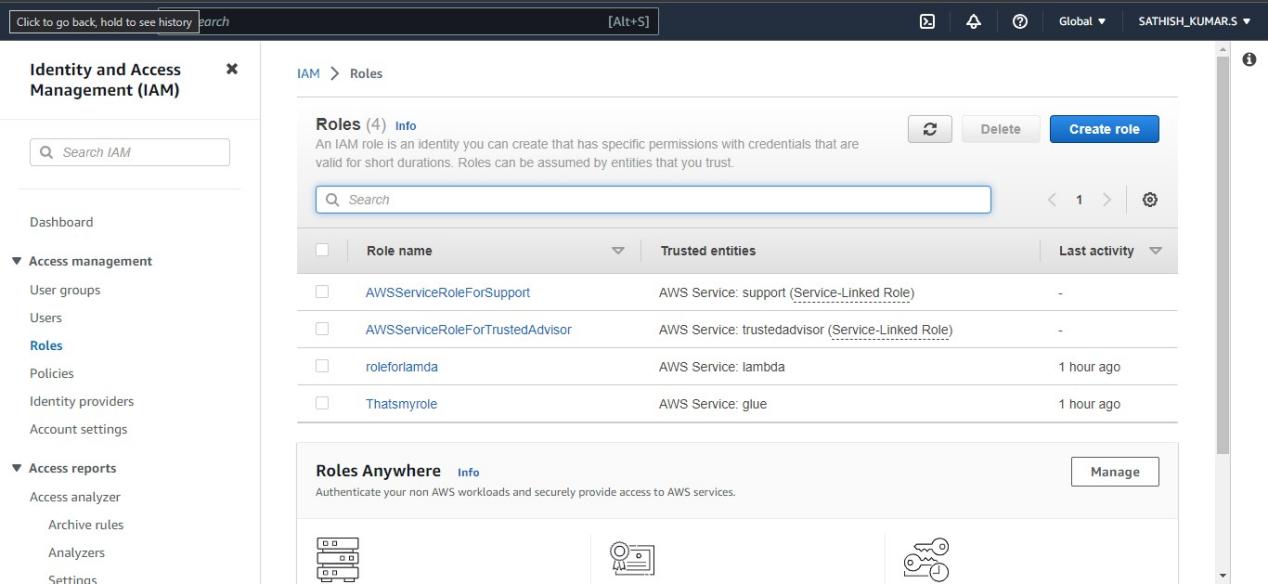


**Implementation of Project Architecture :**

1. **Create IAM Role :**

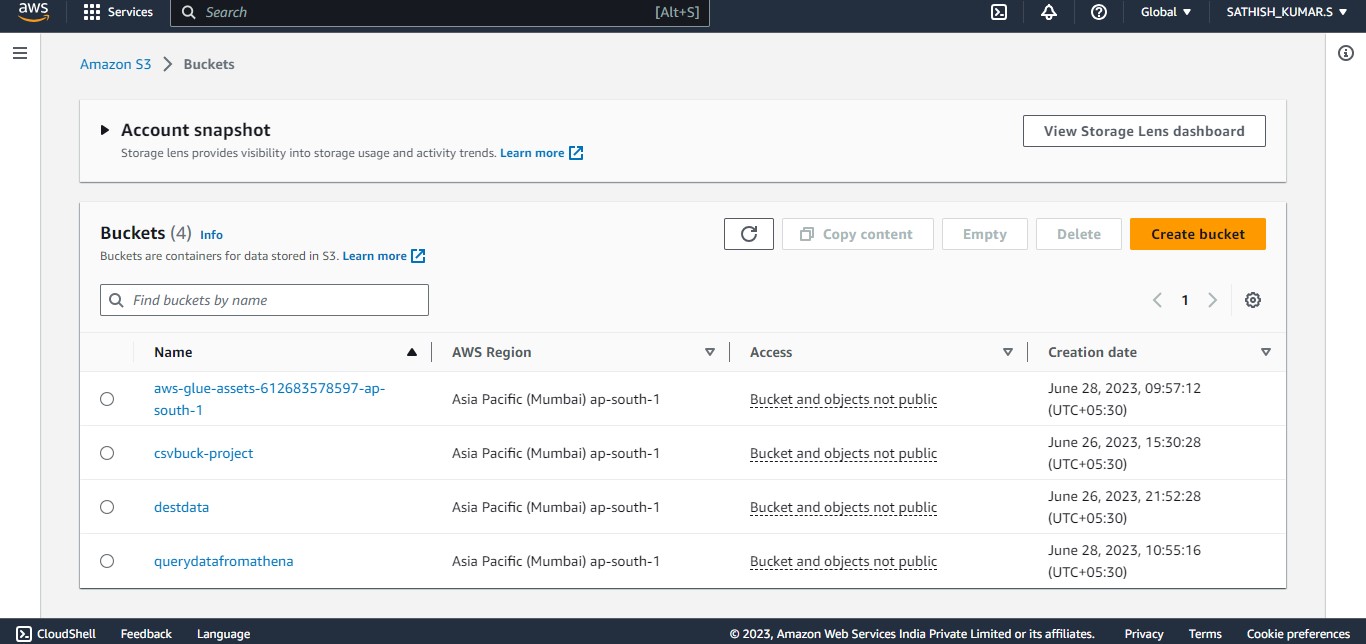
AWS Identity and Access Management (IAM) is a web service provided by Amazon Web Services (AWS) that allows you to manage access to AWS resources securely. IAM enables you to create and manage AWS users, groups, and permissions to control who can access specific AWS resources and what actions they can perform

**Users:** IAM users are entities within your AWS account that represent individual people or applications requiring access to AWS resources. Each user is assigned a unique set of security credentials (access key ID and secret access key) for programmatic access and a password or a password policy for AWS Management Console access.



I have created Two roles for crawler and glue. Which have spesific policies life lambda ,athena,sns,glueand s3.

**2. Create an S3 bucket:** Set up an S3 bucket to store the CSV files.Amazon S3 (Simple Storage Service) is a highly scalable and durable cloud storage service provided by Amazon Web Services (AWS).



**3. Organize subfolders:** Create subfolders within the S3 bucket for crawler to crawl

**SUBFOLDERS** :

1. ATTENDANCE DETAILS

****2.CALENDER

3.CLASS DETAILS

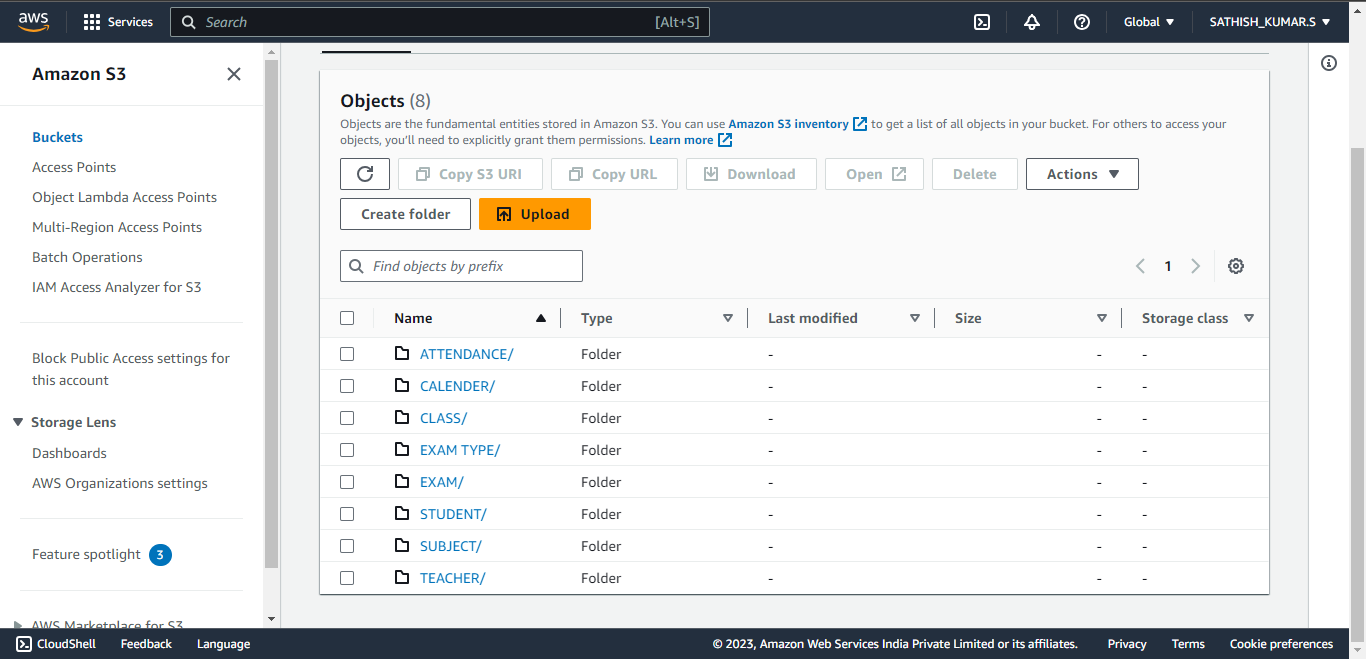
4. EXAM DETAILS

5. EXAM TYPE

6. STUDENT

7. SUBJECT

8. TEACHER DETAILS



The purpose of creating this bucket is to ingest our raw CSV files into it. Eight subfolders has been created inside this bucket to ingest 8 CSV files separately.

