Media Stream Analytics

Datasets & Schema Summary

Ad Revenue

Column	Type	Description
channel_id	int	Unique Channel ID
channel_name	varchar	Name of the channel
date	varchar	Date of revenue
ad_revenue	double	Revenue from ads

Channel Metadata

Column	Type	Description
channel_id	int	Unique Channel ID
channel_name	varchar	Name of the channel
genre	varchar	Genre of the channel
language	varchar	Language of the channel
launch_year	int	Year of launch

Demographics

Column	Type	Description
user_id	int	Unique User ID
gender	varchar	Gender

Column	Type	Description
age_group	varchar	Age group
region	varchar	Region
subscription_type	varchar	Subscription type

Viewership Logs (Real-Time)

Column	Type	Description
user_id	int	Unique User ID
channel_name	varchar	Channel name
channel_id	varchar	Channel ID
timestamp	varchar	Timestamp
duration	int	Duration watched
region	varchar	Region
subscription	varchar	Subscription type
device	varchar	Device used
platform	varchar	Platform used
is_live	boolean	Live stream indicator
genre	varchar	Genre
ads_watched	int	Number of ads watched

Column	Type	Description
ad_revenue	double	Revenue from ads
engagement	int	Engagement score
buffer_count	int	Buffering incidents
completion_pct	int	Completion percentage
session_id	varchar	Session ID
show_name	varchar	Name of the show

Step 1: EC2 Setup for Data Generation

✅ **Launch EC2 Instance**

- Go to EC2 Console → Launch Instance
- **AMI**: Amazon Linux 2
- **Instance Type**: t2.micro (or higher)
- **Key Pair**: Create or use existing
- **Security Group**: Allow SSH (port 22)
- **SSH ssh -i your-key.pem ec2-user@your-ec2-public-ip**
- Upload and Run Data Generator
- nano generate_media_data.py
- python3 generate_media_data.py

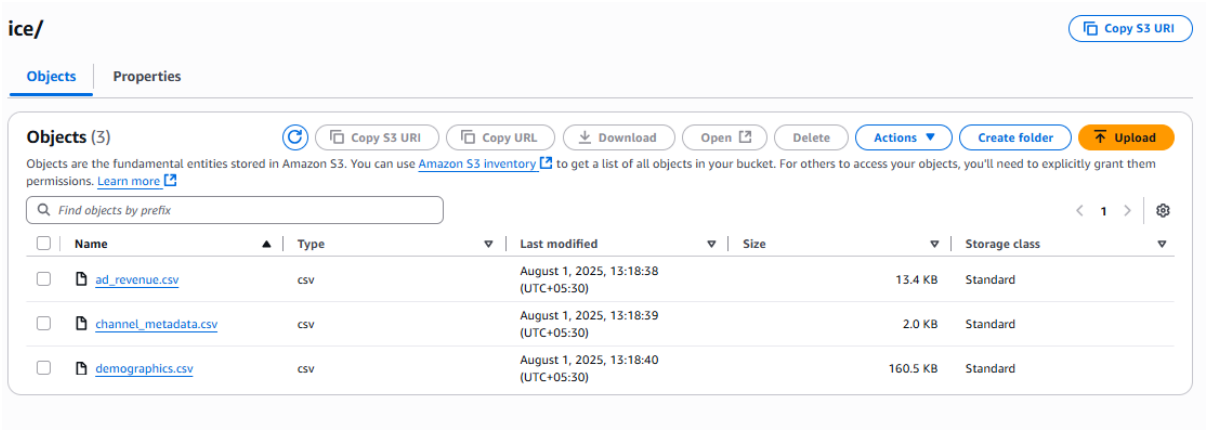
```
import csv
import random
from datetime import datetime, timedelta

# ----------------------------
# Global Lists and Mappings
# ----------------------------
CHANNEL_NAMES = [
  "Star Plus", "Colors TV", "Sony Entertainment Television", "Zee TV", "Star Bharat",
  "Sony SAB", "Dangal TV", "Zee Anmol", "Colors Rishtey", "DD National",
  "Star Sports 1", "Star Sports 2", "Star Sports 3", "Star Sports Select 1", "Star Sports Select 2",
  "Sony Sports Ten 1", "Sony Sports Ten 2", "Sony Sports Ten 3", "DD Sports", "Eurosport India",
  "Aaj Tak", "ABP News", "India TV", "News18 India", "Republic Bharat",
  "Times Now", "CNN-News18", "NDTV India", "Zee News", "India Today",
  "Sun TV", "Star Vijay", "Colors Tamil", "Zee Tamil", "KTV",
  "Star Maa", "Zee Telugu", "ETV Telugu", "Gemini TV", "Asianet",
  "Surya TV", "Zee Kannada", "Colors Kannada", "Zee Marathi", "Zee Bangla",
  "ETV Marathi", "DD Sahyadri", "DD Bangla", "News18 Kerala", "News18 Tamil Nadu"
]

GENRES = ["Entertainment", "News", "Sports", "Kids", "Music", "Movies"]
LANGUAGES = ["Hindi", "English", "Tamil", "Telugu", "Malayalam", "Kannada", "Marathi", "Bengali"]
REGIONS = ["North", "South", "East", "West", "Central", "Northeast", "Pan-India"]
```

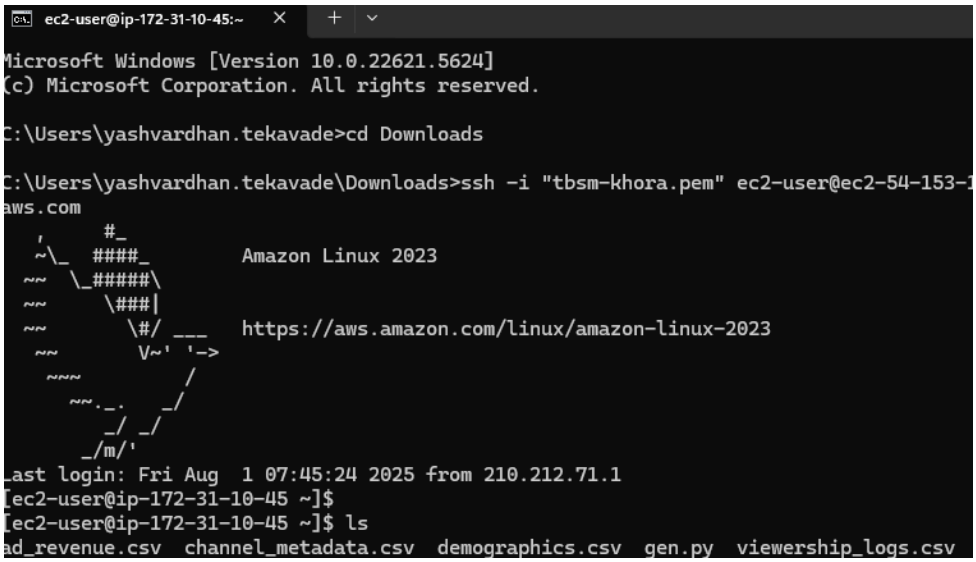
```
DEVICES = ["Mobile", "Tablet", "Smart TV", "Laptop"]
PLATFORMS = ["Android", "iOS", "Web", "FireTV", "Roku"]

CHANNEL_ATTRIBUTES = {
  # genre, language pairs
  "Star Sports 1": ("Sports", "Hindi"), "Sony Sports Ten 1": ("Sports", "English"),
  "DD Sports": ("Sports", "Hindi"), "Star Plus": ("Entertainment", "Hindi"),
  "Sony Entertainment Television": ("Entertainment", "Hindi"), "Zee TV":
("Entertainment", "Hindi"),
  "Colors TV": ("Entertainment", "Hindi"), "SAB TV": ("Entertainment", "Hindi"),
  "Aaj Tak": ("News", "Hindi"), "NDTV India": ("News", "English"),
  "Republic Bharat": ("News", "Hindi"), "Sun TV": ("Entertainment", "Tamil"),
  "ETV Telugu": ("Entertainment", "Telugu"), "Asianet": ("Entertainment", "Malayalam"),
  "Colors Marathi": ("Entertainment", "Marathi"), "Zee Bangla": ("Entertainment",
"Bengali"),
  "Times Now": ("News", "English"), "CNN-News18": ("News", "English"),
  "Zee News": ("News", "Hindi"), "ABP News": ("News", "Hindi"),
  "Star Vijay": ("Entertainment", "Tamil"), "KTV": ("Movies", "Tamil"),
}

# ----------------------------
# Dataset Generators
# ----------------------------

def generate_channel_metadata():
  print("Writing channel meta data.....")
  channel_metadata = []
  channel_id_map = {}
  for i, name in enumerate(CHANNEL_NAMES):
    channel_id = f"CH{i+1:03d}"
    genre, language = CHANNEL_ATTRIBUTES.get(name, (random.choice(GENRES),
random.choice(LANGUAGES)))
    launch_year = random.randint(2000, 2022)
    channel_metadata.append([channel_id, name, genre, language, launch_year])
    channel_id_map[name] = channel_id
  with open("channel_metadata.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["channel_id", "channel_name", "genre", "language",
"launch_year"])
    writer.writerows(channel_metadata)
  print("Writing channel meta data completed .....")
  return channel_id_map

def generate_demographics(num_users=5000):
  demographics = []
  user_ids = set()
  for _ in range(num_users):
```

```
        user_id = f"U{10000 + random.randint(0, 49999)}"
        gender = random.choice(["Male", "Female", "Other"])
        age_group = random.choice(["<18", "18-25", "26-35", "36-45", "46-60", "60+"])
        region = random.choice(REGIONS)
        subscription_type = random.choice(["Free", "Basic", "Premium"])
        demographics.append([user_id, gender, age_group, region, subscription_type])
        user_ids.add(user_id)
    with open("demographics.csv", "w", newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["user_id", "gender", "age_group", "region", "subscription_type"])
        writer.writerows(demographics)
    return user_ids

def generate_ad_revenue(channel_id_map, days=7):
    ad_revenue = []
    for name, cid in channel_id_map.items():
        for i in range(days):
            date = (datetime.now() - timedelta(days=i)).strftime('%Y-%m-%d')
            revenue = round(random.uniform(10000, 100000), 2)
            ad_revenue.append([cid, name, date, revenue])
    with open("ad_revenue.csv", "w", newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["channel_id", "channel_name", "date", "ad_revenue"])
        writer.writerows(ad_revenue)

def generate_viewership_logs(channel_id_map, user_ids, num_records=10000):
    logs = []
    start_time = datetime.now() - timedelta(days=7)
    user_id_list = list(user_ids)
    for _ in range(num_records):
        user_id = random.choice(user_id_list)
        channel_name = random.choice(CHANNEL_NAMES)
        channel_id = channel_id_map[channel_name]
        timestamp = start_time + timedelta(seconds=random.randint(0, 604800))
        duration = random.randint(1, 180)
        region = random.choice(REGIONS)
        subscription = random.choice(["Free", "Basic", "Premium"])
        device = random.choice(DEVICES)
        platform = random.choice(PLATFORMS)
        is_live = random.choice([True, False])
        genre = CHANNEL_ATTRIBUTES.get(channel_name,
(random.choice(GENRES),))[0]
        ads_watched = random.randint(0, 5)
        ad_revenue = round(ads_watched * random.uniform(10, 200), 2)
        engagement = round(random.uniform(0.2, 1.0), 2)
        buffer_count = random.randint(0, 3)
        completion_pct = round(random.uniform(20, 100), 2)
```

```
        session_id = f"SID{random.randint(100000, 999999)}"
        show_name = f"Show_{random.randint(1, 200)}"
        logs.append([
            session_id, user_id, channel_id, channel_name, show_name, genre,
timestamp.strftime('%Y-%m-%d %H:%M:%S'),
            duration, region, subscription, device, platform, is_live, ads_watched,
ad_revenue,
            engagement, buffer_count, completion_pct
        ])
    with open("viewership_logs.csv", "w", newline='') as f:
        writer = csv.writer(f)
        writer.writerow([
            "session_id", "user_id", "channel_id", "channel_name", "show_name", "genre",
"timestamp",
            "duration_minutes", "region", "subscription_type", "device", "platform", "is_live",
            "ads_watched", "ad_revenue", "engagement_score", "buffer_count",
"completion_percentage"
        ])
        writer.writerows(logs)

# ----------------------------
# Master Execution
# ----------------------------
if __name__ == "__main__":
    channel_id_map = generate_channel_metadata()
    user_ids = generate_demographics()
    generate_ad_revenue(channel_id_map)
    generate_viewership_logs(channel_id_map, user_ids, num_records=10000)  #
Adjust as needed
```

- Upload Generated CSVs to S3
- aws s3 cp ad_revenue.csv s3://tbsm-calm/ice/
- aws s3 cp channel_metadata.csv s3://tbsm-calm/ice/
- aws s3 cp demographics.csv s3://tbsm-calm/ice/


ice/
Copy S3 URI
Objects
Properties
Objects (3)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
1
Name
Type
Last modified
Size
Storage class
ad_revenue.csv
csv
August 1, 2025, 13:18:38 (UTC+05:30)
13.4 KB
Standard
channel_metadata.csv
csv
August 1, 2025, 13:18:39 (UTC+05:30)
2.0 KB
Standard
demographics.csv
csv
August 1, 2025, 13:18:40 (UTC+05:30)
160.5 KB
Standard


ec2-user@ip-172-31-10-45:~
Microsoft Windows [Version 10.0.22621.5624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\yashvardhan.tekavade>cd Downloads
C:\Users\yashvardhan.tekavade\Downloads>ssh -i "tbsm-khora.pem" ec2-user@ec2-54-153-1
aws.com
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
Last login: Fri Aug 1 07:45:24 2025 from 210.212.71.1
[ec2-user@ip-172-31-10-45 ~]$
[ec2-user@ip-172-31-10-45 ~]$ ls
ad_revenue.csv channel_metadata.csv demographics.csv gen.py viewership_logs.csv

