Data Lake Construction and Querying with PySpark

Table of Contents

1 INTRODUCTION ………………………………………………………………………………………….. 3

1.1 Purpose of Document ………………………………………………………………………. 3

1.2 Project Overview ………………………………………………………………………………. 3

1.2.1 Objective ……………………………………………………………………………………. 3

2 Project Components …………………………………………………………………………………… 3

3 Technology Stack ……………………………………………………………………………………….. 4

4 GitHub Repository Overview ………………………………………………………………………. 4

4.1 Repository URL …………………………………………………………………………………...5

5 Architectural Diagram …………………………………………………………………………….…..6

6 AWS Services ……………………………………………………………………………………………… 6

6.1 Apache Airflow ……………………………………………………………………………….6

* 1. AWS Lambda …………………………………………………………………………………. 7
  2. S3 Bucket ………………………………………………………………………………………. 8

6.3.1 S3 Folder Structure …………………………………………………………….…. 8

* 1. AWS Glue ……………………………………………………………………………………… 9
  2. AWS Glue Crawler ……………………………………………………………………….. 10
  3. CloudWatch Log ………………………………………………………………………….. 11

6.6.1 CloudWatch Log Link ………………………………………………………..…. 11

* 1. SNS Notification ………………………………………………………………………….. 12
  2. Amazon Athena …………………………………………………………………………… 13
  3. IAM (Identity and Access Management) ………………………….…………… 14

7 AWS Cost Metrics ……………………………………………………………………………………….15

8 Cost Optimization Tips ………………………………………………………………………………..15

9 Conclusion ………………………………………………………………………………………………….16

**1). Introductions**

**1.1). Purpose of Document:**

The purpose of this document is to provide an in-depth overview of the architecture and components involved in constructing and querying a data lake using AWS services such as Amazon S3, AWS Glue, Lambda, and Athena. It describes the workflow and interactions between these services, showcasing how they collaboratively process, transform, store, and analyze data efficiently within a PySpark-powered data lake pipeline

**1.2). Project Overview:**

In our project, a Data Lake will be designed and built to store raw, structured, and unstructured data while enabling querying capabilities using PySpark. The Data Lake will support scalable storage and efficient querying of large datasets, providing a foundation for future analysis and reporting.

**1.2.1). Objectives**

To design and build a scalable Data Lake capable of storing and querying raw, structured, and unstructured data efficiently.

**2). Project Components:**

* **Data Ingestion:** Collect and store data from various sources (e.g., CSV, JSON, logs) in its raw format.
* **Data Processing:** Use PySpark to process and analyze the ingested data.
* **Data Storage:** Save data in a distributed file system such as HDFS, Amazon S3, or Delta Lake.
* **Querying and Reporting:** Enable querying capabilities using Spark SQL and tools like Amazon Athena for analytical insights.

**3). Technology Stack:**

* **Processing:** PySpark (v3.3) – Handles data processing and transformations within the pipeline.
* **Storage:** Amazon S3 – Provides scalable and durable storage for raw, processed, and structured data.
* **Querying:** Spark SQL – Facilitates querying and analysis of data stored in the Data Lake.
* **Orchestration:** Amazon Web Services (AWS) – Integrates and manages workflows using services like AWS Glue and Apache Airflow.
* **Output View:** Athena Table – Enables SQL-based querying and visualization of processed data.

**4). GitHub Repository Overview**

The repository contains all the essential code and configurations for the project "Data Lake Construction and Querying with PySpark." It is organized to ensure seamless execution and ease of use, including the following components:

* **Config Files:**  
  Setup and environment configurations essential for executing the data pipeline. These files include parameters like file types, table names, and queries.

**Example Configuration**:

{

"file\_type": "parquet",

"table\_name": "data\_lake\_query",

"ps\_query": "SELECT \* FROM data\_lake\_query WHERE `Date of birth` BETWEEN '2000-01-01' AND '2024-12-31'"

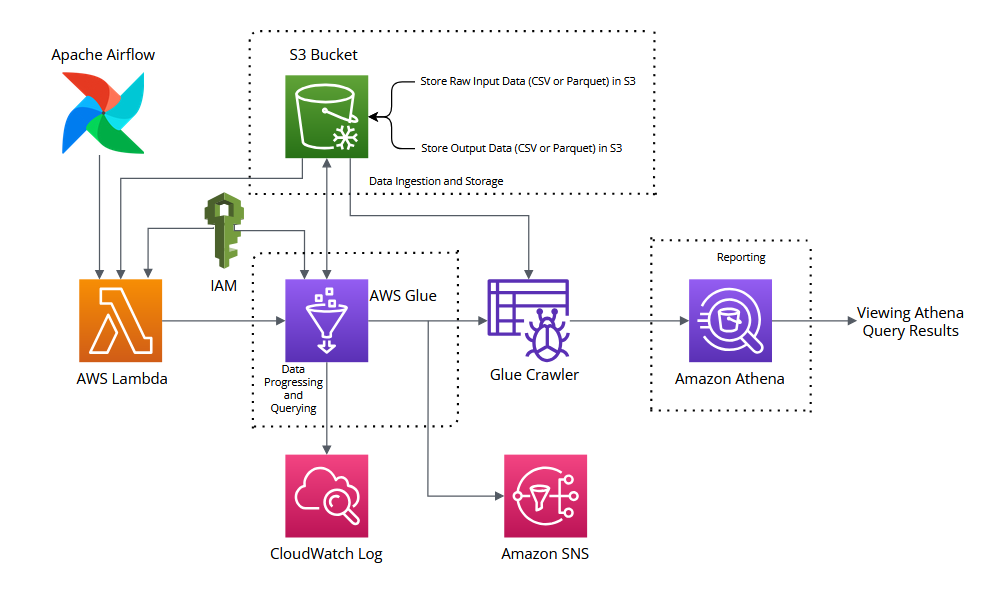
}

* + **File Type**: Defines the data format (e.g., parquet or csv).
  + **Table Name**: Specifies the table for querying (e.g., data\_lake\_query).
  + **Query**: Contains the SQL logic for filtering and selecting data.
* **Airflow DAGs**:  
  Code for orchestrating and scheduling workflows using Apache Airflow. The DAGs define the sequence of tasks for data ingestion, processing, and querying, ensuring smooth execution of the pipeline.
* **Lambda Functions**:  
  Custom serverless functions designed for event-driven processing. These functions are used for tasks such as triggering workflows, processing events, or sending notifications during the data pipeline execution.
* **Glue Scripts**:  
  ETL scripts for transforming, processing, and preparing data for analysis. These scripts enable efficient data transformation and querying in Amazon Athena.