1. **Destination Bucket:** Create new bucket for result of conversion. Same as subfolders created for the source bucket

1. ATTENDANCE DETAILS

2.CALENDER

3.CLASS DETAILS

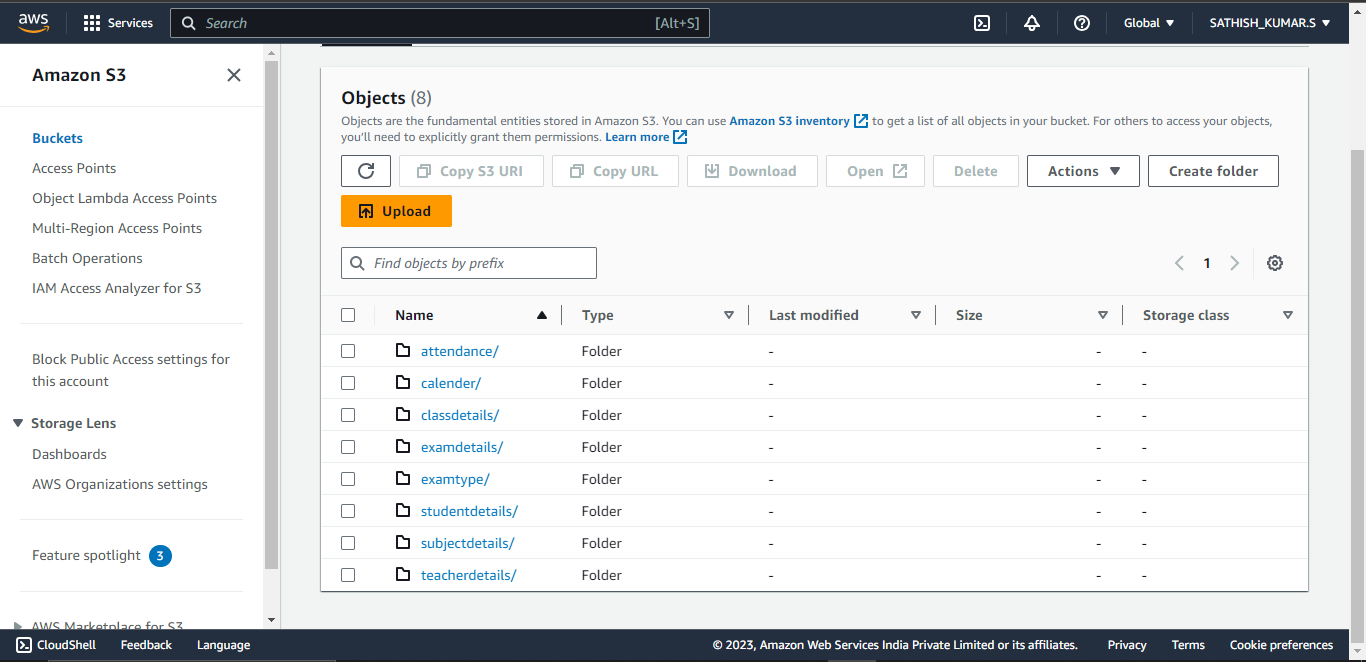
4. EXAM DETAILS

5. EXAM TYPE

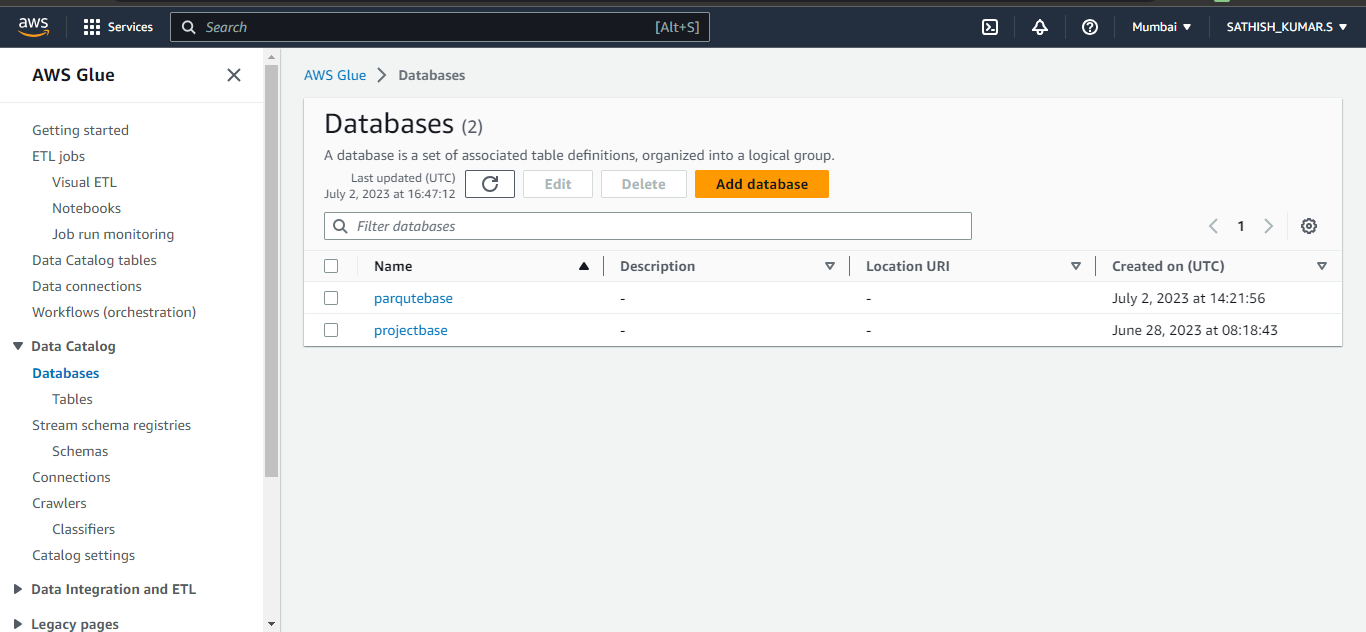
6. STUDENT

7. SUBJECT

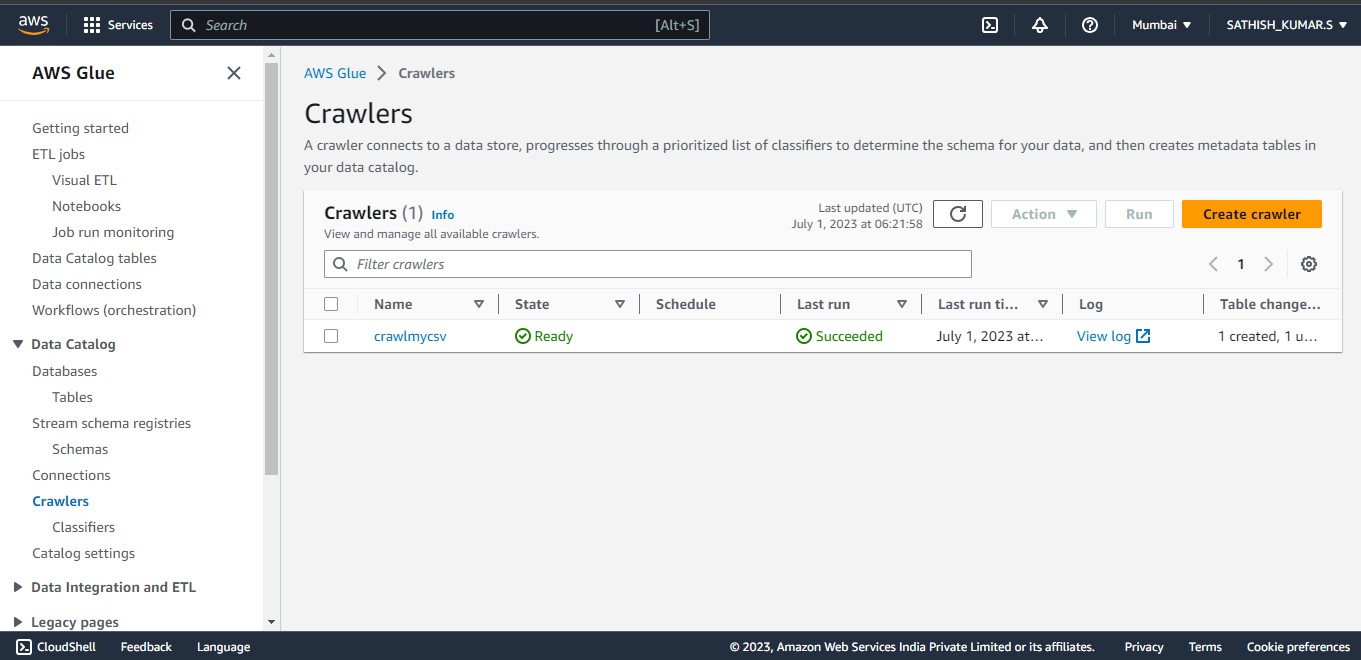
8. TEACHER DETAILS



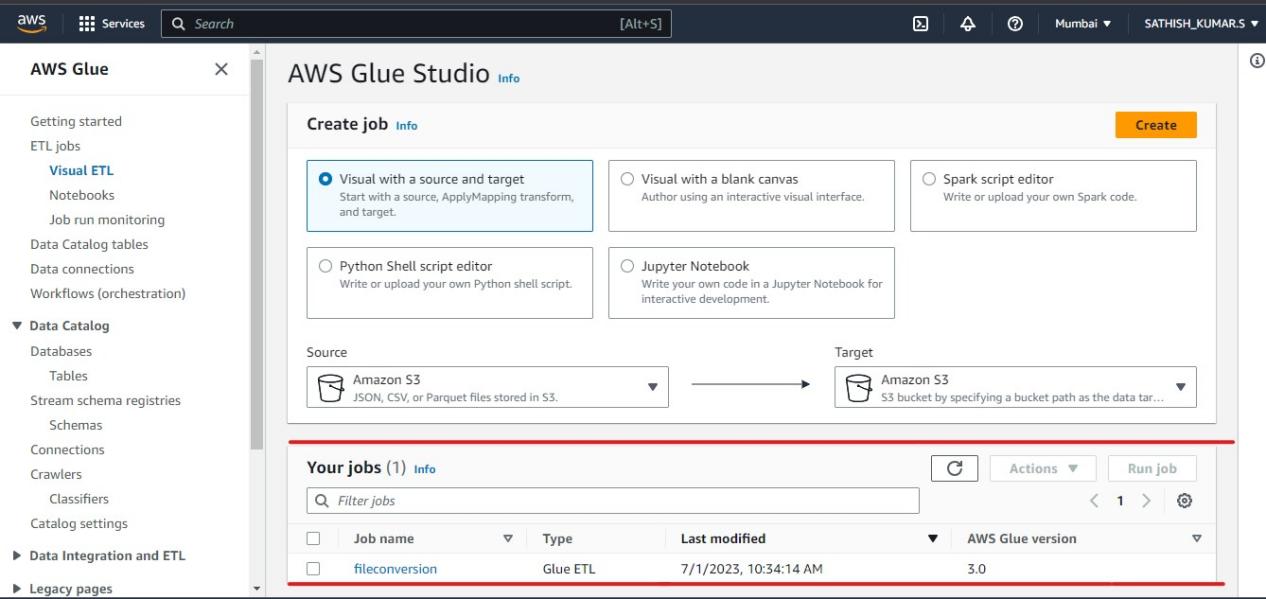
1. **Create Database:** Create two databases. Projectbase is for transformation and other process. We will use Parqutebase to work with the parqute file athena.



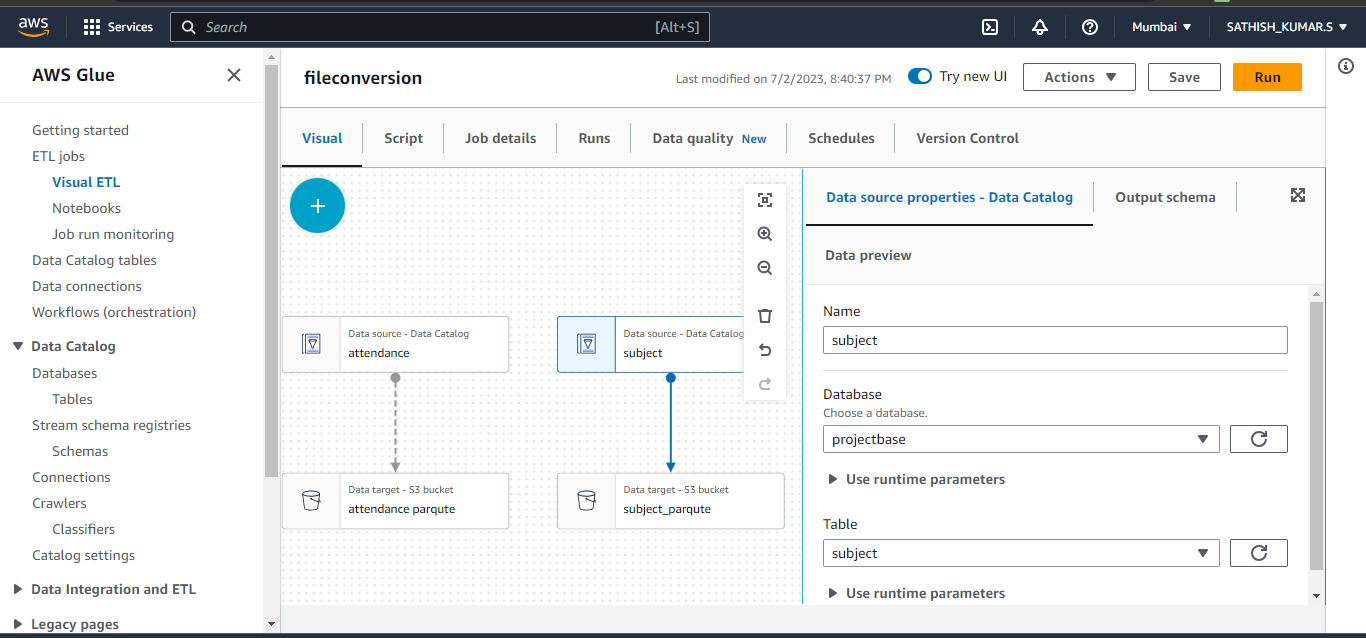
1. **Create Crawler:** Create a crawler with the role mentioned above and give the parameters required. AWS Glue is a fully managed extract, transform, and load (ETL) service provided by Amazon Web Services (AWS). Crawlers are responsible for automatically scanning and cataloging data from different data sources, such as Amazon S3, databases, and data warehouses. They infer the schema and structure of the data and populate the Data Catalog with the metadata.



1. **Create Glue-Job :** Create a glue job for converting the CSV files to PARQUTE file format. give destinatio bucket as the output bucket. Where you want to save the PARQUTE files.



1. **Glue-Job:** Create glue job using ETL visual or using script. The purpose of this glue job is to convert the CSV file to PARQUTE file and later we can use it to crawl the data.



