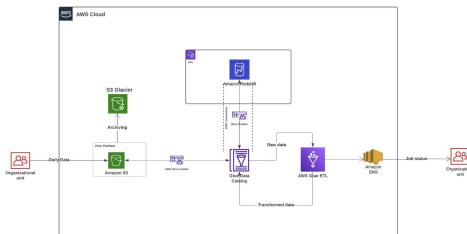


PS/Objective:

Implement an end-to-end, event-driven ETL (Extract, Transform, Load) pipeline on AWS to manage the daily processing of ecommerce transactional data. The solution should automate data extraction, transformation, validation, loading into a Redshift data warehouse, and archiving of outdated data, with process updates communicated via SNS. Additionally, establish a JDBC connection for data extraction or loading processes where necessary.

Develop an automated, event-driven ETL pipeline on AWS to handle daily ecommerce transaction data stored in an S3 bucket with Hive partitions (partitioned by day). The pipeline should filter out customers and products not existing in the current table, transform and load the data into a transactions table, notify stakeholders upon successful data load, and archive data older than a week in S3.



Step 1: Create a Redshift cluster with the Following Parameters

1. Select Redshift service.
2. Click on "Create cluster"
3. Give cluster a name
4. Choose the size of the cluster -> I'll choose
5. Node type -> dc2.large(cuz free trial)
6. Number of nodes -> 2
7. Admin user name -> Give a username
8. Check "Manually add the admin password" -> Add a password
9. Create a IAM role with following permissions

Policy name	Type	Attached entities
AmazonRedshift-CommandsAccessPolicy-202406...	Customer managed	1
AmazonRedshiftAllCommandsFullAccess	AWS managed	3
a. AmazonRedshiftFullAccess	AWS managed	7
AmazonS3FullAccess	AWS managed	19
AWSGlueConsoleFullAccess	AWS managed	5

10. Click on "Create Cluster"

Step2: Prep the Data warehouse

1. Create and Load customers, products table.
2. Create fact_transactions table.
1. Once the cluster is created
2. Select the cluster
3. Open Query editor v2
4. Log in using the above credentials
5. And run these scripts to keep table ready

```

create SCHEMA ecommerce_transactional_data;
CREATE TABLE ecommerce_transactional_data.dim_customers (
customer_id VARCHAR(20) ENCODE lzo,
first_name VARCHAR(255) ENCODE lzo,
last_name VARCHAR(255) ENCODE lzo,
email VARCHAR(100) ENCODE lzo,
membership_level VARCHAR(2) ENCODE lzo)
DISTSTYLE KEY
DISTKEY (membership_level)
SORTKEY (customer_id, first_name, last_name);

INSERT INTO ecommerce_transactional_data.dim_customers (customer_id, first_name, last_name, email, membership_level)
VALUES ('CUST00001', 'John', 'Doe', 'john.doe@example.com', 'BR'),
('CUST00002', 'Jane', 'Smith', 'jane.smith@example.com', 'SL'),
('CUST00003', 'Michael', 'Lee', 'michael.lee@example.com', 'GL'),
('CUST00004', 'Alice', 'Brown', 'alice.brown@example.com', 'BR'),
('CUST00005', 'David', 'Miller', 'david.miller@example.com', 'SL');

CREATE TABLE ecommerce_transactional_data.dim_products (
product_id VARCHAR(20) ENCODE lzo,
product_name VARCHAR(255) ENCODE lzo,
category VARCHAR(100) ENCODE lzo,
price DECIMAL(10,2) ENCODE delta,
supplier_id VARCHAR(20) ENCODE lzo)
DISTSTYLE KEY
  
```

7. These commands are made available on the Github repo.

Step3: Configuring Security Group for Access

We need to open the port on which the database service is running to allow inbound traffic. Our Redshift is running in a VPC, which has a security group. Within this security group, we need to allow inbound traffic on port 3306, the port on which the datawarehouse is running.

1. Under the Connectivity & security section of the database instance, click on the VPC link.
2. Click on the security group associated with the VPC.
3. Click Edit inbound rules.
4. Click Add rule.
5. For Type, select Redshift.
6. For Port range, enter 3306.

- For Source, enter 0.0.0.0/0 (to allow any IP).
- Click Save rules.

Step4: Setup the Data Lake - create an S3 Bucket for Raw Data and a lifecycle rule to archive week old data

Create an S3 bucket where the client can upload CSV data in Hive partitions (using the date as the partition key).