All components are stored in the **master branch**, ensuring easy access and robust version control. This organization helps streamline the project workflow, supporting both development and deployment activities.

**4.1). Repository URL:** <https://github.com/rdinesh808/Data_Lake_Construction_and_Querying_With_PySpark>

**5). Architectural Diagram:**

****

**6). AWS Services:**

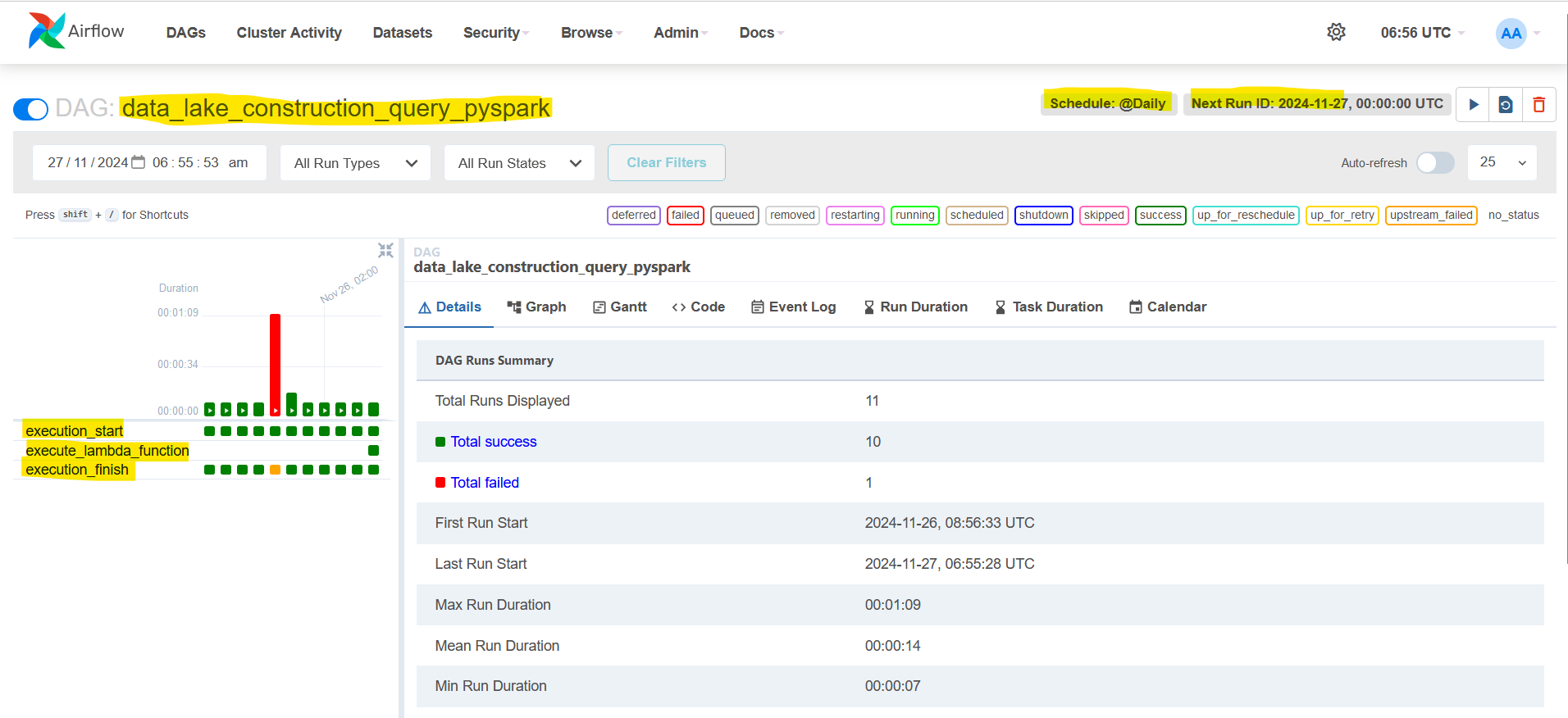
AWS (Amazon Web Services) is a comprehensive and widely adopted cloud platform that offers a collection of services to help businesses build, manage, and scale applications without the need for on-premises infrastructure. These services are hosted in Amazon's global network of data centers, providing scalability, reliability, and security.

**6.1). Apache Airflow**

Apache Airflow triggers the Lambda Function using an API call by programmatically invoking the AWS Lambda service through AWS SDKs. This allows Airflow to directly start the execution of a specific Lambda function as part of the workflow, ensuring seamless integration and automation within the pipeline.

* **DAG Name**: data\_lake\_construction\_query\_pyspark
* **Scheduled Time**: Scheduled to run daily at 12:00 AM (midnight).

Airflow ensures reliable and repeatable execution of workflows, with features like task retries, logging, and monitoring via the Airflow web interface.