**Script of Glue-Job :**

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB\_NAME"])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args["JOB\_NAME"], args)

# Script generated for node attendance

attendance\_node1688309229554 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="attendance",

transformation\_ctx="attendance\_node1688309229554",

)

# Script generated for node Calender

Calender\_node1688309413751 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="calender",

transformation\_ctx="Calender\_node1688309413751",

)

# Script generated for node class

class\_node1688309420898 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="class",

transformation\_ctx="class\_node1688309420898",

)

# Script generated for node exam

exam\_node1688309424173 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="exam",

transformation\_ctx="exam\_node1688309424173",

)

# Script generated for node examtype

examtype\_node1688309426714 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="exam\_type",

transformation\_ctx="examtype\_node1688309426714",

)

# Script generated for node studentdetails

studentdetails\_node1688309428901 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="student",

transformation\_ctx="studentdetails\_node1688309428901",

)

# Script generated for node subject

subject\_node1688309431384 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="subject",

transformation\_ctx="subject\_node1688309431384",

)

# Script generated for node teacher

teacher\_node1688309433763 = glueContext.create\_dynamic\_frame.from\_catalog(

database="projectbase",

table\_name="teacher",

transformation\_ctx="teacher\_node1688309433763",

)

# Script generated for node attendance parqute

attendanceparqute\_node1688309234742 = glueContext.getSink(

path="s3://destdata/parquet files/attendance/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="attendanceparqute\_node1688309234742",

)

attendanceparqute\_node1688309234742.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="attendance"

)

attendanceparqute\_node1688309234742.setFormat("glueparquet")

attendanceparqute\_node1688309234742.writeFrame(attendance\_node1688309229554)

# Script generated for node calender\_parqute

calender\_parqute\_node1688309531682 = glueContext.getSink(

path="s3://destdata/parquet files/calender/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="calender\_parqute\_node1688309531682",

)

calender\_parqute\_node1688309531682.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="calender"

)

calender\_parqute\_node1688309531682.setFormat("glueparquet")

calender\_parqute\_node1688309531682.writeFrame(Calender\_node1688309413751)

# Script generated for node class\_parqute

class\_parqute\_node1688310034644 = glueContext.getSink(

path="s3://destdata/parquet files/classdetails/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="class\_parqute\_node1688310034644",

)

class\_parqute\_node1688310034644.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="classdetails"

)

class\_parqute\_node1688310034644.setFormat("glueparquet")

class\_parqute\_node1688310034644.writeFrame(class\_node1688309420898)

# Script generated for node exam\_parqute

exam\_parqute\_node1688310146213 = glueContext.getSink(

path="s3://destdata/parquet files/examdetails/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="exam\_parqute\_node1688310146213",

)

exam\_parqute\_node1688310146213.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="examdetails"

)

exam\_parqute\_node1688310146213.setFormat("glueparquet")

exam\_parqute\_node1688310146213.writeFrame(exam\_node1688309424173)

# Script generated for node examtype

examtype\_node1688310237710 = glueContext.getSink(

path="s3://destdata/parquet files/examtype/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="examtype\_node1688310237710",

)

examtype\_node1688310237710.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="examtype"

)

examtype\_node1688310237710.setFormat("glueparquet")

examtype\_node1688310237710.writeFrame(examtype\_node1688309426714)

# Script generated for node Amazon S3

AmazonS3\_node1688310348884 = glueContext.getSink(

path="s3://destdata/parquet files/studentdetails/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="AmazonS3\_node1688310348884",

)

AmazonS3\_node1688310348884.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="studentdetails"

)

AmazonS3\_node1688310348884.setFormat("glueparquet")

AmazonS3\_node1688310348884.writeFrame(studentdetails\_node1688309428901)

# Script generated for node subject\_parqute

subject\_parqute\_node1688310428546 = glueContext.getSink(

path="s3://destdata/parquet files/subjectdetails/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="subject\_parqute\_node1688310428546",

)

subject\_parqute\_node1688310428546.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="subject"

)

subject\_parqute\_node1688310428546.setFormat("glueparquet")

subject\_parqute\_node1688310428546.writeFrame(subject\_node1688309431384)

# Script generated for node teacher\_parqute

teacher\_parqute\_node1688310550288 = glueContext.getSink(

path="s3://destdata/parquet files/teacherdetails/",

connection\_type="s3",

updateBehavior="LOG",

partitionKeys=[],

enableUpdateCatalog=True,

transformation\_ctx="teacher\_parqute\_node1688310550288",

)

teacher\_parqute\_node1688310550288.setCatalogInfo(

catalogDatabase="parqutebase", catalogTableName="teacher"

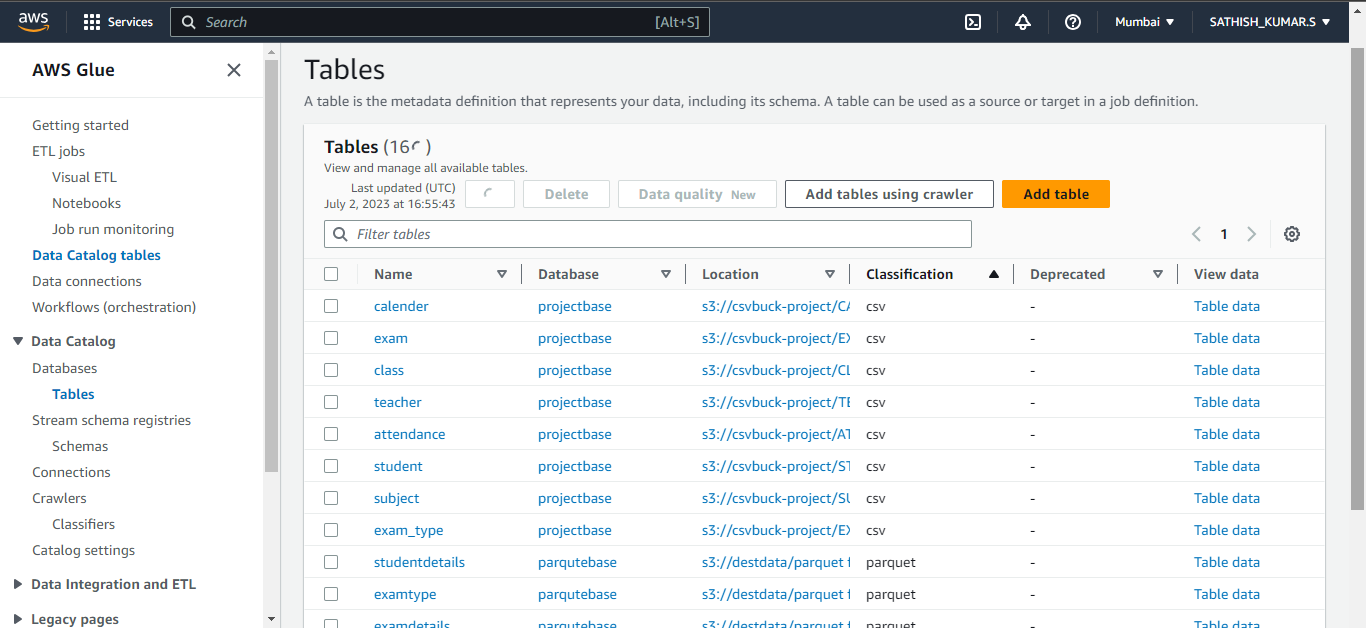
)

teacher\_parqute\_node1688310550288.setFormat("glueparquet")

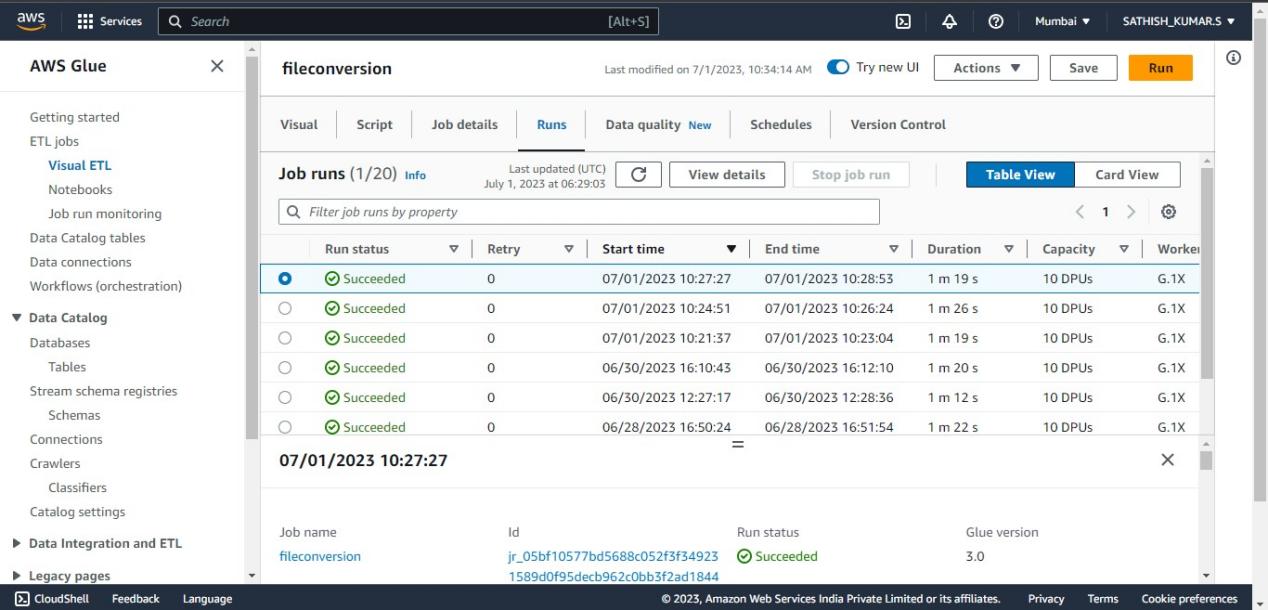
teacher\_parqute\_node1688310550288.writeFrame(teacher\_node1688309433763)

job.commit()

