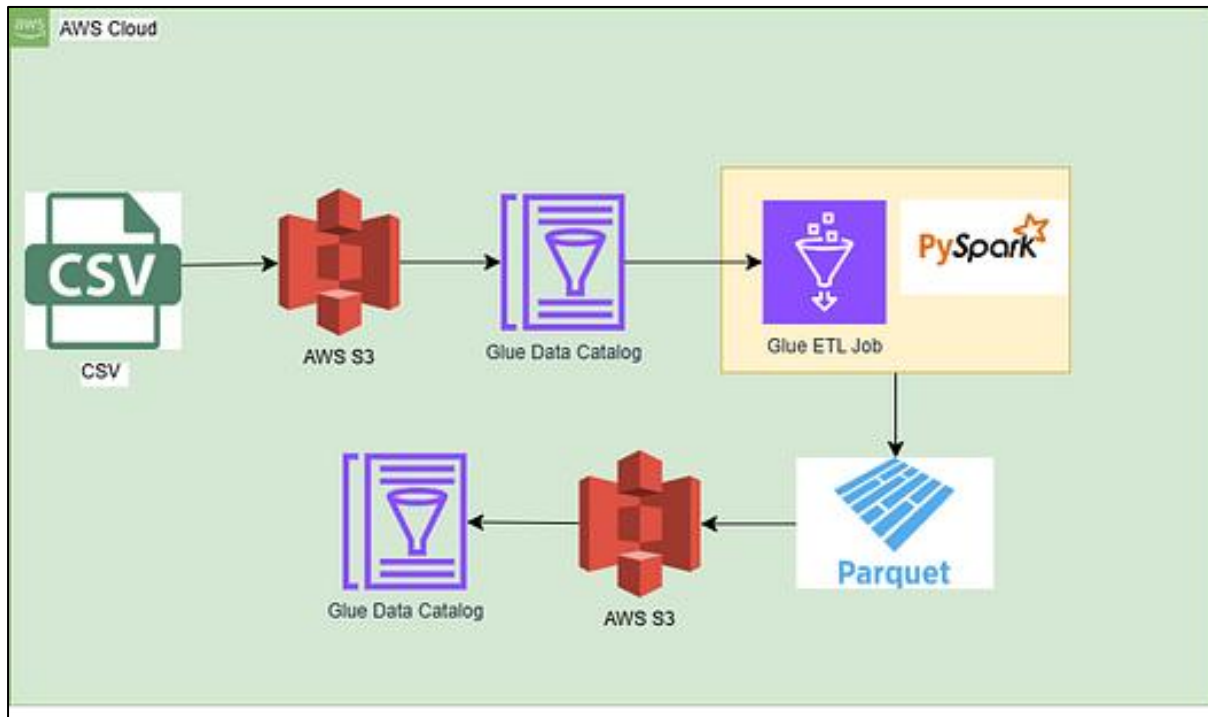


Customer Data ETL Workflow Using AWS Glue Services

AWS Glue ETL Job, S3, PySpark -Bank Customers Dataset



In this project, we have a dataset containing information about the customers of a bank. Our goal is to organize the data in this dataset through PySpark, process missing data, handle erroneous or corrupted format data, and convert it into a proper format. We aim to group the data to make meaningful inferences from it. The goal is to provide the requested data to other departments of the bank (such as management, marketing, sales) in the same format. Through this process, we will explore the tools that Spark offers us in more detail.