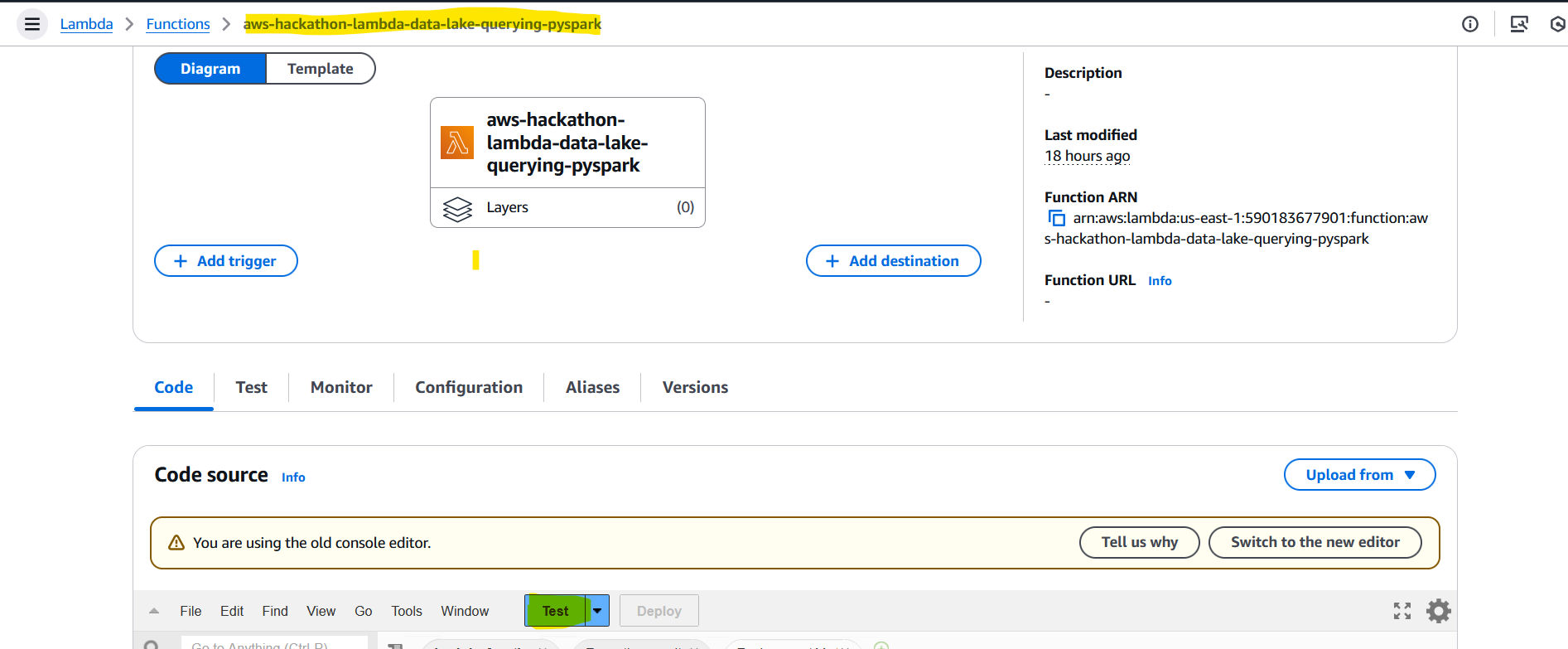
****

**6.2). AWS Lambda**

The Lambda function is designed to automate the process of triggering an AWS Glue job based on configurations stored in an S3 bucket. Here is how it works:

**Lambda Name:** aws-hackathon-lambda-data-lake-querying-pyspark

* **Read Configuration File from S3**:  
  The Lambda function retrieves a configuration file (e.g., JSON or YAML) stored in an S3 bucket. This file contains essential parameters such as the Glue job name and any arguments required for the job.
* **Parse Configuration Parameters**:  
  After fetching the file, the Lambda function extracts the Glue job name and other parameters (like input/output S3 paths or processing options) from the file.
* **Trigger Glue Job with Parameters**:  
  Using the AWS Glue service, the Lambda function triggers the specified Glue job. The extracted parameters are passed as arguments to customize the Glue job's behaviour.

****

**6.3). Amazon S3 Bucket**

Amazon S3 is used to store raw data and configuration files within the data lake, providing scalable and cost-effective storage. It holds raw data ingested from various sources and essential configuration files. Processed output files are also stored in S3, with the storage location determined dynamically based on runtime.