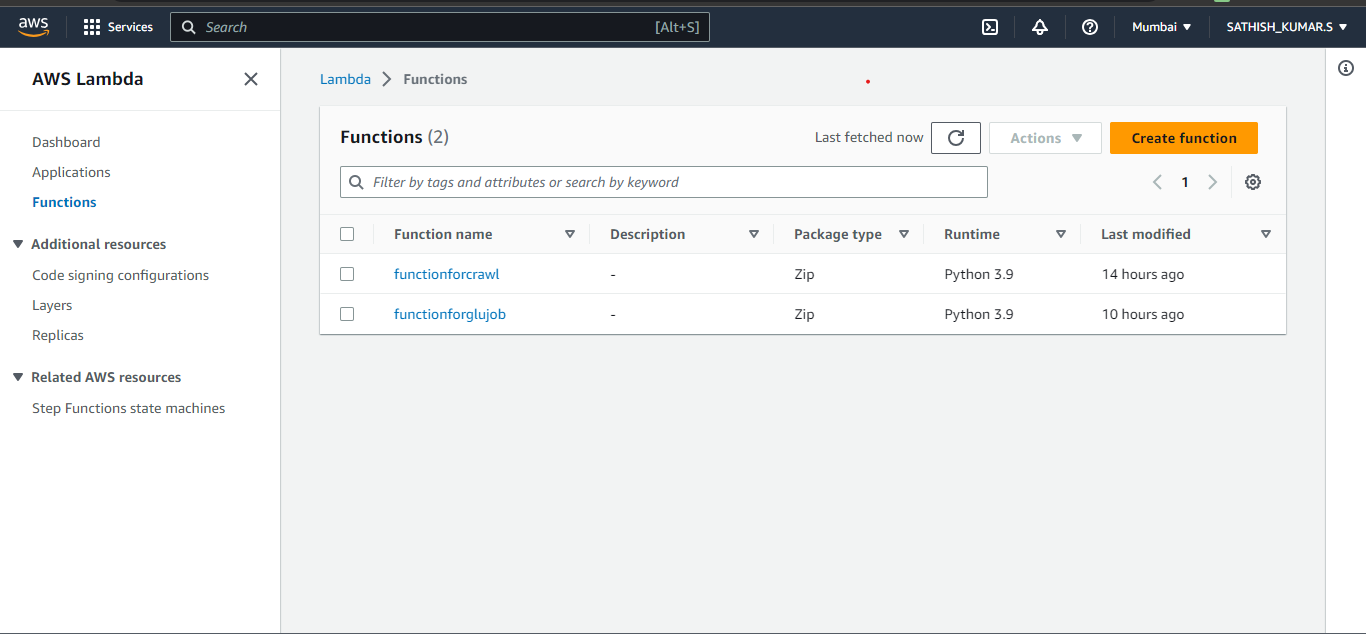
1. **Table creation:** Table will be created when the crawler crawls the CSV file in the source bucket and then it store in parqute database catalog.



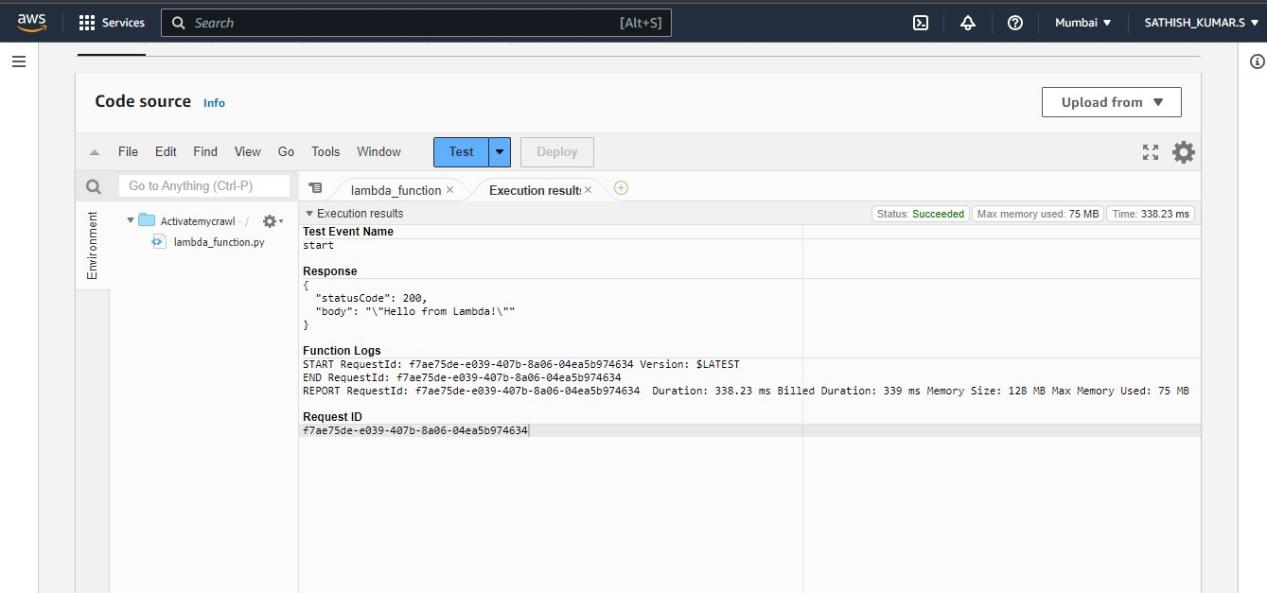
1. **Gluejob run:** The below result represents the logs like start and stop time of Glue-job. And success and failure of the glue jobs can be seen below.



1. **Lambda Functions**: I have created two lambda functions. Function for crawl is used to automate the crawler. When a file is uploaded into the source S3 bucket, this lambda function will automates the crawler.



1. **Function for crawl :** The below Screenshot represents the code successfully executed. Where lambda function automates the crawl.



**Code of functionforcrawl :**

import json

import boto3

glue=boto3.client('glue');

def lambda\_handler(event, context):

# TODO implement

response = glue.start\_crawler(

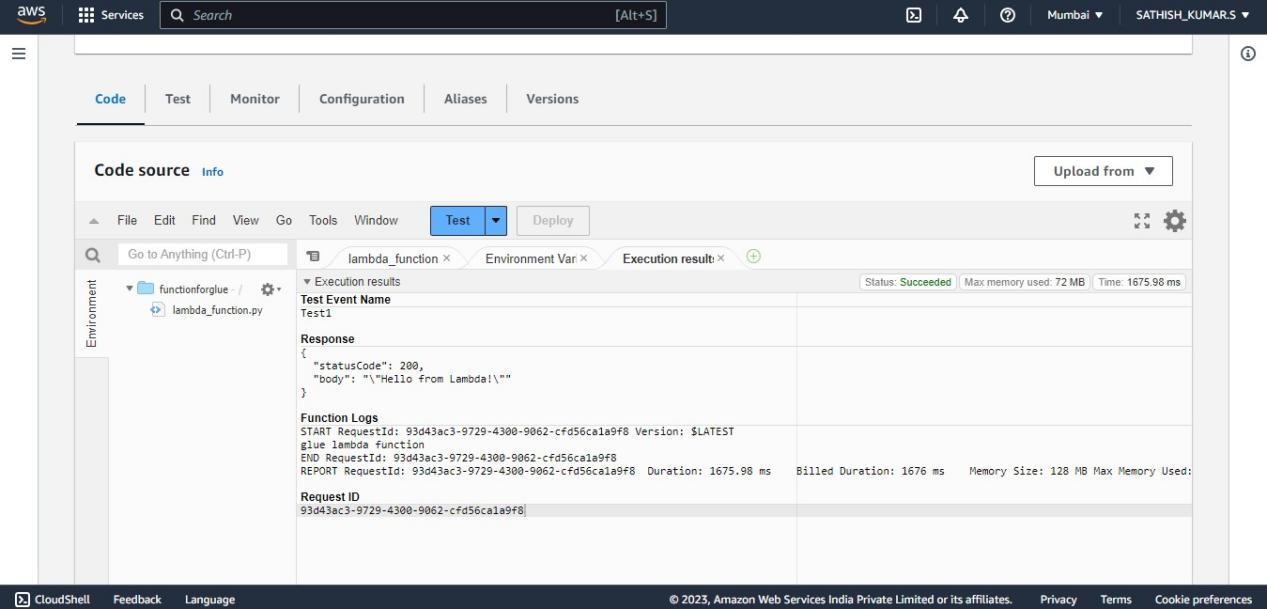
Name='crawlmycsv')

return {

'statusCode': 200,

'body': json.dumps('yeah i started !')}

1. **Function for glue :** The below Screenshot represents the code successfully executed. Where lambda function automates the gluejob.



