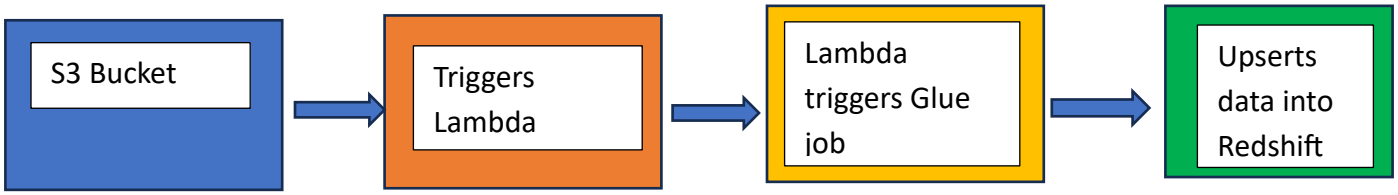
ECOMMERCE DATA PIPELINE USING S3,GLUE,LAMBDA,REDSHIFT

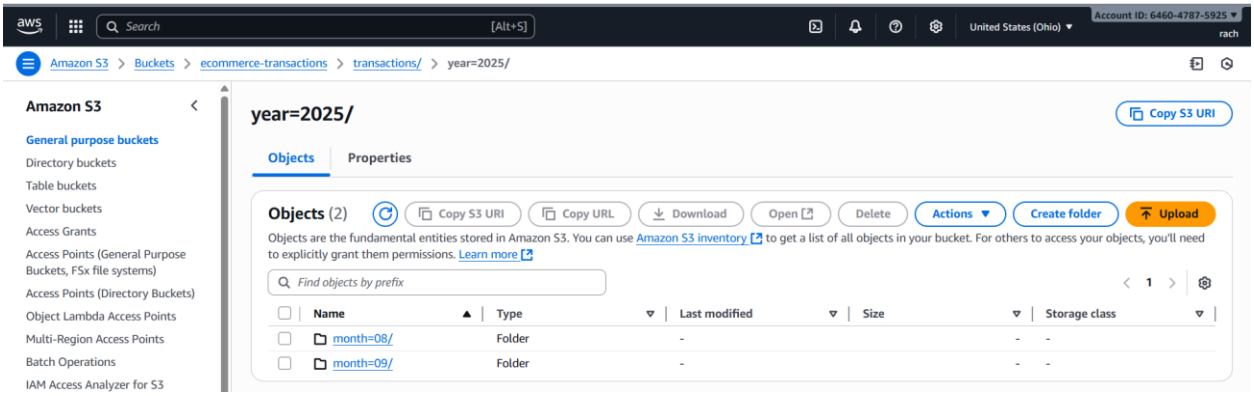
This project involves building a sophisticated event-driven data ingestion and transformation pipeline focusing on e-commerce transactional data. I have designed a system using AWS services such as S3, Lambda, Glue, Redshift, to ingest, transform, validate, and upsert data into Amazon Redshift for analytical purposes.