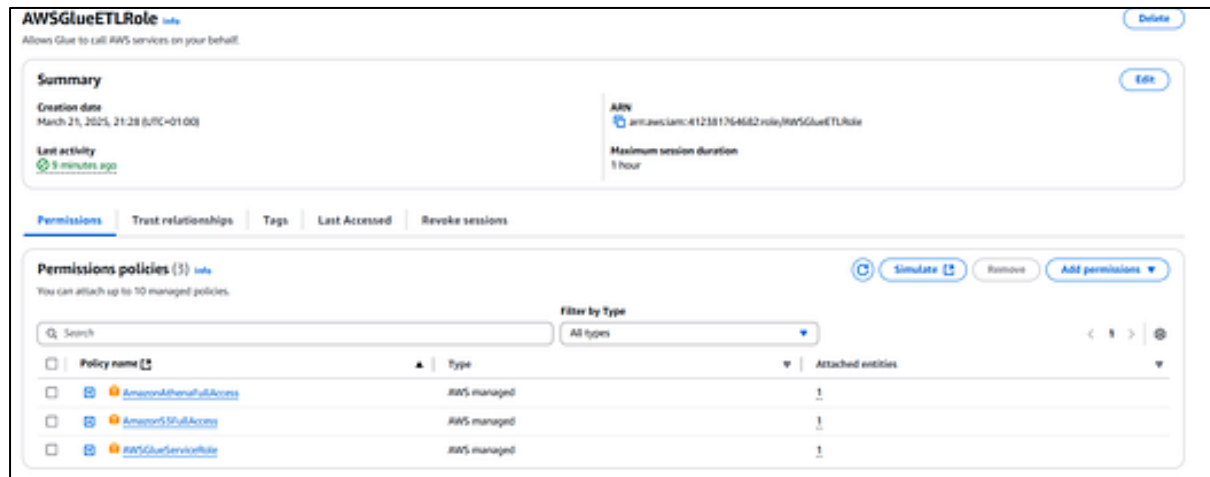
All of these operations will be performed using AWS. In the first phase, we will fetch the CSV-format data from AWS S3, apply the necessary transformations to the data with PySpark support in AWS Glue, then write the data back to S3. Finally, we will transfer the data to AWS Glue Data Catalog for SQL queries via AWS Athena.

First Step: Defining A Role

First, we need to configure the necessary permissions. To do this, follow these steps through the AWS Console:

1. Go to IAM → Roles → Create Role.
2. Choose AWS Service, and for the use case, select Glue.
3. Click Next: Permissions.
4. Add the following permissions by selecting them:
 - AmazonS3FullAccess → (to read/write data from/to S3)
 - AWSGlueServiceRole → (permissions necessary for Glue to operate)
 - AmazonAthenaFullAccess → (to provide access to Athena)
5. Continue with the role creation process and finalize the creation of the role.

By doing this, the Glue service will have the required permissions to read/write data from S3, perform its tasks, and access Athena for querying.



ETL Role definition

Note: For this process, a more restricted IAM Role, User, or custom IAM Policy could be defined. However, I chose to use AWS's pre-defined (Managed) policies to simplify the process. In real-world projects, it is recommended to create customized policies that align with the principle of least privilege. This ensures that users and services only have the minimum permissions necessary to perform their tasks.

Second Step: Creating AWS S3 Buckets

In this step, we are creating S3 buckets to store our CSV file. For our scenario, we assume that data already exists, but I wanted to show this step for those who are working on the project. We plan to create two buckets:

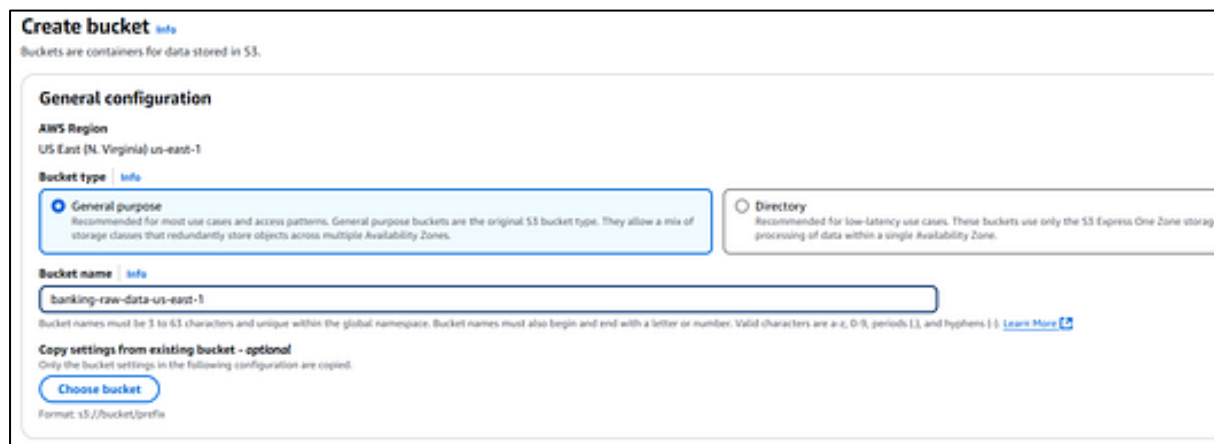
1. The first bucket will be named banking-raw-data-us-east-1-xxxxxxx (Each bucket name must be universally unique). This bucket will store the raw data.
2. The second bucket will be named banking-processed-data-us-east-1-xxxxxxx. This bucket will store the processed data.

