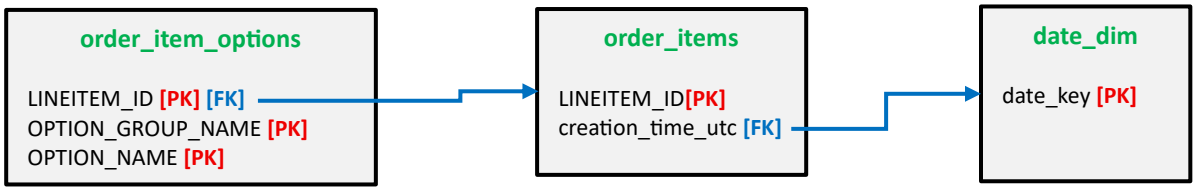
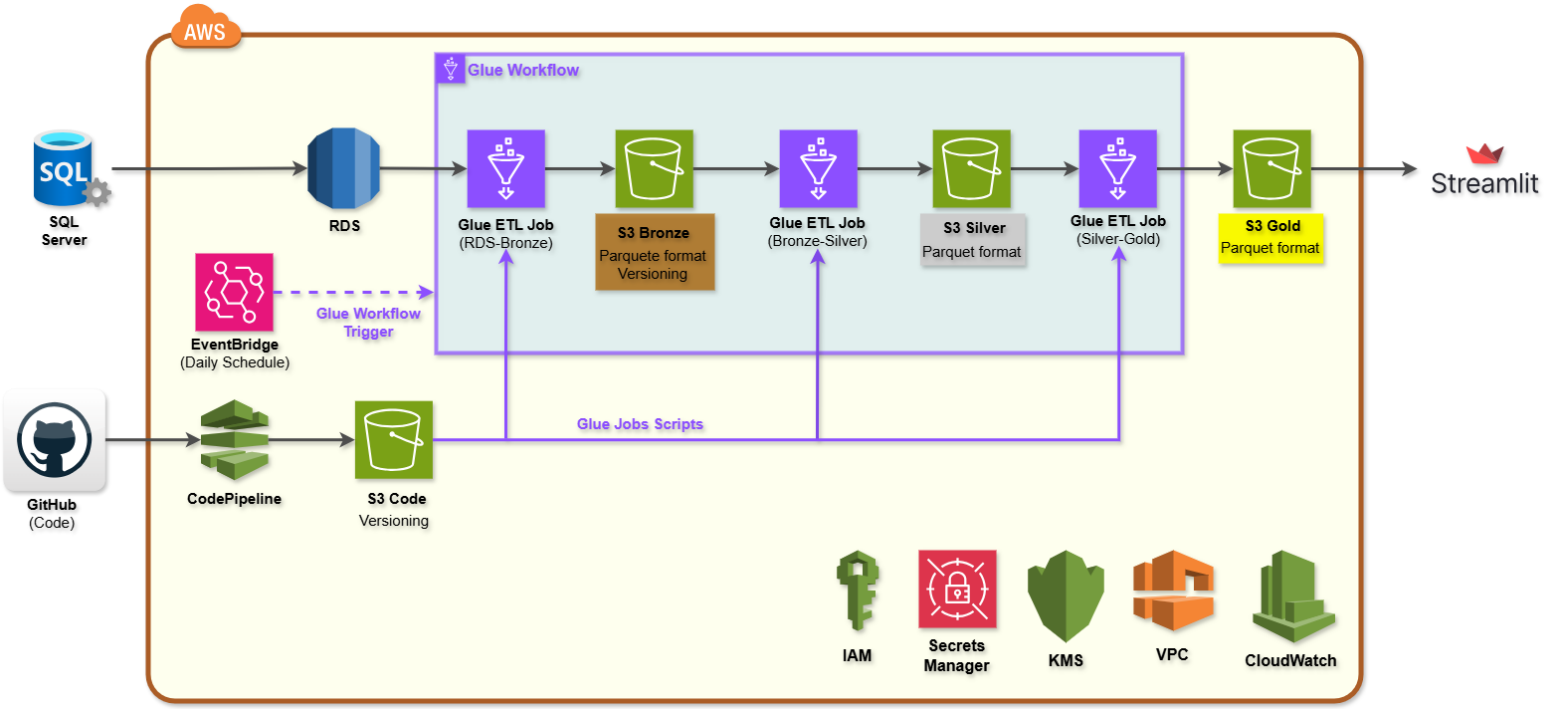
Project 3: Business Insights Assessment

Overview:

After the exploratory alysis of the three source data files (**order_items.csv**, **order_item_options.csv**, and **date_dim.csv**), I have prepared the proposal and got the approval from Jay. Now, here is the data model pipliene, CI/CD operation and ocrhestration from data ingestion, doing ELT jobs, until visualization to support business insights, customer analytics, and reporting requirements.



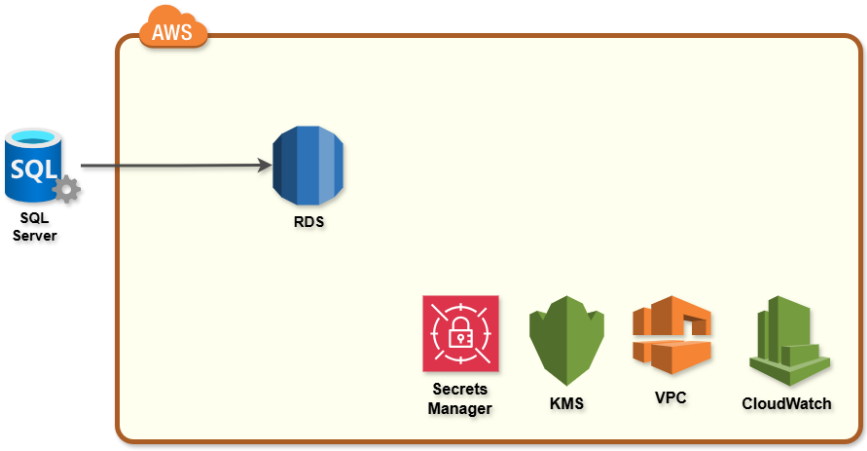
Here is the whole pipeline, and in the following, I am going to state each section in detail.



Step 1 – Data Ingestion (SQL Server to RDS)

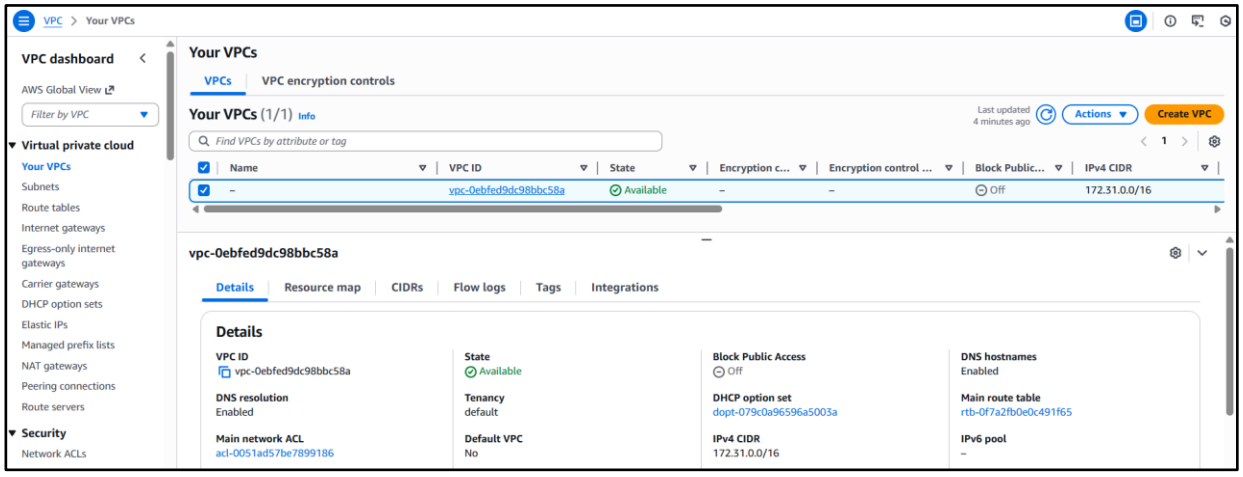
1-1- Objective

The objective of this step is to ingest raw transactional and dimensional data from my local SQL Server database into an Amazon RDS database. This is best solution, as S3 can connect to RDS easily using VPC inside the AWS. We need S3 for Bronze, Silver, and Gold layer as DataLake, and S3 cannot directly get data from my Local SQL Server easily or privately.

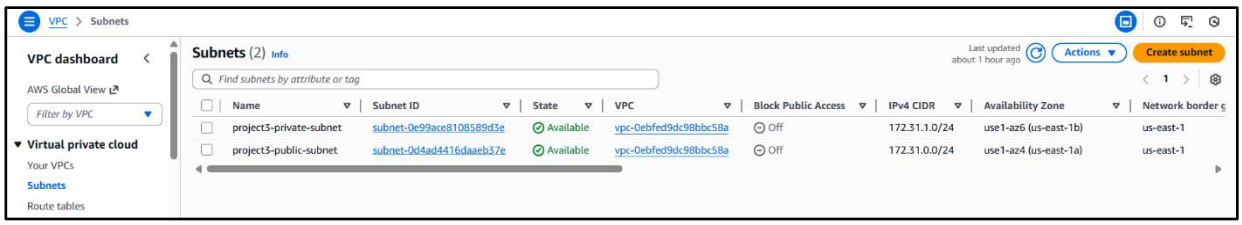


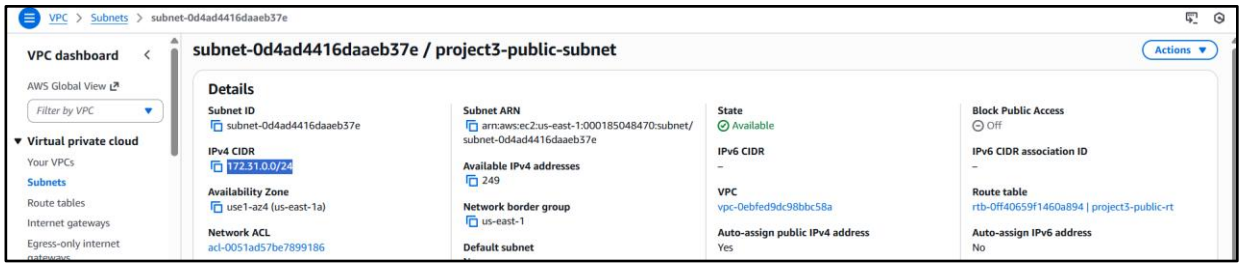
1-2- Connectivity

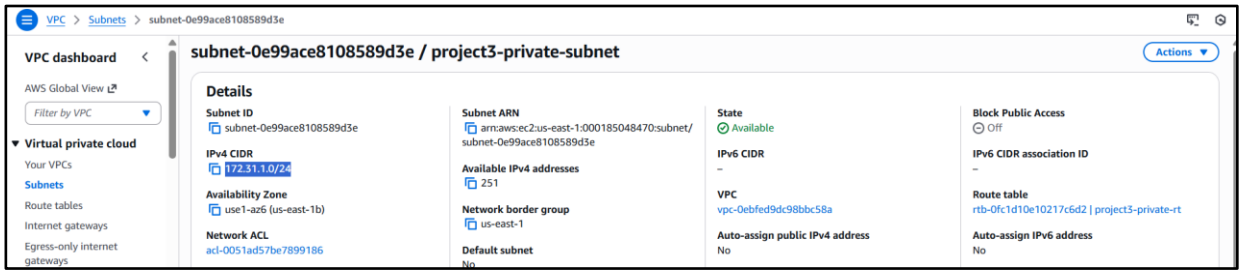
The **Amazon VPC** provides secure network connectivity between AWS RDS and the SQL Server database, ensuring that data transfer occurs within a controlled private network. Inside AWS, I have created a private VPC with Ipv4 CIDR of **172.31.0.0/16**. I don't want to use the default VPC.