Architecture diagram

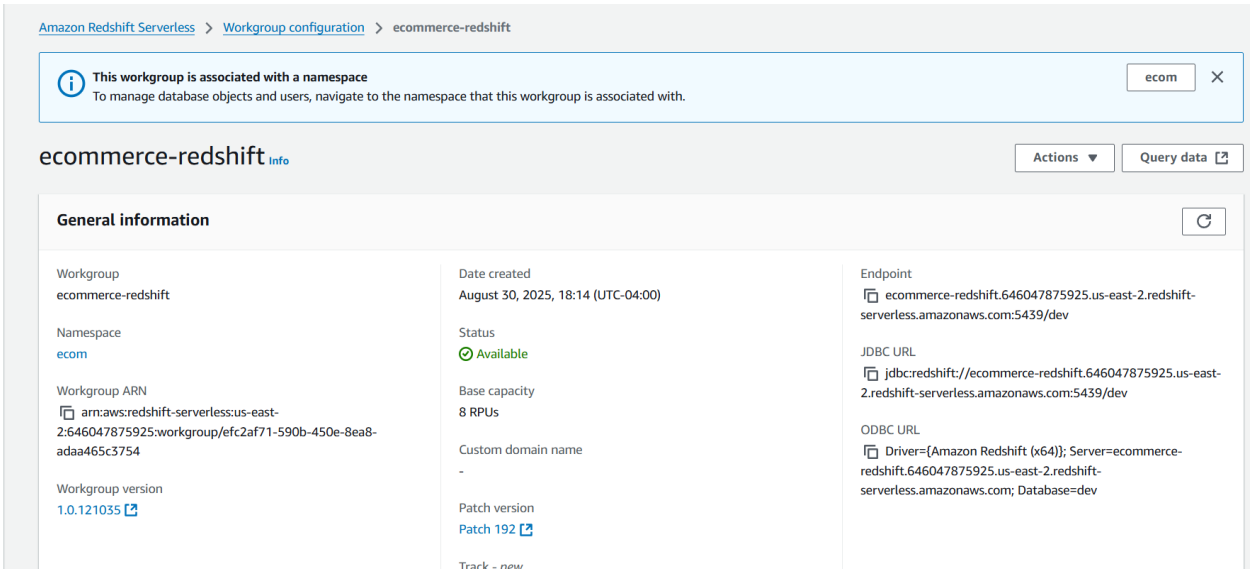


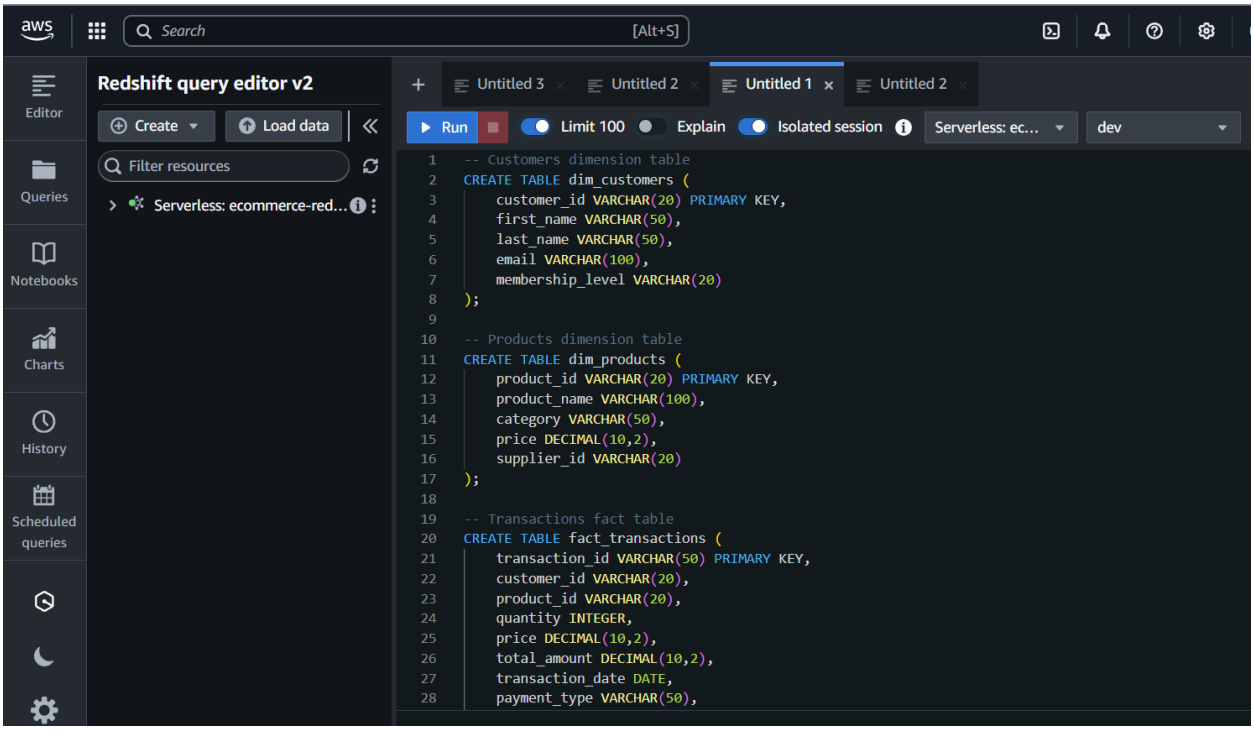
- **Mock Data Generation- script attached**