Since AWS does not charge based on the number of buckets, I believe it is a good practice to use separate buckets for storing raw and processed data to keep them organized.

Steps to Create Buckets:

1. Go to the AWS Console.
2. Navigate to S3.
3. Click on Create Bucket.
4. In the Bucket Name field, enter the desired name for your bucket (make sure the name is unique globally).
5. Leave other settings as default (or customize them as needed).
6. Click Create to finalize the creation of your S3 bucket.

By doing this, you will have your S3 buckets ready to store both raw and processed data.



The screenshot shows the 'Create bucket' page in the AWS console. It includes a 'General configuration' section with the following details:

- AWS Region:** US East (N. Virginia) us-east-1
- Bucket type:** General purpose (selected), Directory (unselected). A note for General purpose states: 'Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.' A note for Directory states: 'Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage processing of data within a single Availability Zone.'
- Bucket name:** banking-raw-data-us-east-1
- Copy settings from existing bucket - optional:** Only the bucket settings in the following configuration are copied. A 'Choose bucket' button is present.
- Format:** s3://bucket/prefix

Creating Buckets



The screenshot shows the 'General purpose buckets (2)' page in the AWS console. It includes a search bar and a table of buckets:

Name	AWS Region	IAM Access Analyzer
banking-processed-data-us-east-1	US East (N. Virginia) us-east-1	View analyzer for us-east-1
banking-raw-data-us-east-1	US East (N. Virginia) us-east-1	View analyzer for us-east-1

Our Buckets

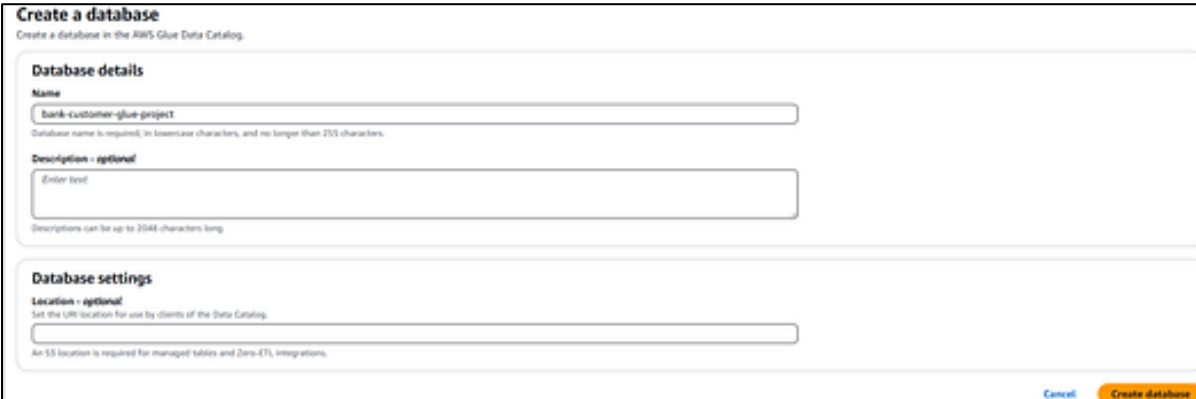
Third Step: Creating an AWS Glue Database

In this step, we are creating our database. We will use a single database for our project, and the name of this database will be bank-customer-glue-project. The database in AWS Glue Data Catalog allows you to store and organize your data in an orderly manner. This way, we will create a centralized structure for both raw and processed data. The database helps you organize your data for ETL processes and provides easier access to it.