**S3 Bucket Name:** aws-hackathon-s3-data-lake-querying-pyspark

**6.3.1). S3 Folder Structure:**

| - config/

| - data\_lake\_config.json

| - input\_data/ (Stores raw data in formats (CSV and Parquet))  
 | - csv/

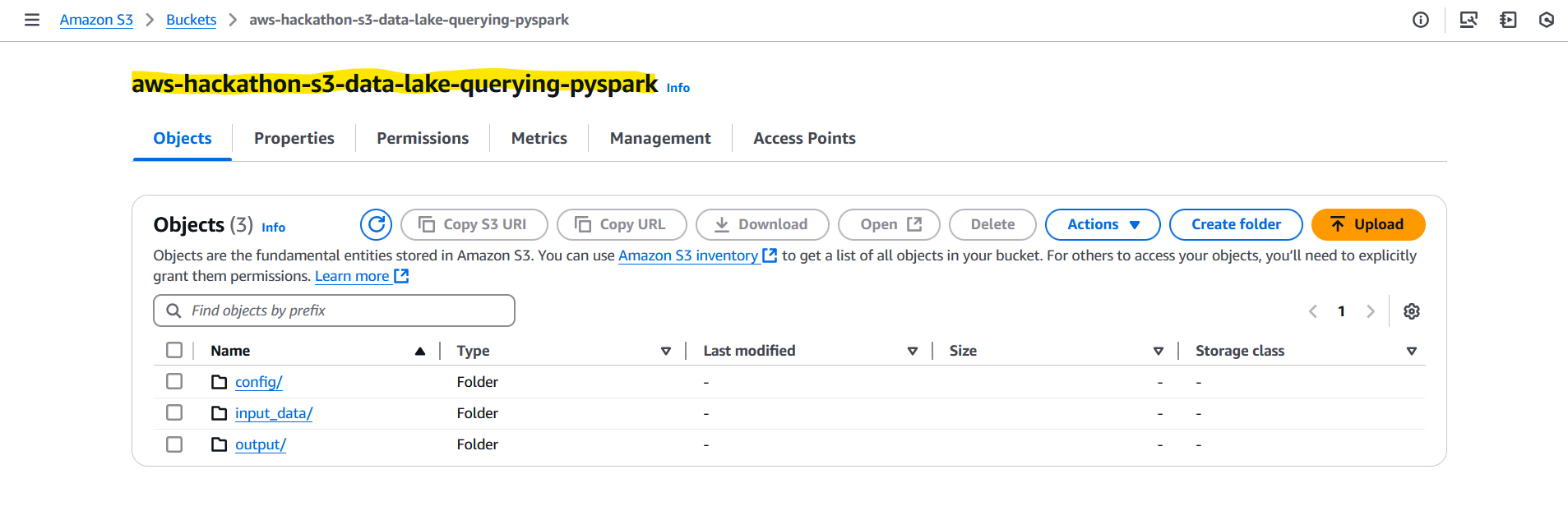
| - people.csv

| - parquet/

| - people.parquet

| - output/

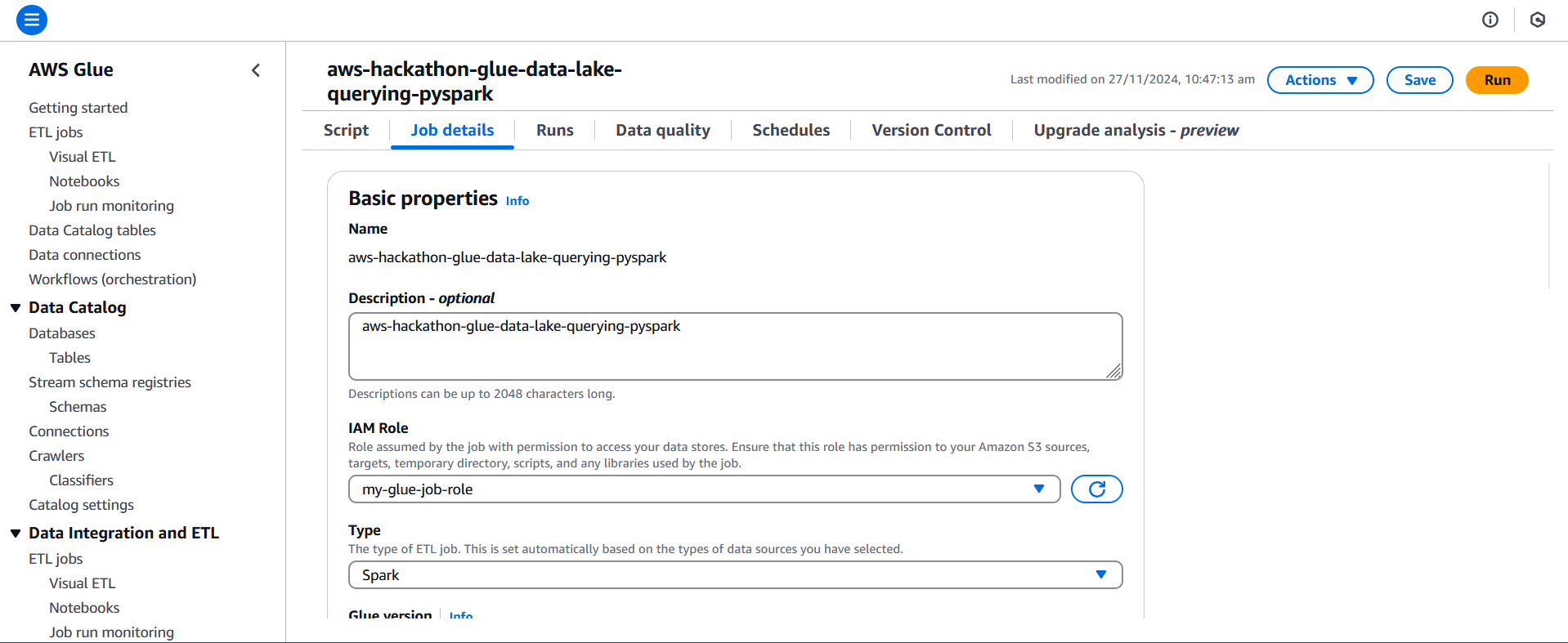
| - Stores processed output files based on runtime (CSV or Parquet).

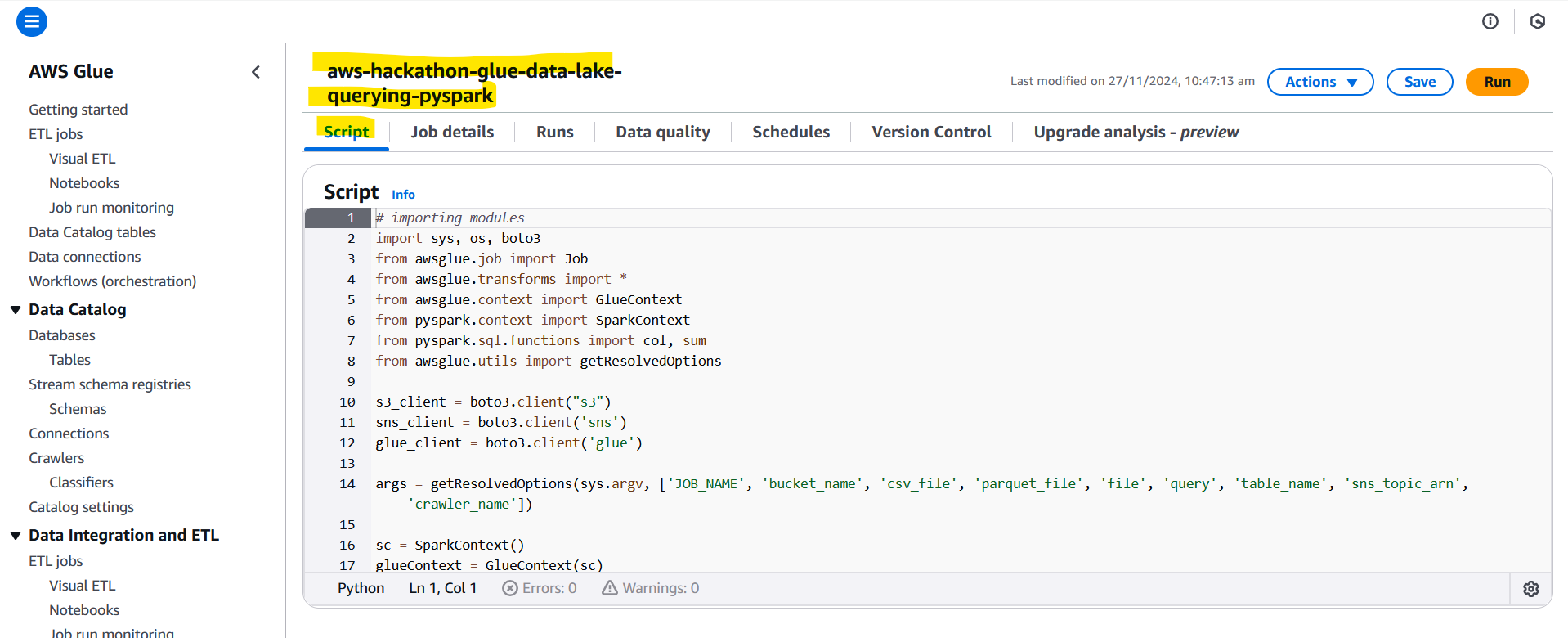


**6.4). AWS Glue**

This AWS Glue script automates the process of reading raw data (CSV or Parquet) from an S3 bucket, performing transformations, and writing the processed data back to S3. It handles transformations such as filling null values, removing duplicates, and aggregating data. The script then runs an SQL query on the transformed data using Spark SQL, allowing for flexible and powerful data analysis. After processing, it triggers an AWS Glue crawler to catalog the new data and sends email notifications via SNS about job status, including successes and failures. This solution simplifies the automation of data processing, transformation, and querying workflows in a data lake environment.