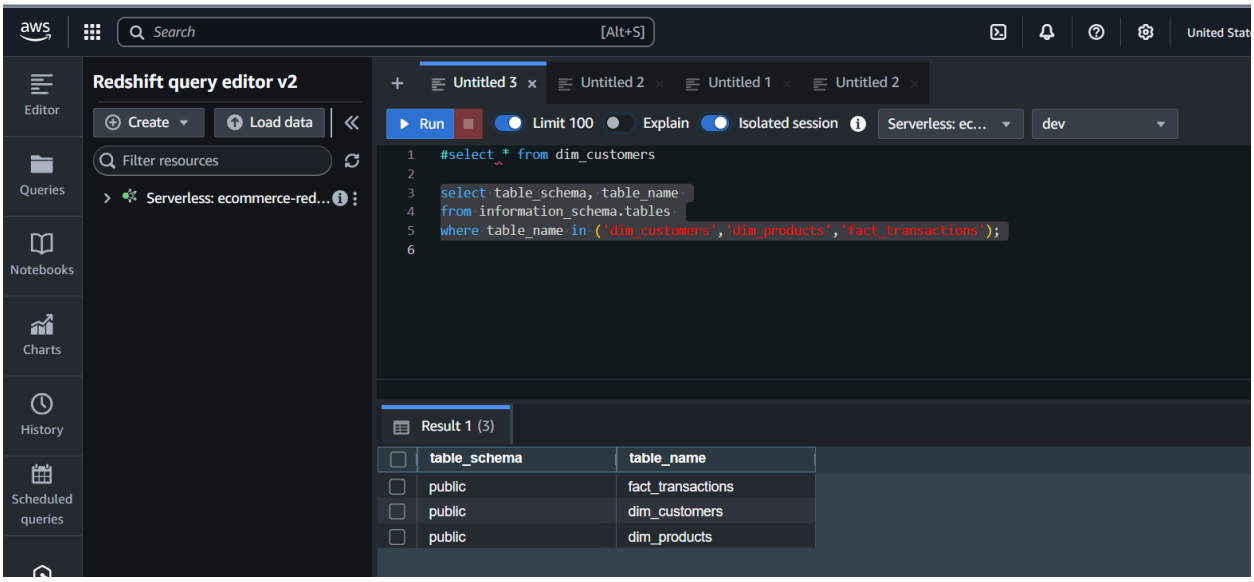
○ Transaction Data: Generate daily transaction files in CSV format, stored using the following hive-style partitioning in S3:



- **Dimension Tables and Sample Records- Pre-load dim_customers, dim_products dimension tables into Redshift as part of the setup process. Setup Redshift cluster first.**