AWS Console Steps:

1. Go to the AWS Console.
2. Navigate to the Glue service.
3. Under the Databases section, click on Add database.
4. Name the database bank-customer-glue-project.
5. Click Create to create the database.

By doing this, you will have a Glue database set up for your project, helping you organize your raw and processed data efficiently.



The screenshot shows the 'Create a database' interface in the AWS Glue console. At the top, it says 'Create a database' and 'Create a database in the AWS Glue Data Catalog.' Below this, there are two main sections: 'Database details' and 'Database settings'. In the 'Database details' section, the 'Name' field is filled with 'bank-customer-glue-project' and has a note that the name is required, lowercase, and no longer than 255 characters. The 'Description - optional' field is empty with a placeholder 'Enter text' and a note that descriptions can be up to 2048 characters long. In the 'Database settings' section, the 'Location - optional' field is empty with a placeholder 'Set the URI location for use by clients of the Data Catalog.' and a note that an S3 location is required for managed tables and Zero-ETL integrations. At the bottom right, there are 'Cancel' and 'Create database' buttons.

Create a database
Create a database in the AWS Glue Data Catalog.

Database details

Name
bank-customer-glue-project
Database name is required, in lowercase characters, and no longer than 255 characters.

Description - optional
Enter text
Descriptions can be up to 2048 characters long.

Database settings

Location - optional
Set the URI location for use by clients of the Data Catalog.
An S3 location is required for managed tables and Zero-ETL integrations.

[Cancel](#) [Create database](#)

Fourth Step: Creating a Table with AWS Glue

In this step, we are creating a table within our database. We will create a table in the Glue Data Catalog to define the structure of the data stored in S3. This table will enable us to process the data using AWS Glue. When creating the table, we specify the data format, schema, and location.

AWS Console Steps:

1. Go to the AWS Console.
2. Navigate to the Glue service.
3. Go to the Tables section and click on Add table.
4. For the Table Name, you can enter bank-customer-full-table.
5. In the Database field, select the bank-customer-glue-project database.
6. For Table Type, choose either Native or Crawler (Crawler automatically discovers the structure of the data and is chargeable).
7. Specify the data format (in our case, it's CSV) and the S3 data source (bucket path).
8. In the Columns section, provide the schema details (column names, data types).
9. Click Add Table to finalize.

After this step, you will have a table added to your database and Glue Data Catalog, which will allow you to process your data. You can use this table in your ETL processes to work with the data.

The screenshot shows the 'Set table properties' interface in the AWS Glue console. On the left, a sidebar indicates the current step is 'Step 1: Set table properties', with other steps being 'Step 2: Choose or define schema', 'Step 3: Review and create', and 'Review and create'. The main area is titled 'Set table properties' and contains two sections: 'Table details' and 'Table format'. In the 'Table details' section, the 'Name' field is populated with 'bank-customer-full-table', and the 'Database' dropdown is set to 'bank-customer-glue-project'. There is a 'Description - optional' field below. In the 'Table format' section, the 'Standard AWS Glue table (Default)' option is selected, while the 'Apache Iceberg table' option is unselected. A 'Create database' button is visible next to the database dropdown.

Table format

Data Catalog managed tables support data optimization for Apache Iceberg table type. [Learn more](#)

☒ Standard AWS Glue table (default)
Create a standard AWS Glue table.

☐ Apache Iceberg table
Create a table in Apache Iceberg table format.

Data store

Select the type of source

☒ S3
☐ Kinesis
☐ Kafka

Data location is specified in

☒ my account
☐ another account

Include path

Path must be in the form s3://bucket/prefix/. It must end with a slash (/) and not include any files.

Data format

Classification

Choose the format of the data in your table.

