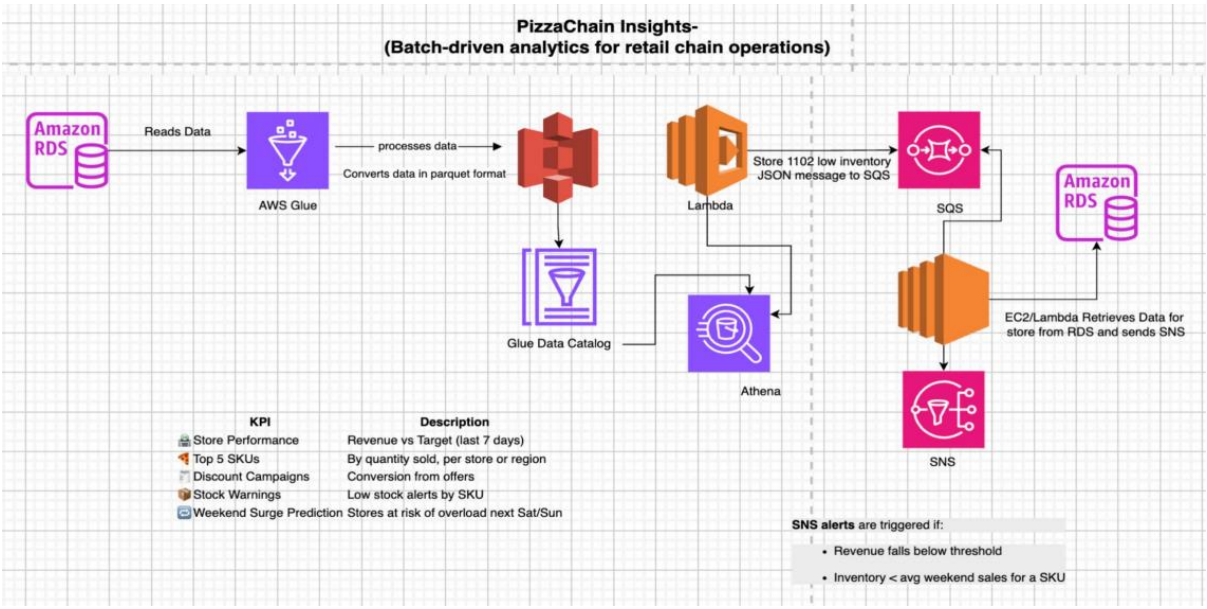
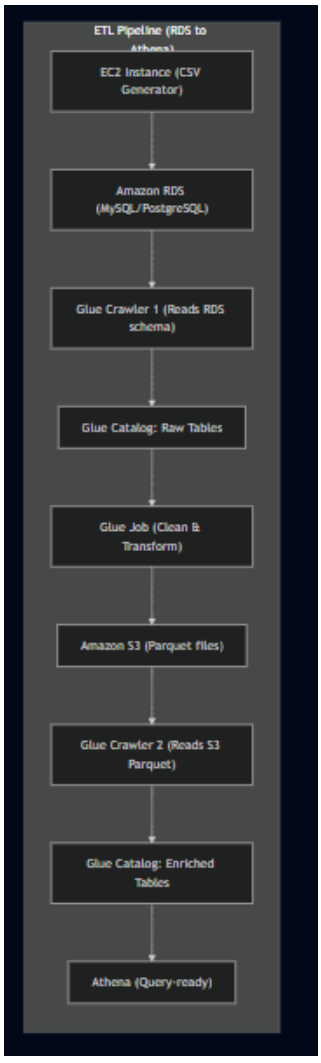
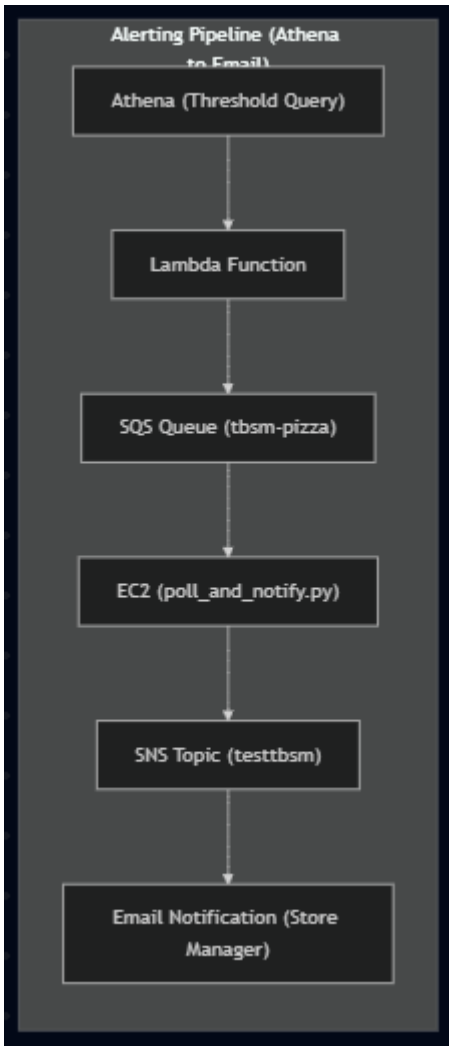
Pizza Chain Insights

Data Flow Architecture/Process Flow







Step 1: Set Up Amazon RDS (MySQL)

🔹 1.1 Launch RDS MySQL Instance

Go to AWS Console → RDS → Create database
Choose:

- Engine: MySQL
- Version: 8.x
- Template: Free Tier (if applicable)
- DB instance identifier: pizzachain-db
- Master username: admin
- Master password: yourpassword (tekavade123)
- DB Instance class: db.t3.micro
- Storage: 20 GB (General Purpose SSD)

🛡️ VPC Settings:

- Public access: Yes
- VPC Security Group: Allow inbound MySQL/Aurora (port 3306) from your IP

Click Create database and wait till status becomes Available

1.2 Connect to RDS from EC2

Once RDS is ready, go to Connectivity & security tab and note Endpoint.

Now from your EC2:

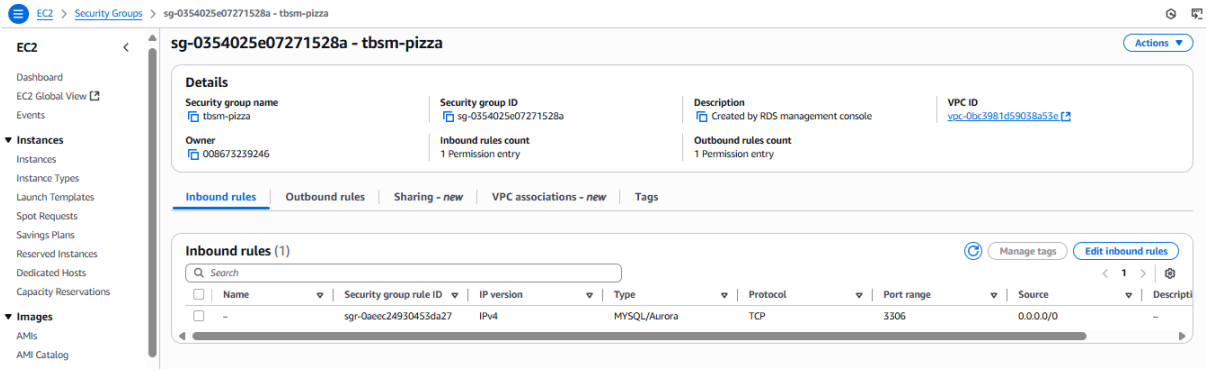
sudo dnf install mariadb105-server mariadb105 -y

mysql -h tbsm-pizza-db.cps8icam4jzk.ap-southeast-2.rds.amazonaws.com -u admin -p

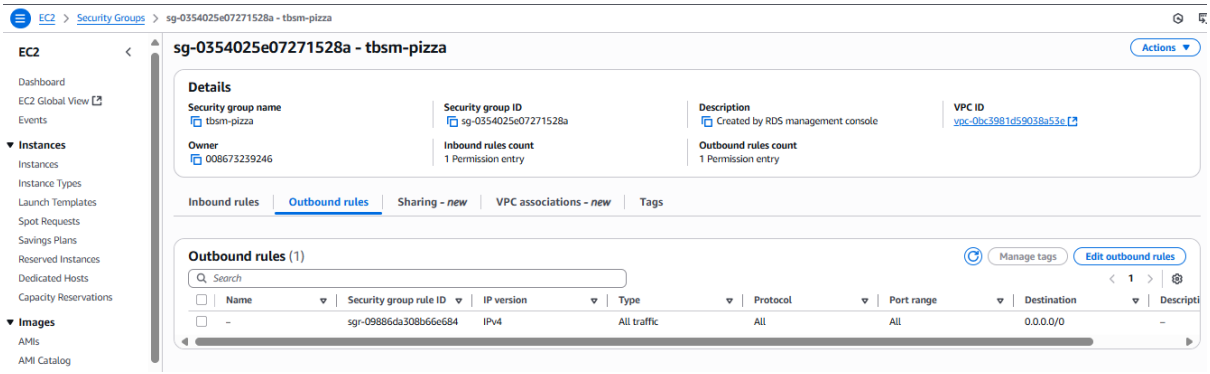
Enter password when prompted.

Potential Error:- check for VPC SG inbound and outbound

Inbound mysql to 0.0.0.0/0



Outbound all



1.3 Create Schema and Tables

```
CREATE DATABASE pizzachain;
USE pizzachain;
```

```
CREATE TABLE discounts_applied (
 discount_code VARCHAR(50),
 discount_amount DOUBLE
);
```

```
CREATE TABLE inventory_logs (
 log_time VARCHAR(50),
 store_id INT,
 sku_id VARCHAR(50),
 current_stock INT,
 restock_threshold INT
);
```

```
CREATE TABLE order_items (
 order_id VARCHAR(50),
 sku_id VARCHAR(50),
 quantity DOUBLE,
 unit_price DOUBLE,
```

```
  discount_code VARCHAR(50),
  discount_amount DOUBLE
);
```

```
CREATE TABLE orders (
  order_id VARCHAR(50),
  customer_id VARCHAR(75),
  store_id VARCHAR(10),
  order_time VARCHAR(50),
  total_amount DOUBLE
);
```

```
CREATE TABLE sku_master (
  sku_id VARCHAR(50),
  item_name VARCHAR(100),
  category VARCHAR(100),
  price DOUBLE,
  created_at TIMESTAMP
);
```

```
create table store_manager (
   store_id varchar(50),
   manager_name varchar(50),
   email varchar(50));
```

1.4 Upload CSV Files to RDS

Use the mysql client

Database changed

MySQL [pizzachain]> LOAD DATA LOCAL INFILE '/home/ec2-user/output/inventory_logs.csv'

-> INTO TABLE inventory_logs

-> FIELDS TERMINATED BY ','

-> ENCLOSED BY '"'

-> LINES TERMINATED BY '\n'

-> IGNORE 1 ROWS;