Amazon S3 > Buckets > ecommerce-transactional-data > transactions_ > year=2024/ > month=6/ > day=8/

day=8/ Copy S3 URI

Objects Properties

Objects (1) Info

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	transactions_2024-06-8.csv	csv	June 9, 2024, 13:13:51 (UTC+05:30)	3.8 KB	Standard

To create life cycle rules

- Select the s3 bucket
- Go to management tab
- Click on LifeCycle rules
- Give a name
- Choose a rule scope -> Apply to all objects in the bucket
- Check -> Move current versions of objects between storage classes option
- Choose storage class transitions -> Glacier Deep Archive
- Days -> 7
- Save

Amazon S3 > Buckets > ecommerce-transactional-data > Lifecycle configuration

Lifecycle configuration Info

To manage your objects so that they are stored cost effectively throughout their lifecycle, configure their lifecycle. A lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects. Lifecycle rules run once per day.

Lifecycle rules (1)

Use lifecycle rules to define actions you want Amazon S3 to take during an object's lifetime such as transitioning objects to another storage class, archiving them, or deleting them after a specified period of time. [Learn more](#)

Find lifecycle rules by name

<input type="checkbox"/>	Lifecycle rule name	Status	Scope	Current version a...	Noncurrent versi...	Expired object de...	Incomplete multi...
<input checked="" type="checkbox"/>	ArchiveCommerceWeekOldData	Enabled	Entire bucket	Transition to Glacier Deep	-	-	-

Step5: Create a database in AWS Glue for the data catalog.

- Select aws Glue
- Select Databases
- Add database
- Give it a name
- Create

AWS Glue > Databases

Databases (1)

Last updated (UTC) June 9, 2024 at 08:56:37 Edit Delete Add database

A database is a set of associated table definitions, organized into a logical group.

Filter databases

<input type="checkbox"/>	Name	Description	Location URI	Created on (UTC)
<input type="checkbox"/>	ecommerceTransactionaldata	-	-	June 8, 2024 at 11:29:19

Step6: Create a JDBC connection to the RedShift.

This connection will be utilized in our crawler to access the redshift tables.

To create the connection:

- Navigate to "Connections" under Data Catalog.
- Choose "Create connection."
- Select "Redshift"
- Choose the Redshift instance for the connection.
- Specify the database you want to crawl in the RDS.
- Input Redshift credentials.
- Provide a name and create the connection.

After creating the connection, it's essential to test it:

- Select the created connection.
- Under actions, click "Test connection."
- Grant permission to connect to Redshift by selecting the previous IAM role and adding the Redshift policy to it.
- Click "Confirm." If the connection is successful, you will receive confirmation.

Step6: Create crawlers

Since, we will be applying our business logic with 4 data pieces we need to create 4 data tables on the glue catalog.
Please refer to my previous project to check how to create crawlers on S3 and JDBC connections.