**Code of functionforglue :**

import json

import boto3

def lambda\_handler(event, context):

client = boto3.client('glue')

client.start\_job\_run(

JobName = 'fileconversion' )

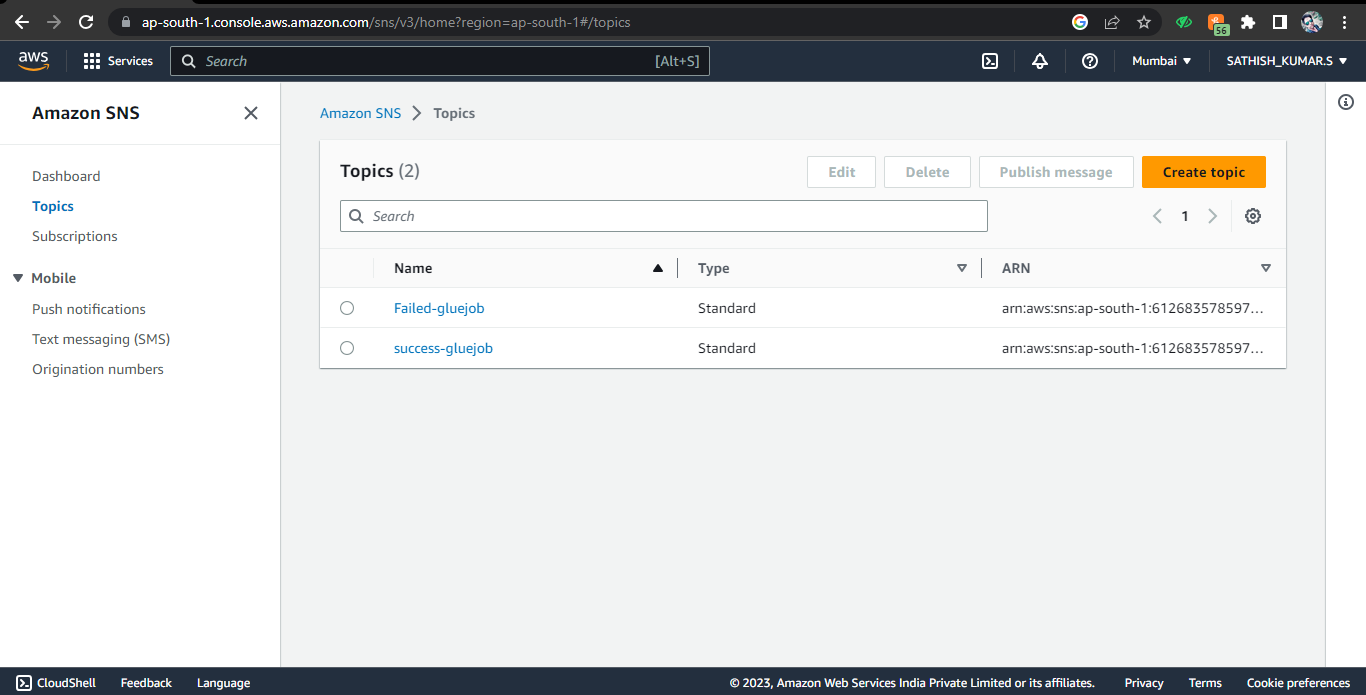
print("glue lambda function")

return{

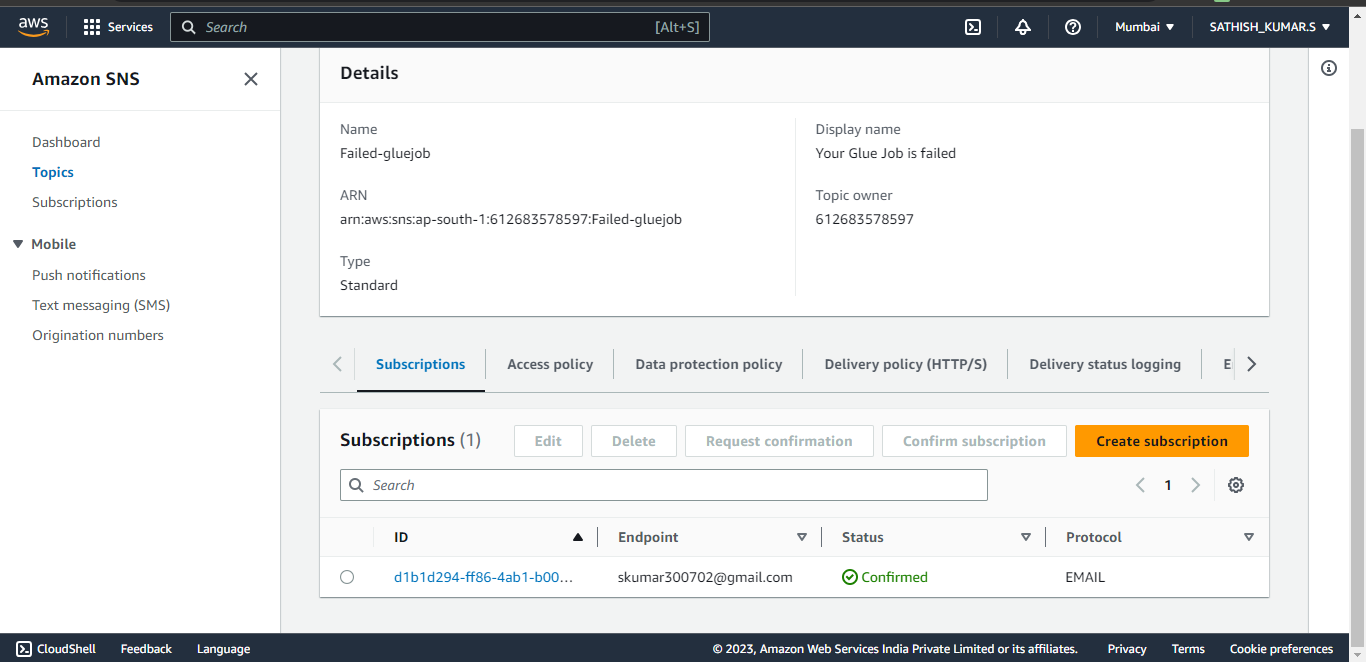
'statusCode': 200,

'body': json.dumps('Hello from Lambda!')}

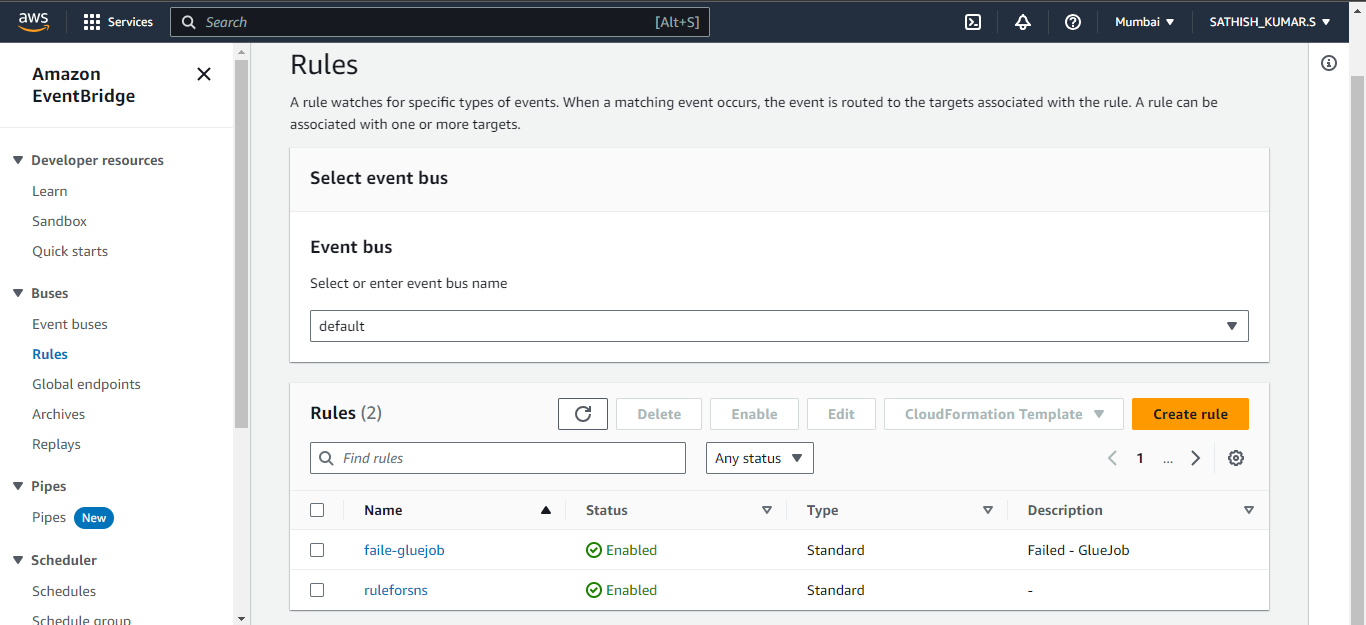
1. **Create SNS topic**: Create two sns topic for success and failure notifcation.