Query OK, 760 rows affected (0.028 sec)

Records: 760 Deleted: 0 Skipped: 0 Warnings: 0

```
MySQL [pizzachain]> LOAD DATA LOCAL INFILE '/home/ec2-
user/output/discounts_applied.csv'
   -> INTO TABLE discounts_applied
   -> FIELDS TERMINATED BY ','
   -> ENCLOSED BY '"'
   -> LINES TERMINATED BY '\n'
   -> IGNORE 1 ROWS;
Query OK, 3 rows affected (0.006 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0

MySQL [pizzachain]> LOAD DATA LOCAL INFILE '/home/ec2-
user/output/order_items.csv'
   -> INTO TABLE order_items
   -> FIELDS TERMINATED BY ','
   -> ENCLOSED BY '"'
   -> LINES TERMINATED BY '\n'
   -> IGNORE 1 ROWS;
Query OK, 3019 rows affected (0.082 sec)
Records: 3019  Deleted: 0  Skipped: 0  Warnings: 0

MySQL [pizzachain]> LOAD DATA LOCAL INFILE '/home/ec2-user/output/orders.csv'
   -> INTO TABLE orders
   -> FIELDS TERMINATED BY ','
   -> ENCLOSED BY '"'
   -> LINES TERMINATED BY '\n'
   -> IGNORE 1 ROWS;
Query OK, 1000 rows affected (0.032 sec)
Records: 1000  Deleted: 0  Skipped: 0  Warnings: 0
```

```
MySQL [pizzachain]> LOAD DATA LOCAL INFILE '/home/ec2-
user/output/sku_master.csv'
   -> INTO TABLE sku_master
   -> FIELDS TERMINATED BY ','
   -> ENCLOSED BY '"'
   -> LINES TERMINATED BY '\n'
   -> IGNORE 1 ROWS;
Query OK, 19 rows affected (0.008 sec)
Records: 19  Deleted: 0  Skipped: 0  Warnings: 0

MySQL [pizzachain]> INSERT INTO store_manager (store_id, manager_name, e
mail) VALUES
   -> ('STORE001', 'Alice Johnson', 'yashvardhan.tekavade@infocepts.com'),
   -> ('STORE002', 'Bob Smith', 'ambrish.solanki@infocepts.com');
Query OK, 2 rows affected (0.005 sec)
MySQL [pizzachain]> show tables;
+----------------------+
| Tables_in_pizzachain |
+----------------------+
| discounts_applied    |
| inventory_logs       |
| order_items          |
| orders               |
| sku_master           |
| store_manager        |
+----------------------+
6 rows in set (0.002 sec)
```

Step 2: Extract Data from RDS to S3 Using AWS Glue

We'll:

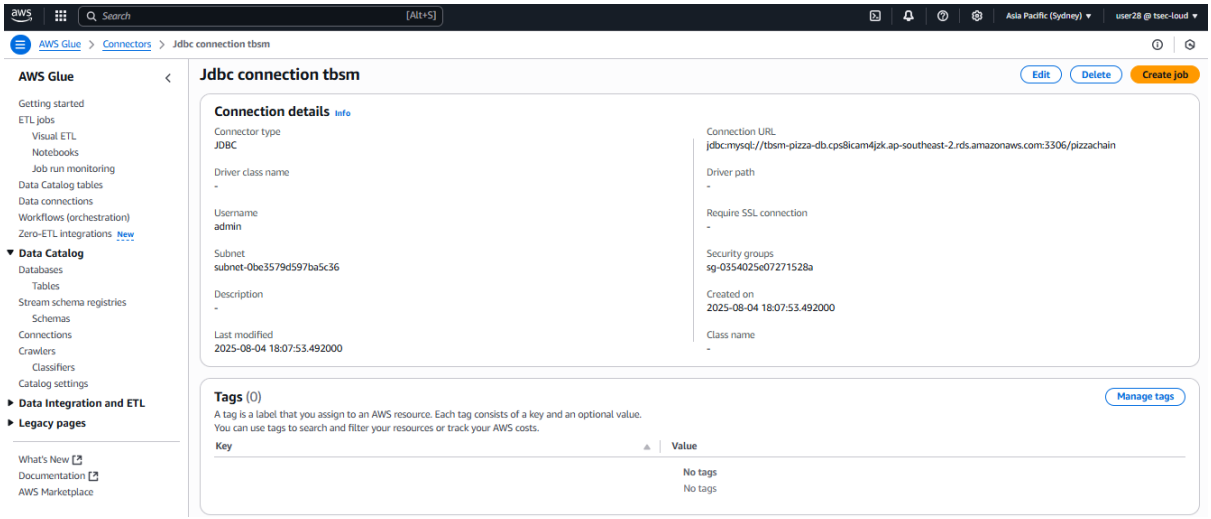
1. Create a **Connection** to RDS
2. Set up a **Glue Crawler** to catalog RDS tables
3. Create a **Glue Job** to transform and write data to S3

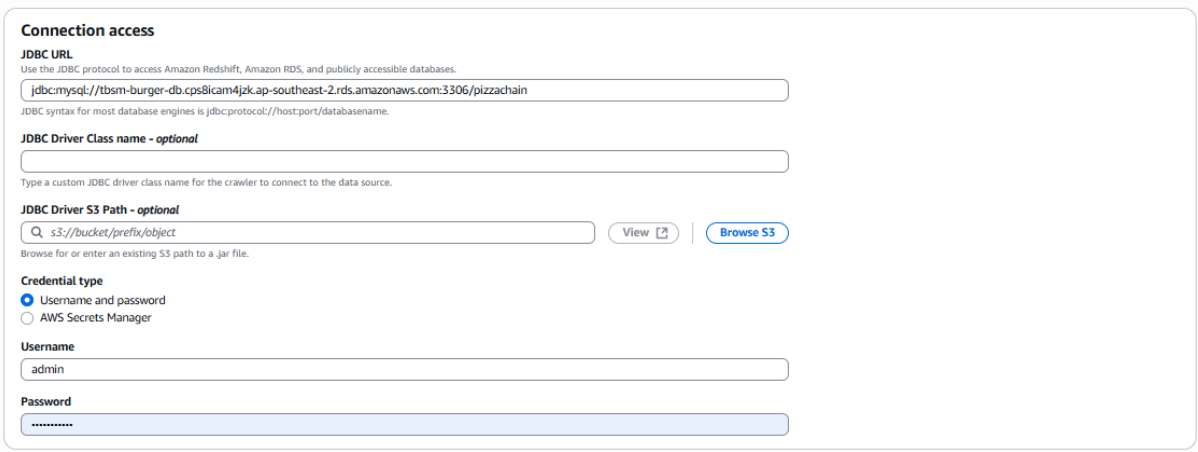
2.1 Create AWS Glue Connection to RDS

Go to **AWS Glue Console** → **Connections** → **Add Connection**

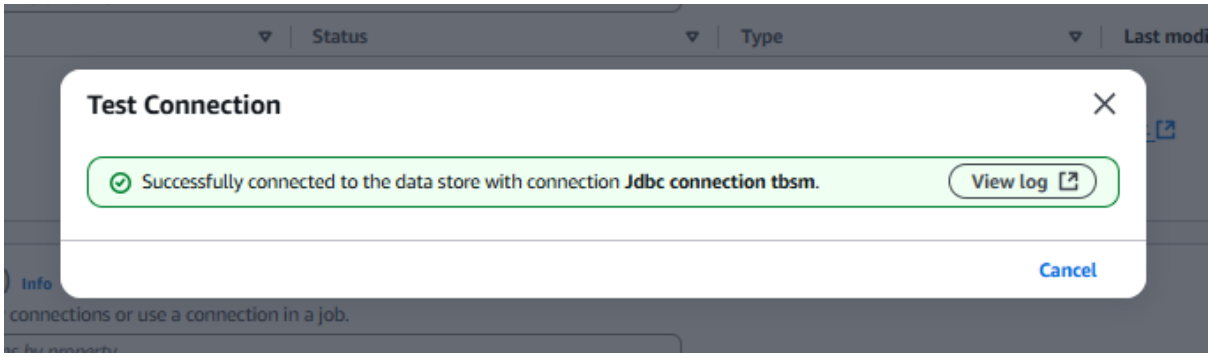
- **Name**: **Jdbc connection tbsm**
- **Connection type**: JDBC
- **JDBC URL**: jdbc:mysql://<your-rds-endpoint>:3306/pizzachain
- **Username**: admin
- **Password**: yourpassword
- **VPC**: Choose same VPC as RDS
- **Subnet**: Use subnet with NAT/public access
- **Security group**: Allow inbound 3306 from Glue

Click **Save connection**





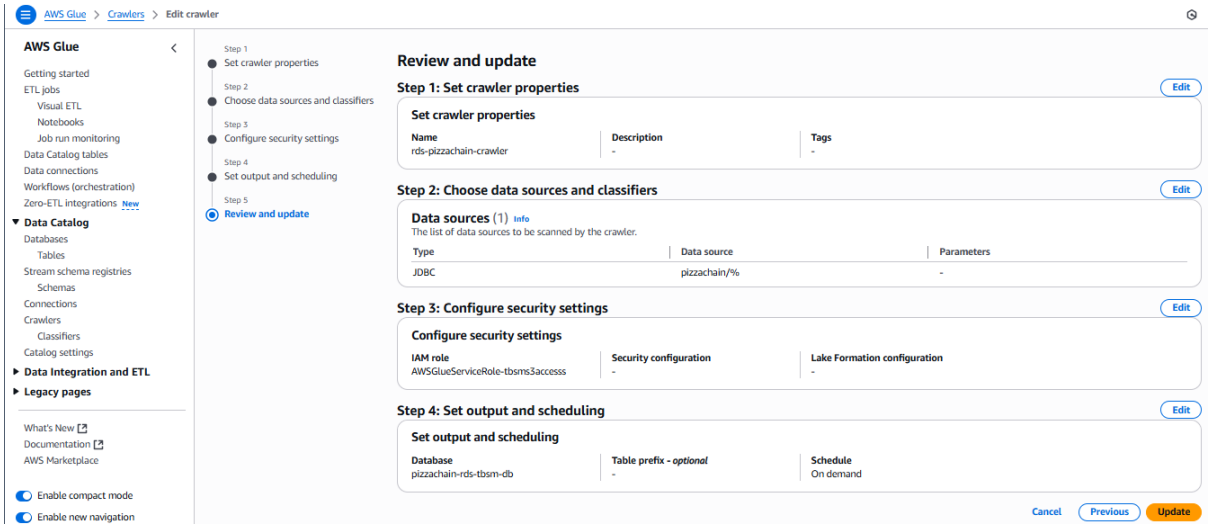
Add vpc and subnet acc to rds subnets with same sg as rds

