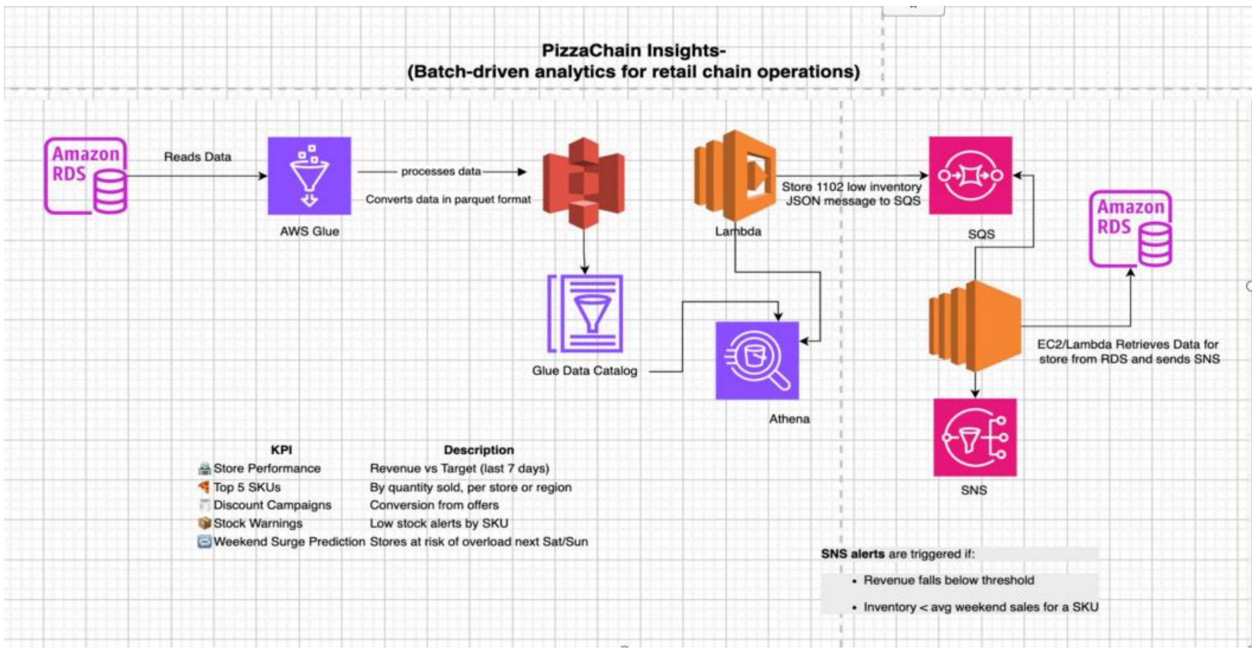
Project documentation – 3

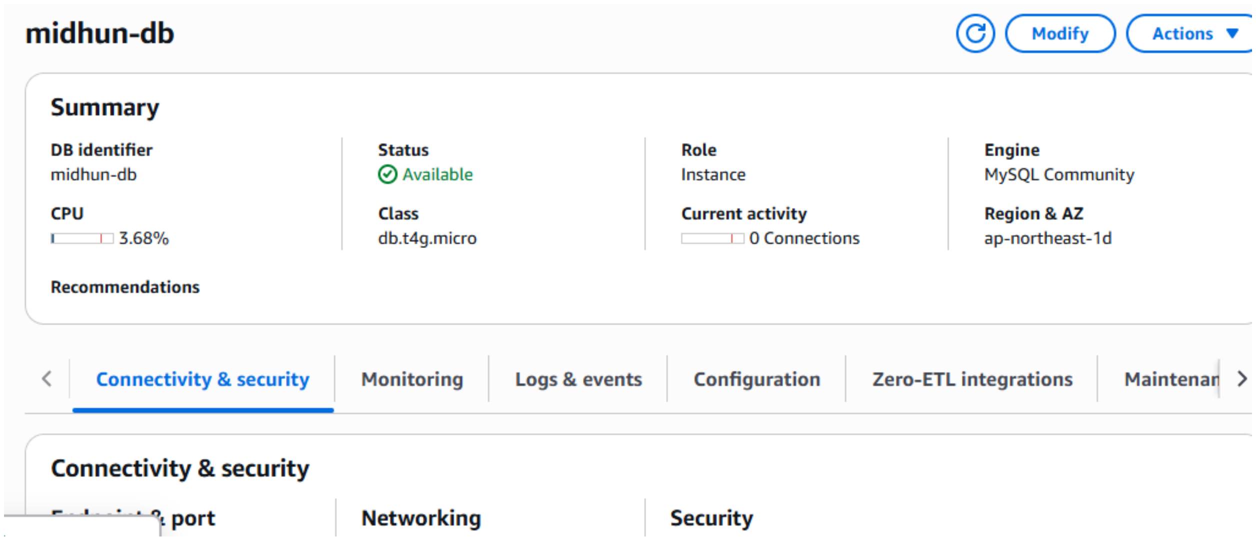

PizzaChain Insights-
(Batch-driven analytics for retail chain operations)
Amazon RDS
Reads Data
AWS Glue
processes data
Converts data in parquet format
Glue Data Catalog
Lambda
Store 1102 low inventory JSON message to SQS
SQS
Athena
Amazon RDS
EC2/Lambda Retrieves Data for store from RDS and sends SNS
SNS

KPI	Description
Store Performance	Revenue vs Target (last 7 days)
Top 5 SKUs	By quantity sold, per store or region
Discount Campaigns	Conversion from offers
Stock Warnings	Low stock alerts by SKU
Weekend Surge Prediction	Stores at risk of overload next Sat/Sun

SNS alerts are triggered if:

- Revenue falls below threshold
- Inventory < avg weekend sales for a SKU

Create an RDS



midhun-db

Summary

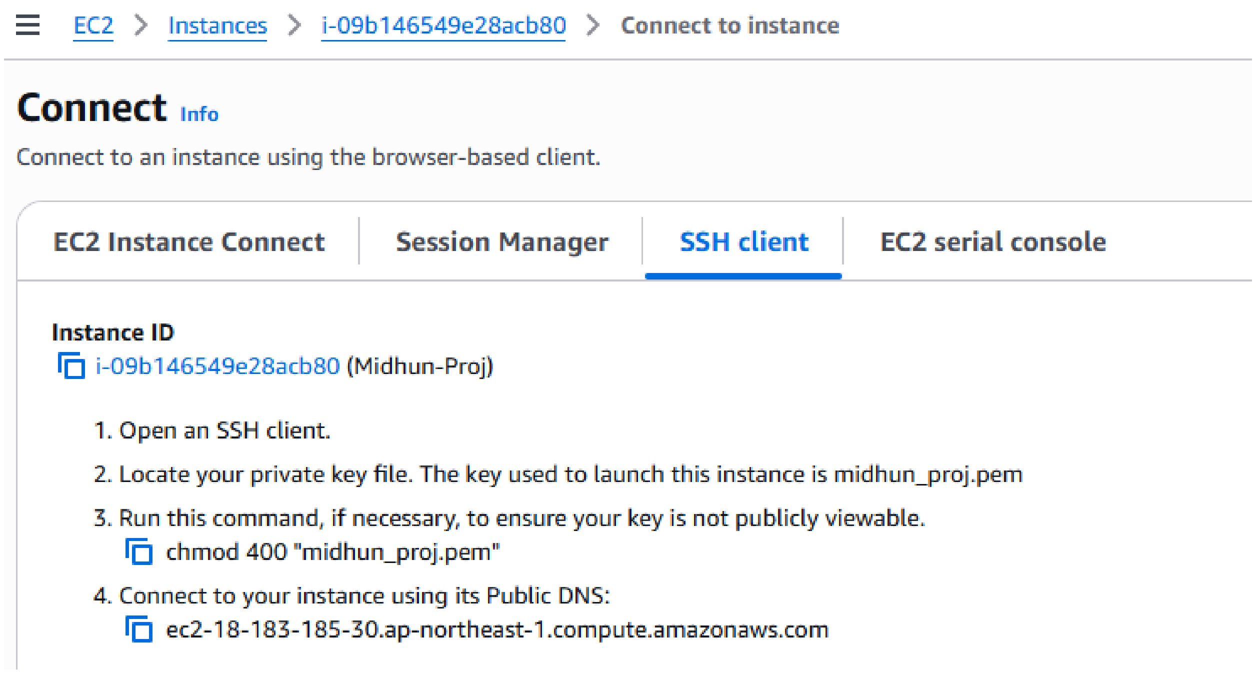
DB identifier midhun-db	**Status** Available	**Role** Instance	**Engine** MySQL Community
CPU 3.68%	**Class** db.t4g.micro	**Current activity** 0 Connections	**Region & AZ** ap-northeast-1d

Recommendations

Connectivity & security

Endpoint & port | **Networking** | **Security**

Create an instance and connect to it



EC2 > Instances > i-09b146549e28acb80 > Connect to instance

Connect Info

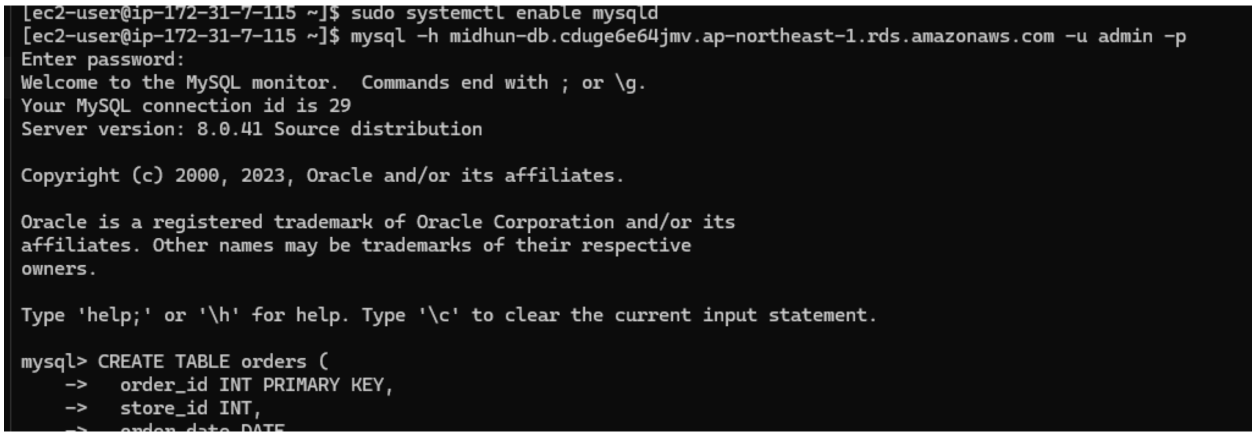
Connect to an instance using the browser-based client.

Instance ID

i-09b146549e28acb80 (Midhun-Proj)