Step 2: AWS Glue Setup

✅ **Create Glue Catalog Database**

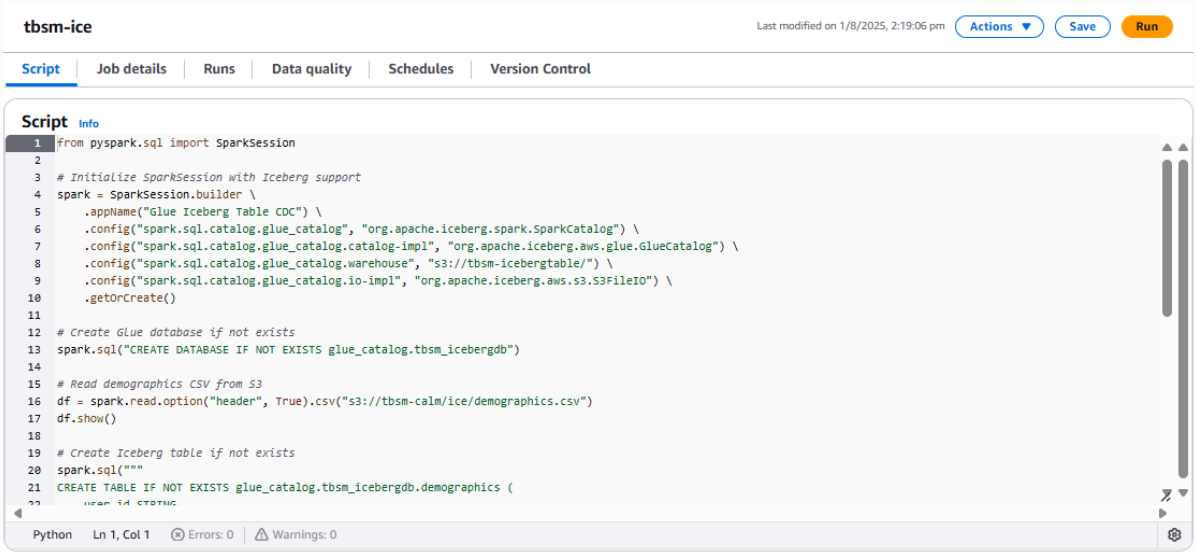
- Go to AWS Glue → Databases → Create
- Name: tbsm_icebergdb

✅ **Create Glue Job**

- Language: Python
- Glue Version: 4.0 or 5.0
- IAM Role: Must have access to S3 and Glue
- Job Parameters:

  Key: --datalake-formats

  Value: iceberg



Code :-

```
from pyspark.sql import SparkSession


# Initialize SparkSession with Iceberg support
spark = SparkSession.builder \
   .appName("Glue Iceberg Table CDC") \
   .config("spark.sql.catalog.glue_catalog", "org.apache.iceberg.spark.SparkCatalog")
\
```

```
   .config("spark.sql.catalog.glue_catalog.catalog-impl",
"org.apache.iceberg.aws.glue.GlueCatalog") \
   .config("spark.sql.catalog.glue_catalog.warehouse", "s3://tbsm-icebergtable/") \
   .config("spark.sql.catalog.glue_catalog.io-impl",
"org.apache.iceberg.aws.s3.S3FileIO") \
   .getOrCreate()

# Create Glue database if not exists
spark.sql("CREATE DATABASE IF NOT EXISTS glue_catalog.tbsm_icebergdb")

# Read demographics CSV from S3
df = spark.read.option("header", True).csv("s3://tbsm-calm/ice/demographics.csv")
df.show()

# Create Iceberg table if not exists
spark.sql("""
CREATE TABLE IF NOT EXISTS glue_catalog.tbsm_icebergdb.demographics (
   user_id STRING,
   gender STRING,
   age_group STRING,
   region STRING,
   subscription_type STRING
)
USING ICEBERG
""")

# Register DataFrame as temporary view
df.createOrReplaceTempView("updates")
```

```
# Perform MERGE (upsert) into Iceberg table
spark.sql("""
MERGE INTO glue_catalog.tbsm_icebergdb.demographics AS target
USING updates AS source
ON target.user_id = source.user_id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
""")

print("✅ Merge operation completed successfully.")
```

Run job

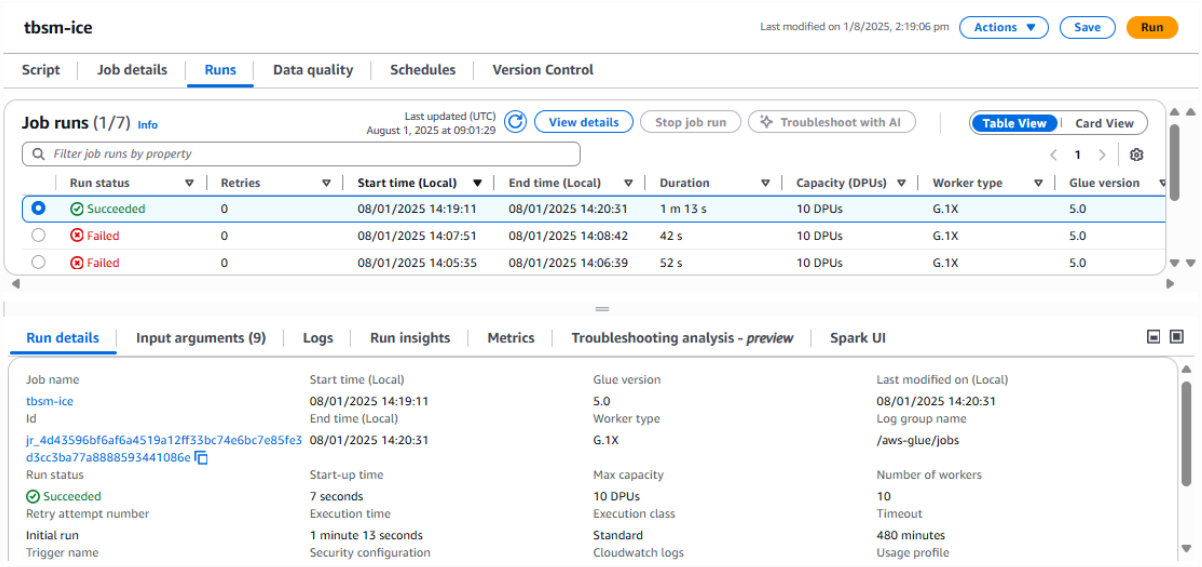
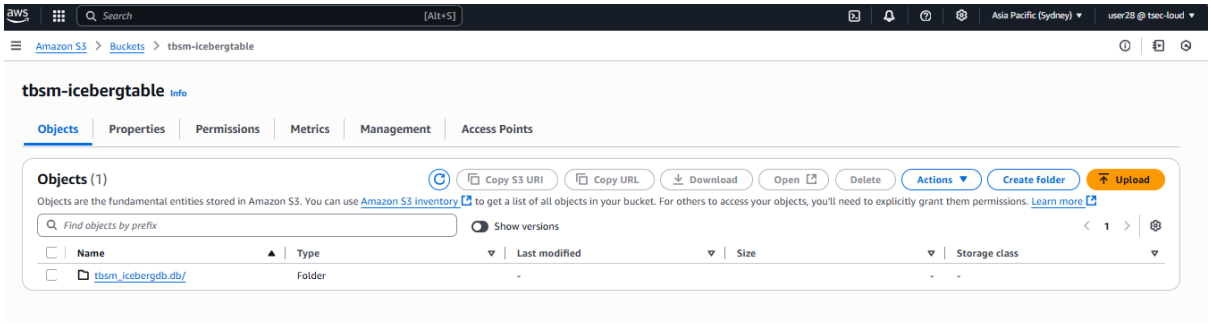


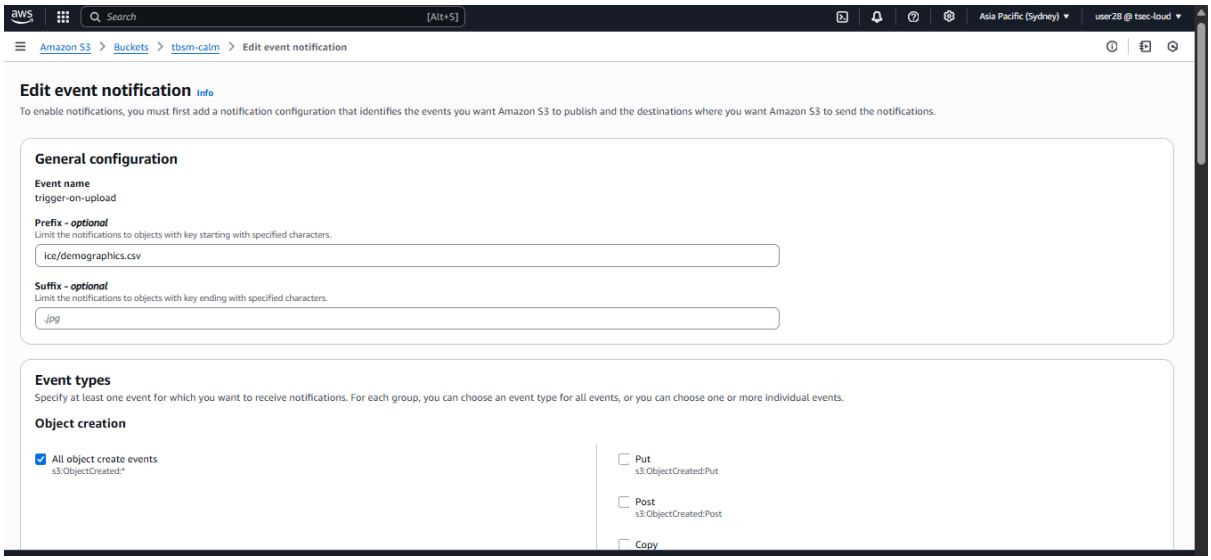
Table created in iceberg s3

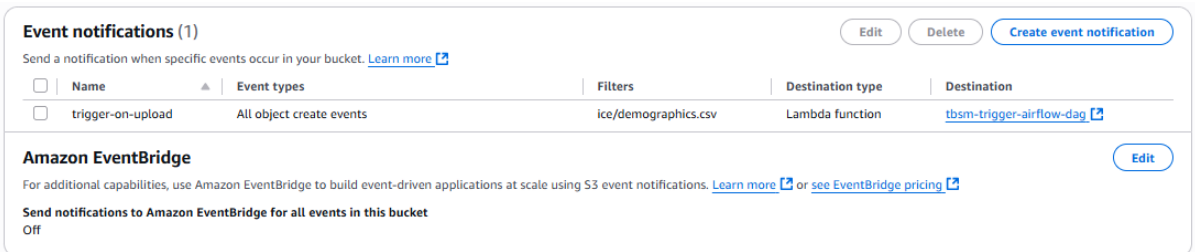


Step 3: S3 Event Notification

✅ Enable Event Notification

- Go to S3 → tbsm-calm → Properties
- Scroll to **Event Notifications**
- Create:
    - Name: trigger-on-upload
    - Event type: PUT
    - Prefix: ice/demographics.csv
    - Destination: Lambda Function