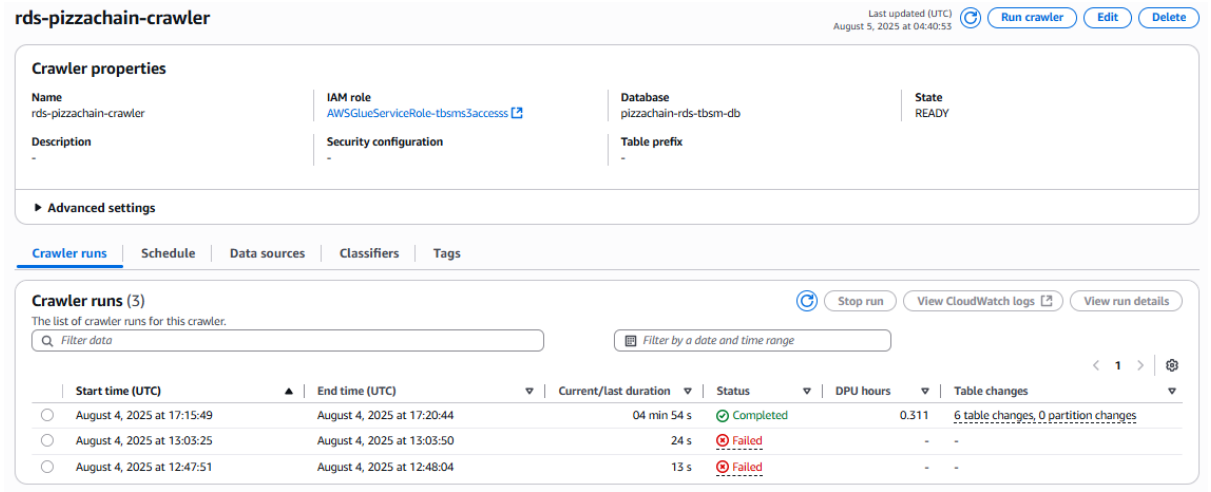


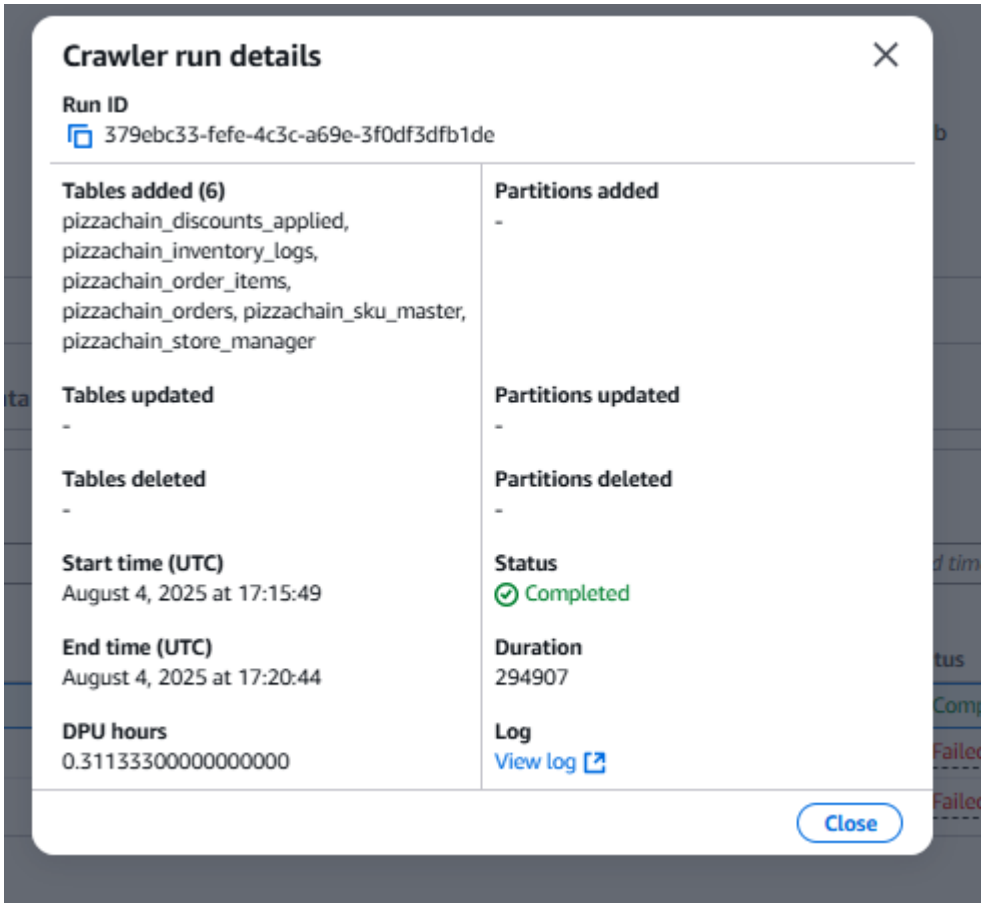
2.2 Create AWS Glue Crawler for RDS

Go to **AWS Glue → Crawlers → Add crawler**

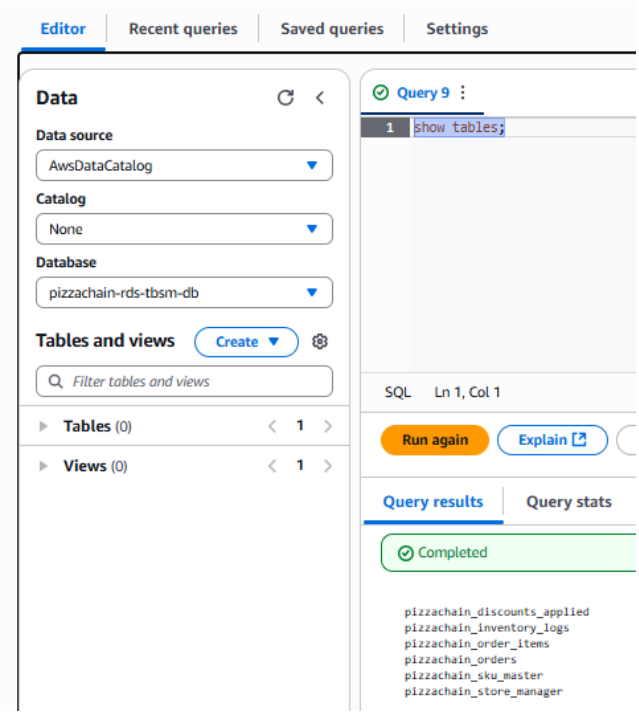
- **Name**: rds-pizzachain-crawler
- **Source**: JDBC
    - Choose the Jdbc connection tbsm
    - **Data source** :- Pizzachain/%
- **IAM Role**: Create or choose a role with Glue and RDS access: AWSGlueServiceRole-tbsms3accesss
- **Output**: Choose a database **:-** pizzachain_rds_tbsm_db
- Click **Run Crawler**
  After it finishes, confirm you see the tables in the Glue Data Catalog.


AWS Glue > Crawlers > Edit crawler
AWS Glue
Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
Data Integration and ETL
Legacy pages
What's New
Documentation
AWS Marketplace
Enable compact mode
Enable new navigation
Step 1 Set crawler properties
Step 2 Choose data sources and classifiers
Step 3 Configure security settings
Step 4 Set output and scheduling
Step 5 Review and update
Review and update
Step 1: Set crawler properties
Edit
Set crawler properties
Name
rds-pizzachain-crawler
Description
-
Tags
-
Step 2: Choose data sources and classifiers
Edit
Data sources (1) Info
The list of data sources to be scanned by the crawler.
Type
Data source
Parameters
JDBC
pizzachain/%
-
Step 3: Configure security settings
Edit
Configure security settings
IAM role
AWSGlueServiceRole-tbsms3accesss
Security configuration
-
Lake Formation configuration
-
Step 4: Set output and scheduling
Edit
Set output and scheduling
Database
pizzachain-rds-tbsm-db
Table prefix - optional
-
Schedule
On demand
Cancel
Previous
Update


rds-pizzachain-crawler
Last updated (UTC)
August 5, 2025 at 04:40:53
Run crawler
Edit
Delete
Crawler properties
Name
rds-pizzachain-crawler
IAM role
AWSGlueServiceRole-tbsms3accesss
Database
pizzachain-rds-tbsm-db
State
READY
Description
-
Security configuration
-
Table prefix
-
Advanced settings
Crawler runs
Schedule
Data sources
Classifiers
Tags
Crawler runs (3)
The list of crawler runs for this crawler.
Stop run
View CloudWatch logs
View run details
Filter data
Filter by a date and time range
1
Start time (UTC)
End time (UTC)
Current/last duration
Status
DPU hours
Table changes
August 4, 2025 at 17:15:49
August 4, 2025 at 17:20:44
04 min 54 s
Completed
0.311
6 table changes, 0 partition changes
August 4, 2025 at 13:03:25
August 4, 2025 at 13:03:50
24 s
Failed
-
-
August 4, 2025 at 12:47:51
August 4, 2025 at 12:48:04
13 s
Failed
-
-



Check tables in athena (data wont be shown as athena only works with S3 not jdbc so only connection is been showed.



Objective

Create and run an **AWS Glue Job** that:

- Reads data from **Glue Catalog tables**
- Applies the transformations
- Writes the enriched data to **S3** in Parquet format

Source Tables

Glue Catalog Database: pizzachain-rds-tbsm-db
Tables:

- pizzachain_orders
- pizzachain_order_items
- pizzachain_sku_master
- pizzachain_discounts_applied
- pizzachain_inventory_logs
- pizzachain_store_manager

S3 Output Bucket

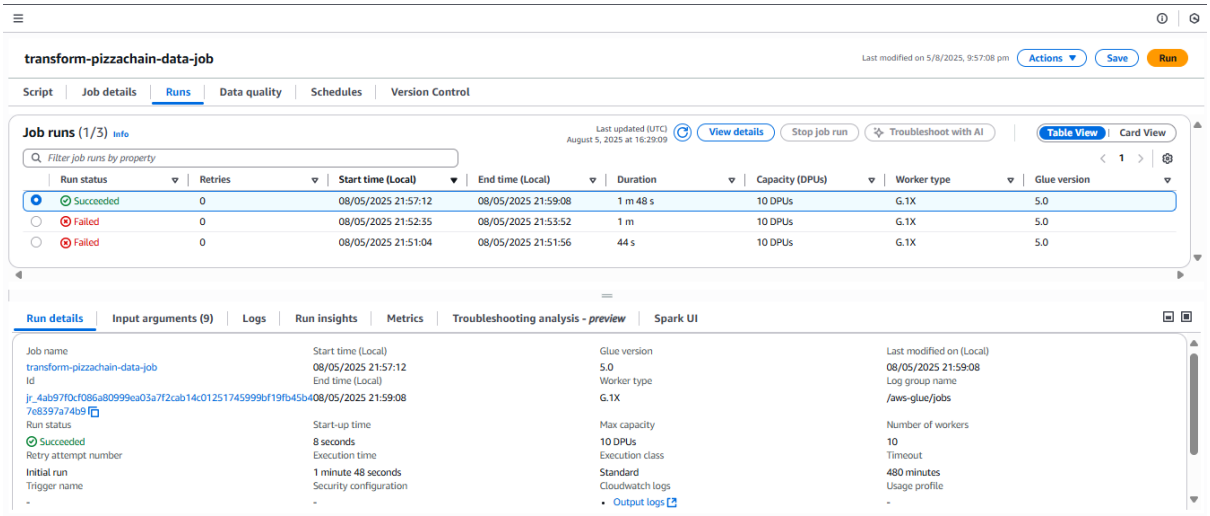
s3://pizzachain-curated-data-tbsm/

Step A: Go to AWS Glue Console → Jobs → Create job

Step B: Job Configuration

- **Job name**: transform-pizzachain-data-job
- **IAM Role**: Select or create role with:
    - S3 full access
    - Glue full access
    - (Optional) RDS access
- **Type**: Spark
- **Glue version**: 3.0 or above