☐ Avro
☒ CSV
☐ JSON
☐ XML
☐ Parquet
☐ ORC

Delimiter

Step 1: Set table properties

Step 2: **Choose or define schema**

Step 3: Review and create

Choose or define schema

☒ Define or upload schema
Manually define schema

☐ Choose from Glue Schema Registry
Select existing schema from your Glue Schema Registry

Schema {1/17}

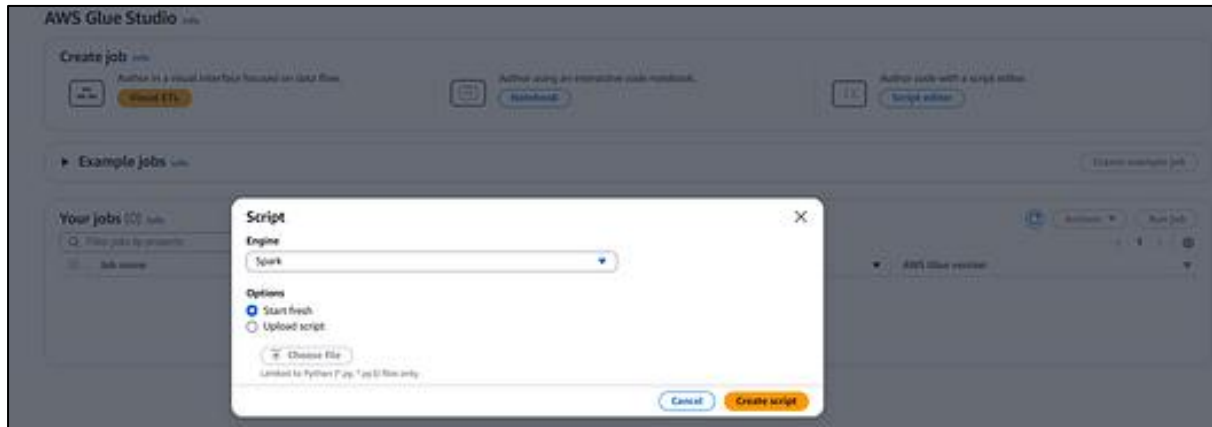
View and manage the table schema.

#	Column name	Data type	Partition key	Comment
<input type="checkbox"/> 1	age	int	-	-
<input type="checkbox"/> 2	job	string	-	-
<input type="checkbox"/> 3	marital	string	-	-
<input type="checkbox"/> 4	education	string	-	-
<input type="checkbox"/> 5	default	string	-	-
<input type="checkbox"/> 6	balance	int	-	-
<input type="checkbox"/> 7	housing	string	-	-
<input type="checkbox"/> 8	loan	string	-	-
<input type="checkbox"/> 9	contact	string	-	-
<input type="checkbox"/> 10	day	int	-	-
<input type="checkbox"/> 11	month	string	-	-
<input type="checkbox"/> 12	duration	string	-	-
<input type="checkbox"/> 13	campaign	int	-	-
<input type="checkbox"/> 14	pdays	int	-	-
<input type="checkbox"/> 15	previous	int	-	-
<input type="checkbox"/> 16	previous	string	-	-
<input checked="" type="checkbox"/> 17	y	string	-	-

Now we can get to the core of the task. We will start by creating a Glue ETL Job to perform the manipulation and analysis processes we planned. In this step, we will leverage the capabilities provided by PySpark, and we will convert our processed data into a more compact and optimal format, Parquet. Below are the steps to follow:

Steps to Create Glue ETL Job:

1. Go to AWS Console → ETL Jobs → Script Editor → Create Script.
2. In the Bucket Name field, specify your output bucket.



Fifth Step: Transforming Data

In this step, we inspect the data and clean it by handling missing, erroneous, and unnecessary data, as well as correcting any malformed formats. We will use the tools provided by PySpark to perform these operations.

```
# Print the schema of the data to inspect columns
```

```
data_from_catalog.printSchema() # Print the schema to check the columns
```

```
# Convert DynamicFrame to Spark DataFrame
```

```
df = data_from_catalog.toDF()
```

```
# Perform operations on the data
```

```
df = df.withColumn("age", df["age"].cast("int")) # Cast 'age' column to integer
```

```
df = df.withColumn("balance", df["balance"].cast("int")) # Cast 'balance' column to integer
```

```
# Drop unnecessary columns
```

```
df = df.drop('contact', 'day_of_week', 'month', 'duration', 'pdays',  
            'previous', 'poutcome', 'y', 'campaign', 'day')
```

```
# Filter out rows with 'unknown' values
```

```
df = df.filter(col('education') != 'unknown')
```

```
df = df.filter(col('job') != 'unknown')
```