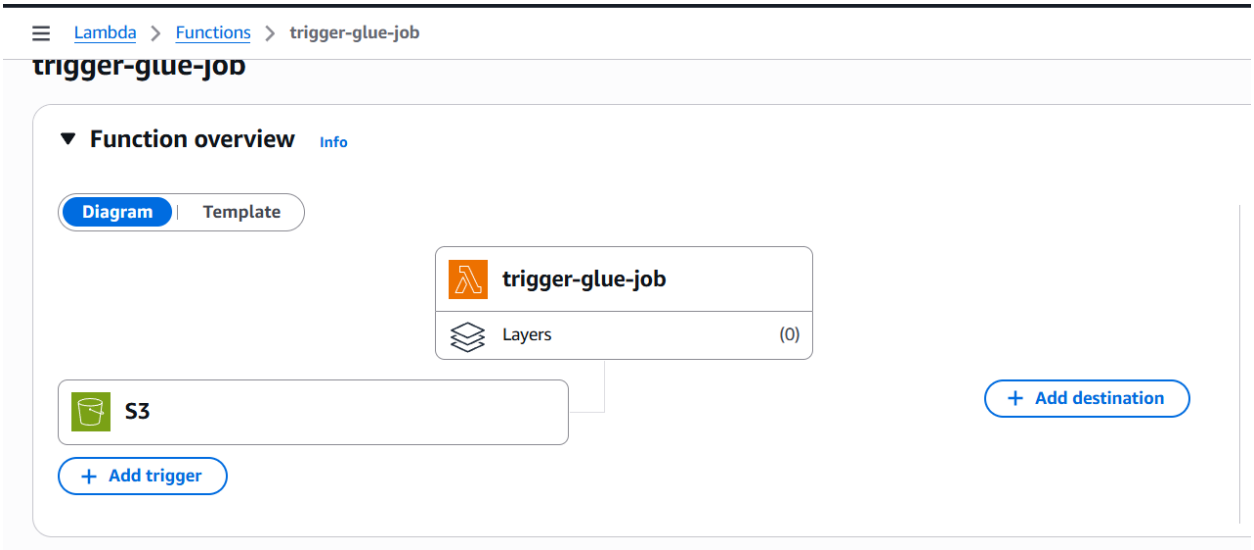

Amazon Redshift Serverless > Workgroup configuration > ecommerce-redshift
This workgroup is associated with a namespace
To manage database objects and users, navigate to the namespace that this workgroup is associated with.
ecom
ecommerce-redshift Info
Actions
Query data
General information
Workgroup
ecommerce-redshift
Namespace
ecom
Workgroup ARN
arn:aws:redshift-serverless:us-east-2:646047875925:workgroup/efc2af71-590b-450e-8ea8-adaa465c3754
Workgroup version
1.0.121035
Date created
August 30, 2025, 18:14 (UTC-04:00)
Status
Available
Base capacity
8 RPUs
Custom domain name
-
Patch version
Patch 192
Endpoint
ecommerce-redshift.646047875925.us-east-2.redshift-serverless.amazonaws.com:5439/dev
JDBC URL
jdbc:redshift://ecommerce-redshift.646047875925.us-east-2.redshift-serverless.amazonaws.com:5439/dev
ODBC URL
Driver={Amazon Redshift (x64)}; Server=ecommerce-redshift.646047875925.us-east-2.redshift-serverless.amazonaws.com; Database=dev


Search
[Alt+S]
Redshift query editor v2
Editor
Queries
Notebooks
Charts
History
Scheduled queries
Create
Load data
Filter resources
Serverless: ecommerce-red...
Untitled 3
Untitled 2
Untitled 1
Untitled 2
Run
Limit 100
Explain
Isolated session
Serverless: ec...
dev
-- Customers dimension table
CREATE TABLE dim_customers (
customer_id VARCHAR(20) PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
email VARCHAR(100),
membership_level VARCHAR(20)
);
-- Products dimension table
CREATE TABLE dim_products (
product_id VARCHAR(20) PRIMARY KEY,
product_name VARCHAR(100),
category VARCHAR(50),
price DECIMAL(10,2),
supplier_id VARCHAR(20)
);
-- Transactions fact table
CREATE TABLE fact_transactions (
transaction_id VARCHAR(50) PRIMARY KEY,
customer_id VARCHAR(20),
product_id VARCHAR(20),
quantity INTEGER,
price DECIMAL(10,2),
total_amount DECIMAL(10,2),
transaction_date DATE,
payment_type VARCHAR(50),



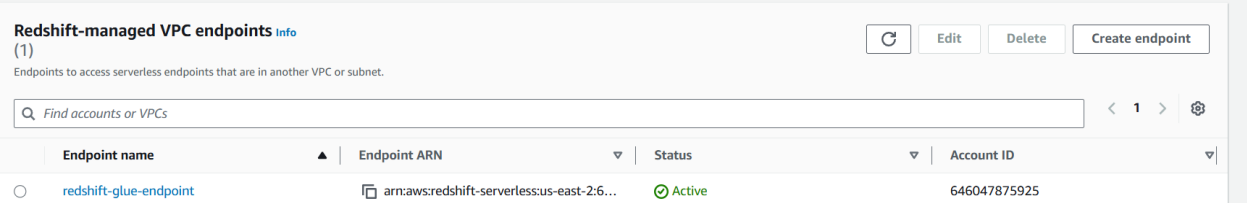
● **Data Ingestion and Transformation with AWS Glue**

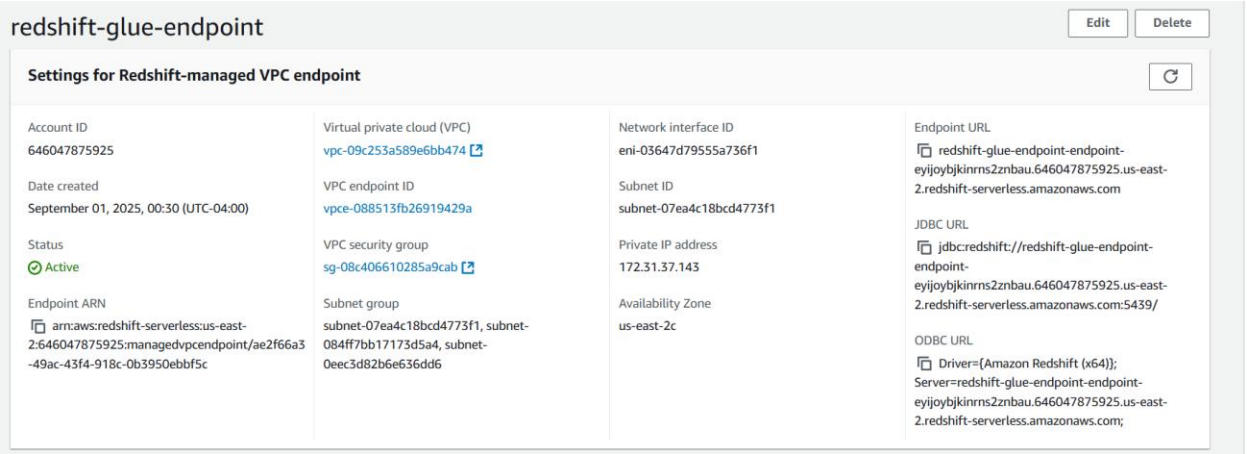
○ **Event-Driven Ingestion: Configure an AWS Lambda function to trigger AWS Glue jobs upon detecting new files in the S3 transactions folder.**