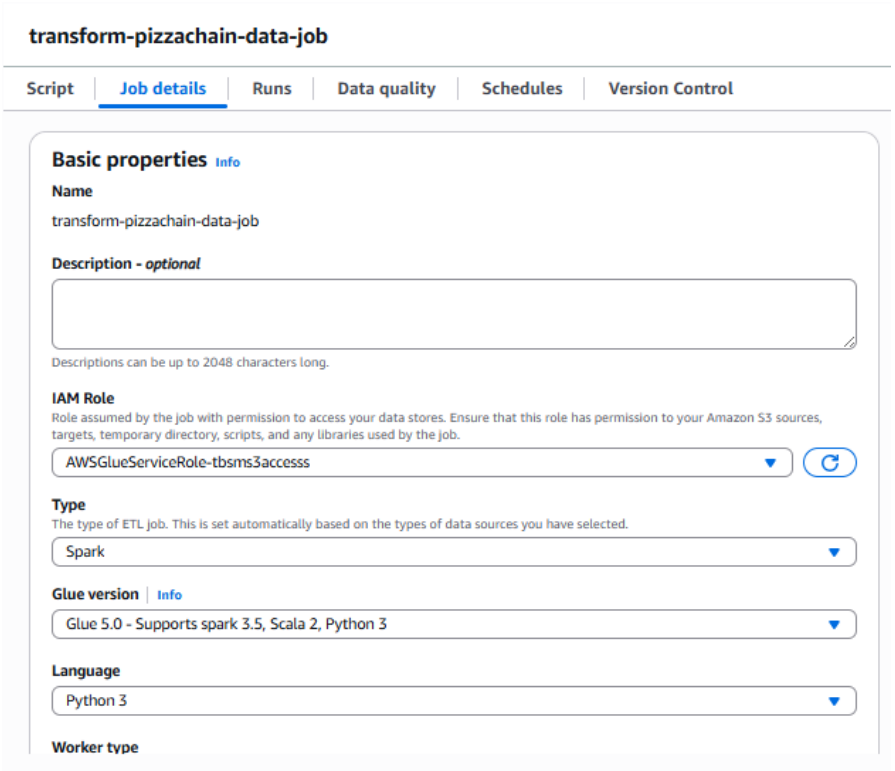
- **Language**: Python
- **Job Timeout**: 10–30 min
- **Worker type**: G.1X
- **Number of workers**: 2–5

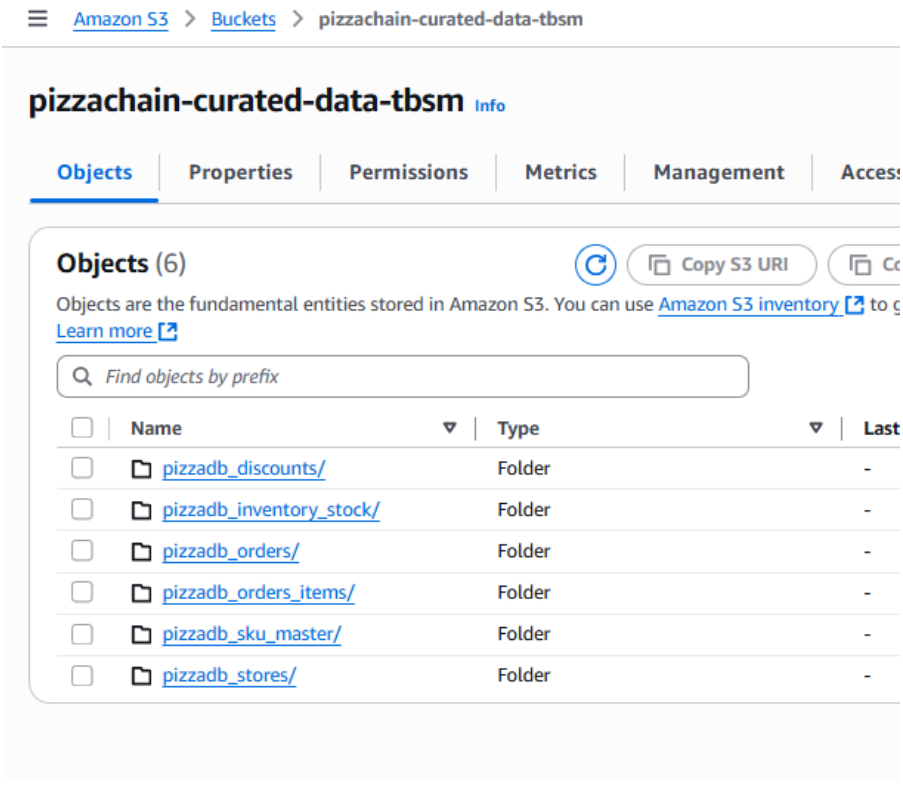



```
import sys
import boto3
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from awsglue.job import Job
from pyspark.sql.functions import *
from awsglue.dynamicframe import DynamicFrame

# Job setup
args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Temp paths
spark._jsc.hadoopConfiguration().set("spark.sql.warehouse.dir", "s3://tbsm-core/glue-temp/")
spark._jsc.hadoopConfiguration().set("hadoop.tmp.dir", "s3://tbsm-core/glue-temp/")

# Config
database_name = "pizzachain-rds-tbsm-db"
s3_output_base = "s3://pizzachain-curated-data-tbsm/"
```