1. Add subsciption: I have added subscription to my personal mail ID. Where the SNS email will be send to the specified mail ID.



1. Create Rule in Eventbridge: Create two events for success and failure this will triggers the SNS to send mail on such cases.



**JSON Rule for failed Glue-job**

{

"source": ["aws.glue"],

"detail-type": ["Glue Job State Change"],

"detail": {

"jobName": ["fileconversion"],

"state": ["FAILED"]

}

}

**JSON Rule for success Glue-job**

{

"source": ["aws.glue"],

"detail-type": ["Glue Job State Change"],

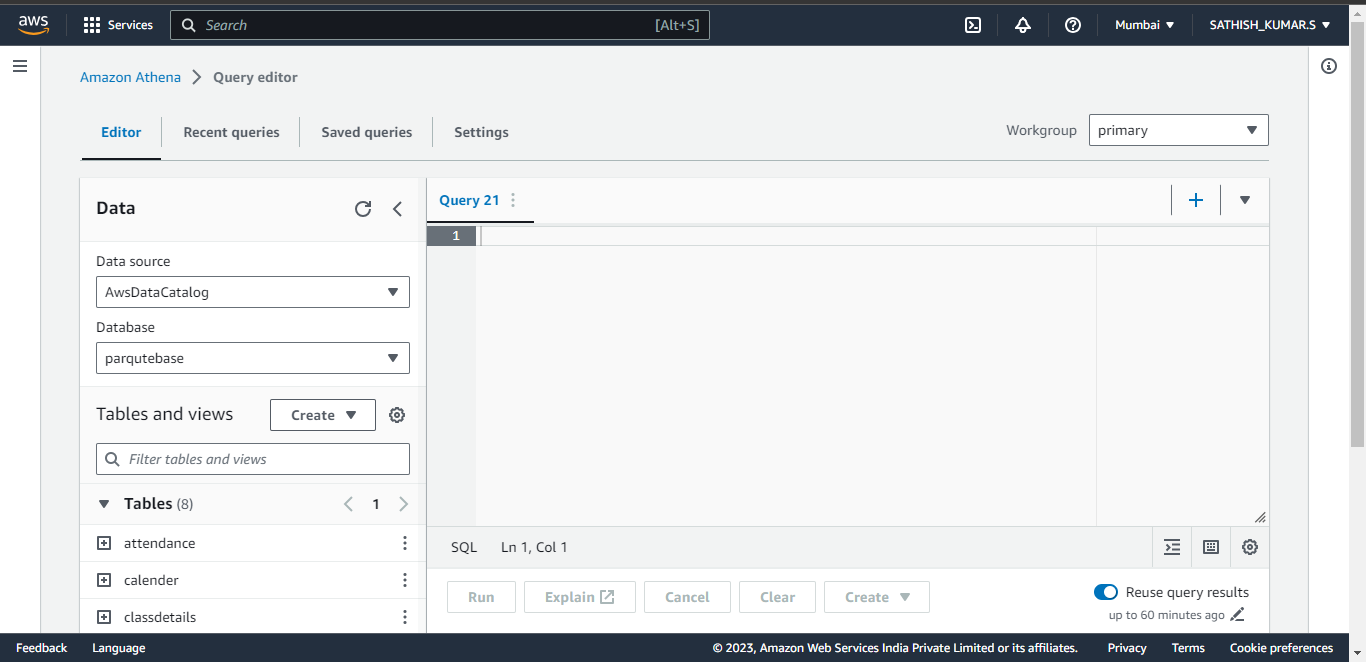
"detail": {

"state": ["Succeeded"]