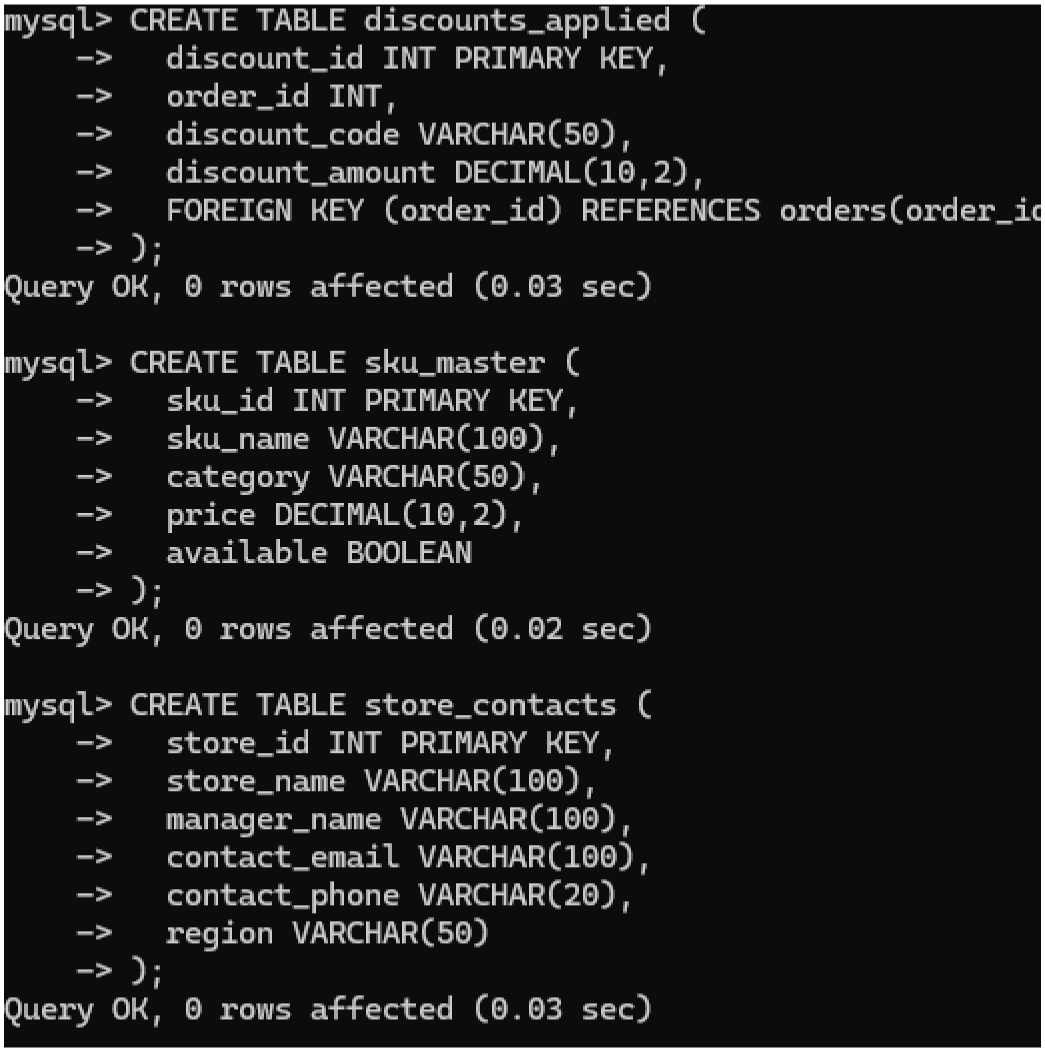
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is midhun_proj.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 "midhun_proj.pem"
4. Connect to your instance using its Public DNS:
   ec2-18-183-185-30.ap-northeast-1.compute.amazonaws.com

Connect to RDS from that instance



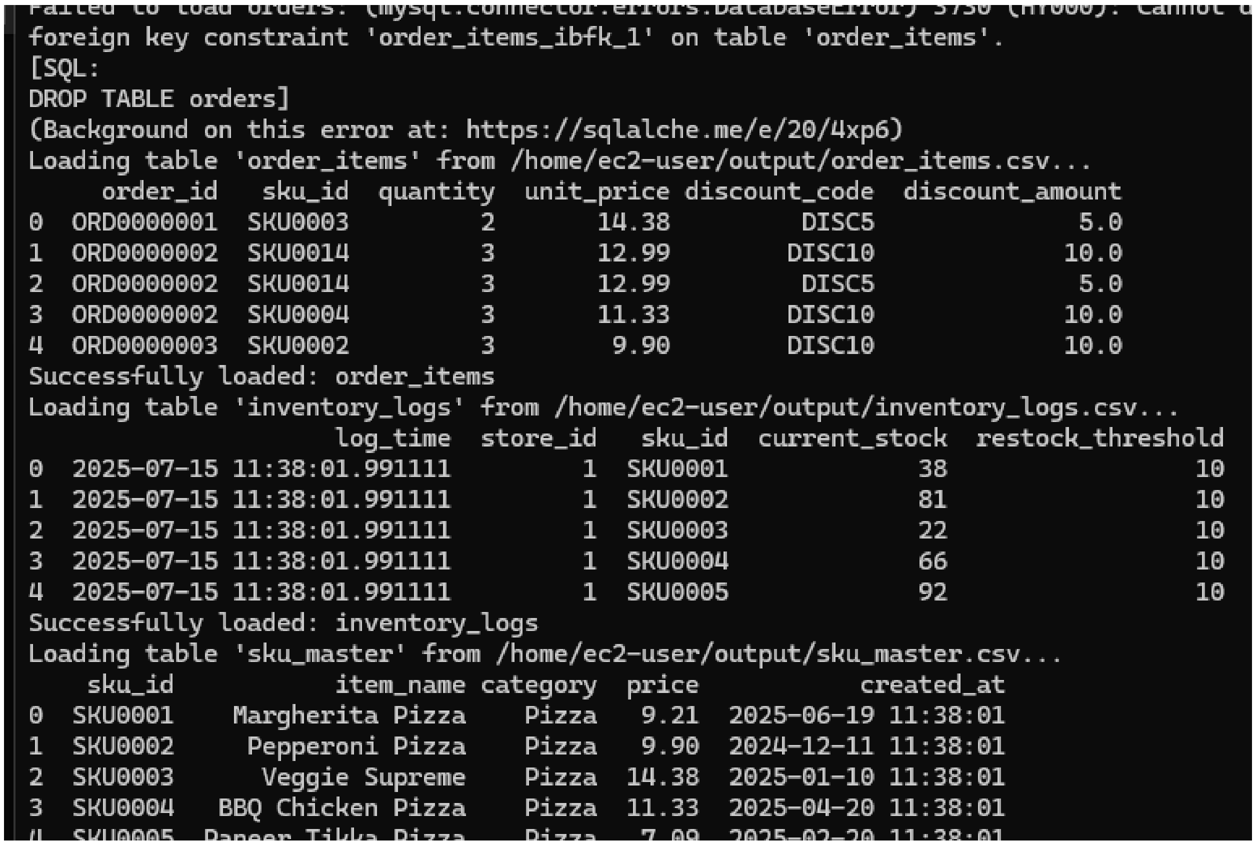
```
[ec2-user@ip-172-31-7-115 ~]$ sudo systemctl enable mysqld
[ec2-user@ip-172-31-7-115 ~]$ mysql -h midhun-db.cduge6e64jmv.ap-northeast-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE TABLE orders (
    ->   order_id INT PRIMARY KEY,
    ->   store_id INT,
    ->   order_date DATE
```

mysql -h midhun-db.cduge6e64jmv.ap-northeast-1.rds.amazonaws.com -u admin -p create DB and tables