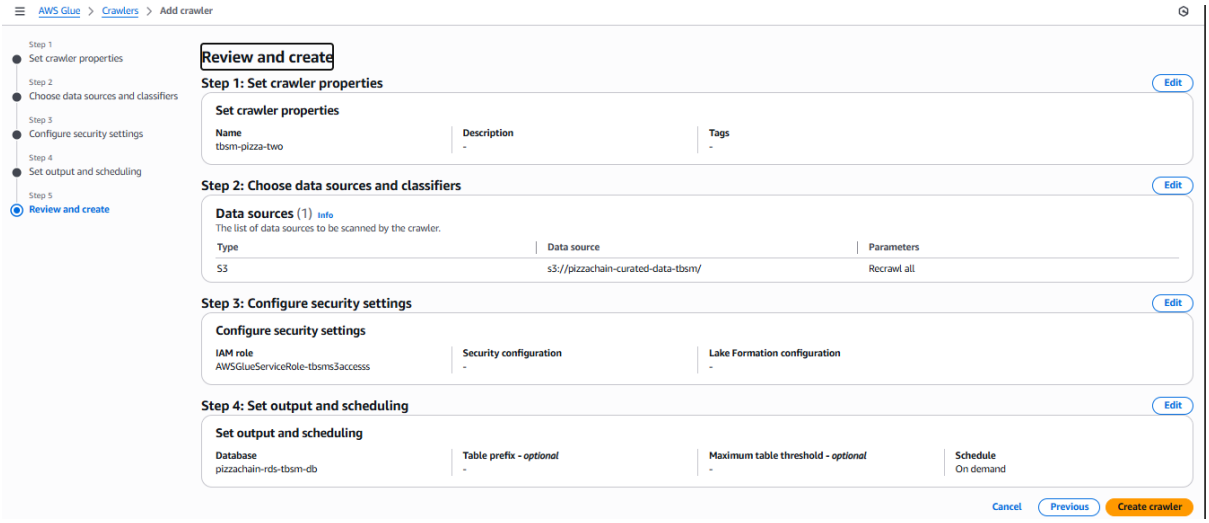
```
# -------- Step 1: sku_master --------
sku_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_sku_master"
).toDF()

sku_df = sku_df.filter("sku_id != ''") \
   .withColumn("item_name", trim(lower(col("item_name")))) \
   .withColumn("category", trim(lower(col("category")))) \
   .withColumn("price", col("price").cast("double"))

# -------- Step 2: discounts --------
discounts_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_discounts_applied"
).toDF()

discounts_df = discounts_df.filter("discount_code != ''") \
   .withColumn("discount_code", trim(col("discount_code"))) \
   .withColumn("line_discount_amount", col("discount_amount").cast("double")) \
   .drop("discount_amount")

# -------- Step 3: orders_items --------
order_items_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_order_items"
).toDF()

order_items_df = order_items_df.filter(
   "order_id != '' AND sku_id != '' AND discount_code != '' AND quantity IS NOT NULL
AND unit_price IS NOT NULL AND quantity != 0 AND unit_price != 0"
) \
   .withColumn("quantity", col("quantity").cast("int")) \
   .withColumn("unit_price", col("unit_price").cast("double")) \
   .withColumn("item_total", col("quantity") * col("unit_price")) \
   .join(sku_df, on="sku_id", how="inner") \
   .join(discounts_df.select("discount_code", "line_discount_amount"),
on="discount_code", how="inner")

# -------- Step 4: orders --------
orders_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_orders"
).toDF()

orders_df = orders_df.filter("order_id != ''")

order_totals = order_items_df.groupBy("order_id").agg(
   sum("item_total").alias("order_total"),
   sum("line_discount_amount").alias("total_discount_amount")
)
```

```
orders_df = orders_df.join(order_totals, on="order_id", how="left") \
   .withColumn("day_of_week", date_format(to_date("order_time"), "EEEE"))

# -------- Step 5: inventory_stock --------
inventory_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_inventory_logs"
).toDF()

inventory_df = inventory_df.filter(
   "sku_id != '' AND store_id IS NOT NULL AND current_stock IS NOT NULL AND
current_stock != 0"
) \
   .withColumnRenamed("current_stock", "stock_qty") \
   .withColumn("stock_qty", col("stock_qty").cast("int")) \
   .join(sku_df.select("sku_id", "price"), on="sku_id", how="inner") \
   .withColumn("stock_value", col("stock_qty") * col("price"))

# -------- Step 6: store --------
store_df = glueContext.create_dynamic_frame.from_catalog(
   database=database_name, table_name="pizzachain_store_manager"
).toDF()

# -------- All Writes at the End --------
sku_df.write.mode("overwrite").parquet(s3_output_base + "pizzadb_sku_master/")
discounts_df.write.mode("overwrite").parquet(s3_output_base +
"pizzadb_discounts/")
order_items_df.write.mode("overwrite").parquet(s3_output_base +
"pizzadb_orders_items/")
orders_df.write.mode("overwrite").parquet(s3_output_base + "pizzadb_orders/")
inventory_df.write.mode("overwrite").parquet(s3_output_base +
"pizzadb_inventory_stock/")
store_df.write.mode("overwrite").parquet(s3_output_base + "pizzadb_stores/")

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

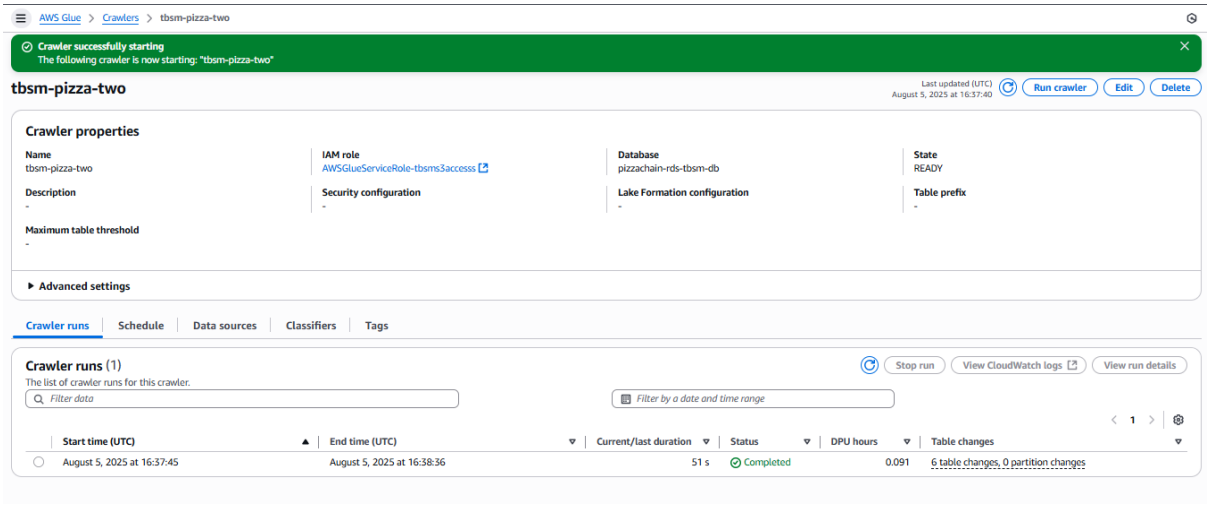

transform-pizzachain-data-job
Script
Job details
Runs
Data quality
Schedules
Version Control
Basic properties Info
Name
transform-pizzachain-data-job
Description - optional
Descriptions can be up to 2048 characters long.
IAM Role
Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.
AWSGlueServiceRole-tbsms3accesss
Type
The type of ETL job. This is set automatically based on the types of data sources you have selected.
Spark
Glue version Info
Glue 5.0 - Supports spark 3.5, Scala 2, Python 3
Language
Python 3
Worker type


Amazon S3 > Buckets > pizzachain-curated-data-tbsm
pizzachain-curated-data-tbsm Info
Objects
Properties
Permissions
Metrics
Management
Acces
Objects (6)
Copy S3 URI
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to g
Learn more
Find objects by prefix
Name
Type
Last
pizzadb_discounts/ Folder -
pizzadb_inventory_stock/ Folder -
pizzadb_orders/ Folder -
pizzadb_orders_items/ Folder -
pizzadb_sku_master/ Folder -
pizzadb_stores/ Folder -

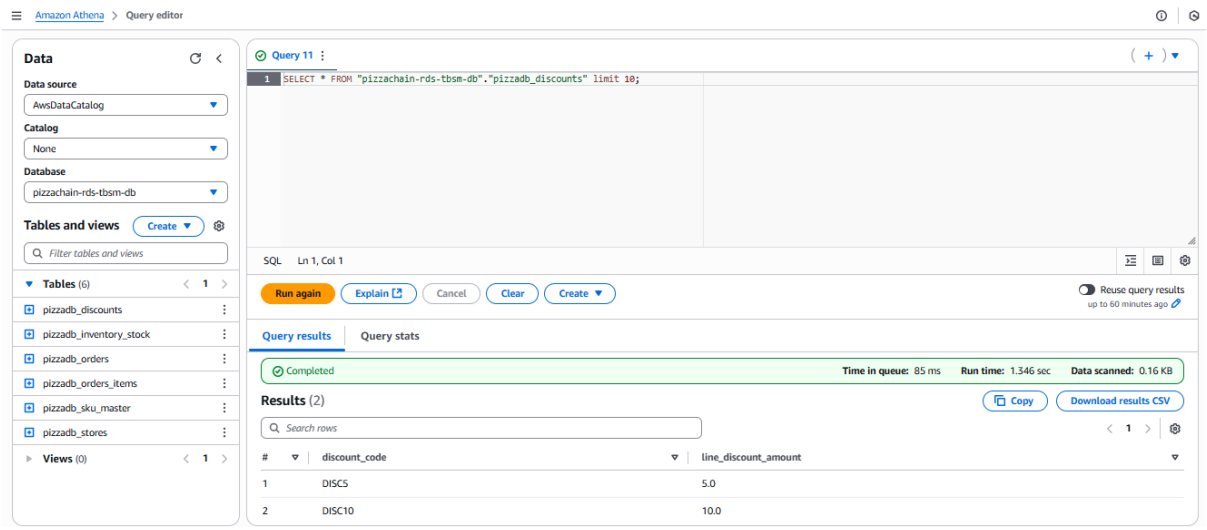
Run crawler again on created parquete files in s3



And crawler again



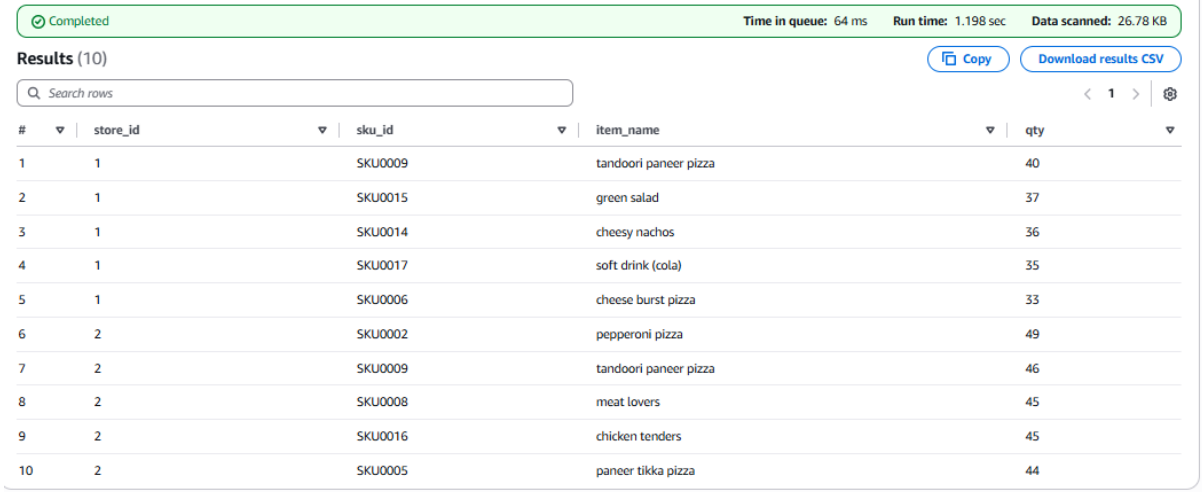
Check Athena query if that table are there with the data



Queries to run.

Top 5 Selling SKUs per Store in the Last 7 Days

```
WITH recent_orders AS (
  SELECT order_id, store_id
  FROM pizzadb_orders
  WHERE CAST(order_time AS TIMESTAMP) >= current_date - INTERVAL '7' day
),
sku_sales AS (
  SELECT r.store_id, oi.sku_id, SUM(oi.quantity) AS qty
  FROM pizzadb_orders_items oi
  JOIN recent_orders r ON oi.order_id = r.order_id
  GROUP BY r.store_id, oi.sku_id
)
SELECT s.store_id, s.sku_id, m.item_name, s.qty
FROM (
  SELECT *, ROW_NUMBER() OVER (PARTITION BY store_id ORDER BY qty DESC) AS
rnk
  FROM sku_sales
) s
JOIN pizzadb_sku_master m ON s.sku_id = m.sku_id
WHERE rnk <= 5
ORDER BY s.store_id, rnk;
```



Completed | Time in queue: 64 ms | Run time: 1.198 sec | Data scanned: 26.78 KB

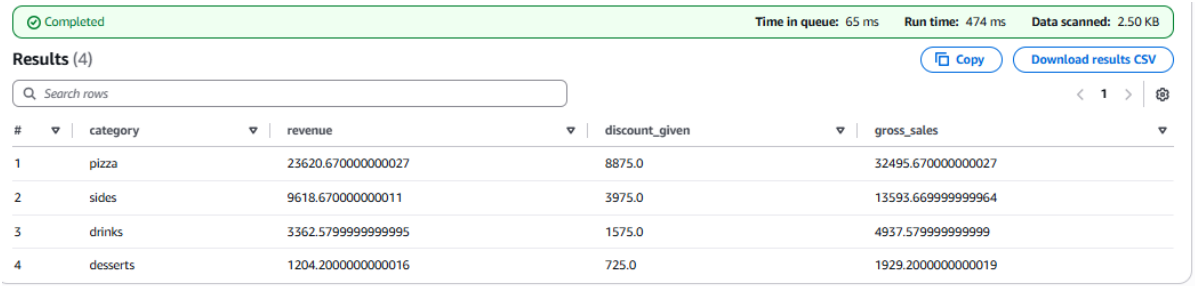
Results (10) | Copy | Download results CSV

Search rows | 1

#	store_id	sku_id	item_name	qty
1	1	SKU0009	tandoori paneer pizza	40
2	1	SKU0015	green salad	37
3	1	SKU0014	cheesy nachos	36
4	1	SKU0017	soft drink (cola)	35
5	1	SKU0006	cheese burst pizza	33
6	2	SKU0002	pepperoni pizza	49
7	2	SKU0009	tandoori paneer pizza	46
8	2	SKU0008	meat lovers	45
9	2	SKU0016	chicken tenders	45
10	2	SKU0005	paneer tikka pizza	44

Category-wise Revenue Breakdown with Discounts Applied