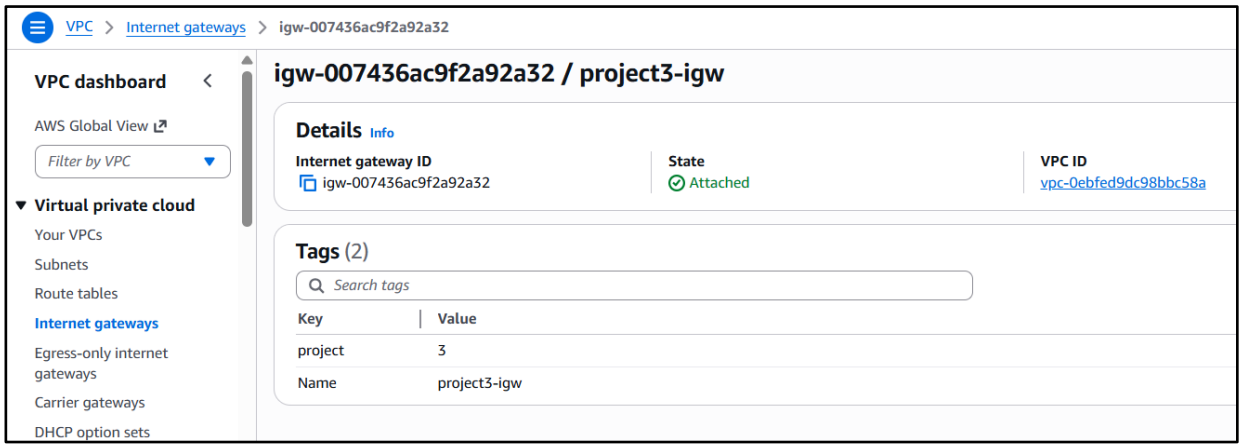
Then, I have created two public (**project3-public-subnet**) and private (**project3-private-subnet**) subnets for the VPC, in different zones of **us-east-1a** and **us-east-1b**. I will use the public subnet for NAT and SMSS networking, and the private for RDS and Glue JDBC job.





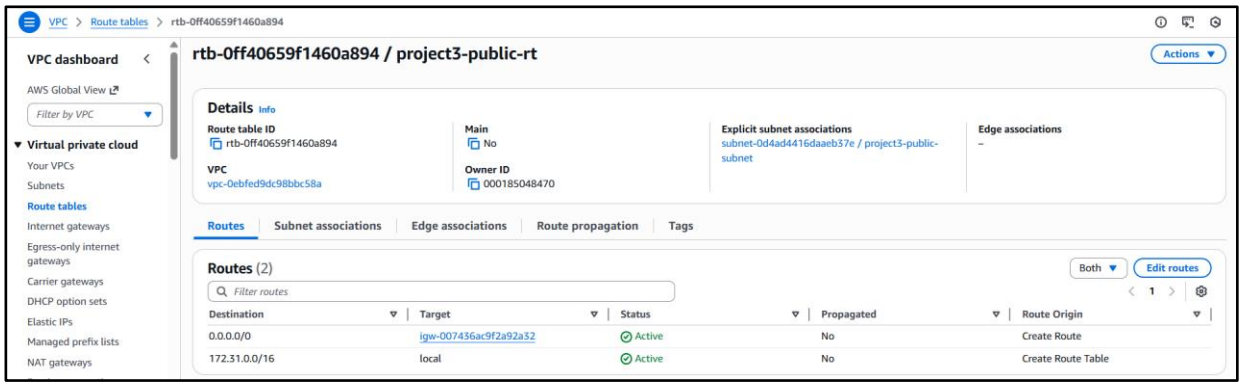


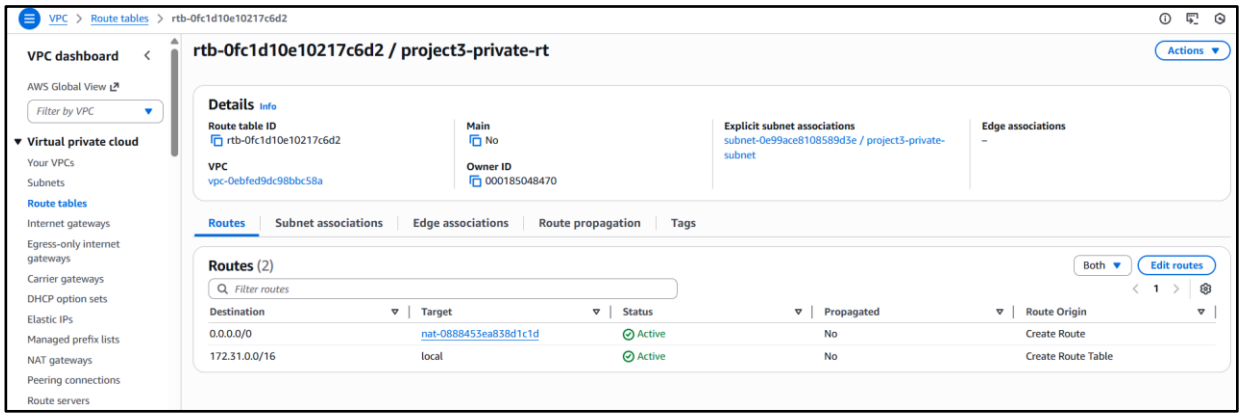
After that, I need to connect SQL Server to AWS via VPC, then I need to create an Internet Gateway and design the Route Table based on public subnet of VPC. First, I created an Internet Gateway, named **project3-igw**.



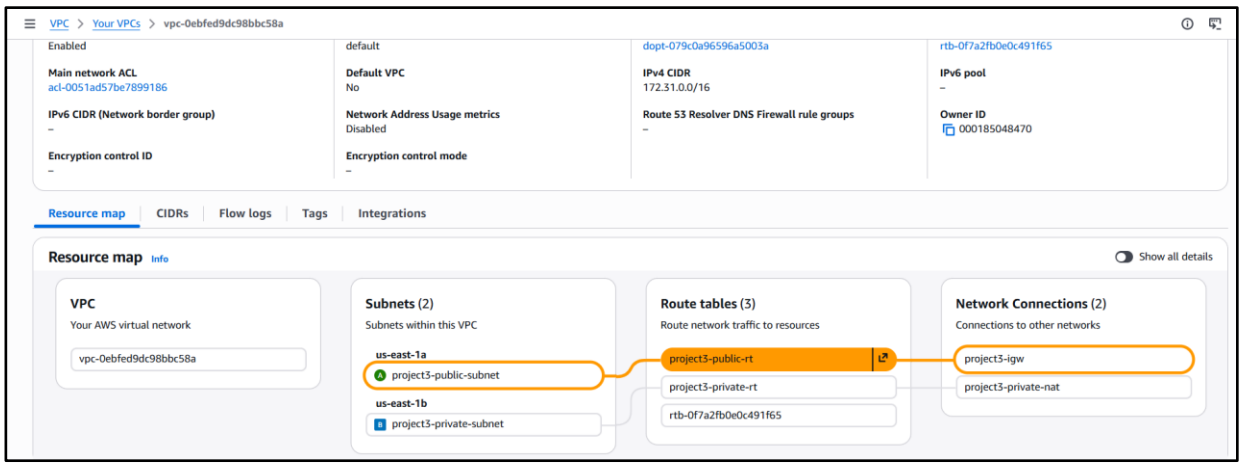
Then, I have created two differenet Route Tables, one for private subnet (**project3-private-rt**) and the other for public subnet (**project3-public-rt**). There is a main route table by default in the AWS VPC, which we will not use it.

If I click on each route table, I can see these information below:





They mean that I connected the public subnet to internet gateway via public route table.



Also, as you can see from the fibure above, we need to conncet private subnet to NAT using private route table. Then, I created a NAT Gateway, named **project3-private-nat**.

And the final and **IMPORTANT** step is creatoing Security Groups for networking. I created two different security groups for RDS (**project3-rds-sg**) and Glue (**project3-glue-sg**). There is also a **default** security group by default in the AWS VPC, which we will not use it.

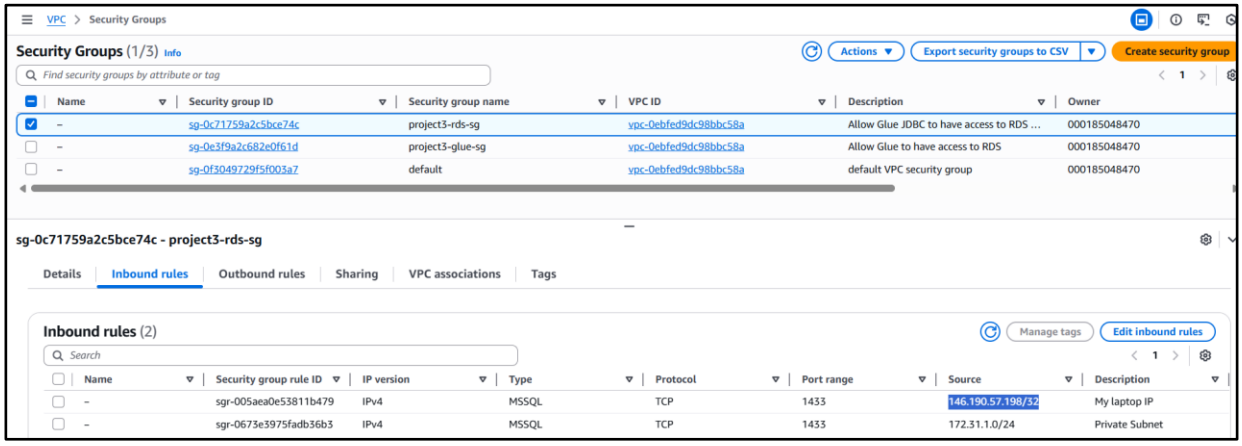
We can access to the Security Groupbs via VPC (above picture) or via EC2 (below picture).

A dedicated security group named **project3-rds-sg** was created for the Amazon RDS instance to control inbound and outbound network traffic. The security group rules were configured as follows:

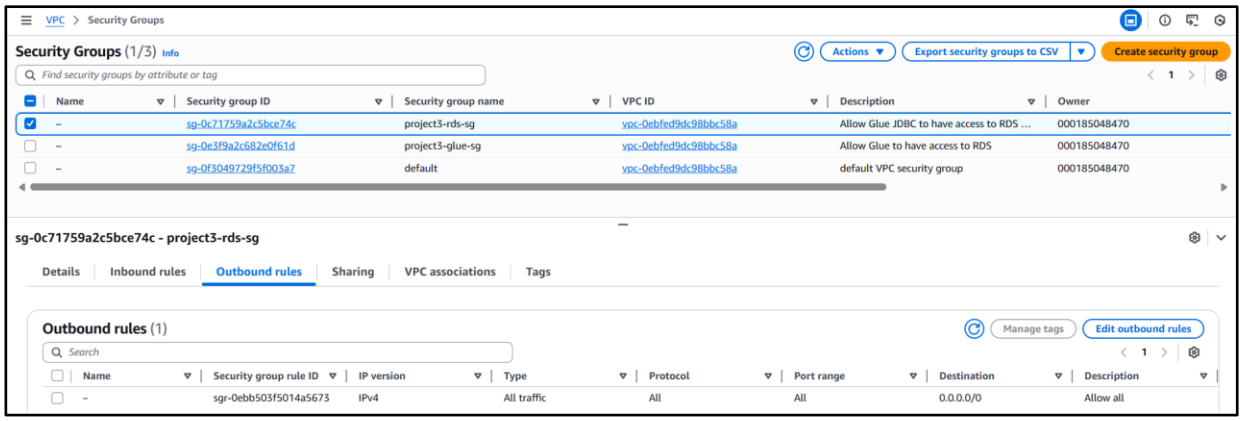
- **Inbound rules:**
  1. MSSQL access from the local machine (*.*.*.*/32) allows Microsoft SQL Server connections from the laptop's public IP address for administrative access and testing.
  2. MSSQL access from the VPC subnet (172.31.1.0/24) allows AWS Glue jobs running within the VPC to connect to the RDS instance for data ingestion.
- **Outbound rules:** All outbound traffic is allowed (0.0.0.0/0), enabling the RDS instance to communicate with required AWS services such as logging, monitoring, and maintenance endpoints.