**AWS Glue Name:** aws-hackathon-glue-data-lake-querying-pyspark

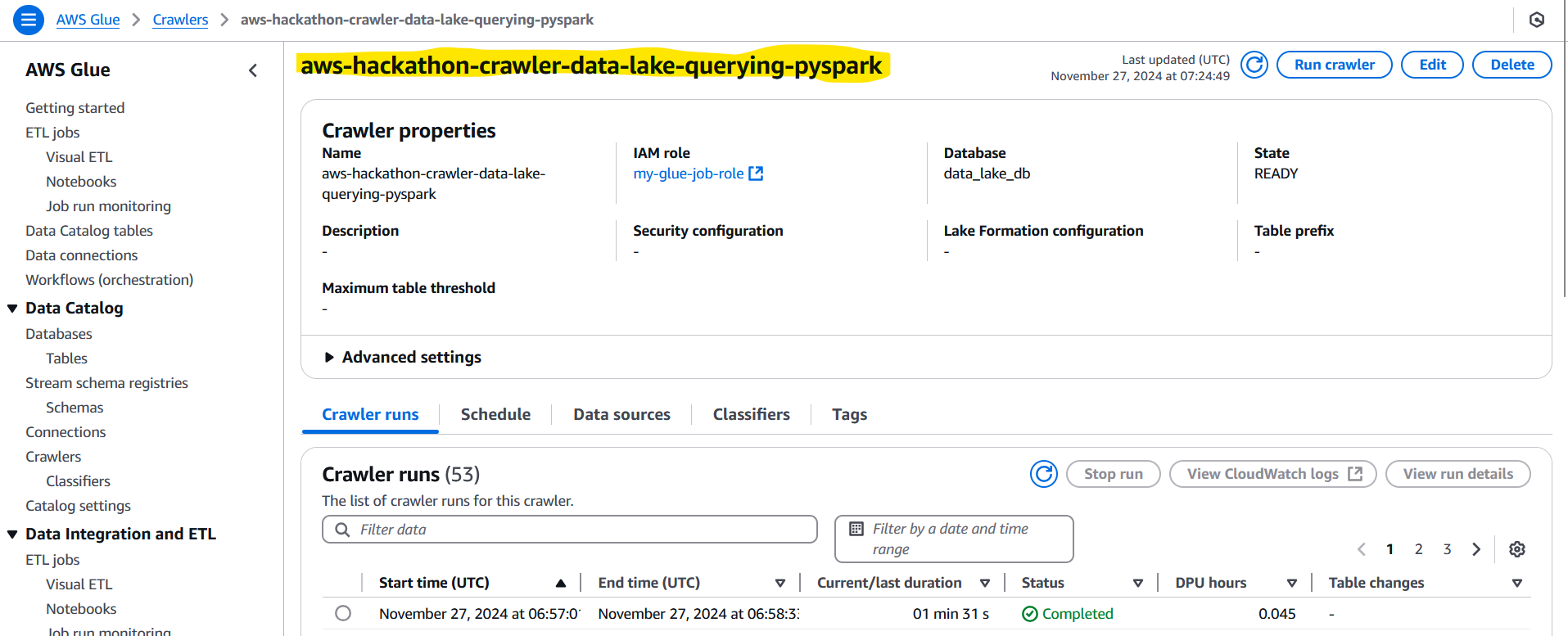




**6.5). AWS Glue Crawler**

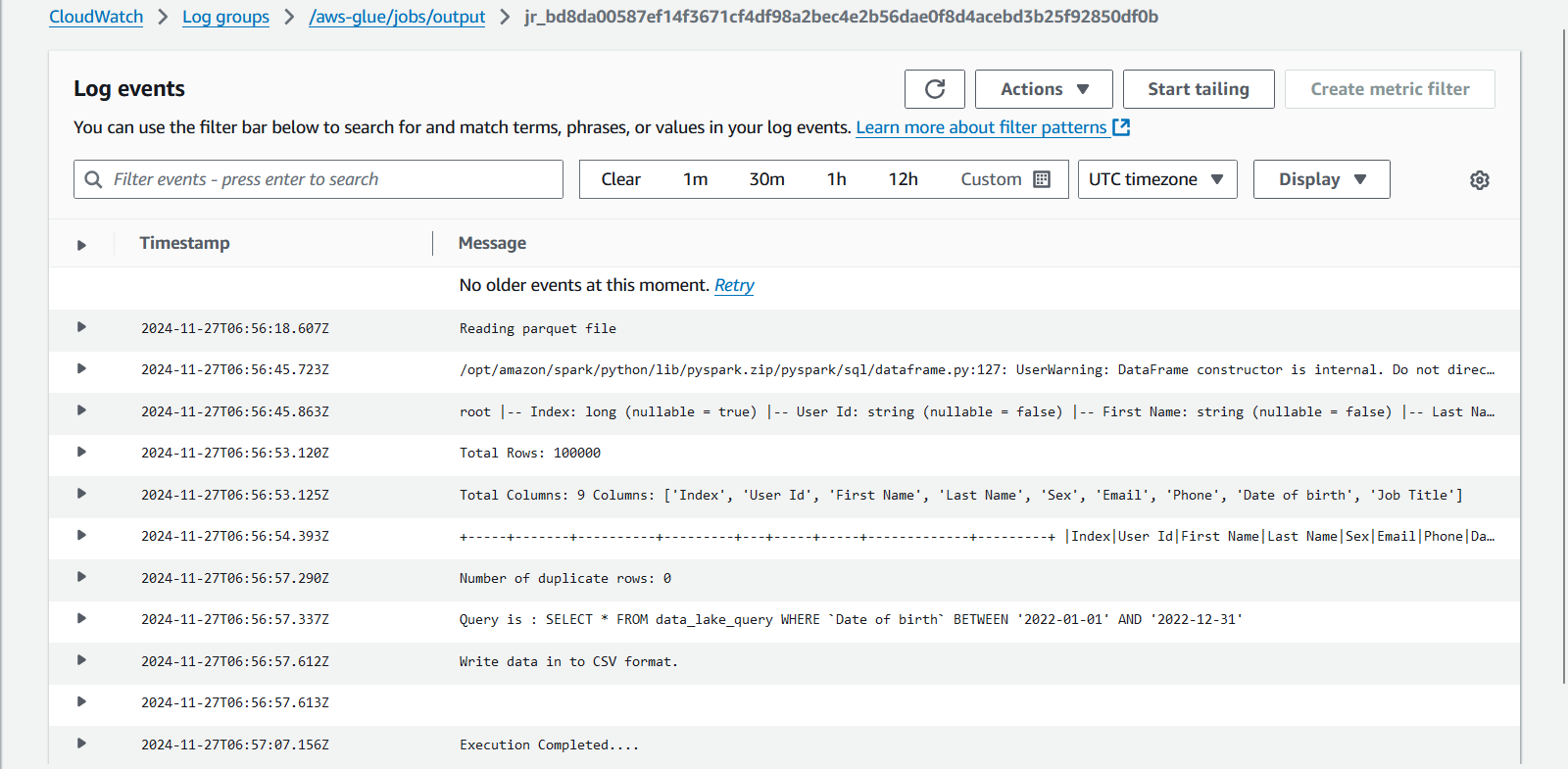
The AWS Glue Crawler automatically discovers and catalogues metadata from data stored in Amazon S3. The Glue Crawler scans the output folder in Amazon S3, infers the schema of the processed data, and updates the AWS Glue Data Catalog by creating tables and defining their structure. This enables seamless querying of the data in Athena. The crawler is triggered based on the data written to the output folder, ensuring that new data is automatically catalogued and available for analysis without manual intervention.