```
SELECT
  category,
  SUM(item_total) AS revenue,
  SUM(discount_amount) AS discount_given,
  SUM(item_total + discount_amount) AS gross_sales
FROM pizzadb_orders_items
GROUP BY category
ORDER BY revenue DESC;
```



Completed | Time in queue: 65 ms | Run time: 474 ms | Data scanned: 2.50 KB

Results (4)

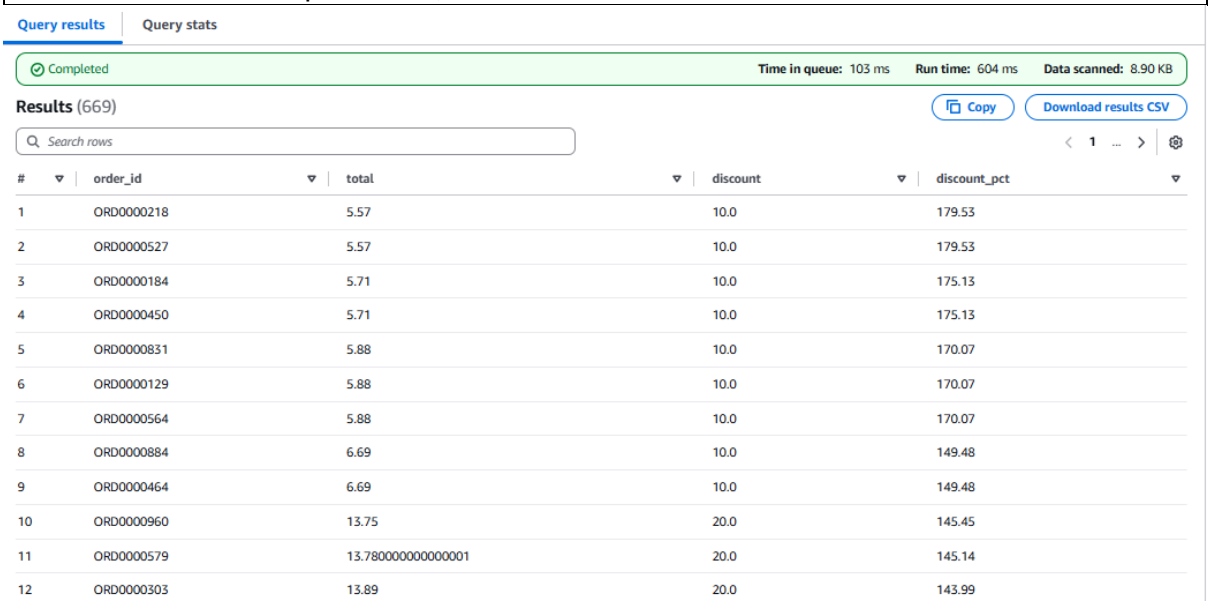
Copy | Download results CSV

Search rows

#	category	revenue	discount_given	gross_sales
1	pizza	23620.670000000027	8875.0	32495.670000000027
2	sides	9618.670000000011	3975.0	13593.669999999964
3	drinks	3362.5799999999995	1575.0	4937.579999999999
4	desserts	1204.2000000000016	725.0	1929.2000000000019

Identify Orders Where Discount > 30% of Total Value

```
SELECT *
FROM (
 SELECT
  order_id,
  SUM(quantity * unit_price) AS total,
  SUM(discount_amount) AS discount,
  ROUND(SUM(discount_amount) * 100 / SUM(quantity * unit_price), 2) AS
discount_pct
 FROM pizzadb_orders_items
 GROUP BY order_id
) t
WHERE discount_pct > 30
ORDER BY discount_pct DESC;
```



Query results | Query stats

Completed | Time in queue: 103 ms | Run time: 604 ms | Data scanned: 8.90 KB

Results (669)

Copy | Download results CSV

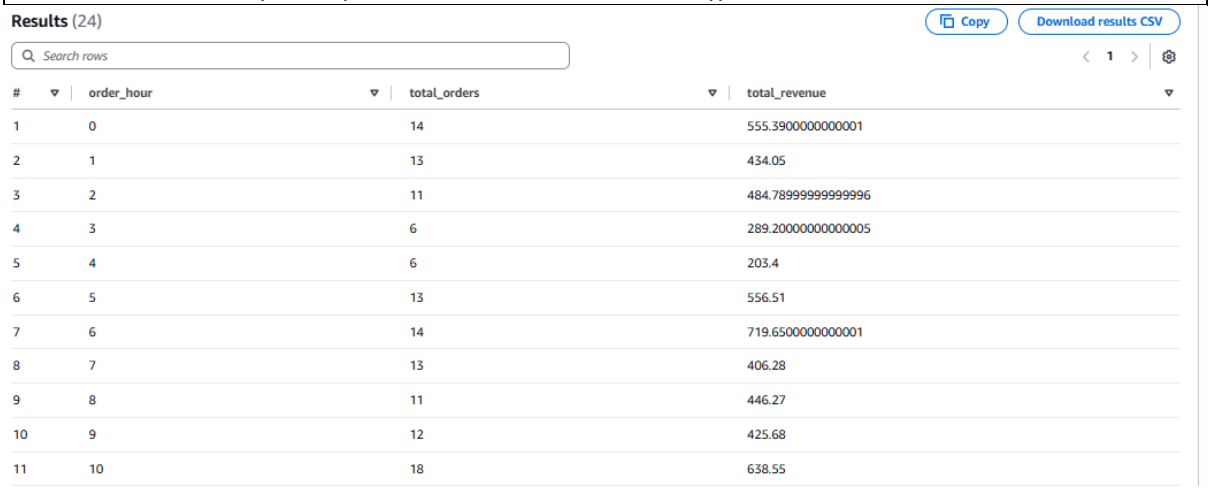
Search rows

#	order_id	total	discount	discount_pct
1	ORD0000218	5.57	10.0	179.53
2	ORD0000527	5.57	10.0	179.53
3	ORD0000184	5.71	10.0	175.13
4	ORD0000450	5.71	10.0	175.13
5	ORD0000831	5.88	10.0	170.07
6	ORD0000129	5.88	10.0	170.07
7	ORD0000564	5.88	10.0	170.07
8	ORD0000884	6.69	10.0	149.48
9	ORD0000464	6.69	10.0	149.48
10	ORD0000960	13.75	20.0	145.45
11	ORD0000579	13.780000000000001	20.0	145.14
12	ORD0000303	13.89	20.0	143.99

Revenue and Orders by Hour of Day

```
SELECT
 HOUR(CAST(order_time AS TIMESTAMP)) AS order_hour,
 COUNT(DISTINCT order_id) AS total_orders,
 SUM(total_amount) AS total_revenue
```

```
FROM pizzadb_orders
WHERE CAST(order_time AS TIMESTAMP) >= current_date - INTERVAL '7' day
GROUP BY HOUR(CAST(order_time AS TIMESTAMP))
ORDER BY HOUR(CAST(order_time AS TIMESTAMP));
```



Results (24)

Copy | Download results CSV

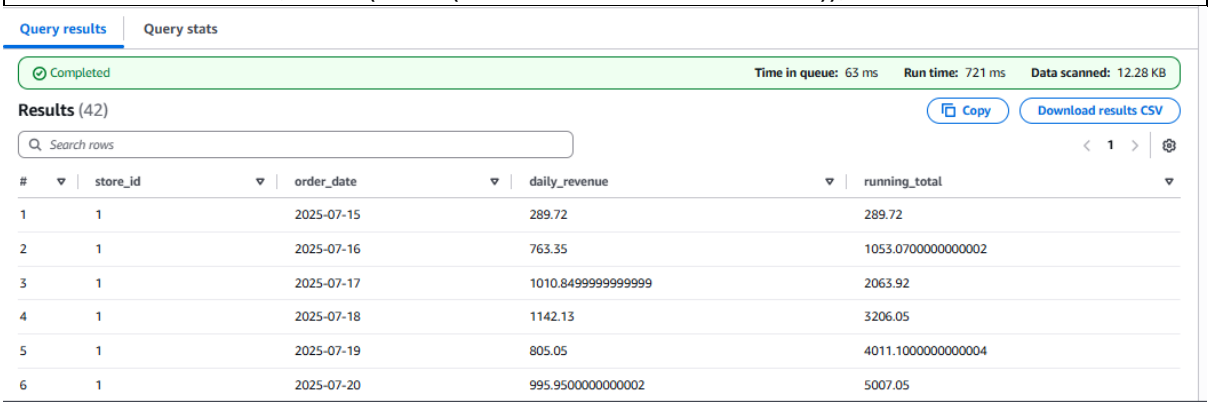
Search rows

1

#	order_hour	total_orders	total_revenue
1	0	14	555.3900000000001
2	1	13	434.05
3	2	11	484.78999999999996
4	3	6	289.20000000000005
5	4	6	203.4
6	5	13	556.51
7	6	14	719.6500000000001
8	7	13	406.28
9	8	11	446.27
10	9	12	425.68
11	10	18	638.55

Running Total of Revenue by Store

```
SELECT
  store_id,
  DATE(CAST(order_time AS TIMESTAMP)) AS order_date,
  SUM(total_amount) AS daily_revenue,
  SUM(SUM(total_amount)) OVER (
    PARTITION BY store_id
    ORDER BY DATE(CAST(order_time AS TIMESTAMP))
  ) AS running_total
FROM pizzadb_orders
GROUP BY store_id, DATE(CAST(order_time AS TIMESTAMP))
ORDER BY store_id, DATE(CAST(order_time AS TIMESTAMP));
```



Query results | Query stats

Completed — Time in queue: 63 ms — Run time: 721 ms — Data scanned: 12.28 KB

Results (42)

Copy | Download results CSV

Search rows

1

#	store_id	order_date	daily_revenue	running_total
1	1	2025-07-15	289.72	289.72
2	1	2025-07-16	763.35	1053.0700000000002
3	1	2025-07-17	1010.8499999999999	2063.92
4	1	2025-07-18	1142.13	3206.05
5	1	2025-07-19	805.05	4011.1000000000004
6	1	2025-07-20	995.9500000000002	5007.05

Customers with High Frequency and High Value Orders

```
SELECT customer_id,
  COUNT(order_id) AS total_orders,
  SUM(total_amount) AS total_spent
FROM pizzadb_orders
WHERE CAST(order_time AS TIMESTAMP) >= current_date - INTERVAL '30' day
```

```
GROUP BY customer_id
HAVING COUNT(order_id) >= 5 AND SUM(total_amount) >= 500
ORDER BY total_spent DESC;
```

Results (28)

Copy | Download results CSV

Search rows

1

#	customer_id	total_orders	total_spent
1	93	16	734.0100000000001
2	79	13	718.34
3	21	17	705.01
4	49	14	691.81
5	95	12	678.13
6	13	14	664.5899999999999
7	89	13	654.46

Most Discounted Products by Total Discount Given