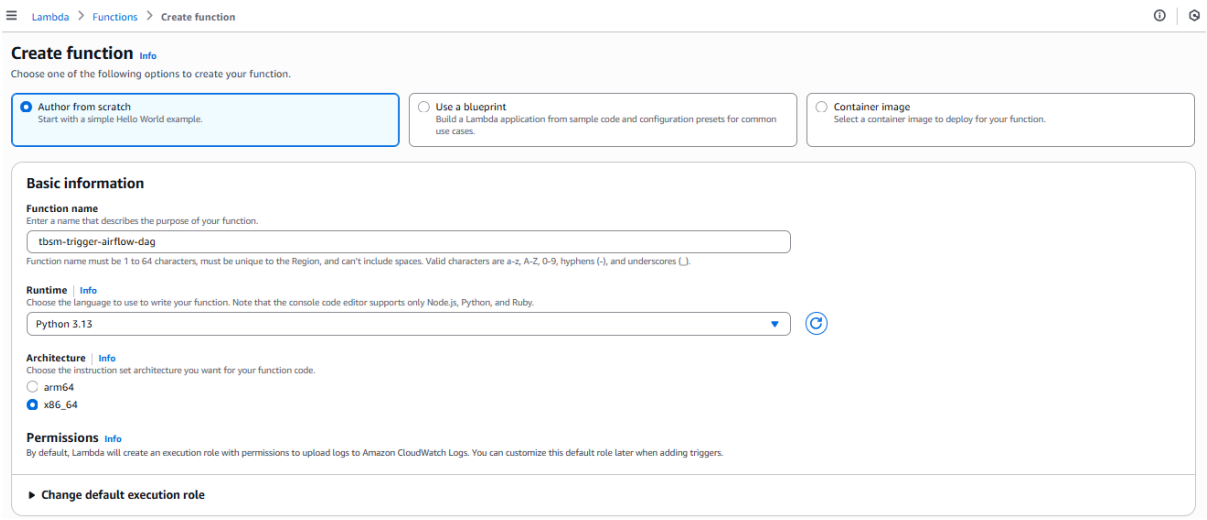


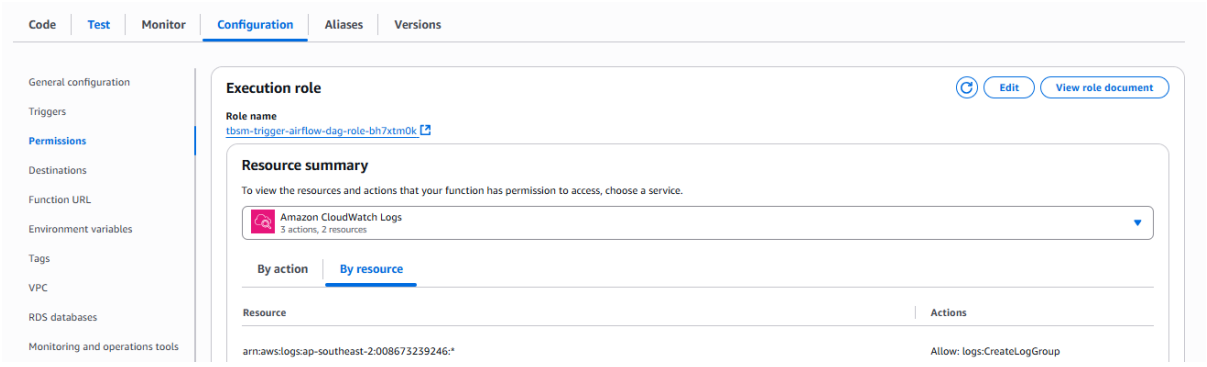


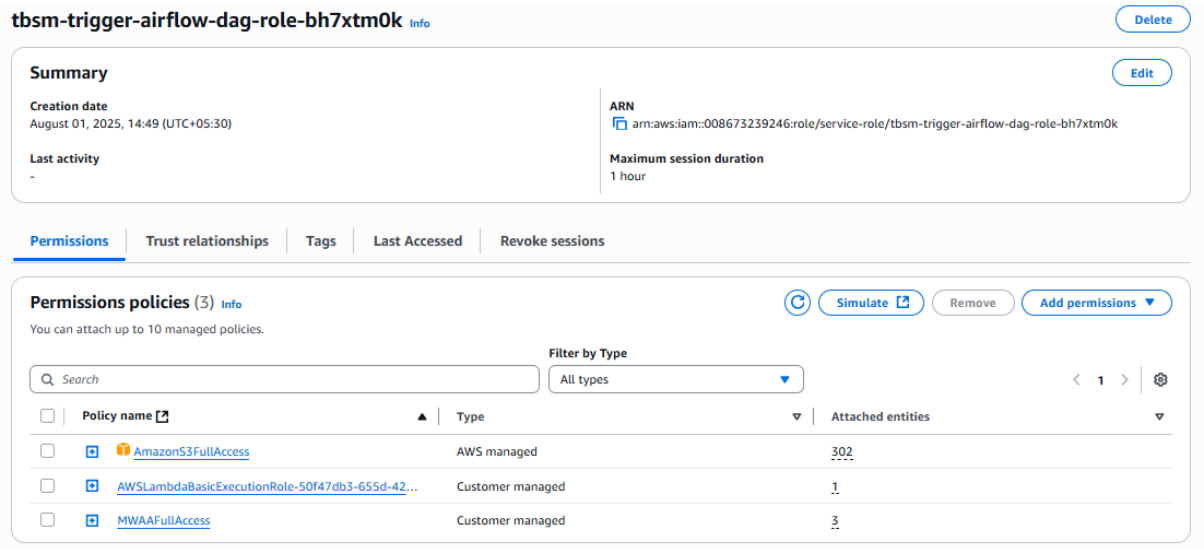
Step 4: Lambda Setup

Create Lambda Function

- Runtime: Python 3.10
- Name: trigger-airflow-dag
- Permissions:
    - MWAAFullAccess
    - AmazonS3ReadOnlyAccess

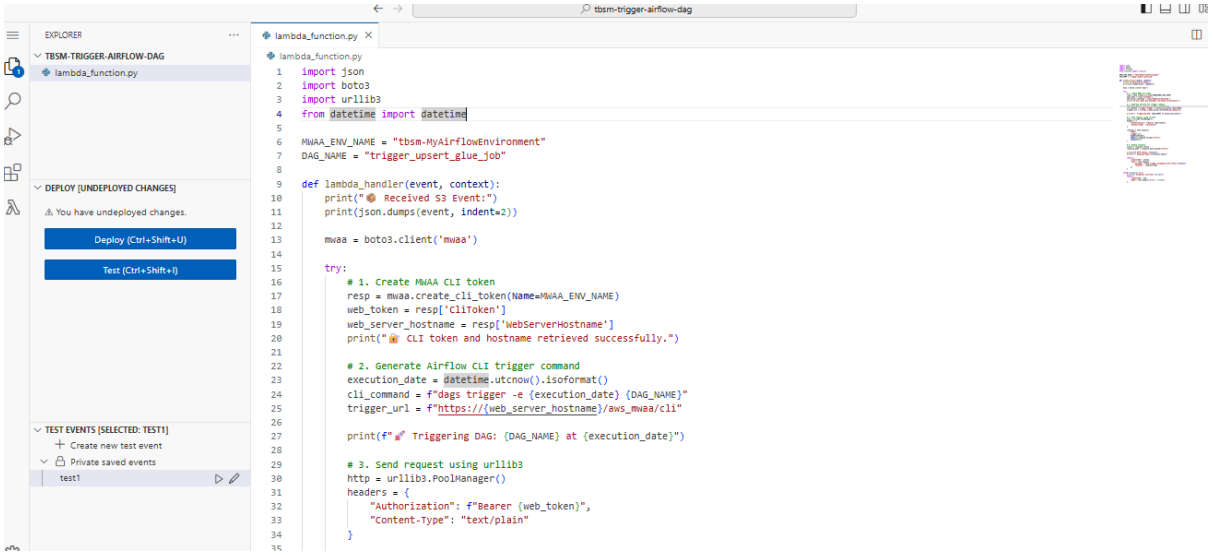


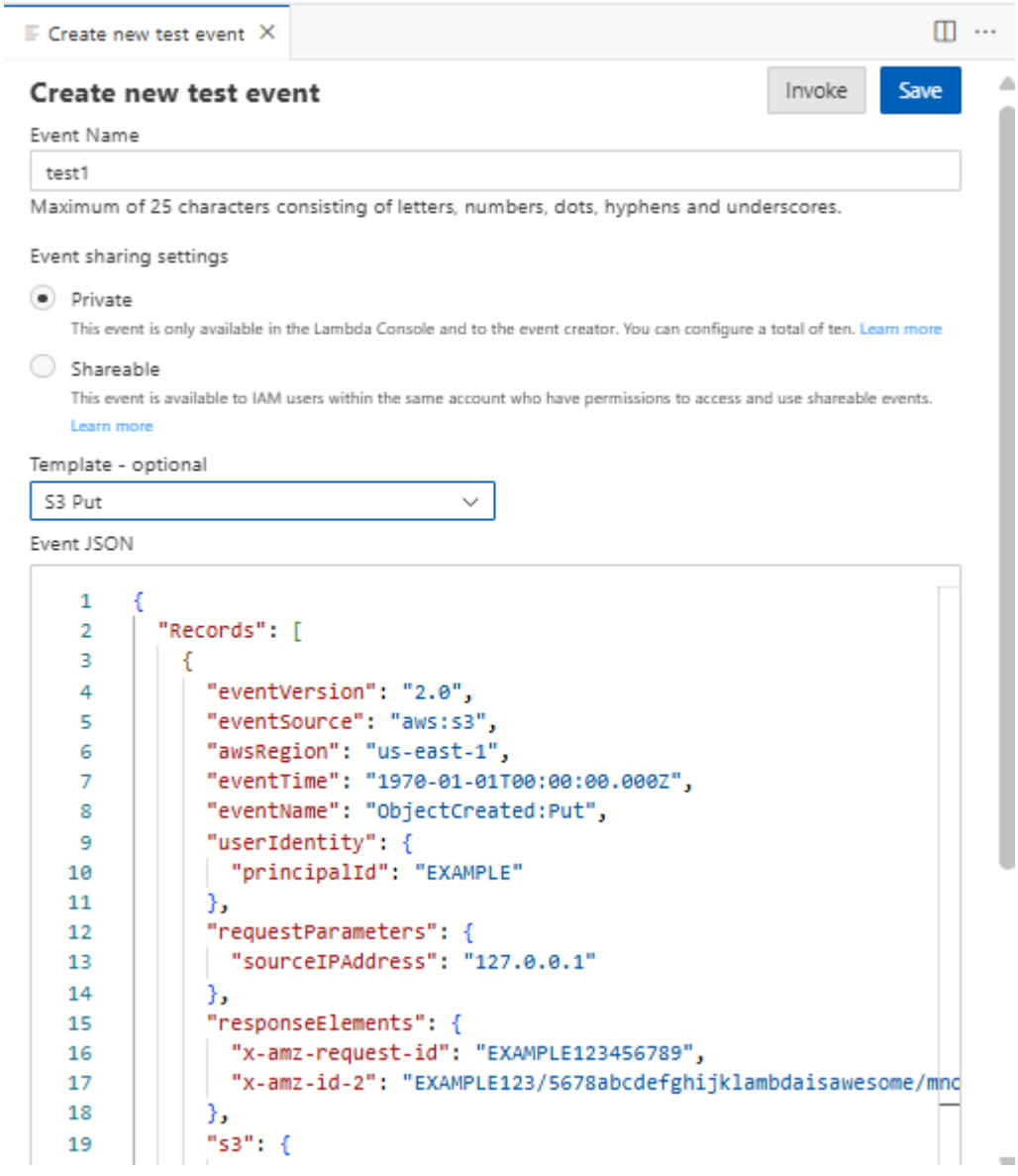


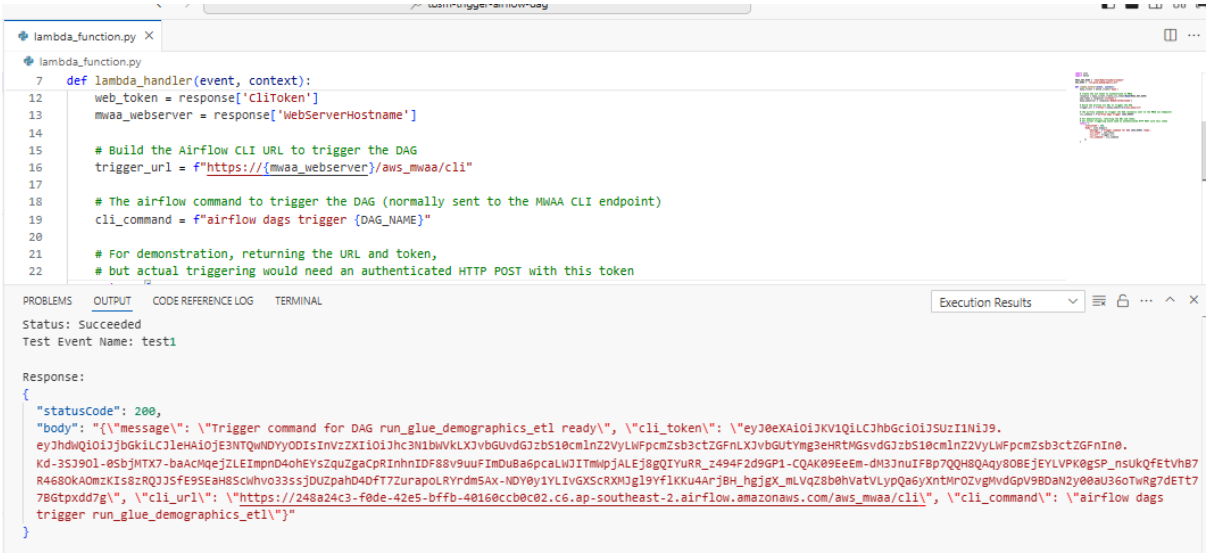



```
import json
import boto3
import urllib3
from datetime import datetime

MWAA_ENV_NAME = "tbsm-MyAirflowEnvironment"
DAG_NAME = "trigger_upsert_glue_job"

def lambda_handler(event, context):
  print("📦 Received S3 Event:")
  print(json.dumps(event, indent=2))

  mwaa = boto3.client('mwaa')

  try:
    # 1. Create MWAA CLI token
    resp = mwaa.create_cli_token(Name=MWAA_ENV_NAME)
    web_token = resp['CliToken']
    web_server_hostname = resp['WebServerHostname']
    print("🔐 CLI token and hostname retrieved successfully.")

    # 2. Generate Airflow CLI trigger command
    execution_date = datetime.utcnow().isoformat()
    cli_command = f"dags trigger -e {execution_date} {DAG_NAME}"
    trigger_url = f"https://{web_server_hostname}/aws_mwaa/cli"

    print(f"🚀 Triggering DAG: {DAG_NAME} at {execution_date}")

    # 3. Send request using urllib3
    http = urllib3.PoolManager()
    headers = {
      "Authorization": f"Bearer {web_token}",
```

```
        "Content-Type": "text/plain"
    }

    response = http.request(
        "POST",
        trigger_url,
        headers=headers,
        body=cli_command.encode("utf-8"),
        timeout=10.0
    )

    # 4. Handle response
    status = response.status
    response_body = response.data.decode("utf-8")

    print(f"🌐 HTTP Status: {status}")
    print(f"📜 Response Body:\n{response_body}")

    return {
        'statusCode': status,
        'body': json.dumps({
            'message': f"DAG trigger attempted with status {status}",
            'response': response_body
        })
    }

except Exception as e:
    print(f"❌ Exception occurred: {str(e)}")
    return {
        'statusCode': 500,
        'body': json.dumps({'error': str(e)})
    }
```