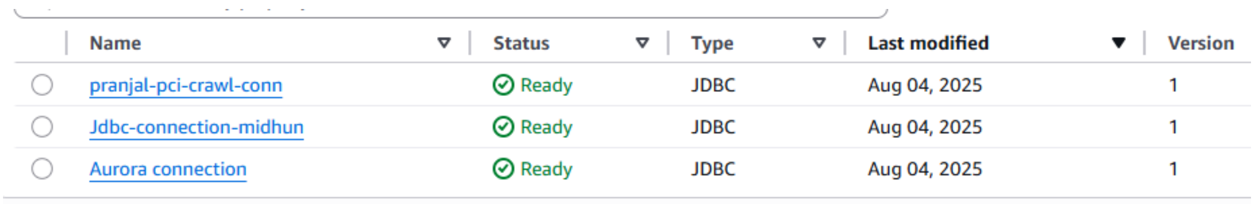
```
mysql> CREATE TABLE discounts_applied (
    ->   discount_id INT PRIMARY KEY,
    ->   order_id INT,
    ->   discount_code VARCHAR(50),
    ->   discount_amount DECIMAL(10,2),
    ->   FOREIGN KEY (order_id) REFERENCES orders(order_ic
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE sku_master (
    ->   sku_id INT PRIMARY KEY,
    ->   sku_name VARCHAR(100),
    ->   category VARCHAR(50),
    ->   price DECIMAL(10,2),
    ->   available BOOLEAN
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE store_contacts (
    ->   store_id INT PRIMARY KEY,
    ->   store_name VARCHAR(100),
    ->   manager_name VARCHAR(100),
    ->   contact_email VARCHAR(100),
    ->   contact_phone VARCHAR(20),
    ->   region VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

load data to the tables



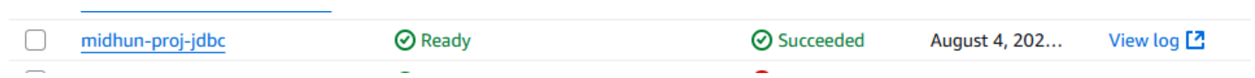
```
Failed to load orders: (mysql.connector.errors.DatabaseError) 3730 (HY000): Cannot d
foreign key constraint 'order_items_ibfk_1' on table 'order_items'.
[SQL:
DROP TABLE orders]
(Background on this error at: https://sqlalche.me/e/20/4xp6)
Loading table 'order_items' from /home/ec2-user/output/order_items.csv...
     order_id   sku_id  quantity  unit_price discount_code  discount_amount
0  ORD0000001  SKU0003         2       14.38         DISC5              5.0
1  ORD0000002  SKU0014         3       12.99        DISC10             10.0
2  ORD0000002  SKU0014         3       12.99         DISC5              5.0
3  ORD0000002  SKU0004         3       11.33        DISC10             10.0
4  ORD0000003  SKU0002         3        9.90        DISC10             10.0
Successfully loaded: order_items
Loading table 'inventory_logs' from /home/ec2-user/output/inventory_logs.csv...
                     log_time  store_id   sku_id  current_stock  restock_threshold
0  2025-07-15 11:38:01.991111         1  SKU0001             38                 10
1  2025-07-15 11:38:01.991111         1  SKU0002             81                 10
2  2025-07-15 11:38:01.991111         1  SKU0003             22                 10
3  2025-07-15 11:38:01.991111         1  SKU0004             66                 10
4  2025-07-15 11:38:01.991111         1  SKU0005             92                 10
Successfully loaded: inventory_logs
Loading table 'sku_master' from /home/ec2-user/output/sku_master.csv...
    sku_id           item_name category  price           created_at
0  SKU0001    Margherita Pizza    Pizza   9.21  2025-06-19 11:38:01
1  SKU0002     Pepperoni Pizza    Pizza   9.90  2024-12-11 11:38:01
2  SKU0003      Veggie Supreme    Pizza  14.38  2025-01-10 11:38:01
3  SKU0004   BBQ Chicken Pizza    Pizza  11.33  2025-04-20 11:38:01
4  SKU0005  Paneer Tikka Pizza    Pizza   7.09  2025-02-20 11:38:01
```

Now create a connection



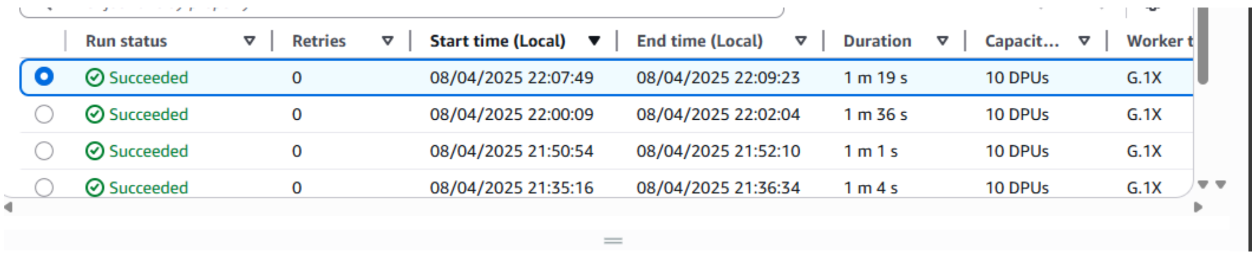
	Name	Status	Type	Last modified	Version
○	pranjal-pci-crawl-conn	✓ Ready	JDBC	Aug 04, 2025	1
○	Jdbc-connection-midhun	✓ Ready	JDBC	Aug 04, 2025	1
○	Aurora connection	✓ Ready	JDBC	Aug 04, 2025	1

Then do a crawler job



☐	midhun-proj-jdbc	✓ Ready	✓ Succeeded	August 4, 202...	View log

After this do etl jobs



	Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacit...	Worker t
●	✓ Succeeded	0	08/04/2025 22:07:49	08/04/2025 22:09:23	1 m 19 s	10 DPUs	G.1X
○	✓ Succeeded	0	08/04/2025 22:00:09	08/04/2025 22:02:04	1 m 36 s	10 DPUs	G.1X
○	✓ Succeeded	0	08/04/2025 21:50:54	08/04/2025 21:52:10	1 m 1 s	10 DPUs	G.1X
○	✓ Succeeded	0	08/04/2025 21:35:16	08/04/2025 21:36:34	1 m 4 s	10 DPUs	G.1X

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://midhun-
tokyo/project3/temp-folder/")
```