Sixth Step (Optional): Small Data Analysis Using PySpark

In this phase, we will use PySpark's tools to perform small data analysis. This step may be needed for some tasks but is generally the responsibility of data analysts. Therefore, I've performed a small analysis here just to demonstrate PySpark's filtering and grouping capabilities, supported by Matplotlib for visualization.

Note: This part is available in the Jupyter notebook version of the project, but it is not present in the Python version.

Seventh Step: Saving the Processed Data in Parquet Format

In the final step, we will convert the cleaned dataset into Parquet format and save it to the target folder. This phase is critical not only for preparing the data for analysis but also for optimizing performance and storage efficiency.

Why Parquet Format?

- **Columnar Storage:** Parquet is a columnar storage format, which offers significant advantages in big data processing. It improves the speed of reading and writing data and reduces storage space.
- **Compression:** Parquet supports data compression, which makes it ideal for storing large datasets.
- **Efficient Reading:** Since only the necessary columns are read, Parquet format offers faster data access, which improves query performance.
- **Cost Efficiency:** Storing data in Parquet reduces storage costs by making data more compact.

In summary, by saving our data in Parquet format, we enhance performance and optimize storage efficiency.

Using AWS Glue DynamicFrame and Glue Sink

We use AWS Glue DynamicFrame and Glue Sink to save our processed data in Parquet format:

1. Converting PySpark DataFrame to DynamicFrame:

- First, we convert the processed PySpark DataFrame to a DynamicFrame. A DynamicFrame is a schema-flexible data structure provided by AWS Glue, which allows us to perform transformations on the data easily. It also facilitates integration with Glue Data Catalog.

2. Writing Data to S3 using Glue Sink:

- After converting to DynamicFrame, we use Glue Sink to save the data to the target folder (e.g., S3). Glue Sink allows us to specify the output format (Parquet) and apply compression (e.g., Snappy) to store the data efficiently.

- Additionally, Glue Sink can update the Glue Data Catalog with database and table information, which helps in keeping the data organized and accessible.

By using Parquet format and Glue Sink, we optimize the data writing process, ensuring fast access and efficient storage. We also catalog the data with Glue Data Catalog, making it easy to query with tools like AWS Athena.

```
# Convert DataFrame back to DynamicFrame
dynamic_frame = DynamicFrame.fromDF(df, glueContext, "dynamic_frame")
```

```
# Define S3 output path
s3_output_path = "s3://banking-processed-data-us-east-1-xxxx/"
```

```
# Set up the sink for writing to S3
s3_sink = glueContext.getSink(
    path=s3_output_path,
    connection_type="s3",
    update_behavior="UPDATE_IN_DATABASE", # Update existing data
    partition_keys=[], # Partition keys, can be added if necessary
    compression="SNAPPY", # Compress the data
    enable_update_catalog=True, # Update the Glue Catalog
    transformation_ctx="s3_sink"
)
```

```
# Define the catalog table name
catalog_table = "bank-customer-full-table-processed"
```

```
# Set the catalog information for the new table
s3_sink.setCatalogInfo(
    catalog_database="bank-customer-glue-project",
    catalog_table_name=catalog_table # New table name
)
```