tbsm-trigger-airflow-dag
EXPLORER
TBSM-TRIGGER-AIRFLOW-DAG
lambda_function.py
DEPLOY [UNDEPLOYED CHANGES]
You have undeployed changes.
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [SELECTED: TEST1]
Create new test event
Private saved events
test1
lambda_function.py
import json
import boto3
import urllib3
from datetime import datetime
MWAA_ENV_NAME = "tbsm-MyAirflowEnvironment"
DAG_NAME = "trigger_upsert_glue_job"
def lambda_handler(event, context):
print("🔔 Received S3 Event:")
print(json.dumps(event, indent=2))
mwaa = boto3.client('mwaa')
try:
# 1. Create MWAA CLI token
resp = mwaa.create_cli_token(Name=MWAA_ENV_NAME)
web_token = resp['CliToken']
web_server_hostname = resp['WebServerHostname']
print("🔐 CLI token and hostname retrieved successfully.")
# 2. Generate Airflow CLI trigger command
execution_date = datetime.utcnow().isoformat()
cli_command = f"dags trigger -e {execution_date} {DAG_NAME}"
trigger_url = f"https://{web_server_hostname}/aws_mwaa/cli"
print(f"🚀 Triggering DAG: {DAG_NAME} at {execution_date}")
# 3. Send request using urllib3
http = urllib3.PoolManager()
headers = {
"Authorization": f"Bearer {web_token}",
"Content-Type": "text/plain"
}


Create new test event
Create new test event
Invoke
Save
Event Name
test1
Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.
Event sharing settings
Private
This event is only available in the Lambda Console and to the event creator. You can configure a total of ten. Learn more
Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events.
Learn more
Template - optional
S3 Put
Event JSON
{
"Records": [
{
"eventVersion": "2.0",
"eventSource": "aws:s3",
"awsRegion": "us-east-1",
"eventTime": "1970-01-01T00:00:00.000Z",
"eventName": "ObjectCreated:Put",
"userIdentity": {
"principalId": "EXAMPLE"
},
"requestParameters": {
"sourceIPAddress": "127.0.0.1"
},
"responseElements": {
"x-amz-request-id": "EXAMPLE123456789",
"x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnc
},
"s3": {

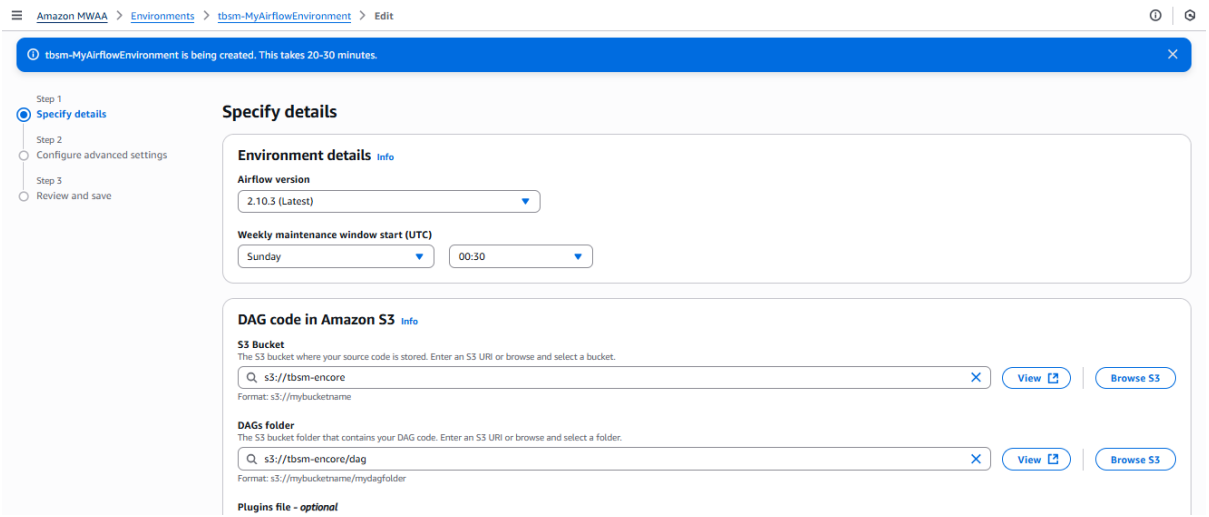


Step 5: Apache Airflow (MWAA) Setup

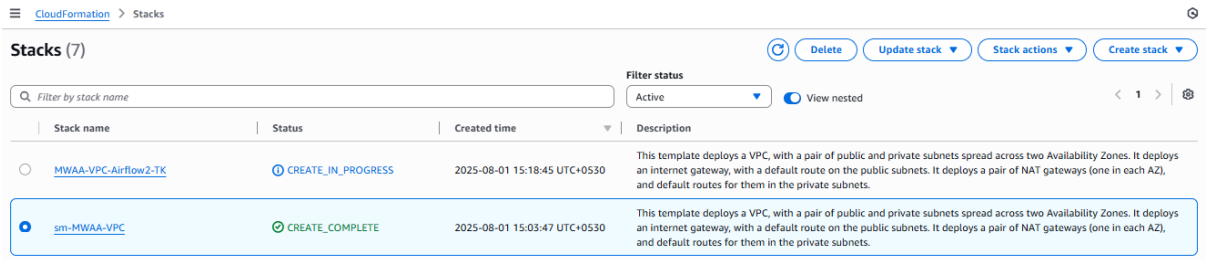
MWAA Environment

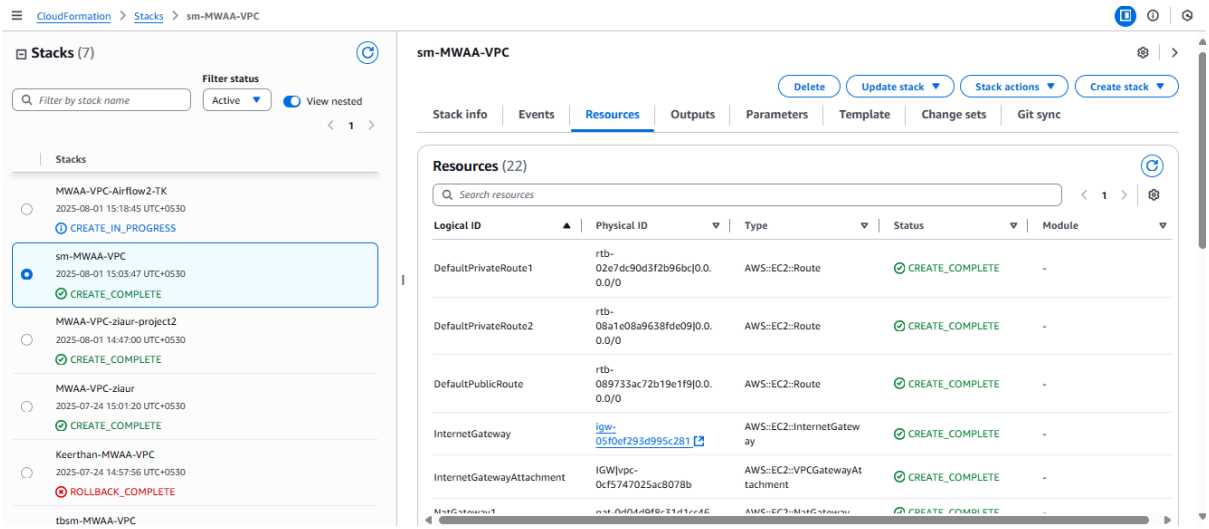
- **Create MWAA environment in VPC**
- **Attach S3 bucket for DAGs**

Upload DAG to /dags/ folder in MWAA S3 bucket

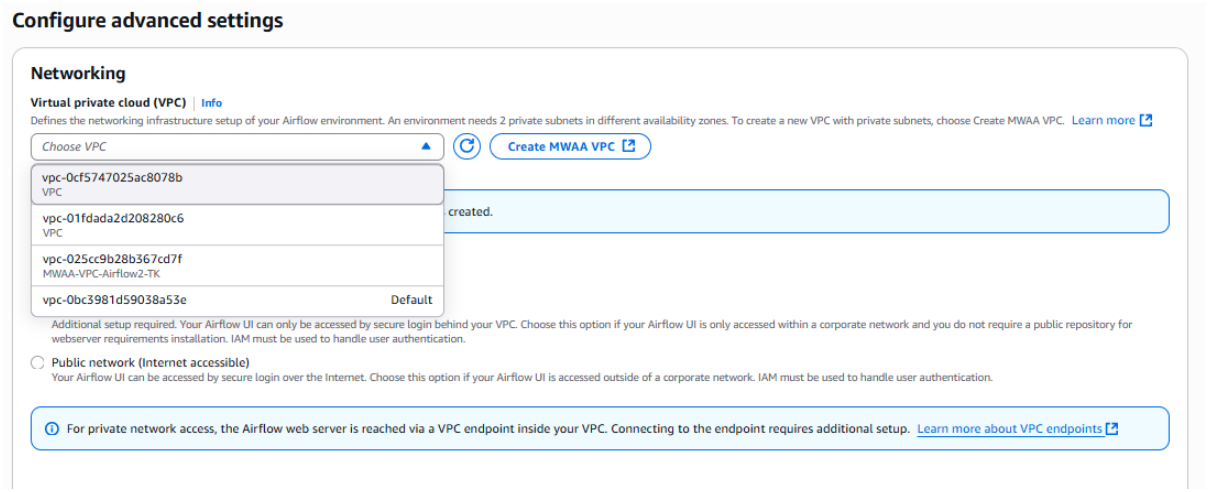


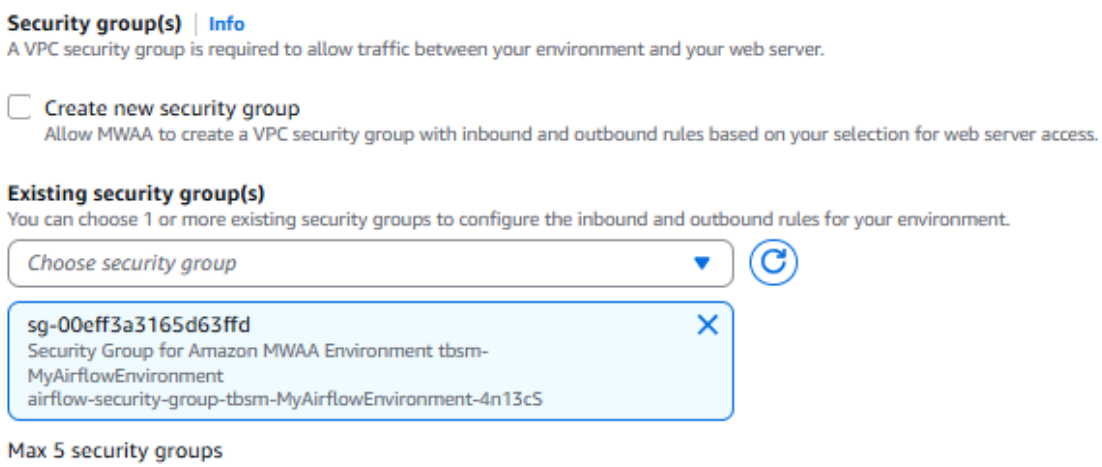
Need to create new MWAA vpc around it




CloudFormation > Stacks > sm-MWAA-VPC
Stacks (7)
Filter by stack name
Active
View nested
Stacks
MWAA-VPC-Airflow2-TK
2025-08-01 15:18:45 UTC+0530
CREATE_IN_PROGRESS
sm-MWAA-VPC
2025-08-01 15:03:47 UTC+0530
CREATE_COMPLETE
MWAA-VPC-ziaur-project2
2025-08-01 14:47:00 UTC+0530
CREATE_COMPLETE
MWAA-VPC-ziaur
2025-07-24 15:01:20 UTC+0530
CREATE_COMPLETE
Keerthan-MWAA-VPC
2025-07-24 14:57:56 UTC+0530
ROLLBACK_COMPLETE
tbsm-MWAA-VPC
sm-MWAA-VPC
Delete
Update stack
Stack actions
Create stack
Stack info
Events
Resources
Outputs
Parameters
Template
Change sets
Git sync
Resources (22)
Search resources
Logical ID
Physical ID
Type
Status
Module
DefaultPrivateRoute1
rtb-02e7dc90d3f2b96bc|0.0.0.0/0
AWS::EC2::Route
CREATE_COMPLETE
DefaultPrivateRoute2
rtb-08a1e08a9638fde09|0.0.0.0/0
AWS::EC2::Route
CREATE_COMPLETE
DefaultPublicRoute
rtb-089733ac72b19e1f9|0.0.0.0/0
AWS::EC2::Route
CREATE_COMPLETE
InternetGateway
igw-05f0ef293d995c281
AWS::EC2::InternetGateway
CREATE_COMPLETE
InternetGatewayAttachment
IGW|vpc-0cf5747025ac8078b
AWS::EC2::VPCGatewayAttachment
CREATE_COMPLETE