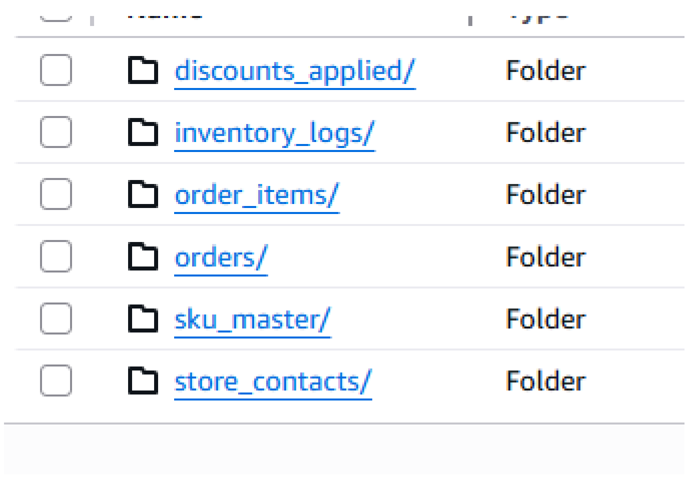
```
spark._jsc.hadoopConfiguration().set("hadoop.tmp.dir", "s3://midhun-
tokyo/project3/temp-folder/")

# Config
database_name = "midhun_proj"
s3_output_base = "s3://midhun-tokyo/project3/output-folder/"

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

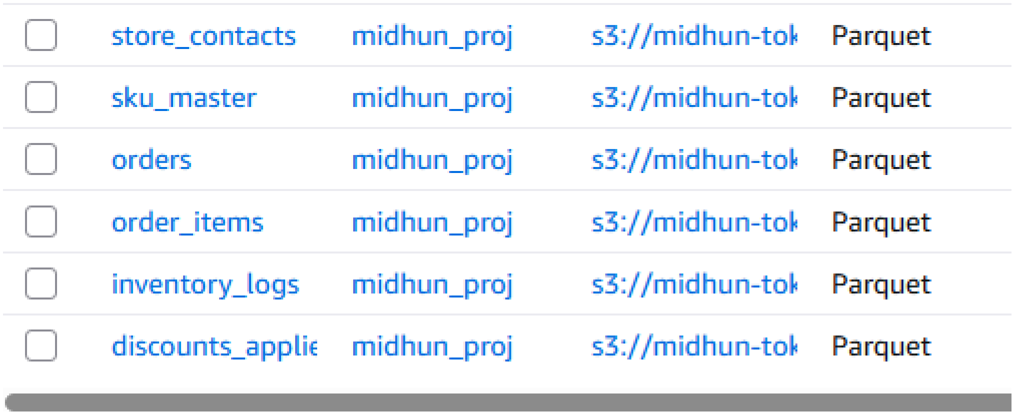
```
# Commit the job