● **Data Transformation and Validation using Glue ETL**

Establish Glue connection to Redshift by creating VPC endpoint





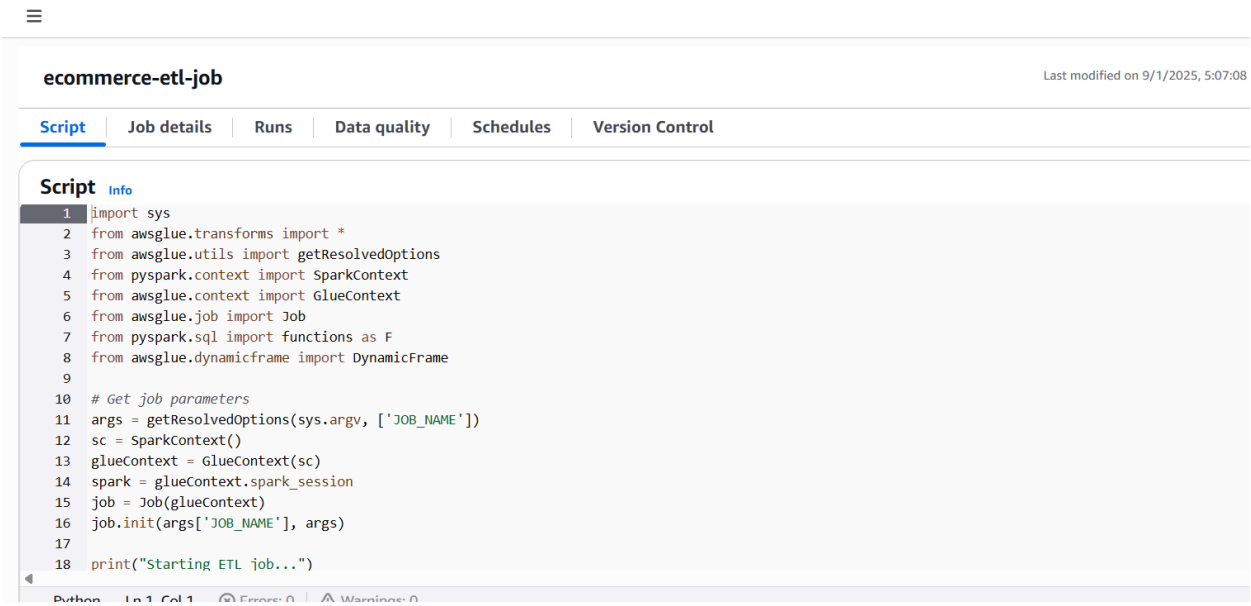
○ **Join Operations**: Enrich transactional data by joining with dim_products and dim_customers based on product_id and customer_id.

○ **Data Validation**: Include validation logic in the Glue job to filter out transactions with invalid customer_id or product_id (e.g., missing in dimension tables).

○ **Additional Transformations**: Calculate the total transaction amount (quantity * price) and categorize transactions into different classes based on the amount (e.g., "Small", "Medium", "Large").

○ **Upsert Operation in Amazon Redshift**

○ Design the Glue job to perform an upsert operation into the fact_transactions table in Redshift, using transaction_id as the key. Consider transaction date and status when determining if an existing record should be updated.


ecommerce-etl-job
Last modified on 9/1/2025, 5:07:08
Script
Job details
Runs
Data quality
Schedules
Version Control
Script Info
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7 from pyspark.sql import functions as F
8 from awsglue.dynamicframe import DynamicFrame
9
10 # Get job parameters
11 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
12 sc = SparkContext()
13 glueContext = GlueContext(sc)
14 spark = glueContext.spark_session
15 job = Job(glueContext)
16 job.init(args['JOB_NAME'], args)
17
18 print("Starting ETL job...")