**Glue Crawler Name:** aws-hackathon-crawler-data-lake-querying-pyspark



**6.6). CloudWatch Log**

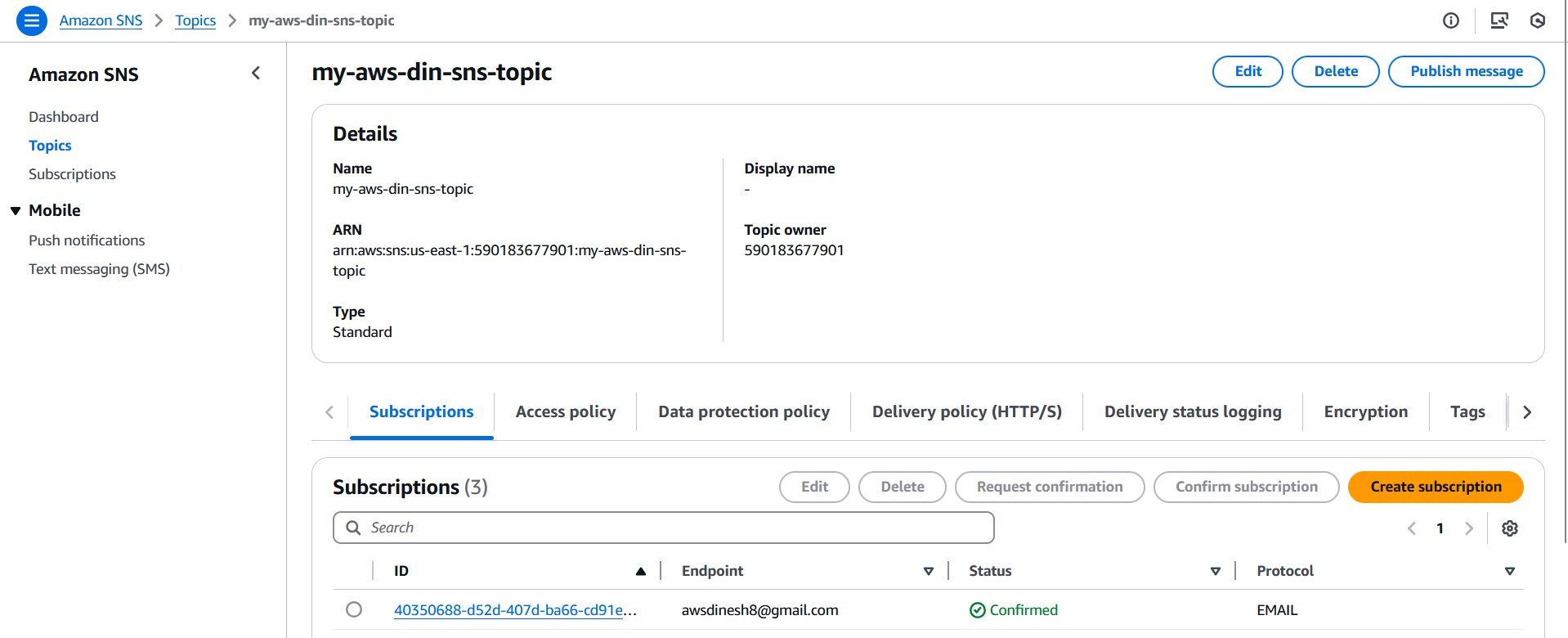
Amazon CloudWatch Logs is used to capture and store logs from AWS services like AWS Glue and Lambda within the data lake pipeline. In our project, CloudWatch Logs helps track the execution details of AWS Glue jobs, including job status, errors, and performance metrics. This enables efficient debugging, performance monitoring, and ensures operational health. By storing detailed logs, it allows easy tracking of the entire data processing workflow, helping maintain smooth and error-free operations.

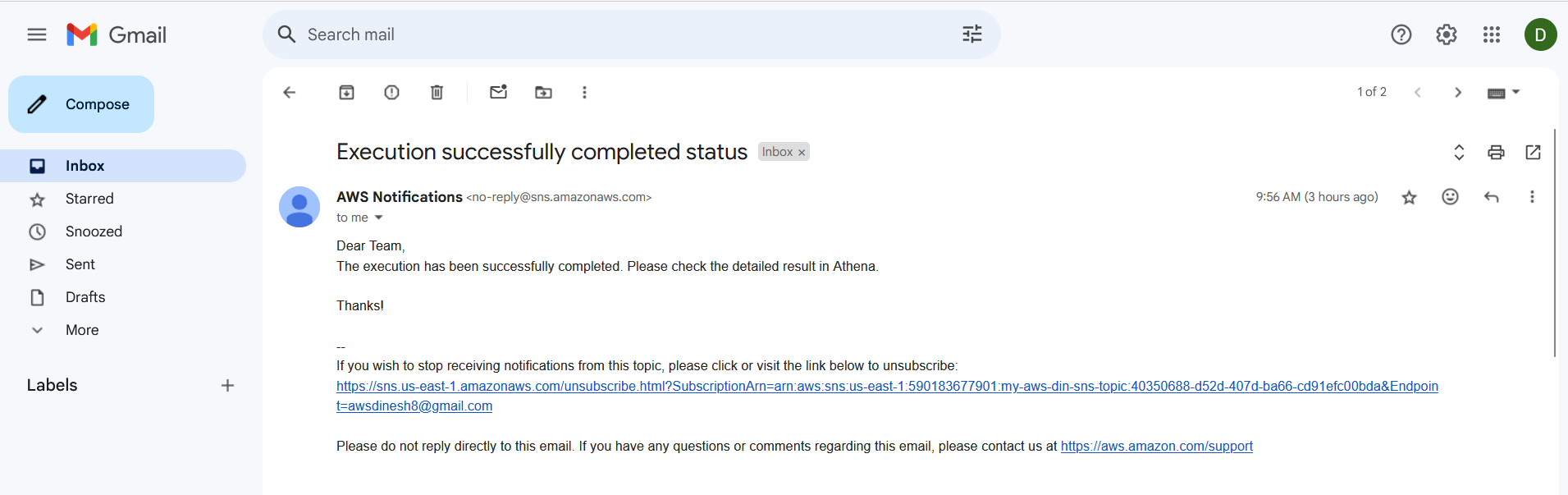


CloudWatch Log Link: [https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:log-groups/log-group/$252Faws-glue$252Fjobs$252Foutput/log-events/jr\_bd8da00587ef14f3671cf4df98a2bec4e2b56dae0f8d4acebd3b25f92850df0b](https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1%23logsV2:log-groups/log-group/$252Faws-glue$252Fjobs$252Foutput/log-events/jr_bd8da00587ef14f3671cf4df98a2bec4e2b56dae0f8d4acebd3b25f92850df0b)

**6.7). SNS Notification**

Amazon Simple Notification Service (SNS) is used in our project to send automated notifications for important events such as job completions, failures, or significant state changes in the data lake pipeline. Based on the execution status, SNS sends a notification indicating either a **successful execution** or a **failure**. If the job completes successfully, SNS sends a success notification, while in the case of an error or failure, a failure notification is sent, keeping the team informed in real-time via email or SMS. This ensures timely communication and proactive monitoring of the pipeline's health.