job.commit()
```

This ETL job writes every table details to S3 as parquet format.



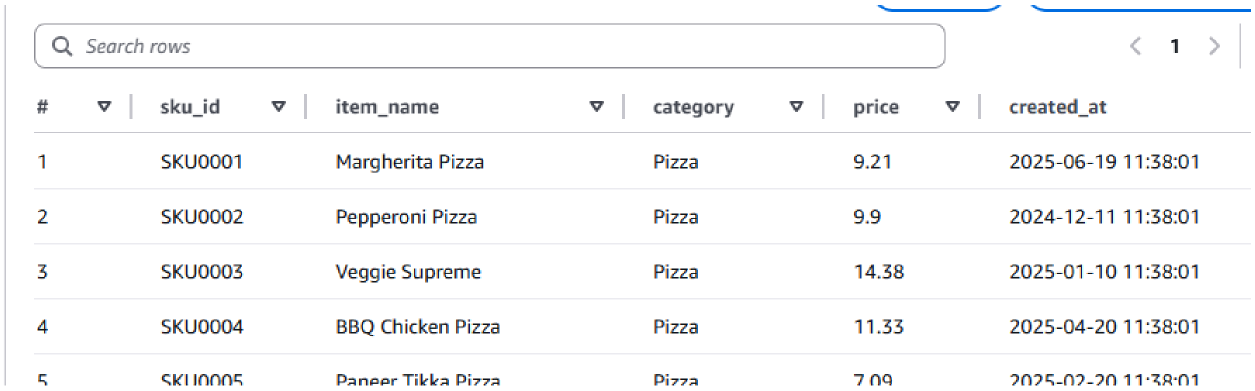
☐	discounts_applied/	Folder
☐	inventory_logs/	Folder
☐	order_items/	Folder
☐	orders/	Folder
☐	sku_master/	Folder
☐	store_contacts/	Folder

Now use crawler to crawl these tables from the s3 so that we can get the catalog



☐	store_contacts	midhun_proj	s3://midhun-tok	Parquet
☐	sku_master	midhun_proj	s3://midhun-tok	Parquet
☐	orders	midhun_proj	s3://midhun-tok	Parquet
☐	order_items	midhun_proj	s3://midhun-tok	Parquet
☐	inventory_logs	midhun_proj	s3://midhun-tok	Parquet
☐	discounts_applie	midhun_proj	s3://midhun-tok	Parquet

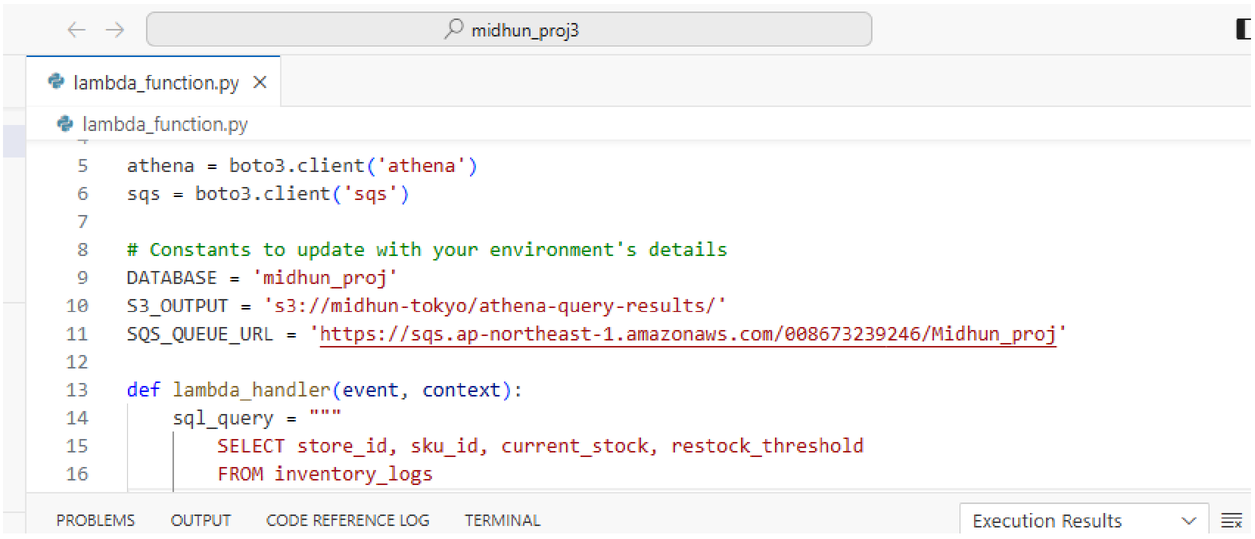
Try to query it in athena



Search rows

#	sku_id	item_name	category	price	created_at
1	SKU0001	Margherita Pizza	Pizza	9.21	2025-06-19 11:38:01
2	SKU0002	Pepperoni Pizza	Pizza	9.9	2024-12-11 11:38:01
3	SKU0003	Veggie Supreme	Pizza	14.38	2025-01-10 11:38:01
4	SKU0004	BBQ Chicken Pizza	Pizza	11.33	2025-04-20 11:38:01
5	SKU0005	Paneer Tikka Pizza	Pizza	7.09	2025-02-20 11:38:01

Create a lambda function



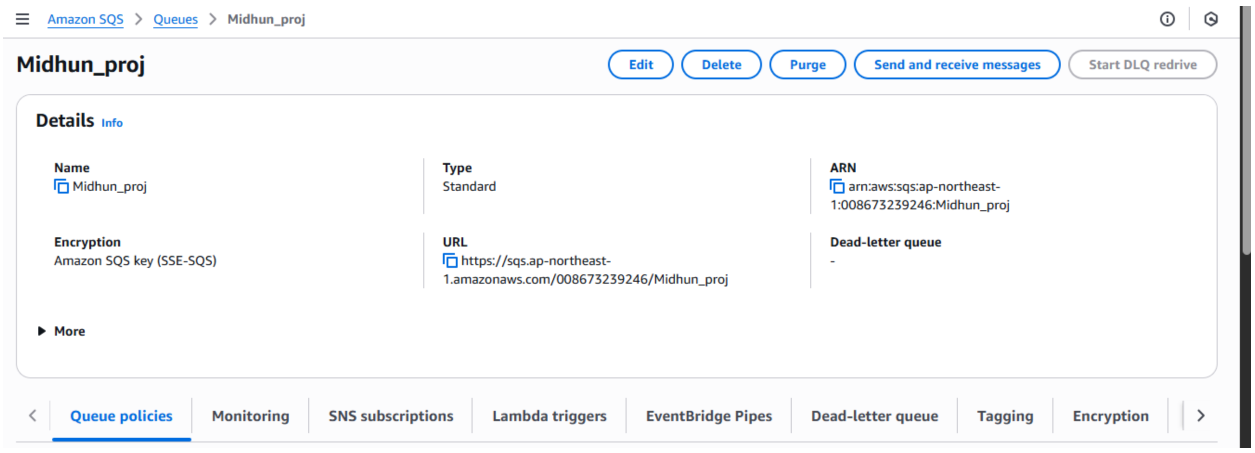
midhun_proj3

lambda_function.py

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

# Constants to update with your environment's details
DATABASE = 'midhun_proj'
S3_OUTPUT = 's3://midhun-tokyo/athena-query-results/'
SQS_QUEUE_URL = 'https://sqs.ap-northeast-1.amazonaws.com/008673239246/Midhun_proj'

def lambda_handler(event, context):
    sql_query = """
        SELECT store_id, sku_id, current_stock, restock_threshold
        FROM inventory_logs
```