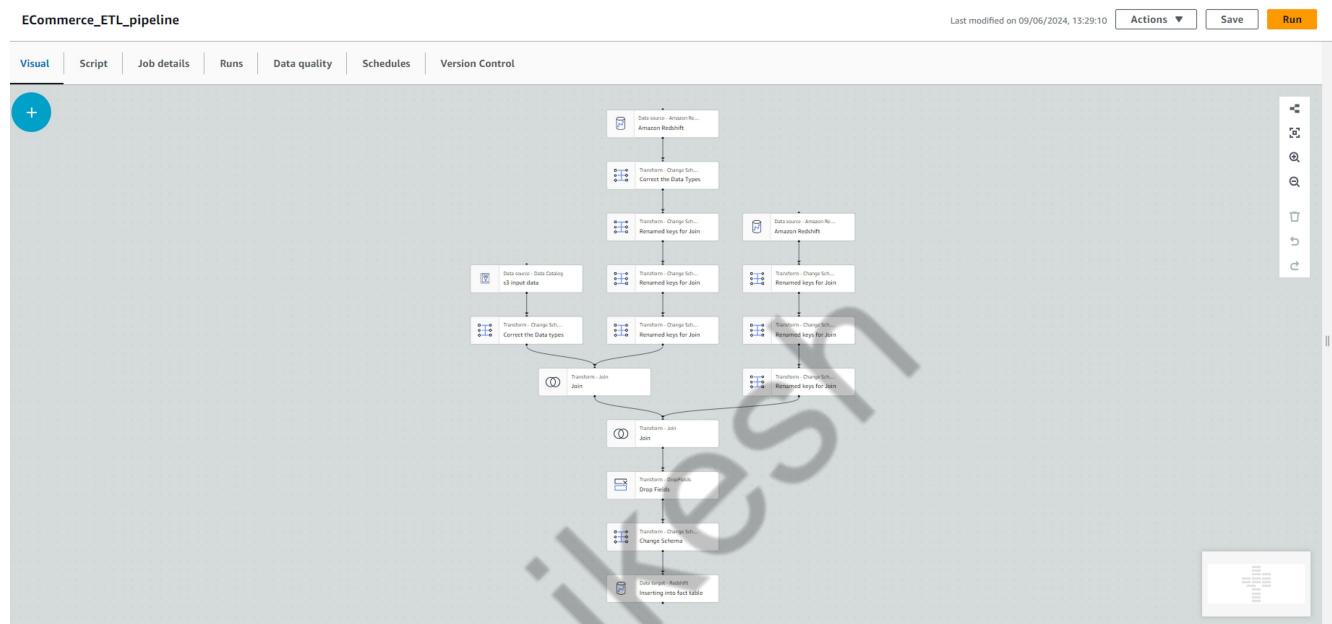
Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (4) Info		Last updated (UTC)	Action	Run	Create crawler
View and manage all available crawlers.					
<input type="text"/> Filter crawlers < 1 > 					
Name	State	Schedule	Last run	Last run time...	Log
InputTransactionData	Ready		Succeeded	June 8, 2024 at 1...	View log
redshift_ecommerce_transactional_customers	Ready		Succeeded	June 8, 2024 at 1...	View log
redshift_ecommerce_transactional_products	Ready		Succeeded	June 8, 2024 at 1...	View log
redshift_ecommerce_transactional_transactions	Ready		Succeeded	June 8, 2024 at 1...	View log

Step7: Create a ETL job in Glue

Check my previous project(Telecom) to find the step by step implementation on how to create a ETL Job.



Step8: Create a Lambda function to invoke the Job upon data arrival on s3

The screenshot shows the AWS Lambda console for creating a new Lambda function.
 1. **Function Overview**: The function is named "ecommerce-transactional-GlueETLJob-Start". It has no layers and is triggered by an S3 event. The last modified time is 3 hours ago. The Function ARN is arn:aws:lambda:ap-southern:ecommerce-transactional-GlueETLJob-Start and the Function URL is [Info](#).
 2. **Code Source**: The code source is named "lambda_function.py". The code is as follows:

```

import json
import boto3
def lambda_handler(event, context):
    glue_client = boto3.client('glue')
    for record in event['Records']:
        s3_event = record['eventName']
        if s3_event == "ObjectCreated:Put":
            try:
                # Start the AWS Glue job
                response = glue_client.start_job_run(
                    JobName="ECommerce_ETL_pipeline"
                )
                print(f"Started Glue job with ID: {response['JobRunId']}")
            except Exception as e:
                print(f"Error starting Glue Job: {e}")
                raise e
  
```

Step9: Create a SNS to update the stakeholder via Email

1. Select SNS service
2. Create a topic
3. Choose standard type
4. Give it a name
5. Click on create
6. Choose Email protocol and add subscriptions.

Amazon SNS > Topics > EcommerceTransactionalProcessingUpdate > Subscription: cb643a1a-3ecd-48ea-97c4-e7c312a0afc8

Subscription: cb643a1a-3ecd-48ea-97c4-e7c312a0afc8

Details	
ARN arn:aws:sns:ap-south-1:771822799155:User/Likesh	Status Confirmed
Endpoint likeshworkdesk@gmail.com	Protocol EMAIL
Topic EcommerceTransactionalProcessingUpdate	
Subscription Principal arn:aws:siam::771822799155:user/Likesh	

Step10: Create Event Bridge Rule to trigger SNS when Glue Job Run is Succeeded

Amazon EventBridge > Rules > EcommerceTransactionalDataProcessingUpdate

EcommerceTransactionalDataProcessingUpdate

Rule details		Info	
Rule name EcommerceTransactionalDataProcessingUpdate	Status Enabled	Event bus name default	Type Standard
Description	Rule ARN arn:aws:events:ap-south-1:771822799155:rule/EcommerceTransactionalDataProcessingUpdate	Event bus ARN arn:aws:events:ap-south-1:771822799155:rule/default	Event