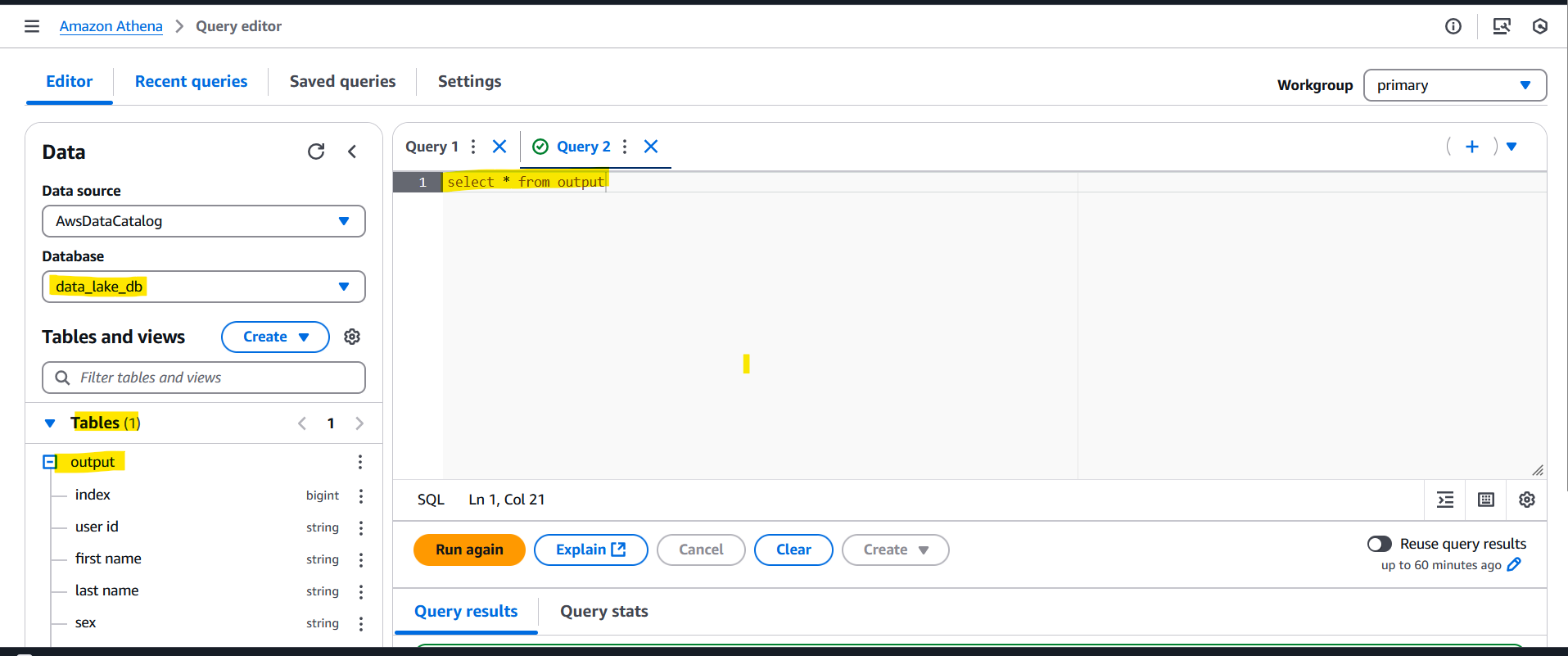


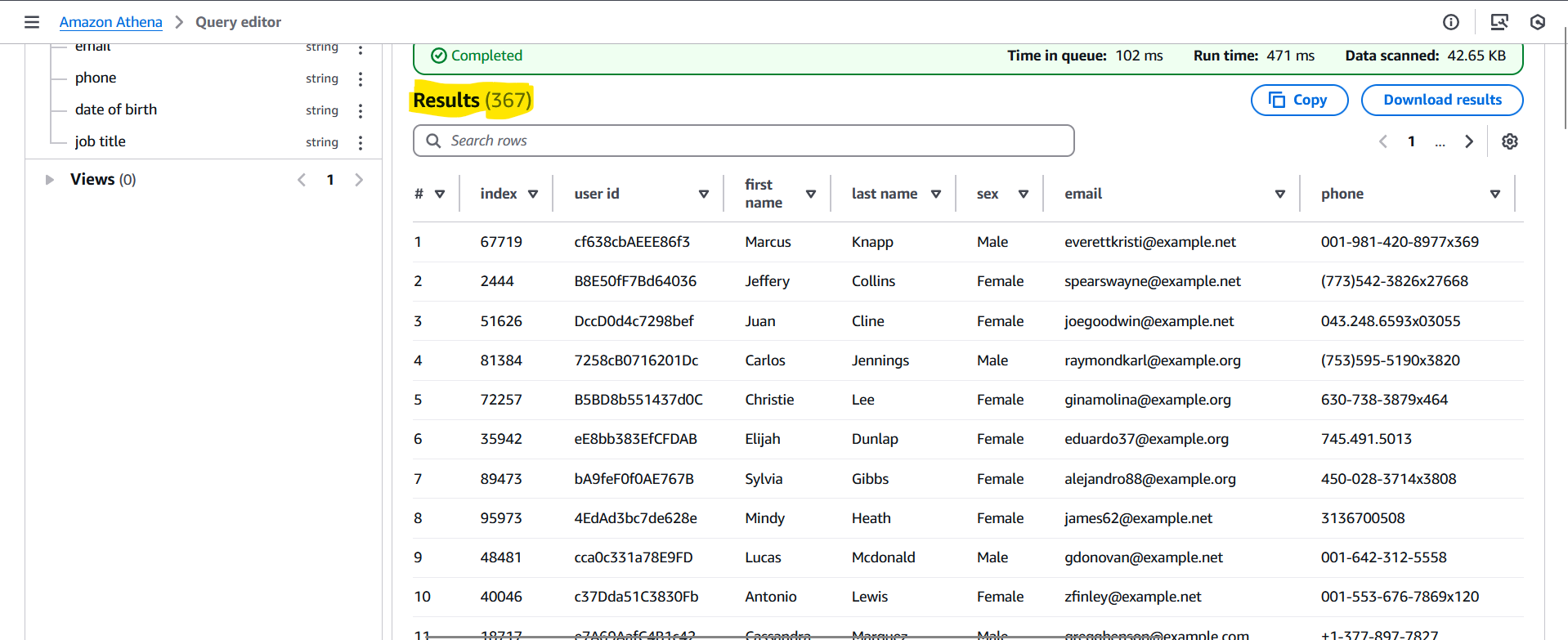


email notification document.