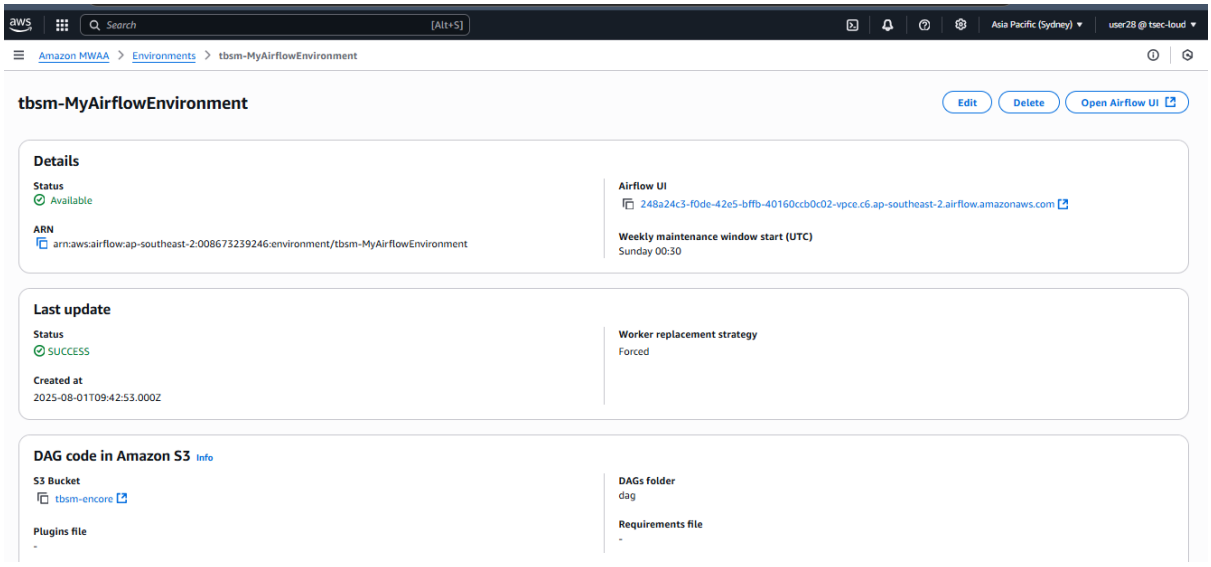
Select VPC

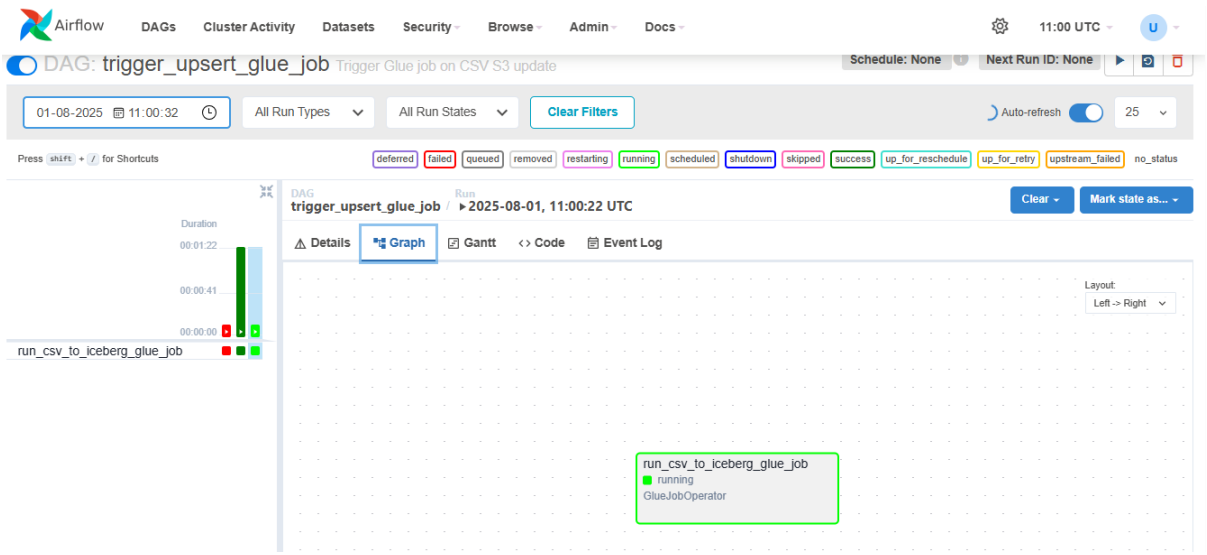

Configure advanced settings
Networking
Virtual private cloud (VPC) | Info
Defines the networking infrastructure setup of your Airflow environment. An environment needs 2 private subnets in different availability zones. To create a new VPC with private subnets, choose Create MWAA VPC. Learn more
Choose VPC
Create MWAA VPC
vpc-0cf5747025ac8078b
VPC
vpc-01fdada2d208280c6
VPC
vpc-025cc9b28b367cd7f
MWAA-VPC-Airflow2-TK
vpc-0bc3981d59038a53e
Default
created.
Additional setup required. Your Airflow UI can only be accessed by secure login behind your VPC. Choose this option if your Airflow UI is only accessed within a corporate network and you do not require a public repository for webserver requirements installation. IAM must be used to handle user authentication.
Public network (Internet accessible)
Your Airflow UI can be accessed by secure login over the Internet. Choose this option if your Airflow UI is accessed outside of a corporate network. IAM must be used to handle user authentication.
For private network access, the Airflow web server is reached via a VPC endpoint inside your VPC. Connecting to the endpoint requires additional setup. Learn more about VPC endpoints


Security group(s) | Info
A VPC security group is required to allow traffic between your environment and your web server.
Create new security group
Allow MWAA to create a VPC security group with inbound and outbound rules based on your selection for web server access.
Existing security group(s)
You can choose 1 or more existing security groups to configure the inbound and outbound rules for your environment.
Choose security group
sg-00eff3a3165d63ffd
Security Group for Amazon MWAA Environment tbsm-MyAirflowEnvironment
airflow-security-group-tbsm-MyAirflowEnvironment-4n13cS
Max 5 security groups

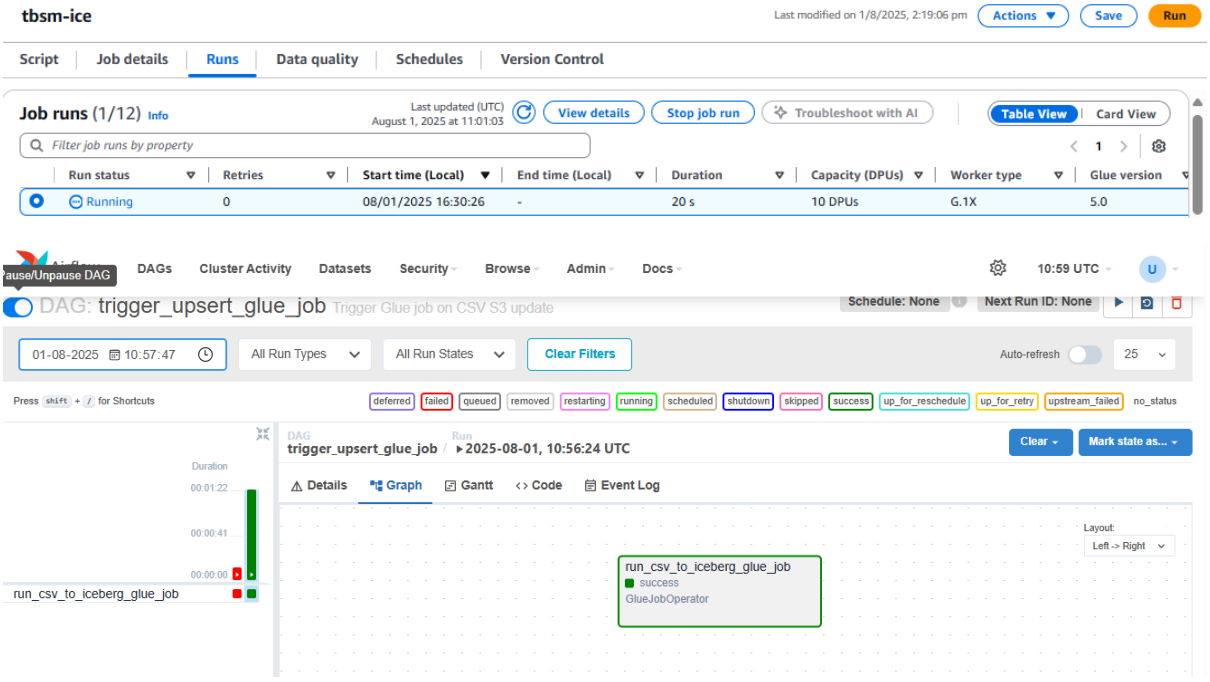
AIRLFOW DAG CODE

```
 GNU nano 8.3        run_glue_demographics_etl.py                from airflow import DAG
from airflow.providers.amazon.aws.operators.glue import AwsGlueJobOpera>from
datetime import datetime

with DAG(
   dag_id='run_glue_demographics_etl',
   schedule_interval=None,
   start_date=datetime(2023, 1, 1),
   catchup=False,
   tags=['iceberg', 'glue', 'etl']
) as dag:

   run_glue_job = AwsGlueJobOperator(
      task_id='run_iceberg_merge_job',
      job_name='tbsm-ice',
      script_location='s3://aws-glue-assets-008673239246-ap-southeast>
aws_conn_id='aws_default',
      region_name='ap-southeast-2'
   )
```

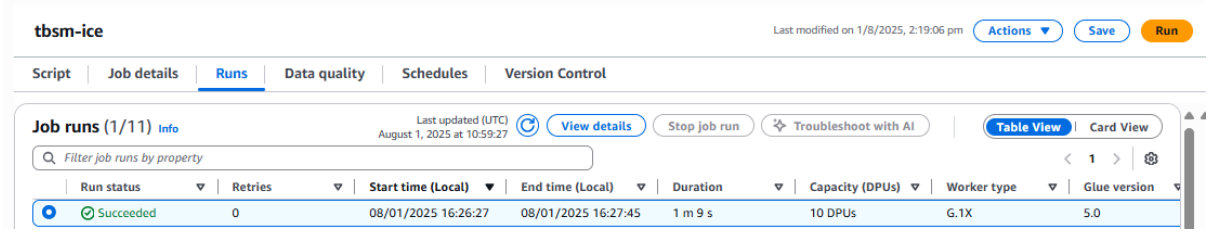




After running airflow it should trigger glue



After airflow successfully you can see glue job completed



Done with airflow and lambda

Step 6: End-to-End Flow Validation

1. **Login to EC2**
2. Modify or regenerate demographics.csv
3. Upload to S3: s3://tbsm-calm/ice/demographics.csv
4. S3 triggers **Lambda**
5. Lambda triggers **Airflow DAG**
6. Airflow runs **Glue Job**
7. Glue performs **MERGE into Iceberg table**
8. Iceberg table stored in: s3://tbsm-icebergtable/
9. Data available in Athena or Spark SQL

>>START HERE

-open ec2 login
-make changes to csv file
-old file

```
U32047,Male,<18,East,Premium
U30279,Other,46-60,Central,Pre
U36000,Female,46-60,North,Free
U33235,Male,36-45,West,Free
U13573,Male,60+,North,Premium
```

-new file

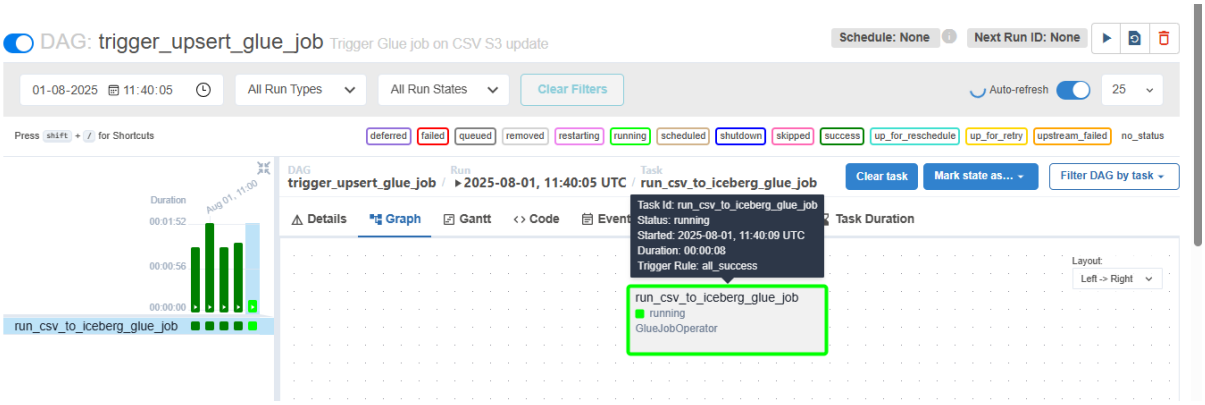
```
U32047,Male,<18,East,Premium
U30279,Other,46-60,Central,Pre
U36000,Female,46-60,North,Free
U33235,Male,36-45,West,Free
U13573,Male,60+,North,Premium
U42900,Other,18-25,West,Basic
```