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL Execution Results

Create an SQS



Amazon SQS > Queues > Midhun_proj

Midhun_proj

Edit Delete Purge Send and receive messages Start DLQ redrive

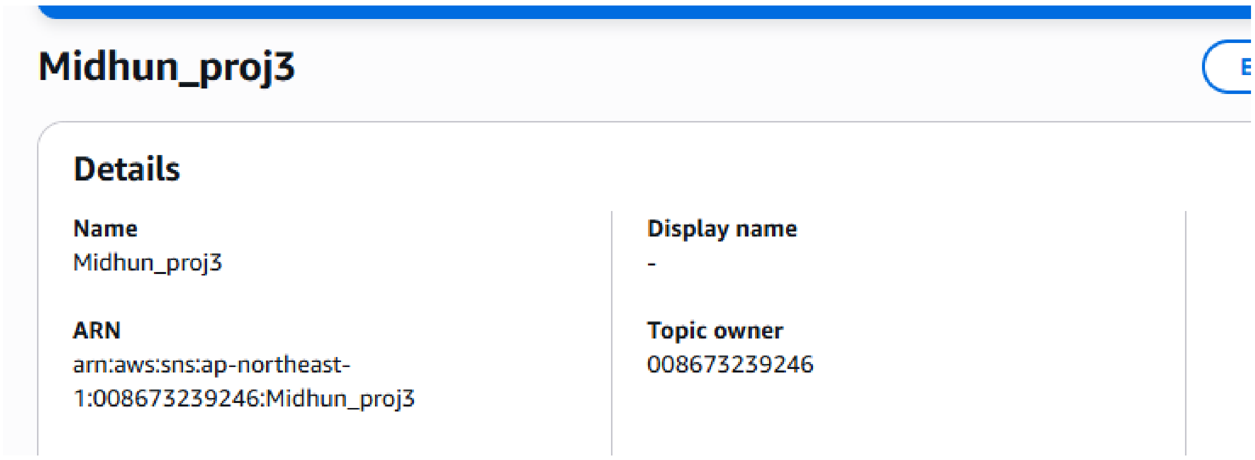
Details Info

Name Midhun_proj	**Type** Standard	**ARN** arn:aws:sqs:ap-northeast-1:008673239246:Midhun_proj
Encryption Amazon SQS key (SSE-SQS)	**URL** https://sqs.ap-northeast-1.amazonaws.com/008673239246/Midhun_proj	**Dead-letter queue** -

▶ More

Queue policies | Monitoring | SNS subscriptions | Lambda triggers | EventBridge Pipes | Dead-letter queue | Tagging | Encryption

Create an SNS



Midhun_proj3

Details

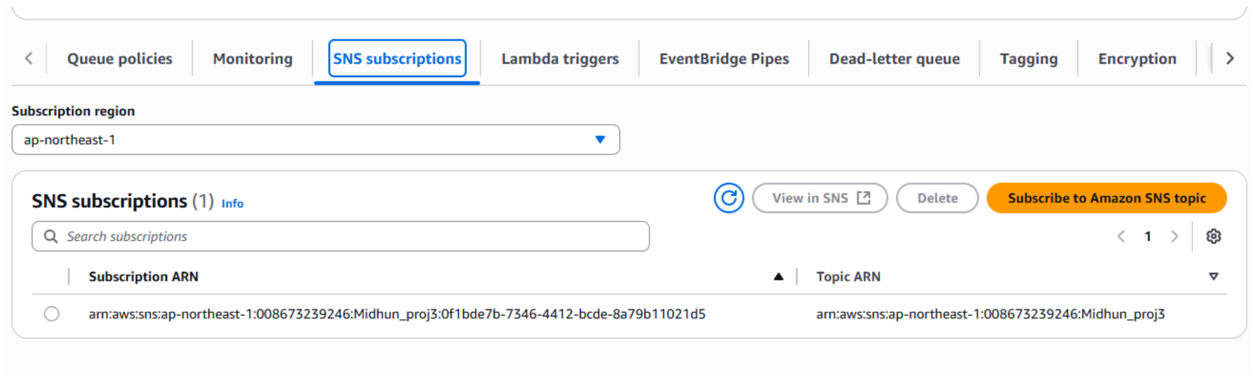
Name
Midhun_proj3

Display name
-

ARN
arn:aws:sns:ap-northeast-1:008673239246:Midhun_proj3

Topic owner
008673239246

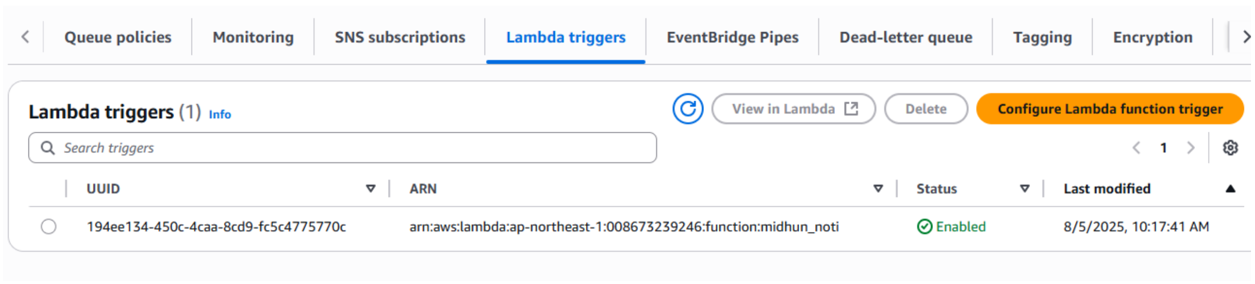
Subscribe to the SNS and add a new lambda trigger from the SQS



Subscription region
ap-northeast-1

SNS subscriptions (1) Info

Subscription ARN	Topic ARN
arn:aws:sns:ap-northeast-1:008673239246:Midhun_proj3:0f1bde7b-7346-4412-bcde-8a79b11021d5	arn:aws:sns:ap-northeast-1:008673239246:Midhun_proj3