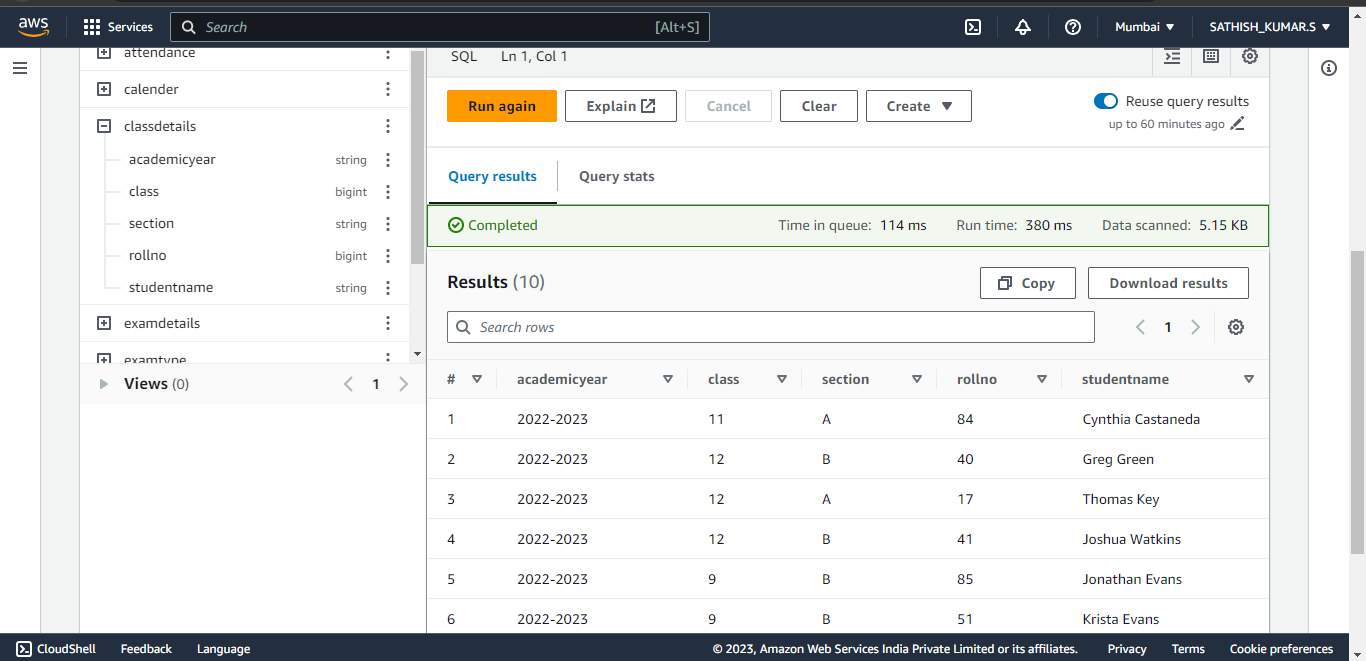
}

}

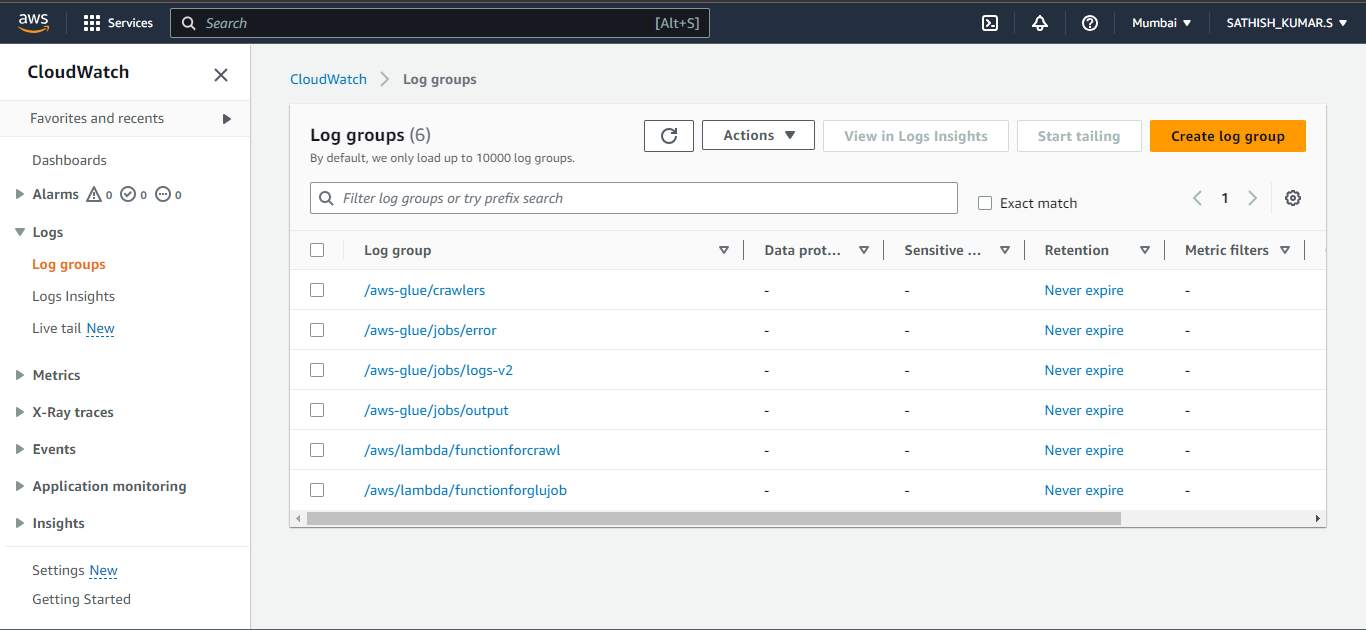
1. **Query in Athena:** Athena can be able to query the crawled files. Here are the catalog tables which hs been crawled and stored in different database. We can able to querry in this database were also the parqute files are stored in the destdata bucket.



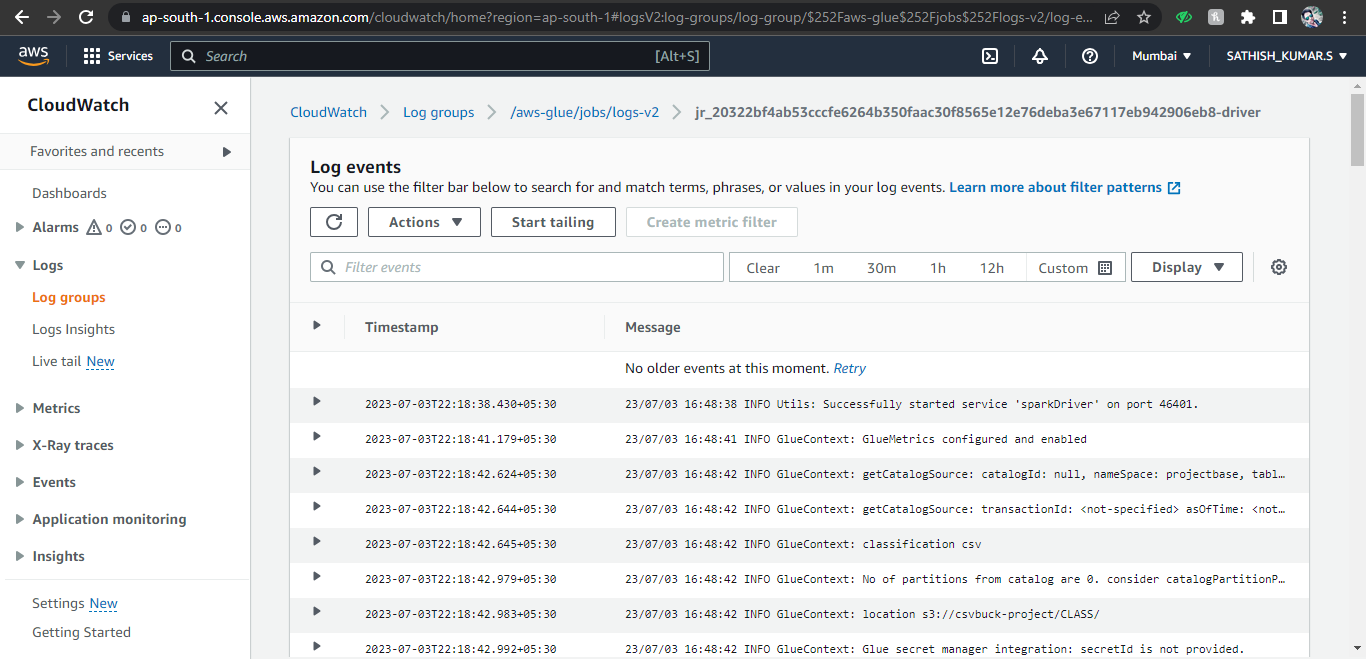
We can able to query in athena. And the datatypes also changed according to the metadata. We can able to analyze the data for further information according to the client need.



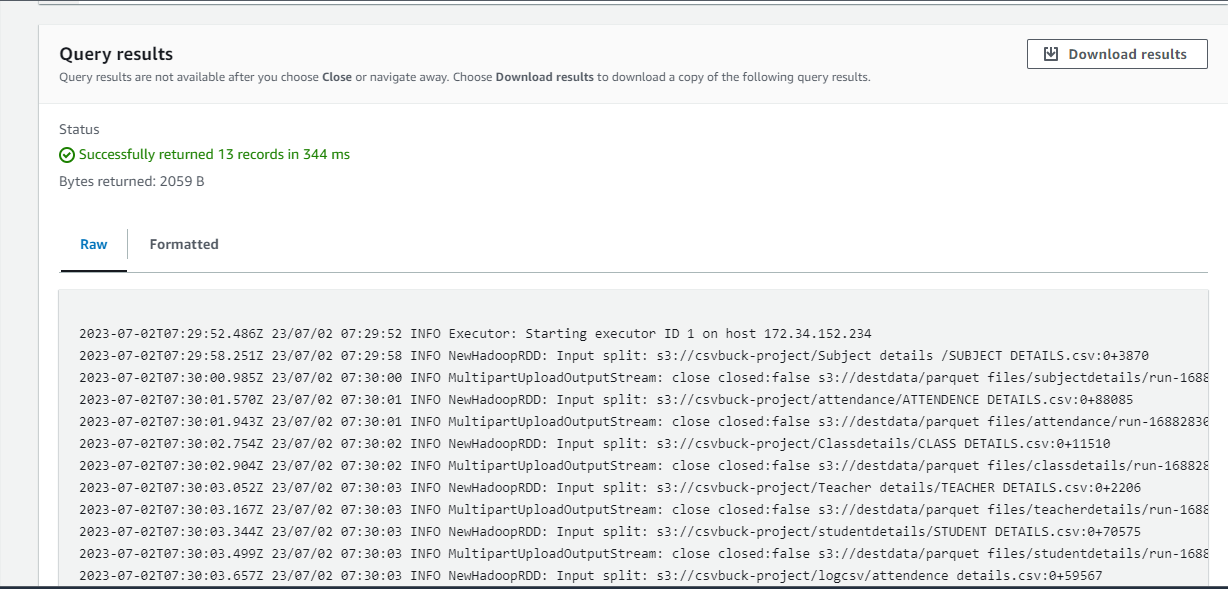
1. **Cloudwatch logs :** Cloudwatch is the service which monitors the activity of AWS services. Here are the logs of the service we used Gluejob, Crawler, lambda. We can export these logs into csv files for further information and references. It contains the informations like start and stop time of gluejob, file name etc.,



1. **Log events :** Here we can able to notice the timestamps and the log messages which are the event logs of gluejob.



1. **Query Results :** Here we can able to view and query the logs. We can also able to separate these log messages into tables and columns of data in which we can able to query.



**Conclusion:**

This project showcases an automated pipeline for converting CSV files to Parquet format, enabling efficient data processing and analysis. By leveraging the power of AWS Glue, Lambda, S3, Athena, and SNS services, users can streamline their data conversion process, automate notifications, and perform sophisticated querying and analysis using Parquet files with ease.