Upload file

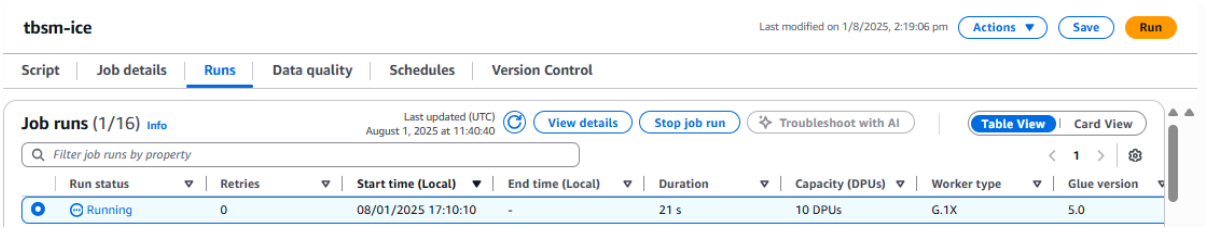
```
[ec2-user@ip-172-31-10-45 ~]$ aws s3 cp demographics.csv s3://tbsm-calm/ice/
upload: ./demographics.csv to s3://tbsm-calm/ice/demographics.csv
```

It will trigger the lambda

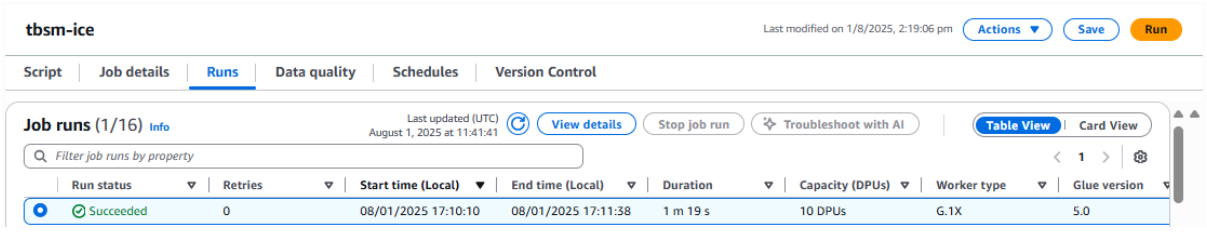
After that airflow will be trigger in running



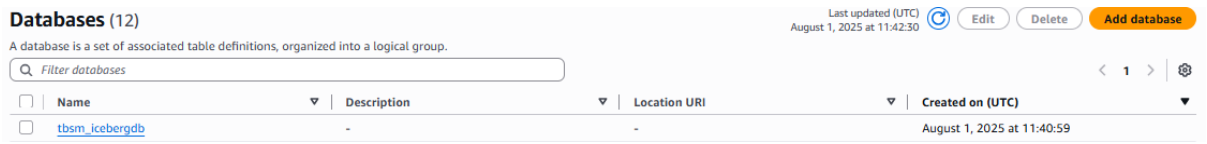
Which will trigger glue in running the job



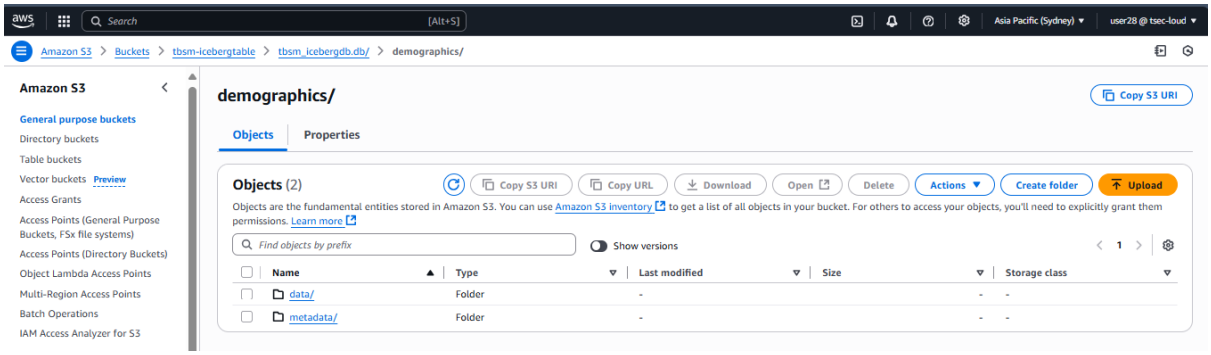
It will be successfully run



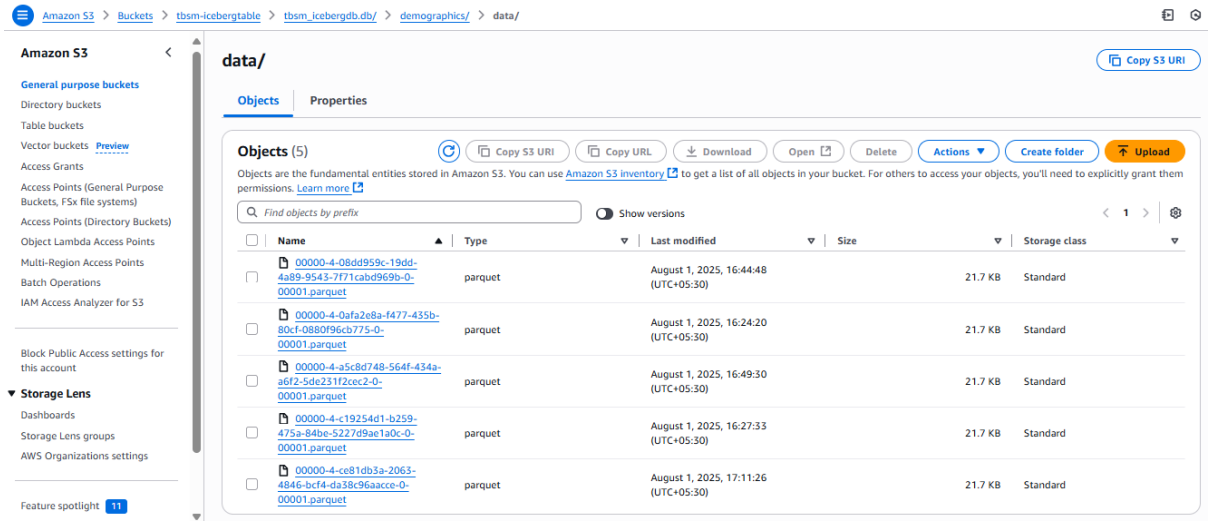
Then you can see table database created



And iceberg s3 is created wih folder named



And in that parquet



Real-Time Streaming Setup: Viewership Logs

🔁 **Data Flow:**

viewership_logs.csv → Python Streamer → Kinesis Data Stream

→ EMR Spark Streaming Job (PySpark)

→ Writes to S3 (raw + processed)

→ Writes to Snowflake (transformed)

Step 1: Kinesis Setup

Create Kinesis Stream

IAM Role for EC2

Attach a role with the following policies:

- AmazonKinesisFullAccess
- AmazonS3FullAccess
- CloudWatchLogsFullAccess (optional for logging)

- **2. Step 2: EC2 Setup for Streaming**
- Launch EC2 Instance
- AMI: Amazon Linux 2
- Security Group: Allow SSH (port 22)
- IAM Role: Attach the role created above
- **SSH into EC2**
- ssh -i your-key.pem ec2-user@your-ec2-public-ip
- Install Dependencies
- sudo yum update -y
- sudo yum install python3 -y
- pip3 install boto3
- Upload and Run Streamer Script
- nano generate_viewership_logs.py
- python3 generate

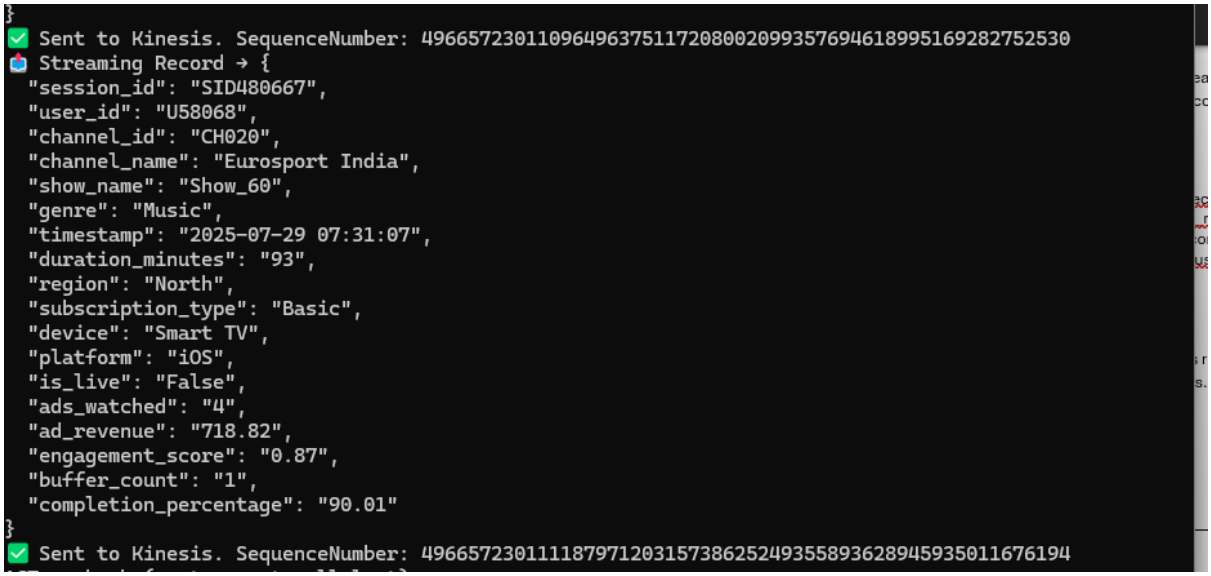
This script should:

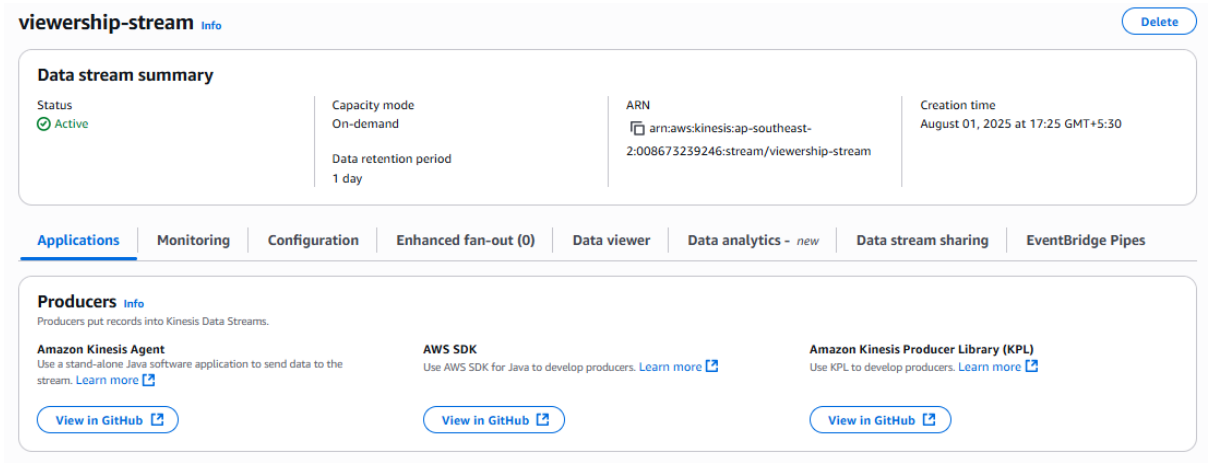
Generate JSON records

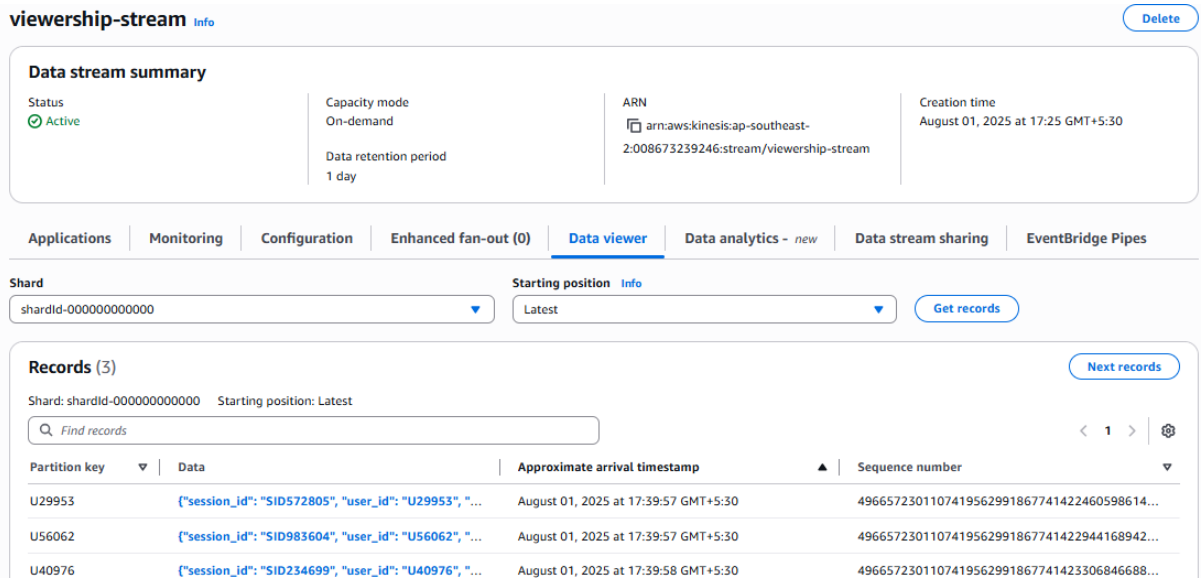
Send them to viewership-stream using boto3

```
import json
import boto3
import time

# Initialize Kinesis client
client = boto3.client('kinesis', region_name="ap-southeast-2")
stream_name = "viewership-stream"

# CSV Header (Must match order in the CSV file)
columns = [
  "session_id", "user_id", "channel_id", "channel_name", "show_name", "genre",
"timestamp",
  "duration_minutes", "region", "subscription_type", "device", "platform",
"is_live",
  "ads_watched", "ad_revenue", "engagement_score", "buffer_count",
"completion_percentage"
]

# Read and process CSV file
with open("viewership_logs.csv") as f:
  lines = f.readlines()[1:]  # Skip header

  for line in lines:
    # Convert line into dictionary
    values = line.strip().split(',')
    record = dict(zip(columns, values))

    # Print what is being streamed
    print("📤 Streaming Record →", json.dumps(record, indent=2))

    # Send to Kinesis
    response = client.put_record(
      StreamName=stream_name,
      Data=json.dumps(record),
      PartitionKey=record["user_id"]
    )

    # Optional: Print Kinesis response (Record ID)
    print("✅ Sent to Kinesis. SequenceNumber:", response['SequenceNumber'])
```

```
        # Simulate real-time
        time.sleep(0.01)
```







STEP 3: Snowflake Setup

SQL Setup

```
CREATE OR REPLACE DATABASE TBSM_DB;
```

```
CREATE OR REPLACE SCHEMA TBSM_DB.PUBLIC;
```

```
CREATE OR REPLACE WAREHOUSE COMPUTE_WH
WITH WAREHOUSE_SIZE = XSMALL
AUTO_SUSPEND = 300
AUTO_RESUME = TRUE
INITIALLY_SUSPENDED = TRUE;
```

```
CREATE OR REPLACE TABLE TBSM_DB.PUBLIC.VIEWERSHIP_LOGS (
    session_id STRING,
    user_id STRING,
    channel_id STRING,
    channel_name STRING,
    show_name STRING,
    genre STRING,
    timestamp STRING,
    duration_minutes STRING,
    region STRING,
    subscription_type STRING,
```

```
    device STRING,
    platform STRING,
    is_live STRING,
    ads_watched STRING,
    ad_revenue STRING,
    engagement_score STRING,
    buffer_count STRING,
    completion_percentage STRING
);
```

```
CREATE OR REPLACE USER spark_user
PASSWORD = 'StrongPassword@123'
DEFAULT_ROLE = ACCOUNTADMIN
DEFAULT_WAREHOUSE = COMPUTE_WH
DEFAULT_NAMESPACE = TBSM_DB.PUBLIC
MUST_CHANGE_PASSWORD = FALSE;
```

```
GRANT USAGE ON WAREHOUSE COMPUTE_WH TO USER spark_user;
GRANT USAGE ON DATABASE TBSM_DB TO USER spark_user;
GRANT USAGE ON SCHEMA TBSM_DB.PUBLIC TO USER spark_user;
GRANT INSERT, SELECT ON ALL TABLES IN SCHEMA TBSM_DB.PUBLIC TO USER
spark_user;
```

STEP 4: EMR Cluster Setup

✅ Create EMR Cluster

- Release: EMR 6.x
- Applications: Spark, Hadoop
- EC2 Key Pair: Use existing .pem
- Networking: Public subnet + Auto-assign Public IP
- IAM Role:
    - AmazonKinesisReadOnlyAccess
    - AmazonS3FullAccess

- Snowflake access via Spark connector

STEP 5: S3 Buckets

- Script Location: s3://tbsm-encore/dag/kinesis_to_snowflake_s3.py
- Temp Location: s3://tbsm-encore/dag/temp/

STEP 6: Install Spark Connectors on EMR

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr, split
from pyspark.sql.types import *

# 1. Define the schema of the CSV-style records
schema = StructType([
  StructField("session_id", StringType()),
  StructField("user_id", StringType()),
  StructField("channel_id", StringType()),
  StructField("channel_name", StringType()),
  StructField("show_name", StringType()),
  StructField("genre", StringType()),
  StructField("timestamp", StringType()),
  StructField("duration_minutes", IntegerType()),
  StructField("region", StringType()),
  StructField("subscription_type", StringType()),
  StructField("device", StringType()),
  StructField("platform", StringType()),
  StructField("is_live", StringType()),
  StructField("ads_watched", IntegerType()),
  StructField("ad_revenue", FloatType()),
  StructField("engagement_score", FloatType()),
  StructField("buffer_count", IntegerType()),
  StructField("completion_percentage", FloatType())
])

# 2. Initialize Spark Session
print("🚀 Initializing Spark session...")
spark = SparkSession.builder \
  .appName("KinesisToS3AndSnowflake") \
  .getOrCreate()

print("✅ Spark session initialized.")
```

```
# 3. Read from Kinesis
print("🔄 Connecting to Kinesis...")
df_raw = spark.readStream \
  .format("kinesis") \
  .option("streamName", "viewership-stream") \
  .option("endpointUrl", "https://kinesis.ap-southeast-2.amazonaws.com") \
  .option("region", "ap-southeast-2") \
  .option("startingPosition", "LATEST") \
  .load()
print("✅ Connected to Kinesis.")

# 4. Convert and split raw records
df_string = df_raw.withColumn("data_string", expr("CAST(data AS STRING)"))
df_split = df_string.withColumn("fields", split(col("data_string"), ","))

# 5. Assign to structured schema
df_parsed = df_split.select(
  col("fields").getItem(0).alias("session_id"),
  col("fields").getItem(1).alias("user_id"),
  col("fields").getItem(2).alias("channel_id"),
  col("fields").getItem(3).alias("channel_name"),
  col("fields").getItem(4).alias("show_name"),
  col("fields").getItem(5).alias("genre"),
  col("fields").getItem(6).alias("timestamp"),
  col("fields").getItem(7).cast("int").alias("duration_minutes"),
  col("fields").getItem(8).alias("region"),
  col("fields").getItem(9).alias("subscription_type"),
  col("fields").getItem(10).alias("device"),
  col("fields").getItem(11).alias("platform"),
  col("fields").getItem(12).alias("is_live"),
  col("fields").getItem(13).cast("int").alias("ads_watched"),
  col("fields").getItem(14).cast("float").alias("ad_revenue"),
  col("fields").getItem(15).cast("float").alias("engagement_score"),
  col("fields").getItem(16).cast("int").alias("buffer_count"),
  col("fields").getItem(17).cast("float").alias("completion_percentage")
)

# 6. Function to write batch to S3 and Snowflake
def write_batch(batch_df, epoch_id):
  print(f"🔁 Processing batch {epoch_id}...")

  record_count = batch_df.count()
  print(f"📦 Records in batch: {record_count}")

  if record_count == 0:
    print("⚠️ Empty batch. Skipping writes.")
    return
```

```
    try:
        # Snowflake options
        snowflake_options = {
            "sfURL": "vdhiuox-lib93136.snowflakecomputing.com",
            "sfUser": "YASHTEKAVADE",
            "sfPassword": "Yashvardhan@420",
            "sfDatabase": "TBSM_DB",
            "sfSchema": "PUBLIC",
            "sfWarehouse": "COMPUTE_WH",
            "sfRole": "ACCOUNTADMIN"
        }

        print("🌨️ Writing to Snowflake...")
        batch_df.write \
            .format("net.snowflake.spark.snowflake") \
            .options(**snowflake_options) \
            .option("dbtable", "VIEWERSHIP_LOGS") \
            .mode("append") \
            .save()
        print("✅ Write to Snowflake succeeded.")

    except Exception as e:
        print("❌ Error writing to Snowflake:", e)

    try:
        print("📩 Writing to S3...")
        batch_df.write \
            .format("csv") \
            .mode("append") \
            .save("s3://tbsm-encore/view/")
        print("✅ Write to S3 succeeded.")

    except Exception as e:
        print("❌ Error writing to S3:", e)

# 7. Start the streaming query
print("📡 Starting streaming query...")
query = df_parsed.writeStream \
    .foreachBatch(write_batch) \
    .outputMode("append") \
    .option("checkpointLocation", "s3://tbsm-encore/dag/") \
    .trigger(processingTime="10 seconds") \
    .start()

print("✅ Streaming started.")
```

```
query.awaitTermination()
```

Install all dependencies

sudo wget https://repo1.maven.org/maven2/net/snowflake/spark-snowflake_2.12/2.11.0-spark_3.3/spark-snowflake_2.12-2.11.0-spark_3.3.jar -P /usr/lib/spark/jars/

sudo wget https://repo1.maven.org/maven2/net/snowflake/snowflake-jdbc/3.13.30/snowflake-jdbc-3.13.30.jar -P /usr/lib/spark/jars/

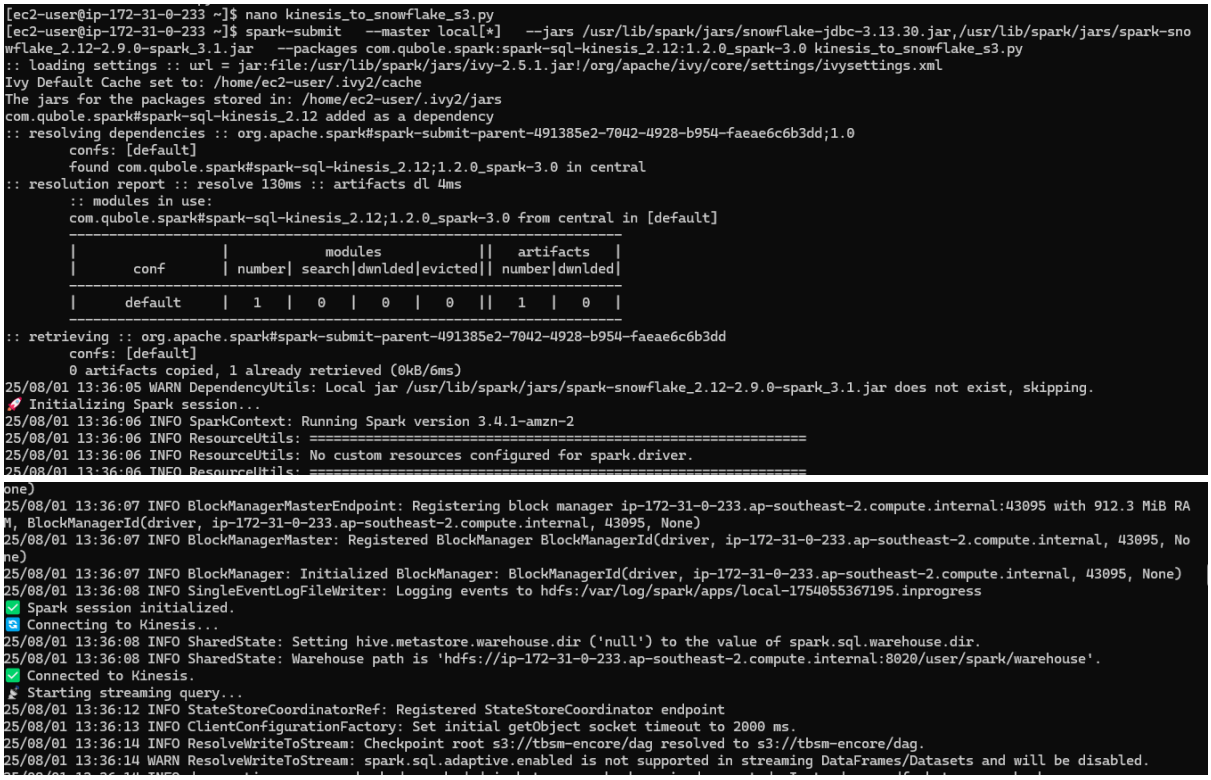
sudo wget https://awslabs-code-us-east-1.s3.amazonaws.com/spark-sql-kinesis-connector/spark-streaming-sql-kinesis-connector_2.12-1.2.1.jar -P /usr/lib/spark/jars/

sudo chmod 755 /usr/lib/spark/jars/spark-streaming-sql-kinesis-connector_2.12-1.2.1.jar

then run the file

spark-submit --master local[*] --jars /usr/lib/spark/jars/snowflake-jdbc-3.13.30.jar,/usr/lib/spark/jars/spark-snowflake_2.12-2.9.0-spark_3.1.jar --packages com.qubole.spark:spark-sql-kinesis_2.12:1.2.0_spark-3.0 kinesis_to_snowflake_s3.py

Screenshots