This configuration ensures that the RDS instance is accessible only from trusted sources while maintaining necessary outbound connectivity.

If I click on the **project3-rds-sg**, I can edit the **inbound** and **outbound** IPv4 for this security group.

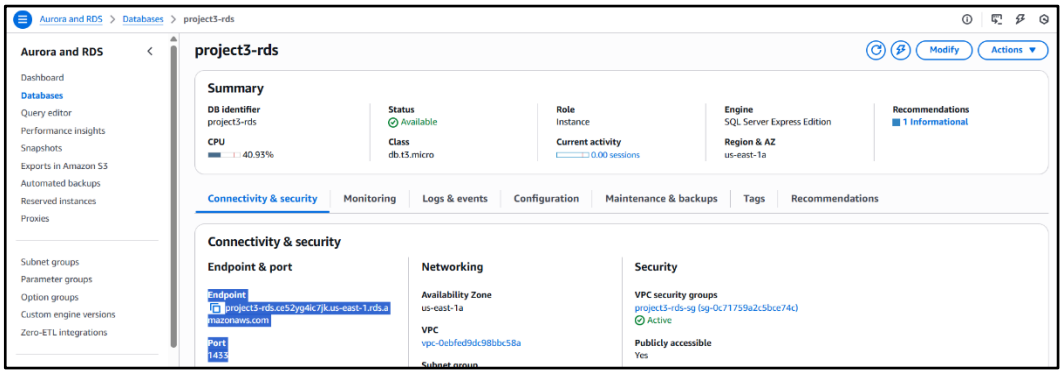


As you can see above, the Inbounde rule has two IPs. One of them is my local laptop's IP address (146.190.57.198/32), which will change as I change my wifi location; it is like *.*.*.*/32 always. The other IP is a private subnet's IP (172.31.1.0/24) and is not change by end of the project. Also, as you can see below, the Outbounde rule has only one general IP, which is open for **All traffic**, 0.0.0.0/0.



1-3- Amazon RDS

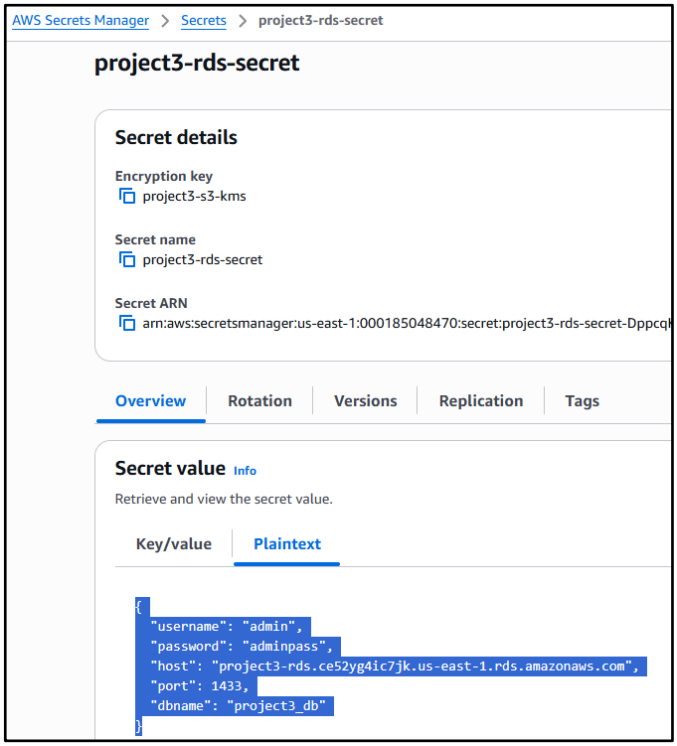
After securing the connection, now it is time to create the RDS database for data ingestion from local SQL Server. I created the **project3-rds** database and connected it to our created VPC and network with minimal configuration.



To have access to this database, we need to have username and password. I have created a secrete inside the Secret Manager and save the RDS credentials in it. Then, I connected SQL Server to RDS using **Endpoint** information and **Port 1433** and with these information.

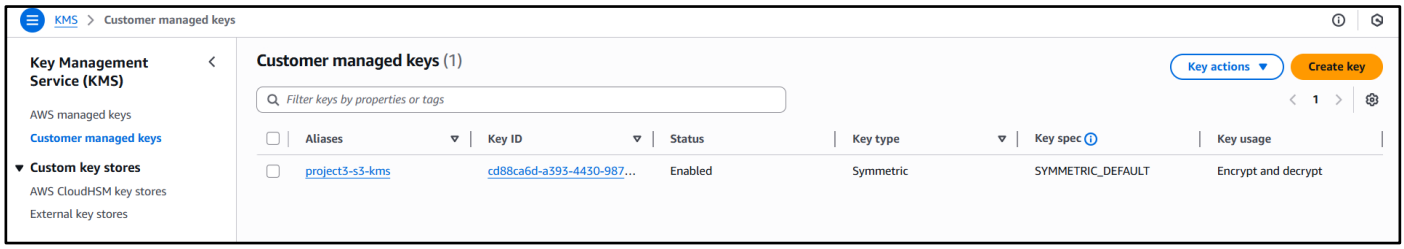
1-4- Secrets Manager

As I mentioned, I created the secret, named **project3-rds-secret**, and put all required information for RDS connection in it. You can see them in the format of JSON format. I do not use the Key/Value format as Glue JDBC can only read them in JSON format. To have strong encryption, I have created a KMS encryption key using **AWS KMS** service and encrypt our sercret with that key.



1-5- AWS KMS

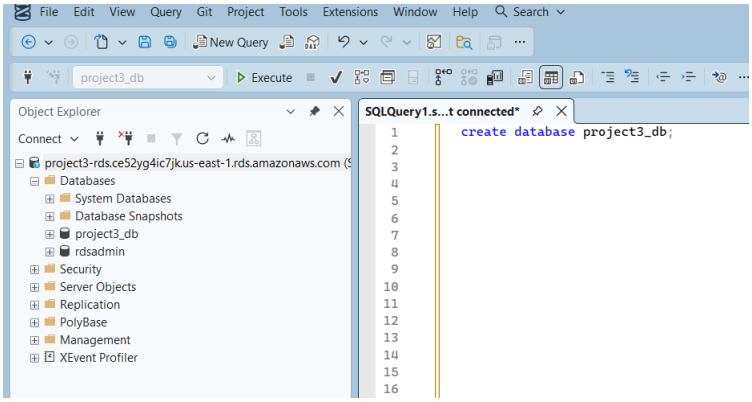
As I mentioned, I create an encryption key using Key Management Service (KMS), named **project3-s3-kms**. I used this key for whole project, specfically in DataLake S3 buckets.



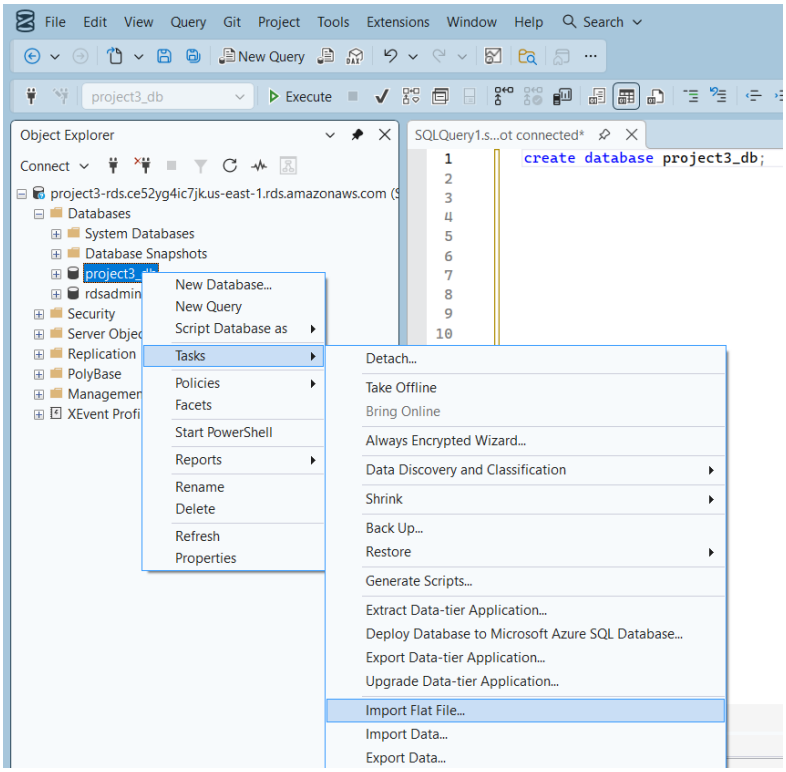
1-6- SQL Server

As requested in the project, our raw data is in local SQL Server. Therefore, I installed the Microsoft SQL Server 2022 (Evaluation Edition) and SQL Server Management Studio (SSMS) (v. 22.1.0) on my local computer. After opening the SSMS, I have connected it to the RDS using the credential and Endpoint and Portal information above. After making secure connection, I have created a database, named **project3_db**, and tried to upload three raw tables inside that database (inside schema **dbo**).

```
create database project3_db;
```

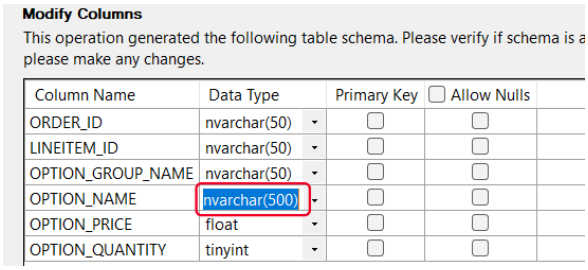


To upload the csv files, I used to import flat files directly into the **project3_db** database.

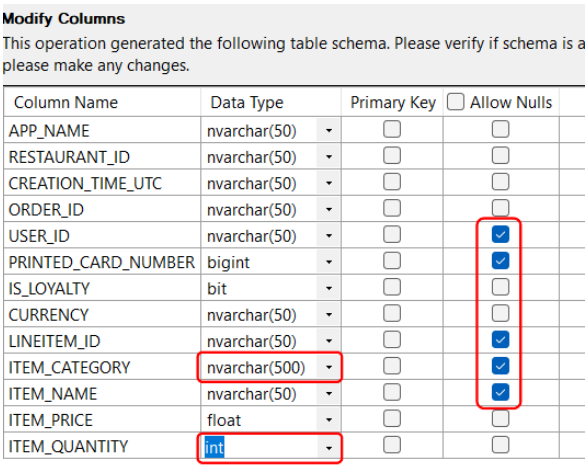


The file of **date_dim.csv** is uploaded easily, but for the other two, I need to change some data format during uploading:

Changes for **order_item_options.csv**:



and for **order_items.csv**:

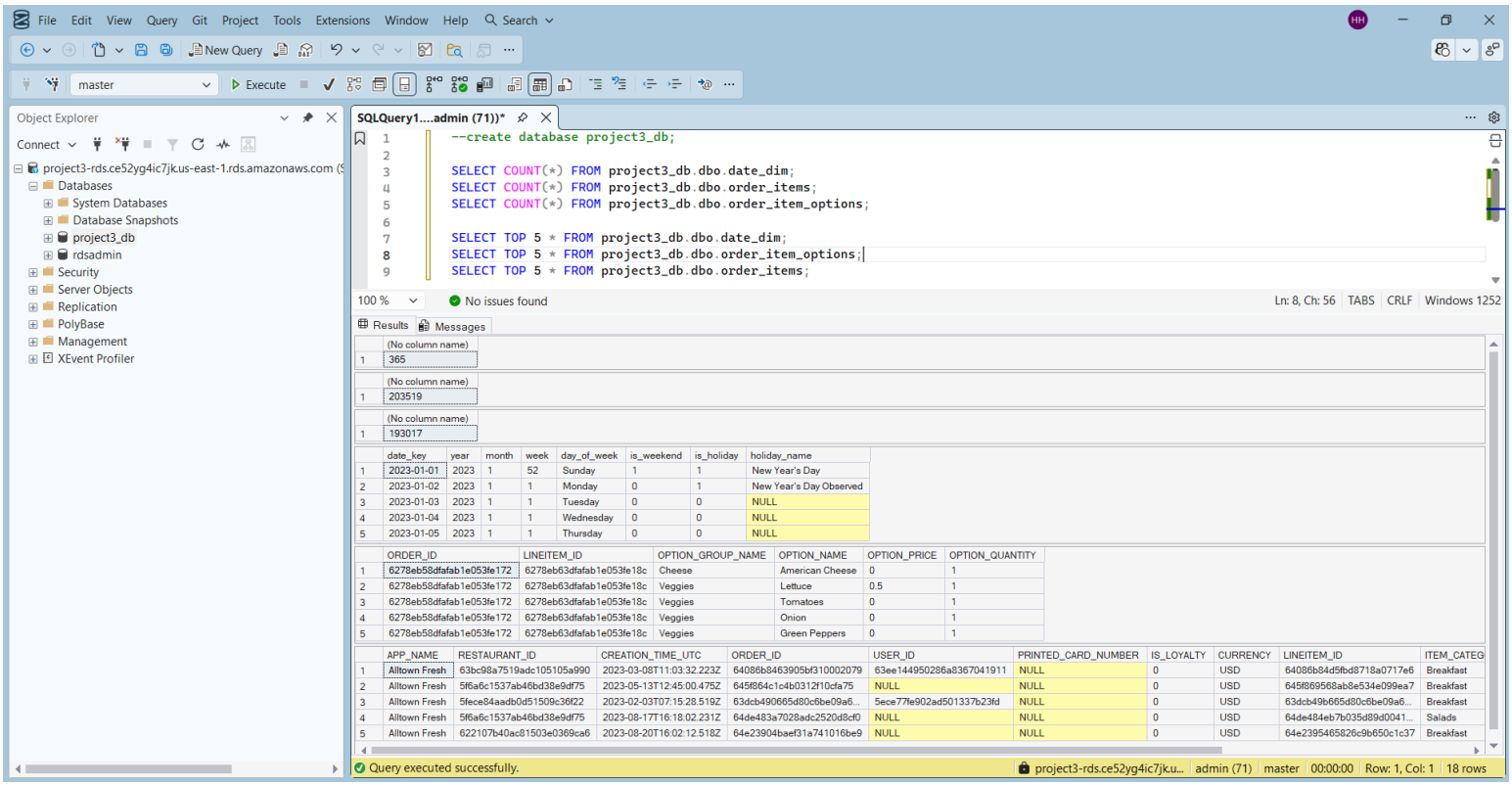


To validate all three tables are uploade inside our local SQL Server, I quickl y did this query:

```
SELECT COUNT(*) FROM project3_db.dbo.date_dim;
SELECT COUNT(*) FROM project3_db.dbo.order_items;
SELECT COUNT(*) FROM project3_db.dbo.order_item_options;

SELECT TOP 5 * FROM project3_db.dbo.date_dim;
SELECT TOP 5 * FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.order_items;
```

And the results show that everything is OK!


File Edit View Query Git Project Tools Extensions Window Help Search
New Query
master
Execute
Object Explorer
Connect
project3-rds.ce52yg4ic7jk.us-east-1.rds.amazonaws.com
Databases
System Databases
Database Snapshots
project3_db
rdsadmin
Security
Server Objects
Replication
PolyBase
Management
XEvent Profiler
SQLQuery1....admin (71))*
--create database project3_db;
SELECT COUNT(*) FROM project3_db.dbo.date_dim;
SELECT COUNT(*) FROM project3_db.dbo.order_items;
SELECT COUNT(*) FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.date_dim;
SELECT TOP 5 * FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.order_items;
100 %
No issues found
Ln: 8, Ch: 56 TABS CRLF Windows 1252
Results
Messages
(No column name)
365
(No column name)
203519
(No column name)
193017
date_key year month week day_of_week is_weekend is_holiday holiday_name
2023-01-01 2023 1 52 Sunday 1 1 New Year's Day
2023-01-02 2023 1 1 Monday 0 1 New Year's Day Observed
2023-01-03 2023 1 1 Tuesday 0 0 NULL
2023-01-04 2023 1 1 Wednesday 0 0 NULL
2023-01-05 2023 1 1 Thursday 0 0 NULL
ORDER_ID LINEITEM_ID OPTION_GROUP_NAME OPTION_NAME OPTION_PRICE OPTION_QUANTITY
6278eb58dfafab1e053fe172 6278eb63dfafab1e053fe18c Cheese American Cheese 0 1
6278eb58dfafab1e053fe172 6278eb63dfafab1e053fe18c Veggies Lettuce 0.5 1
6278eb58dfafab1e053fe172 6278eb63dfafab1e053fe18c Veggies Tomatoes 0 1
6278eb58dfafab1e053fe172 6278eb63dfafab1e053fe18c Veggies Onion 0 1
6278eb58dfafab1e053fe172 6278eb63dfafab1e053fe18c Veggies Green Peppers 0 1
APP_NAME RESTAURANT_ID CREATION_TIME_UTC ORDER_ID USER_ID PRINTED_CARD_NUMBER IS_LOYALTY CURRENCY LINEITEM_ID ITEM_CATEG
Alltown Fresh 63bc98a7519adc105105a990 2023-03-08T11:03:32.223Z 64086b8463905bf310002079 63ee144950286a8367041911 NULL 0 USD 64086b84d5fbd8718a0717e6 Breakfast
Alltown Fresh 5f6a6c1537ab46bd38e9df75 2023-05-13T12:45:00.475Z 645f864c1c4b0312f10cfa75 NULL NULL 0 USD 645f86956ab8e534e099ea7 Breakfast
Alltown Fresh 5fece84aadb0d51509c36f22 2023-02-03T07:15:28.519Z 63dcb490665d80c6be09a6... 5ece77fe902ad501337b23fd NULL 0 USD 63dcb49b665d80c6be09a6... Breakfast
Alltown Fresh 5f6a6c1537ab46bd38e9df75 2023-08-17T16:18:02.231Z 64de483a7028adc2520d8cf0 NULL NULL 0 USD 64de484eb7b035d89d0041... Salads
Alltown Fresh 622107b40ac81503e0369ca6 2023-08-20T16:02:12.518Z 64e23904baef31a741016be9 NULL NULL 0 USD 64e2395465826c9b650c1c37 Breakfast
Query executed successfully.
project3-rds.ce52yg4ic7jku... admin (71) master 00:00:00 Row: 1, Col: 1 18 rows

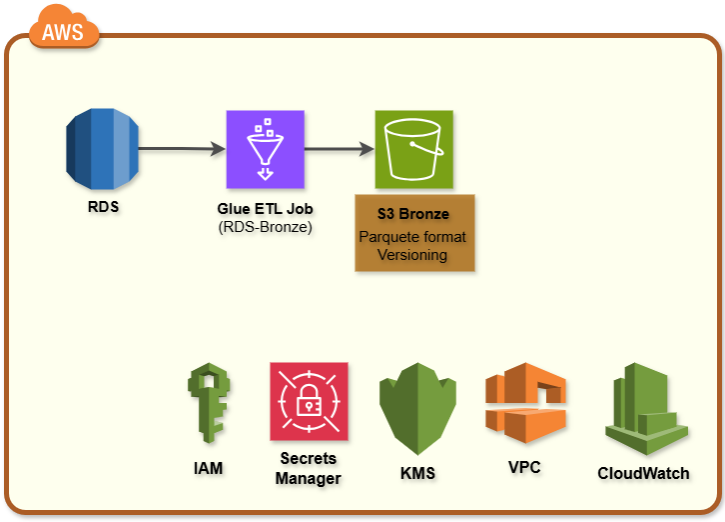
Step 2 – Data Landing (RDS to S3 Bronze)

2-1- Objective

The objective of this step is to land the ingested raw data from RDS database into an Amazon S3-based data lake, forming the Bronze layer. To ensure **data security at rest**, all Amazon S3 buckets (Bronze, Silver, and Gold) are protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**. This approach enables centralized key management and fine-grained access control through **IAM policies** to give permission for Glue ETL job to have access to S3 and RDS services.

Implementation Approach:

- AWS Glue ETL jobs will connect to SQL Server using JDBC.
- Source tables (**order_items**, **order_item_options**, and **date_dim**) will be extracted in batch mode.
- Data will be written to Amazon S3 in **Parquet** format to optimize storage efficiency and downstream processing.
- Enable S3 bucket versioning to retain all versions of the raw Parquet data.



I will give more details in the next sections.

2-2- Landing Storage (Bronze Layer)

First of all, I prefer to create the landing storage as a datalake. I used Amazon S3 service and create a bucet with name of **s3://project3-bronze-bucket/**. I have enabled its **versioning** option to store raw and immutable data to support auditability, reprocessing, and downstream transformations. After that, I created a folder inside that bucket with the name of **raw-data**. Then, all raw data will be written separately into the following S3 bucket: **s3://project3-bronze-bucket/raw-data/**

2-3- IAM Role

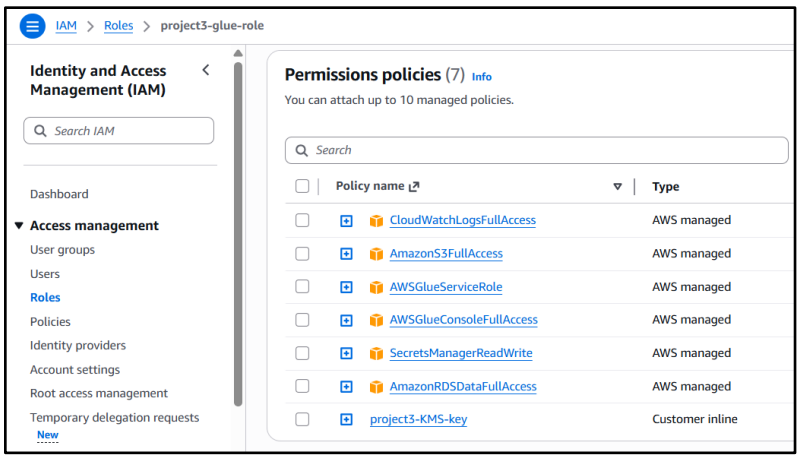
Before starting data ingestion using AWS Glue ETL jobs, I configured an IAM role to enable AWS Glue to securely access the required AWS services during job execution.