```
SELECT sku_id, item_name, SUM(discount_amount) AS total_discount
FROM pizzadb_orders_items
GROUP BY sku_id, item_name
ORDER BY total_discount DESC
LIMIT 10;
```

Results (10)

Copy | Download results CSV

Search rows

1

#	sku_id	item_name	total_discount
1	SKU0013	garlic breadsticks	920.0
2	SKU0002	pepperoni pizza	880.0
3	SKU0008	meat lovers	860.0
4	SKU0004	bbq chicken pizza	860.0
5	SKU0006	cheese burst pizza	850.0
6	SKU0014	cheesy nachos	840.0
7	SKU0003	veggie supreme	830.0
8	SKU0001	margherita pizza	825.0

Lambda Setup (Athena → SQS)

Step:

- Go to **AWS Lambda**
- Click **Create function**
    - Runtime: **Python 3.9 or 3.12**
- Paste the code
- Add **IAM Role** with:
    - Athena Full Access
    - S3 Read/Write
    - SQS SendMessage

Step:

- Create a **test event** to trigger it manually
- Output:
    - Athena query runs
    - Parsed result sent to SQS

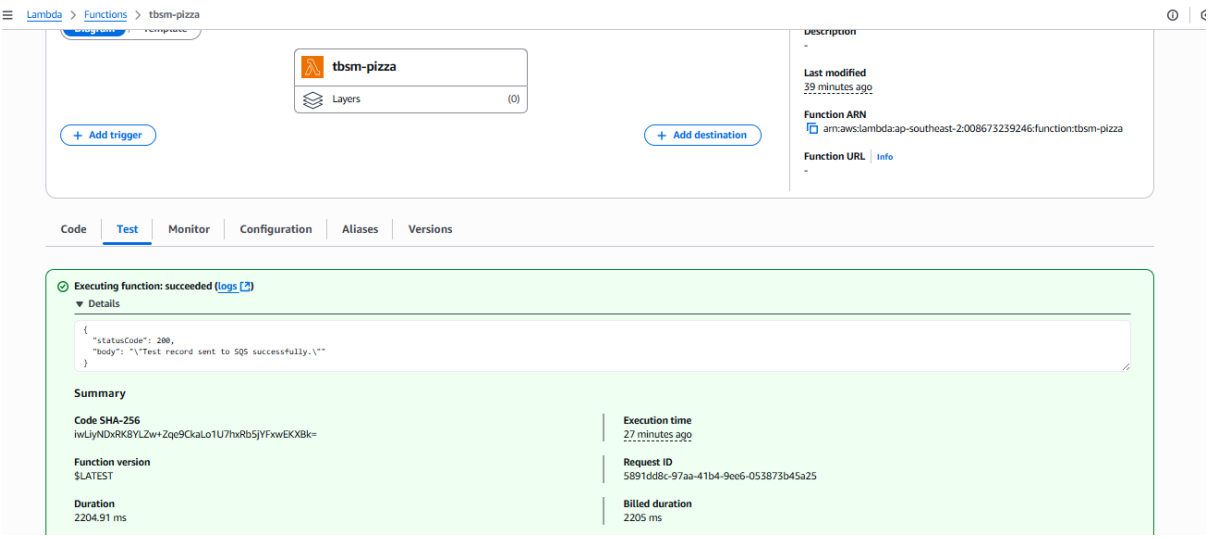
Screenshot:

- Lambda logs
- SQS message count increase

```
import boto3
import time
import json

ATHENA_DB = 'pizzachain-rds-tbsm-db'
ATHENA_OUTPUT = 's3://tbsm-core/output/'
SQS_QUEUE_URL = 'https://sqs.ap-southeast-
2.amazonaws.com/008673239246/tbsm-pizza'

athena = boto3.client('athena')
sqs = boto3.client('sqs')

def run_athena_query(query):
  response = athena.start_query_execution(
    QueryString=query,
    QueryExecutionContext={'Database': ATHENA_DB},
```

```
    ResultConfiguration={'OutputLocation': ATHENA_OUTPUT}
  )
  query_execution_id = response['QueryExecutionId']

  while True:
    result = athena.get_query_execution(QueryExecutionId=query_execution_id)
    state = result['QueryExecution']['Status']['State']

    if state in ['SUCCEEDED', 'FAILED', 'CANCELLED']:
      break
    time.sleep(2)

  if state != 'SUCCEEDED':
    reason = result['QueryExecution']['Status'].get('StateChangeReason', 'Unknown
error')
    raise Exception(f"Athena query failed: {state} - {reason}")

  return query_execution_id

def parse_results(query_execution_id):
  result = athena.get_query_results(QueryExecutionId=query_execution_id)
  rows = result['ResultSet']['Rows']
  headers = [col['VarCharValue'] for col in rows[0]['Data']]
  data = []

  for row in rows[1:]:
    values = [col.get('VarCharValue', '') for col in row['Data']]
    data.append(dict(zip(headers, values)))

  return data

def send_to_sqs(message):
  response = sqs.send_message(
    QueueUrl=SQS_QUEUE_URL,
    MessageBody=json.dumps(message)
  )
  return response

def lambda_handler(event, context):
  # Simple query to fetch any one record from pizzadb_stores
  test_query = """
  SELECT store_id, email
  FROM pizzadb_stores
  LIMIT 1;
  """

  # Run the query
```

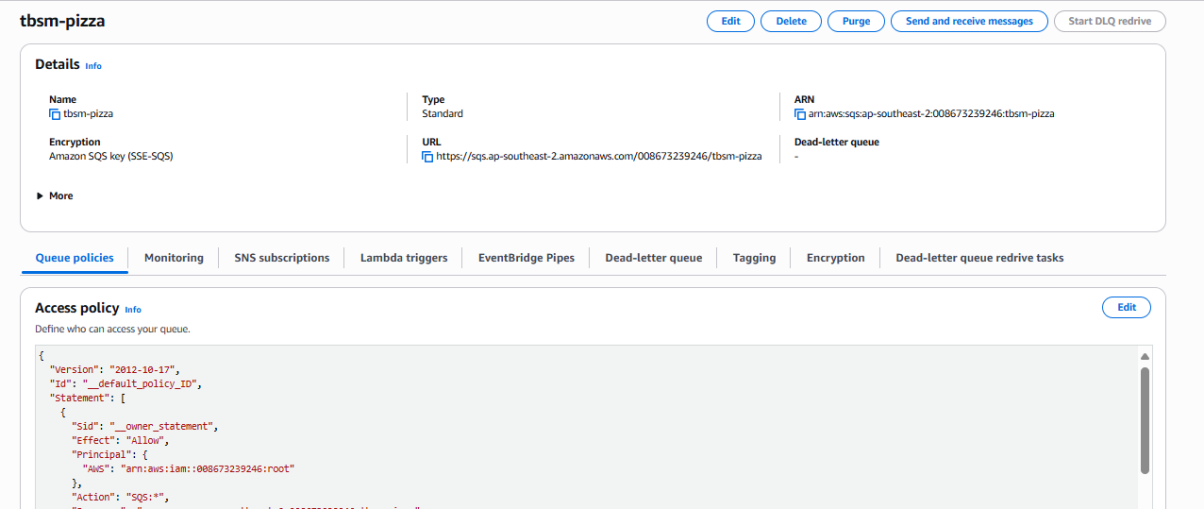
```
query_execution_id = run_athena_query(test_query)
print("Athena QueryExecutionId:", query_execution_id)

# Parse results
result_data = parse_results(query_execution_id)
print("Parsed Athena Results:", result_data)

if not result_data:
    print("No records found in pizzadb_stores.")
    return {
        'statusCode': 200,
        'body': json.dumps('No records found.')
    }

# Send the one record to SQS
item = result_data[0]
message = {
    'store_id': item.get('store_id'),
    'email': item.get('email'),
    'note': 'Test message from Lambda'
}

print("Sending to SQS:", message)
response = send_to_sqs(message)
print("SQS Send Response:", response)

return {
    'statusCode': 200,
    'body': json.dumps('Test record sent to SQS successfully.')
}
```



Amazon SQS Setup

Step:

- Go to **Amazon SQS Console**
- Click **Create Queue**
    - Type: **Standard**
    - Name: tbsm-pizza



Amazon SNS Setup

Step:

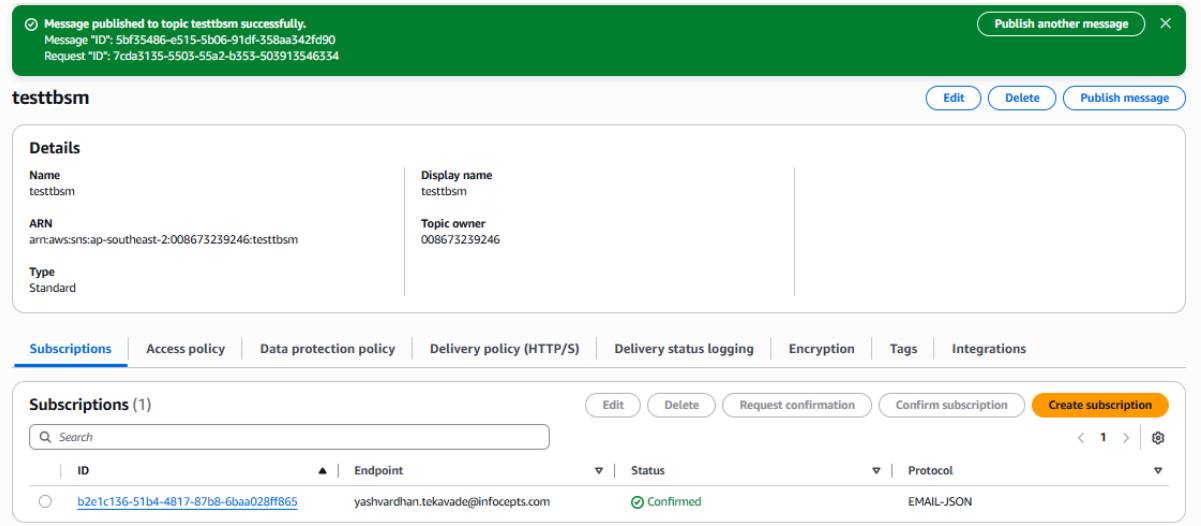
- Go to **SNS Console → Topics**
- Click **Create topic**
    - Type: Standard
    - Name: testtbsm
- Copy the **Topic ARN**

Step:

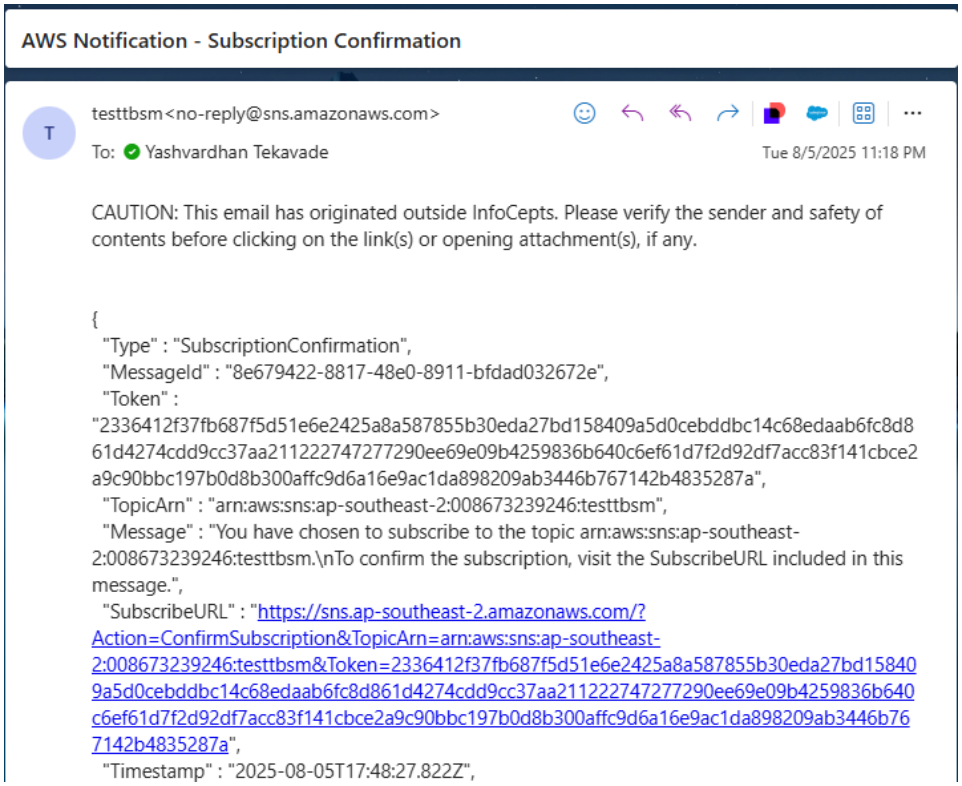
- **Create subscription**
    - Protocol: Email
    - Endpoint: your email (e.g., user@example.com)
- Click **Create subscription**
- Go to your email and **confirm the subscription**

Screenshot:

- SNS topic + Subscriptions list
- Screenshot of confirmation email ✅



Subscription confirmation



EC2 Setup (SQS Poller → SNS Publisher)

Step:

- Launch EC2 (Amazon Linux 2 or Ubuntu)
- Install Python + Boto3

sudo yum update -y

sudo yum install python3 -y

pip3 install boto3

Step:

- Copy your **EC2 Python code** to a .py file (e.g., poll_and_notify.py)
- Update SQS_QUEUE_URL and SNS_TOPIC_ARN with your values

Step:

- Run the script:

python3 poll_and_notify.py

IAM:

- Ensure EC2 has an IAM Role attached with:
    - SQS Read/Delete
    - SNS Publish