```
ata using algorithm version 1
25/08/01 13:36:15 INFO CheckpointFileManager: Renamed temp file s3://tbsm-encore/dag/.metadata.1da34b83-f07b-4d97-a93f-e0f5d975362a.tmp to s3://tbsm
-encore/dag/metadata
25/08/01 13:36:15 INFO MicroBatchExecution: Starting [id = 9d4bd1a7-4eec-4d2e-9266-41b64cadb15f, runId = 57a7f866-e2ef-41b4-8803-d99883f3d2c0]. Use
s3://tbsm-encore/dag to store the query checkpoint.
✅ Streaming started.
25/08/01 13:36:15 INFO MicroBatchExecution: Using Source [KinesisSource[viewership-stream]] from DataSourceV1 named 'kinesis' [DataSource(org.apache
.spark.sql.SparkSession@7d81617d,kinesis,List(),None,List(),None,Map(streamName -> viewership-stream, endpointUrl -> https://kinesis.ap-southeast-2.
amazonaws.com, region -> ap-southeast-2, startingPosition -> LATEST),None)]
25/08/01 13:36:15 INFO OffsetSeqLog: BatchIds found from listing:
25/08/01 13:36:15 INFO OffsetSeqLog: BatchIds found from listing:
25/08/01 13:36:15 INFO MicroBatchExecution: Starting new streaming query.
25/08/01 13:36:15 INFO MicroBatchExecution: Stream started from {}
25/08/01 13:36:15 INFO KinesisSource: This is the first batch. Returning Empty sequence
25/08/01 13:36:16 INFO KinesisReader: List shards in Kinesis Stream:  Buffer({ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,Ending
HashKey: 85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber: 49665723011074195629913717369258928093946445813681
```

```
000000000001,AT_TIMESTAMP,1754055375802), ShardInfo(shardId-000000000002,AT_TIMESTAMP,1754055375802), ShardInfo(shardId-000000000003,AT_TIMESTAMP,17
54055375802))
25/08/01 13:36:16 INFO KinesisSource: GetBatch generating RDD of offset range: ShardInfo(shardId-000000000000,AT_TIMESTAMP,1754055375802), ShardInfo
(shardId-000000000001,AT_TIMESTAMP,1754055375802), ShardInfo(shardId-000000000002,AT_TIMESTAMP,1754055375802), ShardInfo(shardId-000000000003,AT_TIM
ESTAMP,1754055375802)
25/08/01 13:36:18 INFO CodeGenerator: Code generated in 302.676248 ms
🔁 Processing batch 0...
25/08/01 13:36:18 INFO CodeGenerator: Code generated in 14.966234 ms
25/08/01 13:36:18 INFO CodeGenerator: Code generated in 14.890572 ms
25/08/01 13:36:18 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
25/08/01 13:36:18 INFO DAGScheduler: Registering RDD 5 (start at NativeMethodAccessorImpl.java:0) as input to shuffle 0
25/08/01 13:36:18 INFO DAGScheduler: Got job 0 (start at NativeMethodAccessorImpl.java:0) with 1 output partitions
25/08/01 13:36:18 INFO DAGScheduler: Final stage: ResultStage 1 (start at NativeMethodAccessorImpl.java:0)
25/08/01 13:36:18 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 0)
25/08/01 13:36:18 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 0)
25/08/01 13:36:18 INFO DAGScheduler: Submitting ShuffleMapStage 0 (MapPartitionsRDD[5] at start at NativeMethodAccessorImpl.java:0), which has no mi
ssing parents
25/08/01 13:36:18 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 157.3 KiB, free 912.1 MiB)
25/08/01 13:36:18 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 58.3 KiB, free 912.1 MiB)
```

```
25/08/01 13:36:20 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
25/08/01 13:36:20 INFO DAGScheduler: ResultStage 1 (start at NativeMethodAccessorImpl.java:0) finished in 0.123 s
25/08/01 13:36:20 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
25/08/01 13:36:20 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
25/08/01 13:36:20 INFO DAGScheduler: Job 0 finished: start at NativeMethodAccessorImpl.java:0, took 2.162825 s
📦 Records in batch: 163
☁️ Writing to Snowflake...
25/08/01 13:36:21 WARN SnowflakeConnectorUtils$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed
to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of
 Spark.
25/08/01 13:36:21 INFO CodeGenerator: Code generated in 21.067161 ms
25/08/01 13:36:21 WARN DefaultJDBCWrapper$: JDBC 3.13.30 is being used. But the certified JDBC version 3.13.22 is recommended.
25/08/01 13:36:21 INFO BlockManagerInfo: Removed broadcast_1_piece0 on ip-172-31-0-233.ap-southeast-2.compute.internal:43095 in memory (size: 6.2 Ki
B, free: 912.2 MiB)
25/08/01 13:36:23 INFO SparkConnectorContext$: Spark Connector system config: {
  "spark_connector_version" : "2.11.0",
  "spark_version" : "3.4.1-amzn-2",
  "application_name" : "KinesisToS3AndSnowflake",
  "scala_version" : "2.12.15",
  "java_version" : "1.8.0_452",
```

```
25/08/01 13:36:30 INFO StageWriter$: Succeed to write in 3.10 seconds at 2025-08-01T13:36:30.158
25/08/01 13:36:30 INFO StageWriter$: Spark Connector Master: Total job time is 5.29 seconds including read & upload time: 1.57 seconds and COPY time
: 3.73 seconds.
25/08/01 13:36:30 WARN SnowflakeConnectorUtils$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed
to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of
 Spark.
25/08/01 13:36:30 WARN DefaultJDBCWrapper$: JDBC 3.13.30 is being used. But the certified JDBC version 3.13.22 is recommended.
25/08/01 13:36:31 INFO SnowflakeSQLStatement: Spark Connector Master: execute query with bind variable: select * from VIEWERSHIP_LOGS where 1 = 0
✅ Write to Snowflake succeeded.
🪣 Writing to S3...
25/08/01 13:36:31 INFO PathOutputCommitterFactory: No output committer factory defined, defaulting to class org.apache.hadoop.mapreduce.lib.output.F
ileSystemOptimizedOutputCommitterFactory
25/08/01 13:36:31 INFO FileSystemOptimizedOutputCommitterFactory: EMR Optimized Committer: ENABLED
25/08/01 13:36:31 INFO FileOutputCommitter: File Output Committer Algorithm version is 2
25/08/01 13:36:31 INFO FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failure
s: true
25/08/01 13:36:31 INFO SQLConfCommitterProvider: Using output committer class org.apache.hadoop.mapreduce.lib.output.FileSystemOptimizedCommitter
25/08/01 13:36:31 INFO FileSystemOptimizedCommitter: Nothing to setup as successful task attempt outputs are written directly
25/08/01 13:36:31 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
25/08/01 13:36:31 INFO DAGScheduler: Got job 2 (start at NativeMethodAccessorImpl.java:0) with 4 output partitions
```

```
25/08/01 13:36:32 INFO FileFormatWriter: Write Job dbc3aaf3-f142-4f3f-a019-fb275b78b40e committed. Elapsed time: 64 ms.
25/08/01 13:36:32 INFO FileFormatWriter: Finished processing stats for write job dbc3aaf3-f142-4f3f-a019-fb275b78b40e.
✅ Write to S3 succeeded.
25/08/01 13:36:32 INFO CheckpointFileManager: Writing atomically to s3://tbsm-encore/dag/commits/0 using temp file s3://tbsm-encore/dag/commits/.0.6
07a15f1-0202-4d99-a790-e235ba185a59.tmp
25/08/01 13:36:32 INFO MultipartUploadOutputStream: close closed:false s3://tbsm-encore/dag/commits/.0.607a15f1-0202-4d99-a790-e235ba185a59.tmp
25/08/01 13:36:33 INFO S3NativeFileSystem: rename s3://tbsm-encore/dag/commits/.0.607a15f1-0202-4d99-a790-e235ba185a59.tmp s3://tbsm-encore/dag/comm
its/0 using algorithm version 1
25/08/01 13:36:33 INFO CheckpointFileManager: Renamed temp file s3://tbsm-encore/dag/commits/.0.607a15f1-0202-4d99-a790-e235ba185a59.tmp to s3://tbs
m-encore/dag/commits/0
25/08/01 13:36:33 INFO MicroBatchExecution: Streaming query made progress: {
  "id" : "9d4bd1a7-4eec-4d2e-9266-41b64cadb15f",
  "runId" : "57a7f866-e2ef-41b4-8803-d99883f3d2c0",
  "name" : null,
  "timestamp" : "2025-08-01T13:36:15.825Z",
  "batchId" : 0,
  "numInputRows" : 1426,
  "inputRowsPerSecond" : 0.0,
  "processedRowsPerSecond" : 82.13812568400438,
  "durationMs" : {
    "addBatch" : 15125,
    "commitOffsets" : 341,
    "getBatch" : 128,
    "getOffset" : 538,
    "queryPlanning" : 733,
    "triggerExecution" : 17360,
    "walCommit" : 370
  },
```

```
25/08/01 13:36:34 INFO TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool
25/08/01 13:36:34 INFO DAGScheduler: ResultStage 5 (start at NativeMethodAccessorImpl.java:0) finished in 0.022 s
25/08/01 13:36:34 INFO DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
25/08/01 13:36:34 INFO TaskSchedulerImpl: Killing all running tasks in stage 5: Stage finished
25/08/01 13:36:34 INFO DAGScheduler: Job 3 finished: start at NativeMethodAccessorImpl.java:0, took 0.385827 s
📦 Records in batch: 98
❄️ Writing to Snowflake...
25/08/01 13:36:34 WARN SnowflakeConnectorUtils$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed
to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of
 Spark.
25/08/01 13:36:34 WARN DefaultJDBCWrapper$: JDBC 3.13.30 is being used. But the certified JDBC version 3.13.22 is recommended.
25/08/01 13:36:34 INFO SnowflakeSQLStatement: Spark Connector Master: execute query with bind variable: alter session set timezone = 'UTC' , timesta
mp_ntz_output_format = 'YYYY-MM-DD HH24:MI:SS.FF3', timestamp_ltz_output_format = 'TZHTZM YYYY-MM-DD HH24:MI:SS.FF3', timestamp_tz_output_format = '
TZHTZM YYYY-MM-DD HH24:MI:SS.FF3' ;
25/08/01 13:36:35 INFO SnowflakeSQLStatement: Spark Connector Master: execute query with bind variable: create  temporary stage if not exists identi
fier(?)
25/08/01 13:36:36 INFO CloudStorageOperations$: Spark Connector Master: Begin to process and upload data for 4 partitions: directory=WvLCncThFI CSV
true
25/08/01 13:36:36 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
25/08/01 13:36:36 INFO DAGScheduler: Got job 4 (start at NativeMethodAccessorImpl.java:0) with 4 output partitions
25/08/01 13:36:36 INFO DAGScheduler: Final stage: ResultStage 6 (start at NativeMethodAccessorImpl.java:0)
```

```
25/08/01 13:36:39 INFO StageWriter$: Succeed to write in 2.19 seconds at 2025-08-01T13:36:39.486
25/08/01 13:36:39 INFO StageWriter$: Spark Connector Master: Total job time is 4.06 seconds including read & upload time: 1.22 seconds and COPY time
: 2.84 seconds.
25/08/01 13:36:40 WARN SnowflakeConnectorUtils$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed
to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of
 Spark.
25/08/01 13:36:40 WARN DefaultJDBCWrapper$: JDBC 3.13.30 is being used. But the certified JDBC version 3.13.22 is recommended.
25/08/01 13:36:40 INFO SnowflakeSQLStatement: Spark Connector Master: execute query with bind variable: select * from VIEWERSHIP_LOGS where 1 = 0
✅ Write to Snowflake succeeded.
🪣 Writing to S3...
25/08/01 13:36:40 INFO PathOutputCommitterFactory: No output committer factory defined, defaulting to class org.apache.hadoop.mapreduce.lib.output.F
ileSystemOptimizedOutputCommitterFactory
25/08/01 13:36:40 INFO FileSystemOptimizedOutputCommitterFactory: EMR Optimized Committer: ENABLED
25/08/01 13:36:40 INFO FileOutputCommitter: File Output Committer Algorithm version is 2
25/08/01 13:36:40 INFO FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failure
s: true
25/08/01 13:36:40 INFO SQLConfCommitterProvider: Using output committer class org.apache.hadoop.mapreduce.lib.output.FileSystemOptimizedCommitter
25/08/01 13:36:40 INFO FileSystemOptimizedCommitter: Nothing to setup as successful task attempt outputs are written directly
25/08/01 13:36:40 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
```

STEP 8: Validate Output

- **Check Snowflake Table**: TBSM_DB.PUBLIC.VIEWERSHIP_LOGS
- **Check S3 Output**: s3://tbsm-encore/dag/temp/
- **Monitor EMR Logs**: via EMR Console or CloudWatch