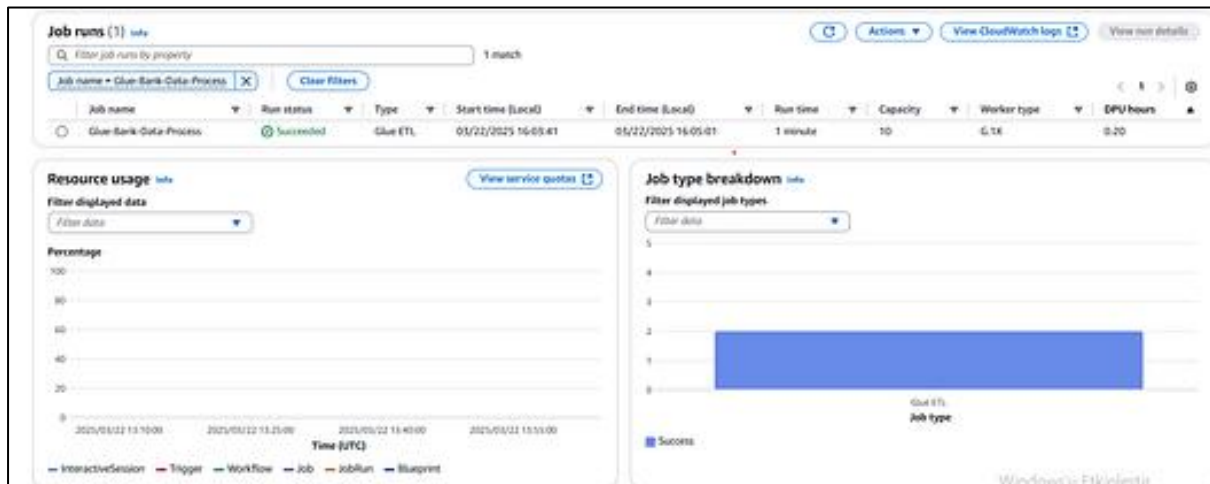
```
# Save the data in Parquet format
s3_sink.setFormat("glueparquet")
```

```
# Write the DynamicFrame to S3
s3_sink.writeFrame(dynamic_frame)
```

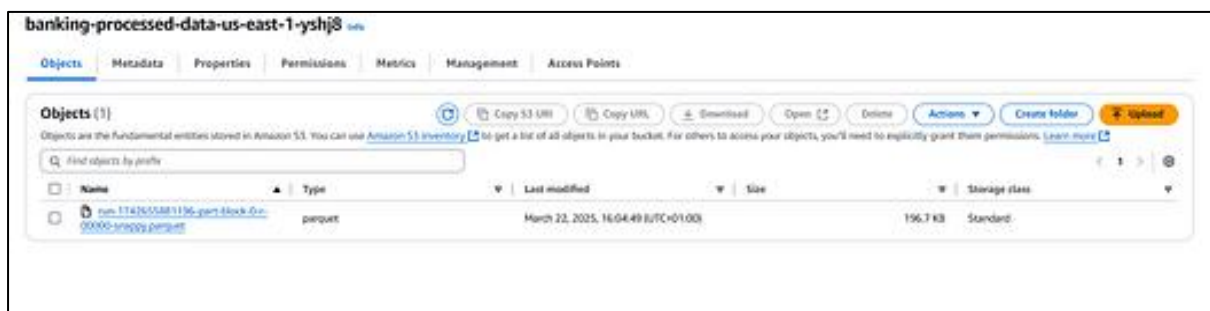
```
# Commit the job when done
job.commit()
```

Eighth Step: Monitoring

In this step, we focus on monitoring the performance and status of the Glue ETL job. AWS Glue provides built-in tools for tracking job execution, logging errors, and checking the status of your transformations. You can monitor the job using AWS Glue Console, where you can view the logs, check the execution time, and diagnose issues if they arise. AWS CloudWatch can also be used for more detailed monitoring and alerts. It's essential to monitor the job regularly to ensure smooth data processing and resolve any potential issues quickly. Below, we can see that the process has successfully completed. Additionally, we can check whether our Parquet-format file has been saved to the target location.



Successfully Processed



Successfully Uploaded

Conclusion and Evaluation:

In this study, an ETL process was performed on banking customer data using AWS Glue and PySpark. By cleaning, transforming, and writing the data in an optimized format to S3, the following benefits were achieved:

- Automated large-scale data processing,
- A more efficient structure for data analysis and reporting,
- Reduced maintenance overhead thanks to AWS Glue's managed services.

This approach provides a scalable and cost-effective data processing solution, offering significant advantages for industries like finance that handle large datasets.