**6.8). Amazon Athena**

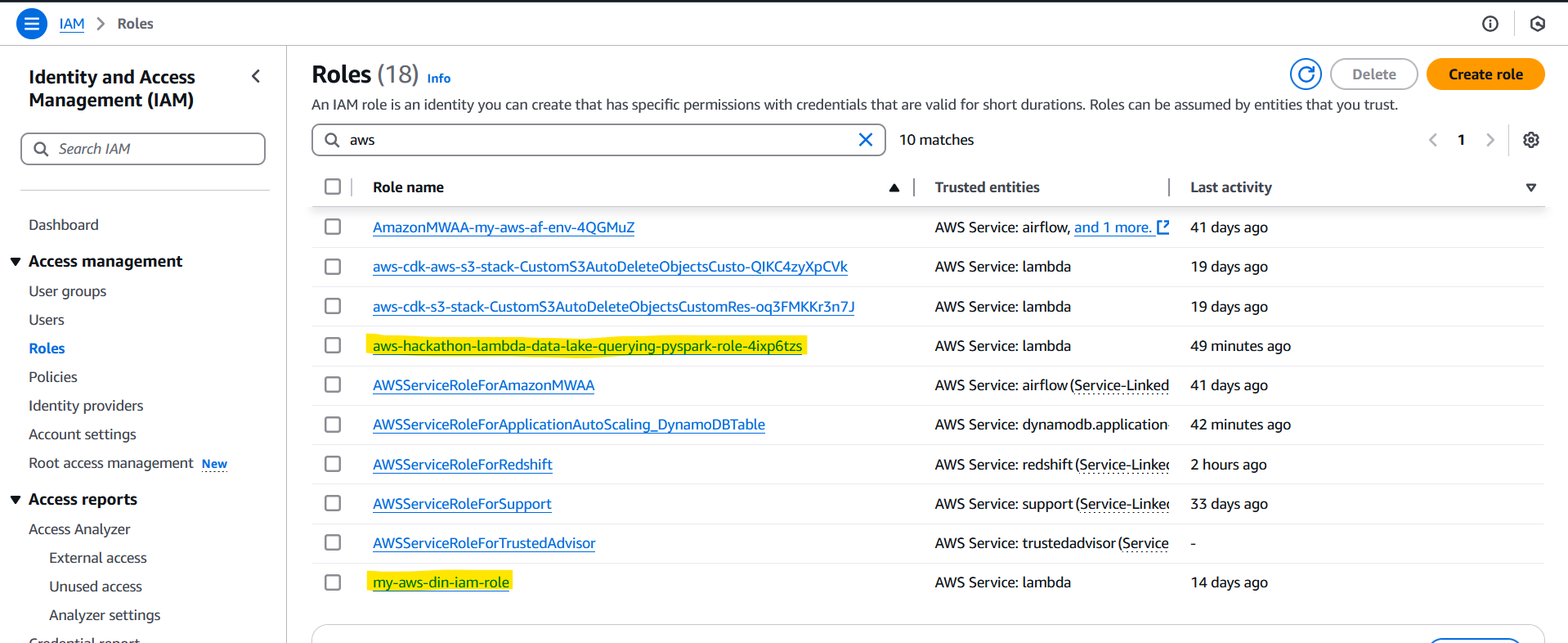
Amazon Athena is a serverless interactive query service that enables you to analyze data directly in Amazon S3 using standard SQL. In our project, Athena is used to query the processed data stored in the data lake on S3. After the AWS Glue Crawler scans and catalogues the data, it automatically creates tables in the Athena Data Catalog, defining the structure of the data. Using Athena’s query editor, users can run SQL queries on these tables to retrieve insights and view the results.





**6.9). IAM (Identity and Access Management)**

AWS Identity and Access Management (IAM) allows you to securely control access to AWS services and resources. In our project, IAM is used to define and manage the roles and permissions for various AWS services to interact securely. For **Lambda**, IAM roles grant access to **S3** for reading and writing data, as well as to **AWS Glue** for triggering ETL jobs. For **AWS Glue**, IAM roles provide necessary permissions for accessing **S3** (for data storage) and **CloudWatch Logs** (for logging job execution details). These roles ensure that each service has the appropriate permissions to perform its tasks securely, while enforcing security policies to prevent unauthorized access.



**7). AWS Cost Metrics**

The estimated monthly cost for AWS services used in the data lake architecture is as follows:

| Service | Estimated Monthly Cost |
| --- | --- |
| AWS Glue | $5.00 |
| Glue Crawlers | $1.00 |
| Amazon Athena | $0.50 |
| AWS Lambda | $0.90 |
| Amazon S3 | $5.00 |
| Amazon CloudWatch | $2.00 |
| Amazon SNS | $0.10 |
| Total Estimated Monthly Cost | ~$14.50 |

**8). Cost Optimization Tips**

To reduce AWS costs, consider the following tips for each service:

* **AWS Glue:** Reduce unnecessary transformations and use partitioned datasets to save on processing costs.
* **Glue Crawlers:** Limit crawler runs and scan only relevant directories to reduce costs.
* **Amazon Athena:** Query only partitioned data and optimize queries to minimize data scanned.
* **AWS Lambda:** Consolidate functions, set appropriate memory limits, and manage execution time to reduce costs.
* **Amazon S3:** Use appropriate storage classes (e.g., Glacier for archived data), implement lifecycle policies, and clean up unused files.
* **Amazon CloudWatch:** Set retention policies for logs and use custom metrics sparingly.
* **Amazon SNS:** Limit SNS usage to essential notifications and batch publish messages when possible.

**9). Conclusion:**

This project demonstrates the effective use of AWS services to design and implement a scalable data lake for processing and querying large datasets. It automates key workflows, including data ingestion, transformation, and analysis. The architecture leverages Amazon S3, AWS Glue, Lambda, and Athena to ensure cost-efficient and seamless data processing. This solution provides a robust foundation for further enhancements, offering flexibility to scale and adapt as data needs grow. The project sets the stage for efficient and future-ready data analytics.