```
import boto3
import json
import time

# AWS Resources

SQS_QUEUE_URL = 'https://sqs.ap-southeast-
2.amazonaws.com/008673239246/tbsm-pizza'
SNS_TOPIC_ARN = 'arn:aws:sns:ap-southeast-2:008673239246:testtbsm'

# Clients

sqs = boto3.client('sqs', region_name='ap-southeast-2')
sns = boto3.client('sns', region_name='ap-southeast-2')
def poll_and_forward():
while True:
        print("Polling messages from SQS...")
```

```
response = sqs.receive_message(
QueueUrl=SQS_QUEUE_URL,
MaxNumberOfMessages=10,
WaitTimeSeconds=10,  # Long polling
)
messages = response.get('Messages', [])
if not messages:
print("No messages in queue. Waiting...")
time.sleep(5)
continue
for message in messages:
try:
        body = json.loads(message['Body'])
        print("Received message:", body)
        # Send to SNS
        sns_response = sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Subject='SQS to SNS Alert',
        Message=json.dumps(body, indent=2)
        )
        print("Published to SNS. MessageId:", sns_response['MessageId'])
        # Delete message from SQS
        sqs.delete_message(
        QueueUrl=SQS_QUEUE_URL,
```

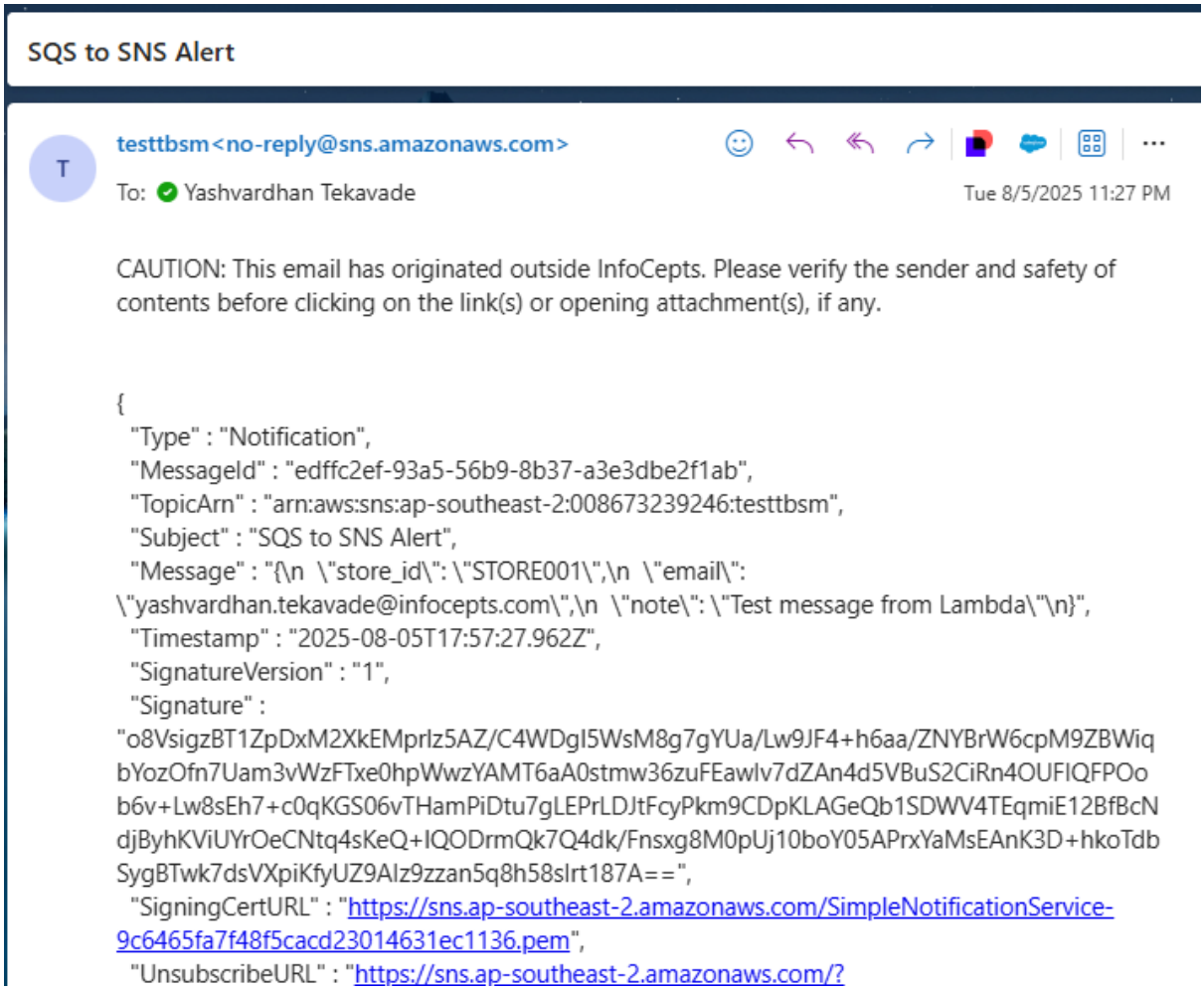
```
                ReceiptHandle=message['ReceiptHandle']

                )

                print("Deleted message from SQS.")
        except Exception as e:
                print("Error processing message:", e)
        time.sleep(2)

if __name__ == '__main__':

poll_and_forward()
```

Final Output Verification

- When Lambda sends a message → SQS receives it → EC2 picks it → SNS sends email

Screenshot Ideas:

- EC2 terminal showing:
    - “Received message: {...}”
    - “Published to SNS. MessageId: ...”
- Email inbox showing the alert



SQS to SNS Alert

T

testtbsm<no-reply@sns.amazonaws.com>

To: Yashvardhan Tekavade

Tue 8/5/2025 11:27 PM

CAUTION: This email has originated outside InfoCepts. Please verify the sender and safety of contents before clicking on the link(s) or opening attachment(s), if any.

{
"Type" : "Notification",
"MessageId" : "edffc2ef-93a5-56b9-8b37-a3e3dbe2f1ab",
"TopicArn" : "arn:aws:sns:ap-southeast-2:008673239246:testtbsm",
"Subject" : "SQS to SNS Alert",
"Message" : "{\n \"store_id\": \"STORE001\",\n \"email\": \"yashvardhan.tekavade@infocepts.com\",\n \"note\": \"Test message from Lambda\"\n}",
"Timestamp" : "2025-08-05T17:57:27.962Z",
"SignatureVersion" : "1",
"Signature" : "o8VsigzBT1ZpDxM2XkEMprlz5AZ/C4WDgI5WsM8g7gYUa/Lw9JF4+h6aa/ZNYBrW6cpM9ZBWiqbYozOfn7Uam3vWzFTxe0hpWwzYAMT6aA0stmw36zuFEawlv7dZAn4d5VBuS2CiRn4OUFIQFPOob6v+Lw8sEh7+c0qKGS06vTHamPiDtu7gLEPrLDJtFcyPkm9CDpKLAGeQb1SDWV4TEqmiE12BfBcNdjByhKViUYrOeCNtq4sKeQ+IQODrmQk7Q4dk/Fnsxg8M0pUj10boY05APrxYaMsEAnK3D+hkoTdbSygBTwk7dsVXpiKfyUZ9AIz9zzan5q8h58slrt187A==",
"SigningCertURL" : "https://sns.ap-southeast-2.amazonaws.com/SimpleNotificationService-9c6465fa7f48f5cacd23014631ec1136.pem",
"UnsubscribeURL" : "https://sns.ap-southeast-2.amazonaws.com/?


```
[ec2-user@ip-172-31-8-179 ~]$ python3 testto.py
Polling messages from SQS...
Received message: {'store_id': 'STORE001', 'email': 'yashvardhan.tekavade@infocepts.com', 'note': 'Test message from Lambda'}
Published to SNS. MessageId: fddacac3-a7cc-55b5-aec0-abcfe02a567d
Deleted message from SQS.
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
Received message: {'store_id': 'STORE002', 'email': 'ambrish.solanki@infocepts.com', 'note': 'Test message from Lambda'}
Published to SNS. MessageId: f1febc44-d8e5-5f17-b179-dd4890f3a23a
Deleted message from SQS.
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
No messages in queue. Waiting...
Polling messages from SQS...
Received message: {'store_id': 'STORE001', 'email': 'yashvardhan.tekavade@infocepts.com', 'note': 'Test message from Lambda'}
Published to SNS. MessageId: a43da6e6-d51d-57e5-b572-c7ea9d5890f7
Deleted message from SQS.
```