The following permissions were configured:

- **AWSGlueConsoleFullAccess:** Attached to my IAM user to allow me to create and manage AWS Glue resources, such as ETL jobs, workflows, and connections, through the AWS Management Console.
- **AWSGlueServiceRole:** Attached to the Glue job runtime role, allowing AWS Glue to assume the role at execution time and access required services, including Amazon S3, AWS Secrets Manager, Amazon CloudWatch Logs, and VPC networking resources.
- **AmazonS3FullAccess:** Granted to allow AWS Glue to read from and write to the Bronze, Silver, and Gold S3 buckets.
- **CloudWatchLogsFullAccess:** Enabled logging and monitoring of Glue job execution in Amazon CloudWatch.
- **SecretsManagerReadWrite:** Provided access to retrieve and manage secrets, including the RDS credentials.
- **AmazonRDSDataFullAccess:** Allowed AWS Glue to read raw source data from Amazon RDS using JDBC connectivity.
- **project3-KMS-key (Inline Policy):** A custom inline policy was created to allow encryption and decryption of sensitive data, including S3 bucket objects and RDS credentials stored in AWS Secrets Manager. The policy grants permissions to retrieve secret values and perform cryptographic operations using a customer-managed KMS key, as shown below:


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:us-east-1:000185048470:secret:project3-rds-secret-DppcqK"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Encrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:000185048470:key/cd88ca6d-a393-4430-987a-f47fd77c4124"
    }
  ]
}
```


The IAM role was named **project3-glue-role** and tagged with **project: 3**, and it was attached to the AWS Glue ETL jobs to ensure secure and authorized access during execution.



2-4- VPC Security Group

As internal VPC networking is used between Amazon RDS and AWS Glue jobs, a separate security group was created specifically for AWS Glue and named **project3-glue-sg**.

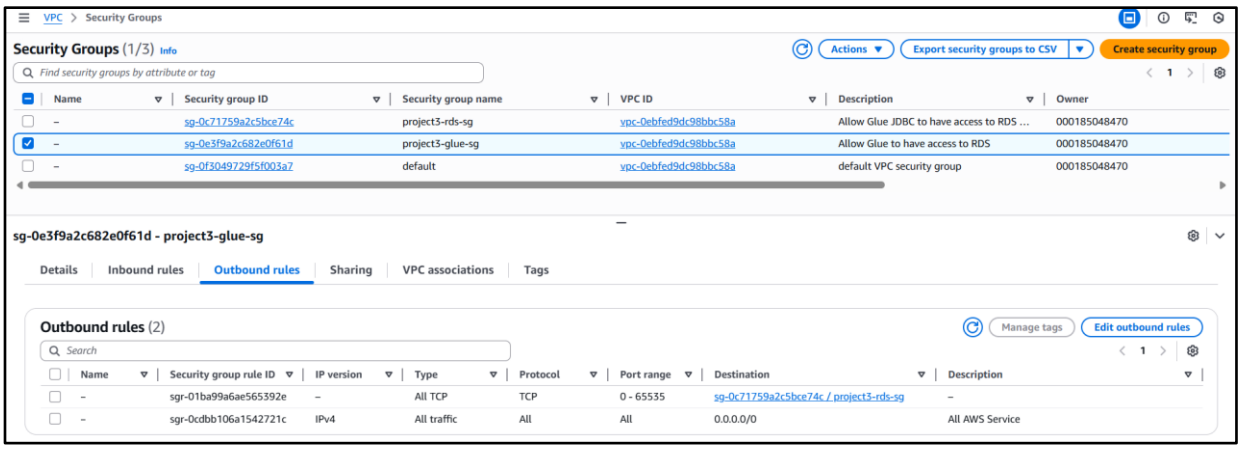
Within the **project3-glue-sg**, inbound and outbound IPv4 rules were configured as follows:

- **Inbound rules:** The inbound traffic is allowed from **project3-glue-sg** itself, enabling Glue components within the same security group to communicate with each other.
- **Outbound rules:** Outbound traffic is allowed to **project3-rds-sg**, permitting the AWS Glue job to connect to the Amazon RDS instance for data ingestion.

This configuration ensures secure, controlled communication between AWS Glue and Amazon RDS within the VPC while restricting unnecessary network access.

If I click on the **project3-glue-sg**, I can edit the **inbound** and **outbound** IPv4 for this security group.

As you can see above, the inbounde rule has one source, **project3-glue-sg**.



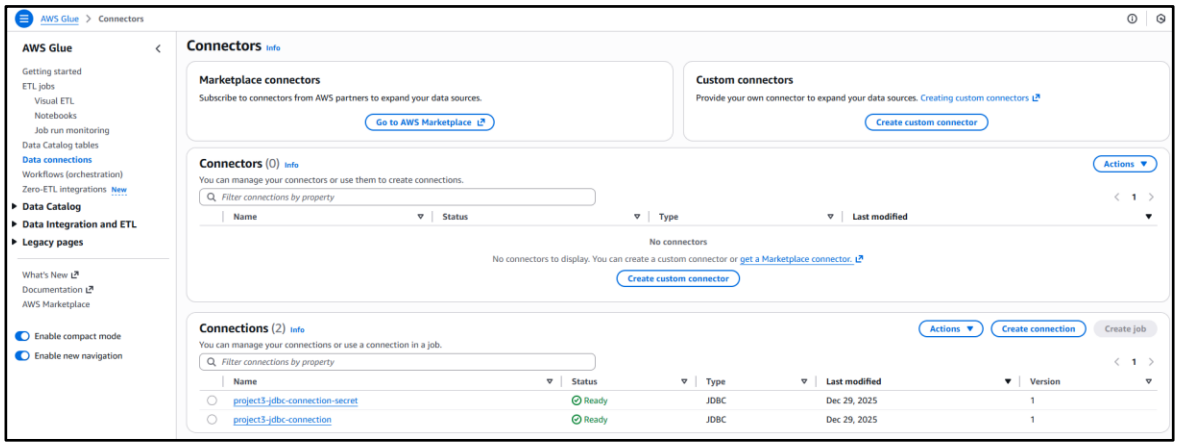
There are two outbound rules. One of them is the source of **project3-rds-sg**, and the other is general IP, which is open for All AWS Services, 0.0.0.0/0.

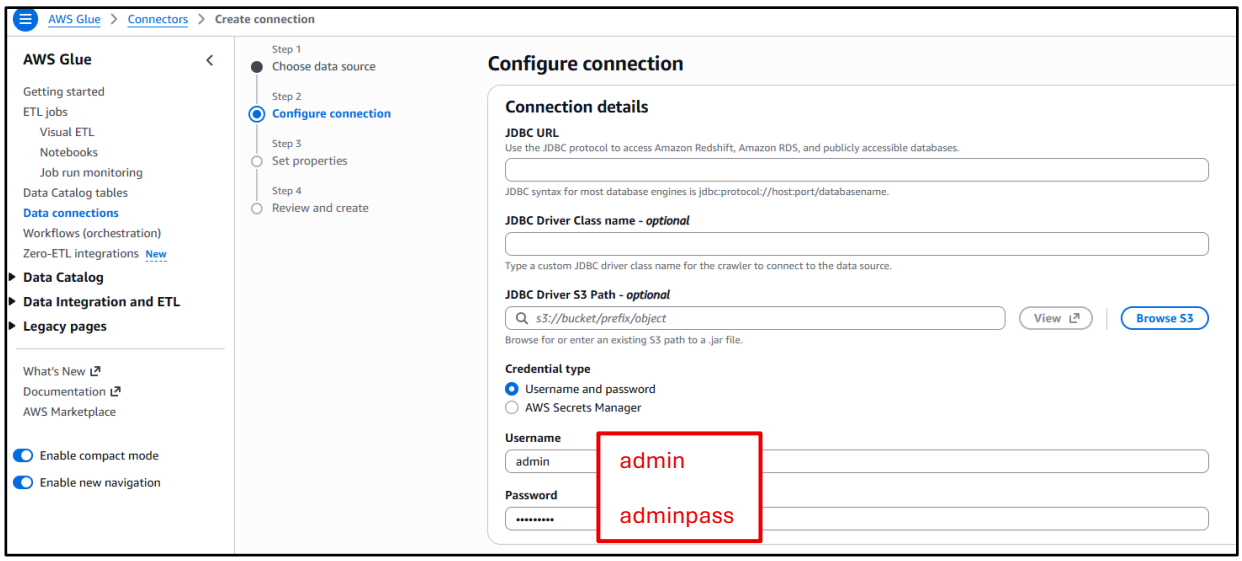
2-5- Glue JDBC Connection

To comply with the project requirement that all transformation logic be implemented using Spark, AWS Glue ETL job with JDBC connectivity will be used for data ingestion instead of AWS Lambda. This approach ensures that the ingestion process is both scalable and aligned with Spark-based processing standards. AWS Glue provides a distributed Spark environment that is well suited for:

- Reading relational data using JDBC
- Handling incremental and batch ingestion
- Scaling efficiently for future data growth
- Maintaining consistency across ingestion and transformation layers

In the section of AWS Glue → Data Connections, I created two connectors. The first one, named **project3-jdbc-connection**, which connects Glue to the RDS and I put the RDS credentials in the configuration of this connection, which is not secure (only for test). Then, I created the second connection, **project3-jdbc-connection-secret**, which all credentials are inside out secret in the secrets manager service. To create both connectons, I used JDBC data source.

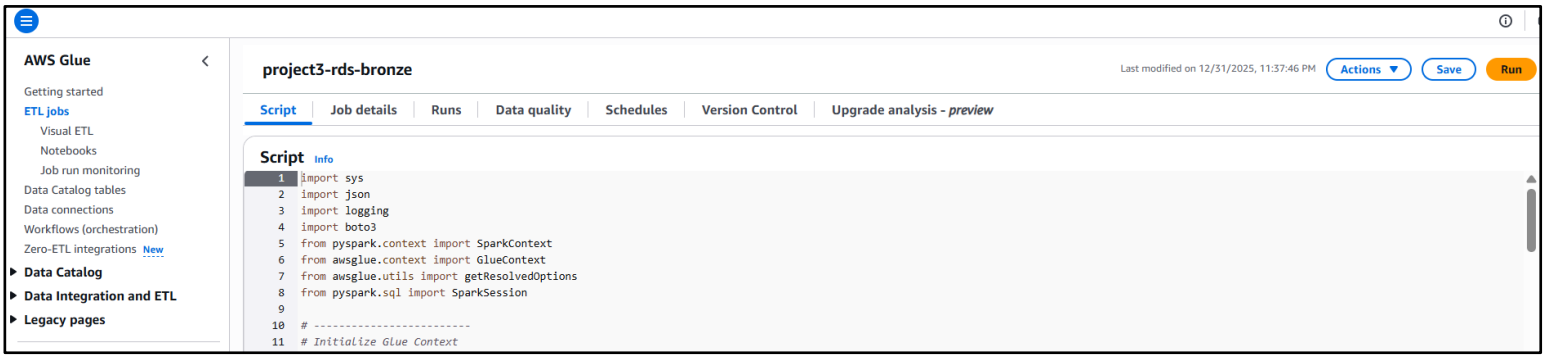




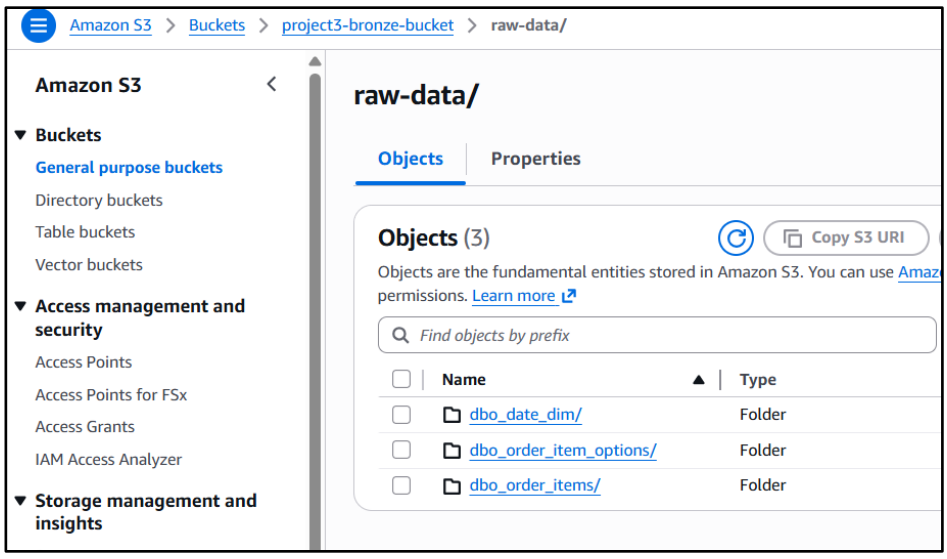
The important point is that the secret should be in the plaintext, not key/value, format. In that case Glue can read the secret successfully.

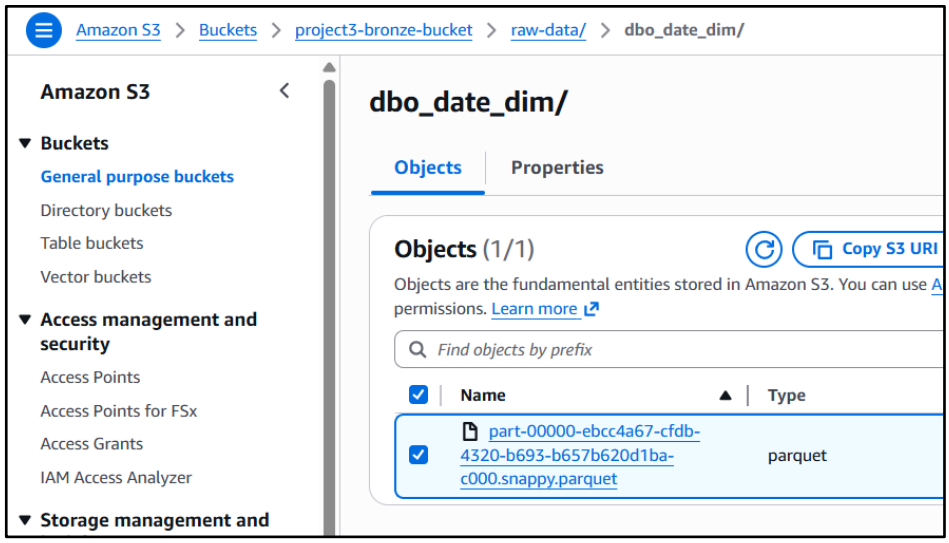
2-6- Glue ETL Job (RDS - Bronze)

Finally, after securing the connection between RDS and Glue, now, I am going to create the first AWS Glue PySpark ETL job, named **project3-rds-bronze**, to perform raw data ingestion from RDS database into S3 Bronze bucket and convert the file format from CSV into Parquet. For test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put all Glue ETL Jobs' script in another S3 code bucket. The script file name in the PySpark language is **project3-rds-bronze.py**.