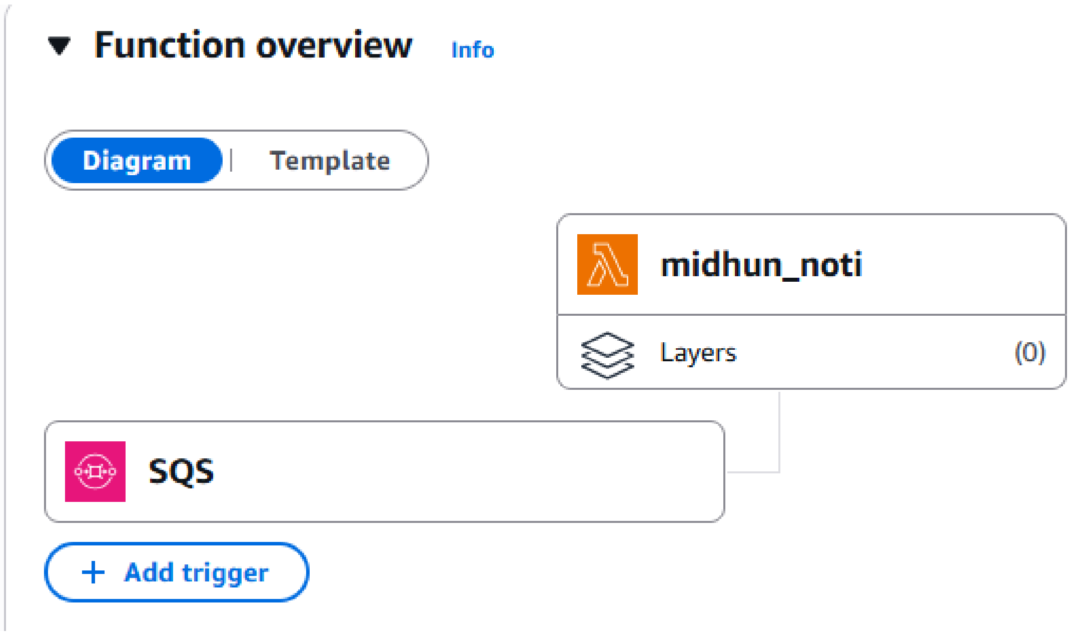


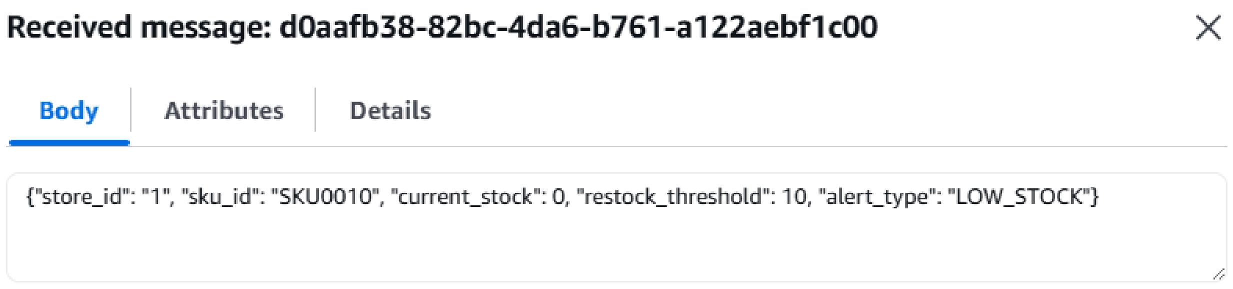
Lambda triggers (1) Info

UUID	ARN	Status	Last modified
194ee134-450c-4caa-8cd9-fc5c4775770c	arn:aws:lambda:ap-northeast-1:008673239246:function:midhun_noti	Enabled	8/5/2025, 10:17:41 AM

The above lambda function checks if the stock is low and if the stock is low it will trigger an SQS

We are using this SQS to trigger another lambda function which sends SNS notification to the email ids of the shops which are configured





```
{"store_id": "1", "sku_id": "SKU0010", "current_stock": 0, "restock_threshold": 10, "alert_type": "LOW_STOCK"}
```

```
import boto3
import json
import time

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

# Constants to update with your environment's details
DATABASE = 'midhun_proj'
S3_OUTPUT = 's3://midhun-tokyo/athena-query-results/'
```

```
SQS_QUEUE_URL = 'https://sqs.ap-northeast-
1.amazonaws.com/008673239246/Midhun_proj'


def lambda_handler(event, context):
  sql_query = """
    SELECT store_id, sku_id, current_stock, restock_threshold
    FROM inventory_logs
    WHERE current_stock < restock_threshold
  """

  # Start Athena query execution
  response = athena.start_query_execution(
    QueryString=sql_query,
    QueryExecutionContext={'Database': DATABASE},
    ResultConfiguration={'OutputLocation': S3_OUTPUT}
  )
  query_execution_id = response['QueryExecutionId']

  # Wait for the query to complete
  while True:
    query_status = athena.get_query_execution(QueryExecutionId=query_execution_id)
    state = query_status['QueryExecution']['Status']['State']
    if state in ['SUCCEEDED', 'FAILED', 'CANCELLED']:
      break
    time.sleep(1)  # Wait and poll again
```

```
if state == 'SUCCEEDED':
    results_paginator = athena.get_paginator('get_query_results')
    page_iterator = results_paginator.paginate(QueryExecutionId=query_execution_id)

    # Skip header row, process rows page-wise
    first_page = True
    for page in page_iterator:
        rows = page['ResultSet']['Rows']
        if first_page:
            # Skip header row in first page
            rows = rows[1:]
            first_page = False

        for row in rows:
            data = row['Data']
            store_id = data[0].get('VarCharValue', None)
            sku_id = data[1].get('VarCharValue', None)
            current_stock = int(data[2].get('VarCharValue', 0))
            restock_threshold = int(data[3].get('VarCharValue', 0))

            # Prepare message JSON
            message = {
                'store_id': store_id,
                'sku_id': sku_id,
                'current_stock': current_stock,
                'restock_threshold': restock_threshold,
                'alert_type': 'LOW_STOCK'
            }

            # Send message to SQS
            sqs.send_message(
                QueueUrl=SQS_QUEUE_URL,
                MessageBody=json.dumps(message)
            )
        return {
            'statusCode': 200,
            'body': f"Successfully processed low stock alerts."
        }
    else:
        error_message = f"Athena query failed with state: {state}"
        print(error_message)
        return {
            'statusCode': 500,
            'body': error_message
        }
```

SQS

```
import json
import boto3

sns = boto3.client('sns')
```

```
SNS_TOPIC_ARN = 'arn:aws:sns:ap-northeast-1:008673239246:Midhun_proj3'


def lambda_handler(event, context):
  for record in event['Records']:
    alert = json.loads(record['body'])
    message = (
      f"Low Stock Alert!\n"
      f"Store: {alert['store_id']}\n"
      f"SKU: {alert['sku_id']}\n"
      f"Current Stock: {alert['current_stock']}\n"
      f"Restock Threshold: {alert['restock_threshold']}"
    )

    sns.publish(
      TopicArn=SNS_TOPIC_ARN,
      Subject="PizzaChain: Low Stock Notification",
      Message=message
    )
  return {"status": "done"}
```