Event pattern | Targets | Monitoring | Tags

Event pattern Info

```
1 {
  "source": ["aws:glue"],
  "detail-type": ["Glue Job Run Status"],
  "detail": {
    "jobName": ["Commerce_ETL_pipeline"],
    "state": ["Succeeded"]
  }
}
```

Execute the pipeline now,

Upload the csv in S3 and check the logs

- Lambda has invoked the Glue job

CloudWatch > Log groups > /aws/lambda/e-commerce-transactional-GlueETLJob-Start > 2024/06/09[\$LATEST]4c801b91fb524e5ba0f0f3cac7cc7f4b

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Timestamp	Message
No older events at this moment. Retry	
2024-06-09T07:43:50.932Z	INIT_START Runtime Version: python:3.12.v27 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:598ba9ea2c7b29b2fa9d54c92efdfb6c6fe22f81dcfa10c877191021672c
2024-06-09T07:43:51.234Z	START RequestId: 84fe9236-b966-4bba-b0df-1b16aac88cc4 Version: \$LATEST
2024-06-09T07:43:53.966Z	Started Glue job with ID: jr_3e43f4d1c535922b3c8b81a58c5a890bb9118fb22e75da7cd707446634431ds
2024-06-09T07:43:53.994Z	END RequestId: 84fe9236-b966-4bba-b0df-1b16aac88cc4
2024-06-09T07:43:53.985Z	REPORT RequestId: 84fe9236-b966-4bba-b0df-1b16aac88cc4 Duration: 2750.99 ms Billed Duration: 2751 ms Memory Size: 128 MB Max Memory Used: 83 MB Init Duration: 301.03 ms
No newer events at this moment. Auto resume paused. Resume	

- Glue Job has been successfully executed.

ECommerce_ETL_pipeline

Last modified on 09/06/2024, 15:29:10

Visual | Script | Job details | **Runs** | Data quality | Schedules | Version Control

Job runs (1/5) Info

Last updated (UTC) June 9, 2024 at 09:20:09 < 1 >

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
4. <input checked="" type="radio"/> Succeeded	0	06/09/2024 13:42:27	06/09/2024 13:44:05	1 m 25 s	10 DPUs	G.1X	4.0
<input type="radio"/> Succeeded	0	06/09/2024 13:32:22	06/09/2024 13:33:51	1 m 16 s	10 DPUs	G.1X	4.0
<input type="radio"/> Succeeded	0	06/09/2024 13:29:15	06/09/2024 13:31:01	1 m 32 s	10 DPUs	G.1X	4.0
<input type="radio"/> Succeeded	0	06/09/2024 13:13:53	06/09/2024 13:15:57	1 m 40 s	10 DPUs	G.1X	4.0
<input type="radio"/> Succeeded	0	06/09/2024 13:08:54	06/09/2024 13:10:53	1 m 45 s	10 DPUs	G.1X	4.0

- Transactions data is reflected on the warehouse.

Redshift query editor v2

Untitled 1

Run Limit 100 Explain Isolated session ecommerce-tr... dev

Filter resources

ecommerce-transaction-data

awsdatacatalog dev

Tables: dim_customers, dim_products, fact_transactions

Views: 0

Functions: 0

Stored procedures: 0

public sample_data_dev

Result 1 (100)

	transaction_id	customer_id	customer_email	product_id	product_name	quantity	price
1	TXN09543918	CUST00003	NULL	PROD00001	NULL	1	799.99
2	TXN18948791	CUST00001	NULL	PROD00001	NULL	4	3199.96
3	TXN32145631	CUST00005	NULL	PROD00004	NULL	2	39.96
4	TXN35633264	CUST00004	NULL	PROD00002	NULL	4	1999.96
5	TXN41208683	CUST00003	NULL	PROD00004	NULL	3	59.97
6	TXN45913877	CUST00003	NULL	PROD00001	NULL	1	799.99
7	TXN47038438	CUST00005	NULL	PROD00003	NULL	1	79.99
8	TXN52865917	CUST00002	NULL	PROD00004	NULL	1	19.99
9	TXN60379884	CUST00002	NULL	PROD00005	NULL	1	49.99
10	TXN63067618	CUST00001	NULL	PROD00003	NULL	4	319.96
11	TXN65433256	CUST00004	NULL	PROD00005	NULL	3	149.97
12	TXN65264695	CUST00002	NULL	PROD00003	NULL	4	319.96

Row 1, Col 1, Chr 65

Export Chart

Query ID 204852 Elapsed time: 5624 ms Total rows: 100