After running the Glue ETL job, all three files landed from RDS database into S3 Bronze bucket successfully.





As you can see, there are three foders, with schema name and table name, in **s3://project3-bronze-bucket/raw-data/**, and inside each folder, there is a Parquet file.

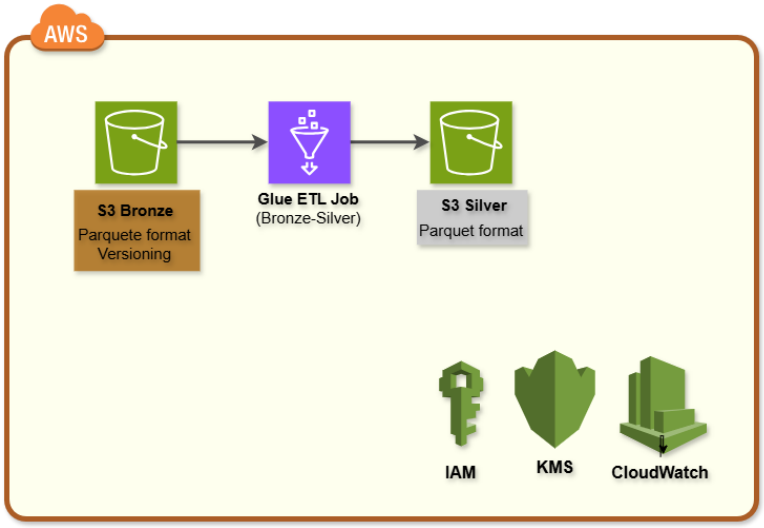
Step 3 – Transformation (S3 Bronze to S3 Silver)

3-1- Objective

The objective of the Silver layer is to transform raw ingested data into **clean**, **standardized**, and **analytically usable datasets**. This step focuses on improving data quality by resolving **missing values**, **removing duplicates**, **enforcing schemas**, and **preparing fact and dimension tables** for downstream analytics and metric computation.

Implementation Approach:

- AWS Glue ETL jobs will transform and clean up the raw data.
- Two fact tables of **fact_orders** and **fact_options**, and one dime table of **dim_date** will created.
- Data will be written to Amazon S3 in **Parquet** format to optimize storage efficiency and downstream processing.
- Amazon S3 Silver bucket is protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**.



3-2- Transformed Storage (Silver Layer)

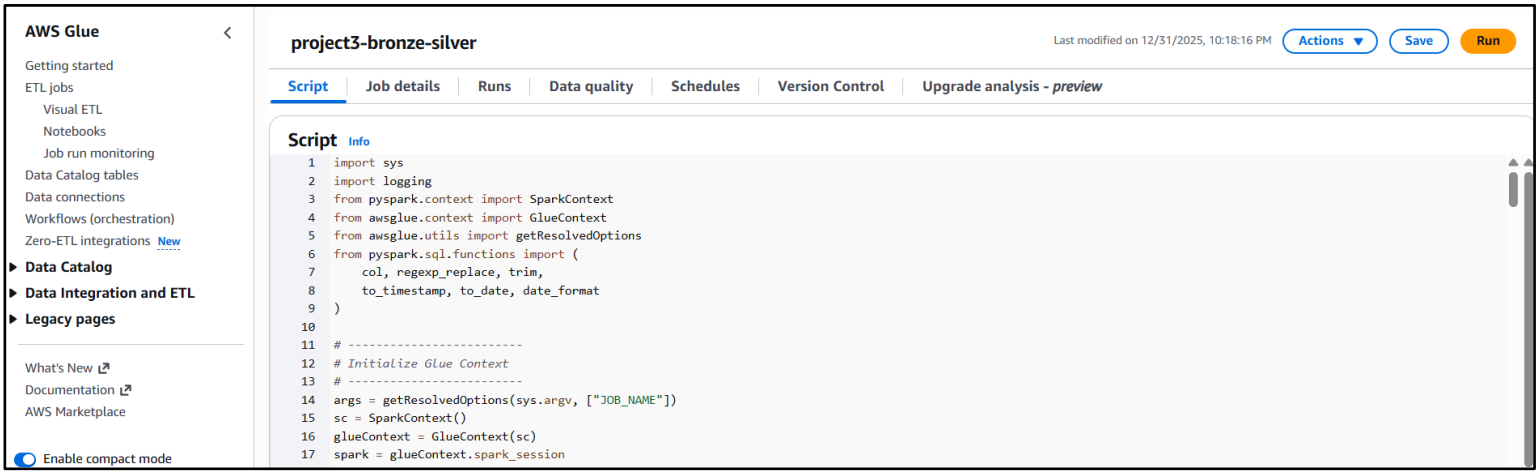
Data in this layer will be structured into fact and dimension–oriented datasets, enabling efficient joins and aggregations in subsequent processing stages. All three transformed data will be stored in a dedicated S3 bucket: **s3://project3-silver-bucket/**. I removed schema name from raw raw folders and classified them into dim and fact tables. Then, all transformed data will be written separately into their relatd folders. I have enabled the **SSE-KMS Encryption** for S3 Silver bucket.

3-3- Glue ETL Job (Bronze - Silver)

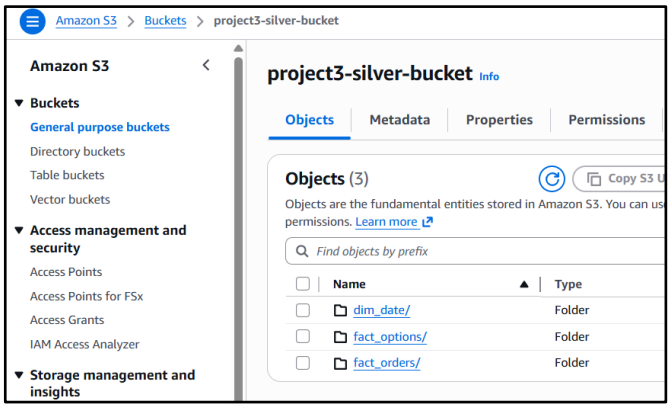
An AWS Glue PySpark ETL job, named **project3-bronze-silver**, will be executed to perform the following transformations:

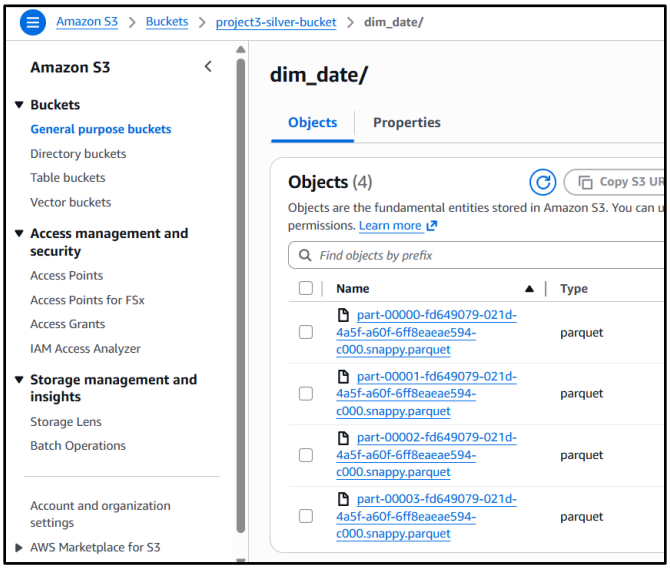
- **Schema Enforcement**
  - Explicitly define schemas for fact and dimension tables to prevent schema drift.
  - Ensure consistent column ordering and naming conventions.
- **Data Type Standardization**
  - Cast columns to appropriate data types (e.g., timestamps, numeric fields, boolean flags).
  - Ensure consistency across datasets for join keys and metrics.
- **Duplicate Handling**
  - Identify and remove duplicate records based on defined primary or composite keys.
  - Preserve only the most recent or valid records where applicable.
- **Missing Value Management**
  - Remove records with critical missing identifiers (e.g., primary keys).
  - Retain non-critical nulls where they are analytically meaningful (e.g., holiday names).
- **Column Normalization**
  - Standardize column names (e.g., lowercase, snake_case) for consistency and usability.
  - Align naming conventions across all datasets.
- **Optimized Storage Format**
  - Persist transformed datasets in **Parquet format** to improve query performance and reduce storage costs.

Again, for test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put all Glue ETL Jobs' script in another S3 code bucket. The script file name in the PySpark language is **project3-bronze-silver.py**.



After running the Glue ETL job, all three dim and fact folders with Parquet files inside them is created in the S3 Silver bucket successfully.





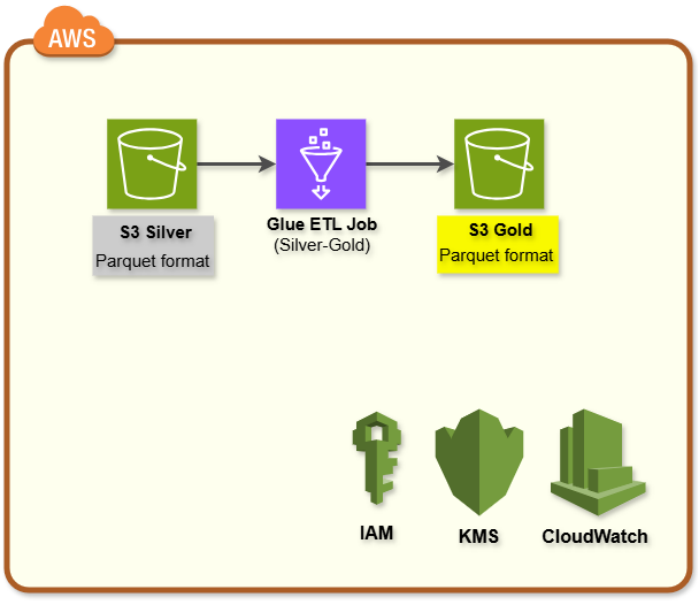
Step 4 – Analytics & Aggregation (S3 Silver to S3 Gold)

4-1- Objective

The objective of this step is to produce analytics-ready, aggregated datasets by joining and summarizing the dimensional and fact tables created in the Silver layer. The Gold layer serves as the single source of truth for business metrics and is optimized for direct consumption by visualization and reporting tools.

Notes:

- The Gold layer is explicitly optimized for **Streamlit dashboards**, enabling fast data access, simplified querying, and seamless visualization without additional transformation logic.
- The transformed datasets will be persisted in **Parquet format** to improve query performance and reduce storage costs.
- Amazon S3 Gold bucket is protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**.



4-2- Gold Layer Storage

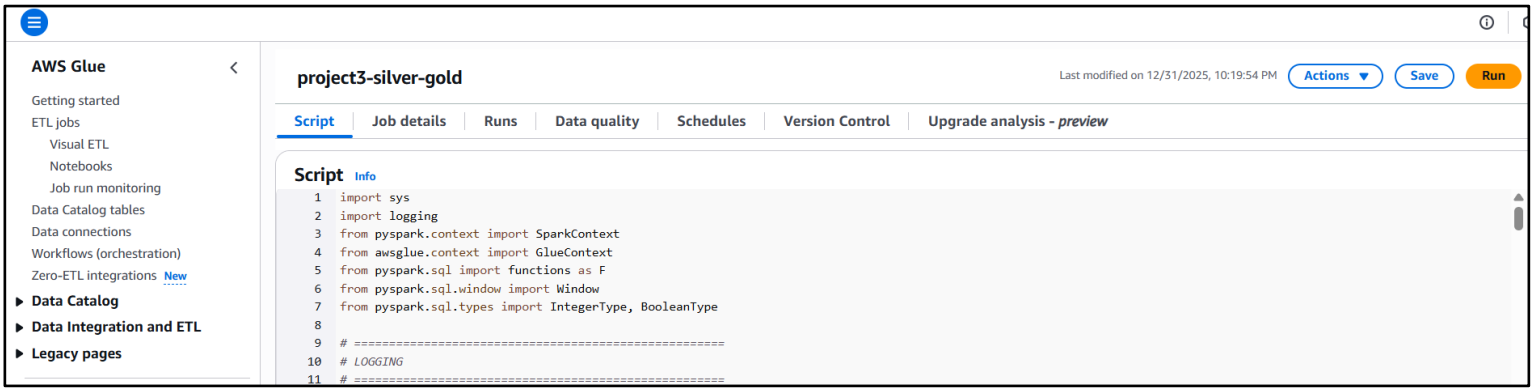
Data in this layer will be structured into two different fact tables and stored in a dedicated S3 bucket: **s3://project3-gold-bucket/**. Again, I have enabled the **SSE-KMS Encryption** for S3 Gold bucket. All fact tables are stored in the Parquet format.

4-3- Glue ETL Job (Silver - Gold)

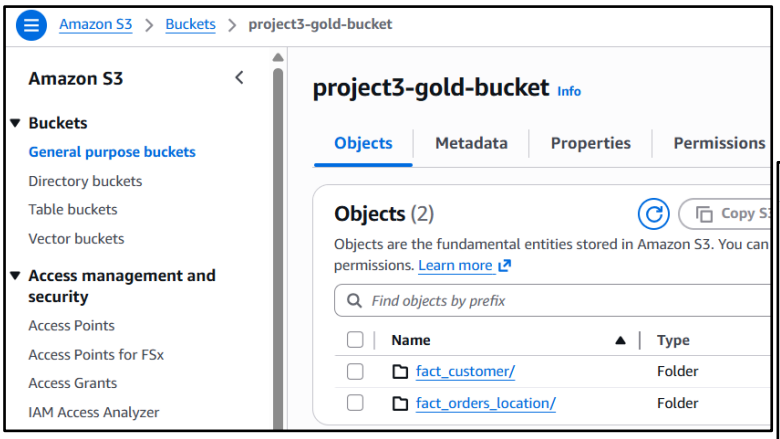
An AWS Glue PySpark ETL job, named **project3-silver-gold**, will be generate curated datasets and metrics aligned with the business analytics questions, including but not limited to:

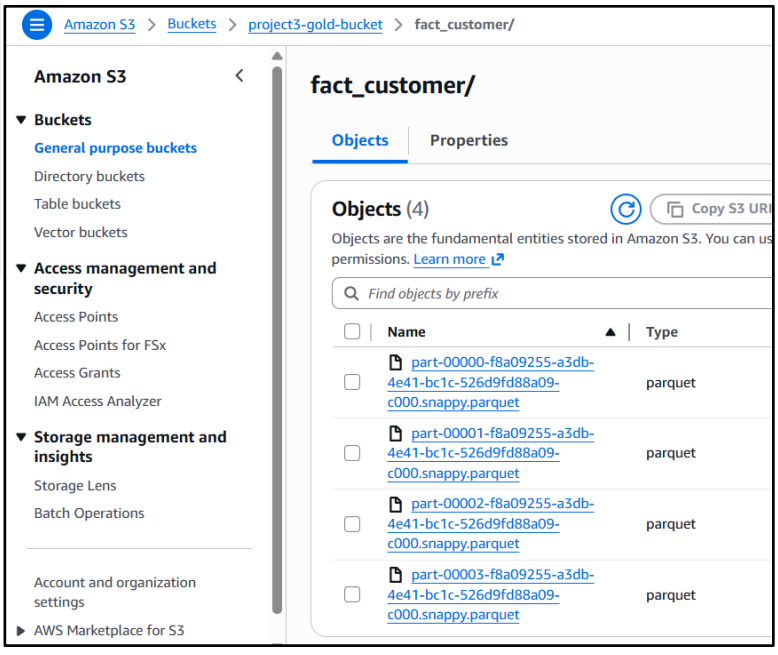
- Customer Lifetime Value (CLV) evolution (daily)
- RFM (Recency, Frequency, Monetary) snapshot
- Sales trends over time
- Location-level performance metrics
- Customer Churn indicators
- Other relevant KPI metrics

Again, for test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put the script, **project3-silver-gold.py**, in another S3 code bucket.



After running the Glue ETL job, two fact folders with Parquet files inside them (**fact_customer** and **fact_orders_location**) are created in the S3 Gold bucket successfully.





Step 5 – Visualization (Streamlit)

5-1- Objective

In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket. I have created two **key buttons** inside the dashboard for manual triggering the orchestration. I used Pandas for data handling (sufficient for moderate-sized parquet files), and built interactive dashboards for stakeholders with filters, charts, and KPI cards.

Step 6 – Jobs Orchestration and CI/CD

6-1- Objective

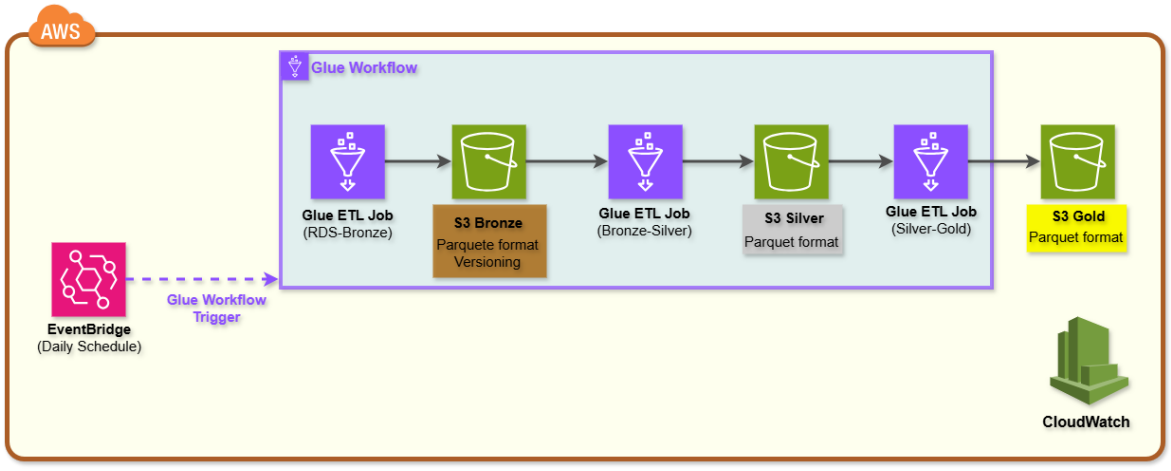
The objective of this component is to automate the end-to-end batch data pipeline on a daily schedule while ensuring reliable orchestration, clear job dependencies, and maintainable CI/CD practices. All data is ingested from SQL Server into AWS RDS, transformed through Bronze, Silver, and Gold layers, and made available for analytics and visualization without manual intervention. To support continuous development and deployment, all AWS Glue Spark scripts are version-controlled in a GitHub repository and automatically deployed to AWS S3 code bucket when changes occur.

6-2- Scheduling and Orchestration

For occhestraion of Glue ETL jobs, I used Glue Workflow, named **project3-bronze-silver-gold**, which acts as the central orchestration engine for the pipeline. This Glow Workflow is triggered **manually** or using **EventBridge**. EventBridge is scheduled to trigger the Glue Workflow daily (10 minutes for test).

The Glue Workflow executes jobs in a dependency-based sequence:

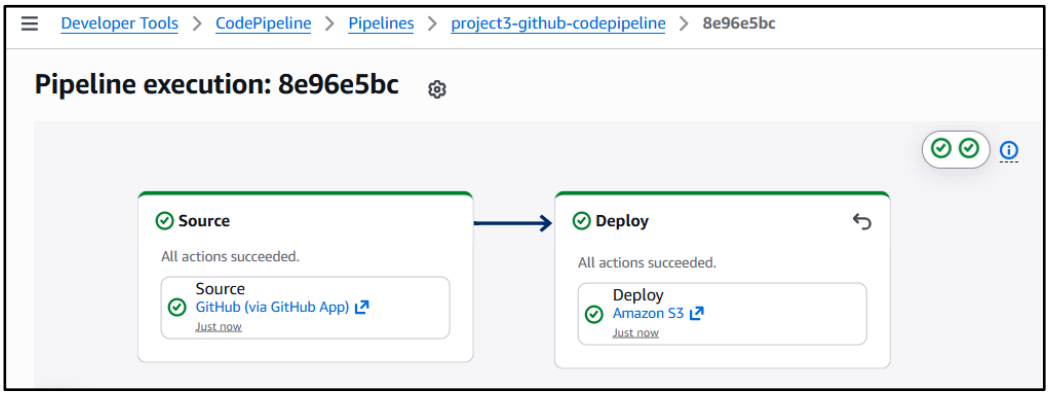
Glue ETL Job (RDS - Bronze) → Glue ETL Job (Bronze - Silver) → Glue ETL Job (Silver - Gold)



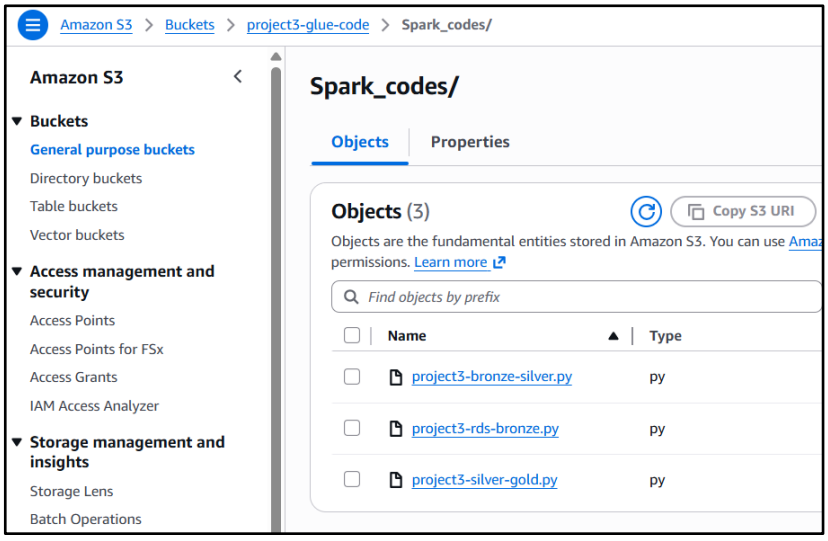
6-3- CI/CD Strategy

This approach enables automated deployment, traceability, and reproducibility of data pipelines. All PySpark scripts (**project3-rds-bronze.py**, **project3-bronze-silver.py**, **project3-silver-gold.py**) for three Glue ETL jobs are stored in a **GitHub** repository**:** https://github.com/Hadi2468/ELT_Project3/tree/main/Spark_codes.

The CI/CD pipeline automatically detects changes or additions to Glue job scripts in the GitHub repo and immediately deploys updated scripts to another S3 (code) bucket, name **s3://project3-code-bucket/** using AWS CodePipeline service.

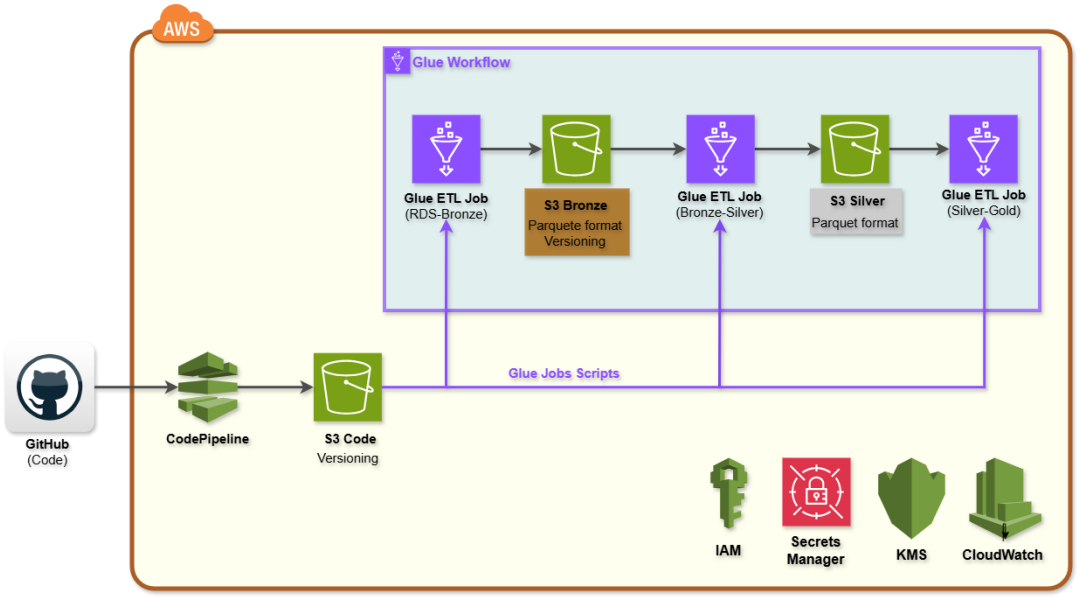


I have enabled its **versioning** option to store all versions of codes to support auditability, reprocessing, and downstream transformations. Also, I have enabled the SSE_KMS encryption in Amazon S3 Code bucket using AWS Key Management Service (KMS) for more security.



All Glue jobs reference their scripts directly from the S3 code bucket, ensuring:

- Version consistency
- Easy rollback
- No manual code uploads

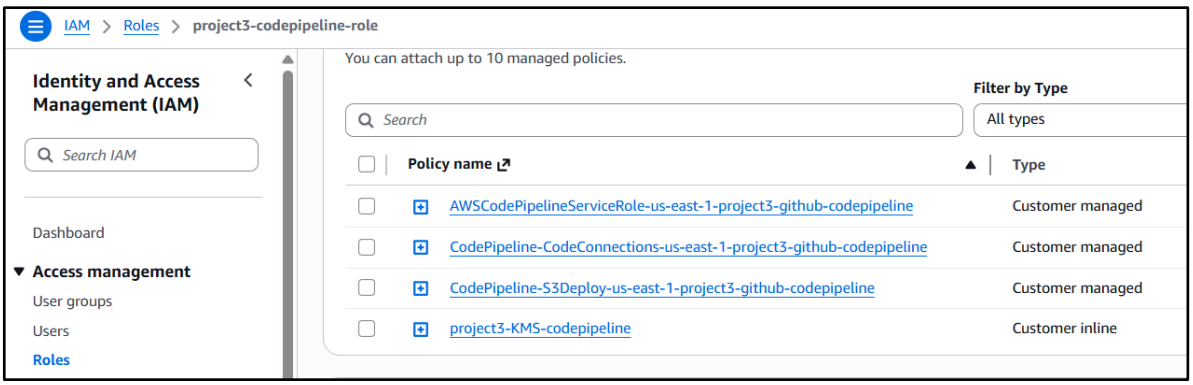


6-4- IAM Role

Before starting the code transfer using AWS CodePipeline, I configured an IAM role named **project3-codepipeline-role** and tagged with **project: 3** This role grants CodePipeline the required permissions to securely connect to the GitHub repository, retrieve the PySpark source code, and deploy the artifacts to Amazon S3. The following permissions were configured:

- **AWSCodePipelineServiceRole**: Grants CodePipeline core permissions to orchestrate pipeline stages and interact with AWS services on behalf of the pipeline.
- **CodePipeline-CodeConnections**: Allows CodePipeline to establish and manage secure connections to external source providers such as GitHub using AWS CodeStar Connections.
- **CodePipeline-S3Deploy**: Provides permission for CodePipeline to upload, read, and manage build artifacts in Amazon S3 buckets used during the deployment process.
- **project3-KMS-codepipeline (Inline Policy)**: A custom inline policy was created to allow AWS CodePipeline to use a customer-managed KMS key for encrypting and decrypting pipeline artifacts, as shown below Enables CodePipeline to encrypt and decrypt artifacts stored in Amazon S3 using AWS Key Management Service (KMS).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Resource": "arn:aws:kms:us-east-1:000185048470:key/cd88ca6d-a393-4430-987a-f47fd77c4124"
    }
  ]
}
```



The benefits of this orchestration strategy are that this design ensures:

- Clear and deterministic job dependencies
- Controlled execution order
- Parallel processing where appropriate
- Minimal operational complexity
- Scalable and production-ready CI/CD practices
- Failure isolation and observability using **Amazon CloudWatch**

Step 7 – Demonestration

Now, we want to demonestrate the project and run the EventBridge to trigger the Glue Workflue function to start getting data from local SQL Server, run Glue ETL jobs for data ingestion, transformation, and aggregation analysis in the approach of Bronze-Silver-Gold. The first Glue job does some data clean-up stuff on the raw CSV files and convert them into parquet files and saves dim and fact tables inside Silver S3 bucket. The second and third Glue jobs will be run after the previous Glue job finished, and all final BI-related dim and fact tables will be created in the Gold S3 bucket.

7-1- CI/CD approach

As soon as we click on **GitHub Deployment via CodePipeline** button in the Streamlit dashboard (maually), or any small change in the GitHub repository, the CodePipeline automatically gets the update from GitHub repo and shares it inside the S3 code bucket. Then, the PySpark scripts are available for Glue ETL jobs immedietely.

7-2- Trigger Glue Workflow

As soon as we click on **Trigger Glue Job Manually** button in the Streamlit dashboard (maually), or the scheduled time on the EventBridge is approached, the EventBridge will trigger the Glue Workflow and all three Glue ETL jobs get their PySpark scripts from the S3 code bucket and start one after one successfully.

As you can see in the **project3-rds-bronze.py** file, the first Glue ETL job ingests all CSV files from RDS database and put them in the S3 Bronze bucket with Parquet format. After succeeding in this process, the Glue Workflow triggers the second Glue ETL job, and the orchestration keep ing on.

As shown in the **project3-rds-bronze.py** file, the first AWS Glue ETL job ingests all CSV files from the RDS database and stores them in the Amazon S3 Bronze bucket in Parquet format. Once this process is completed successfully, the Glue Workflow automatically triggers the second Glue ETL job, and the orchestration continues sequentially.

To validate, I have downloaded all Parquet files from AWS Bronze, Silver, and Gold buckets into my local machine, and checked them carefully.

```
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_date_dim/ ./data/s3/date_dim --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_order_items/ ./data/s3/order_items --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_order_item_options/ ./data/s3/order_item_options --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/fact_options/ ./data/s3/fact_options --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/fact_orders/ ./data/s3/fact_orders --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/dim_date/ ./data/s3/dim_date --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-gold-bucket/fact_customer/ ./data/s3/fact_customer/ --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-gold-bucket/fact_orders_location/ ./data/s3/fact_orders_location/ --recursive --exclude "*" --include "*.parquet"
```

After downloading Parquet files for both dim and fact tables, now in Jupyter Notebook and using Pandas library, explored on those tables for validation. In the Bronze layer, everything is OK. Also, in the Silver layer, we can see, all the cleaning actions are done successfully.

date_dim table:

- Rename table **date_dim** into **dim_date**

order_items table:

- Rename table **order_items** into **fact_orders**
- Normalizes column names (lower_case and snake_case)
- Drops one record with **NULL 'LINEITEM_ID'**
- Splits **'CREATION_TIME_UTC'** into three columns of **'creation_date'** (DATE), **'creation_time'** (TIME), and removes T, Z, and milliseconds in **'creation_ts_utc'**
- Writes clean Parquet to Silver bucket

order_item_options table:

- Rename table **order_items** into **fact_optios**
- Normalizes column names (lower_case and snake_case)
- Deduplicates correctly 2299 rows from **fact_options**

Here are the results of transforamtion in the S3 Gold layer:

fact_customer_daily_metrics	
Columns	**Type**
user_id [PK]	object
creation_d	datetime64[ns]
orders_daily	int64
revenue_daily	float64
loyalty_revenue	float64
loyalty_orders	int64
recency_days	Int32
churn_risk_flag	object
clv_cumulative	float64
frequency_7d	float64
monetary_7d	float64
frequency_30d	float64
monetary_30d	float64
frequency_90d	float64
monetary_90d	float64

fact_orders_location_performance_daily	
Columns	**Type**
creation_date	datetime64[ns]
restaurant_id [PK]	object
total_orders	int64
total_revenue	float64
avg_order_value	float64
loyalty_revenue	float64
loyalty_orders	int64
non_loyalty_revenue	float64
non_loyalty_orders	int64
period_day	Int32
period_month	Int32
period_week	Int32
year	Int32
revenue_rank_day	Int32
revenue_rank_week	Int32
revenue_rank_month	Int32

Conclusion

In this project, an end-to-end, production-oriented data engineering pipeline was designed and implemented to transform raw transactional data into analytics-ready business insights. Starting from exploratory data analysis on local CSV files, the project systematically addressed data quality issues, schema design, and primary key definition before moving into a fully automated cloud-based architecture on AWS.

The pipeline follows a **Bronze–Silver–Gold** data lake architecture using Amazon S3, ensuring clear separation of raw, cleaned, and curated datasets. Data ingestion was securely performed from a local SQL Server into Amazon RDS and subsequently landed into the S3 Bronze layer using AWS Glue PySpark jobs with JDBC connectivity. Strong security practices were applied throughout the system, including private VPC networking, dedicated security groups, IAM least-privilege roles, AWS Secrets Manager for credential management, and SSE-KMS encryption across all S3 buckets.

Data transformation logic implemented in the Silver layer successfully enforced schemas, removed duplicates, handled missing values, normalized column naming, and produced well-structured fact and dimension tables. The Gold layer aggregated these cleaned datasets into analytics-ready fact tables optimized for business intelligence use cases such as customer lifetime value (CLV), RFM analysis, churn indicators, sales trends, and location-level performance metrics.

Automation and reliability were achieved through **AWS Glue Workflows** for orchestration and **Amazon EventBridge** for scheduling, enabling deterministic, dependency-driven execution of all ETL jobs. Additionally, a robust **CI/CD pipeline** using **AWS CodePipeline** and **GitHub** was implemented to manage Glue job scripts, ensuring version control, reproducibility, easy rollback, and zero manual deployment effort.

Finally, the curated Gold datasets were consumed directly by **Streamlit dashboards**, demonstrating how the pipeline supports real-time business insights and decision-making. Overall, this project demonstrates a scalable, secure, and maintainable data platform that closely mirrors real-world production data engineering systems and effectively bridges raw data ingestion with actionable analytics.