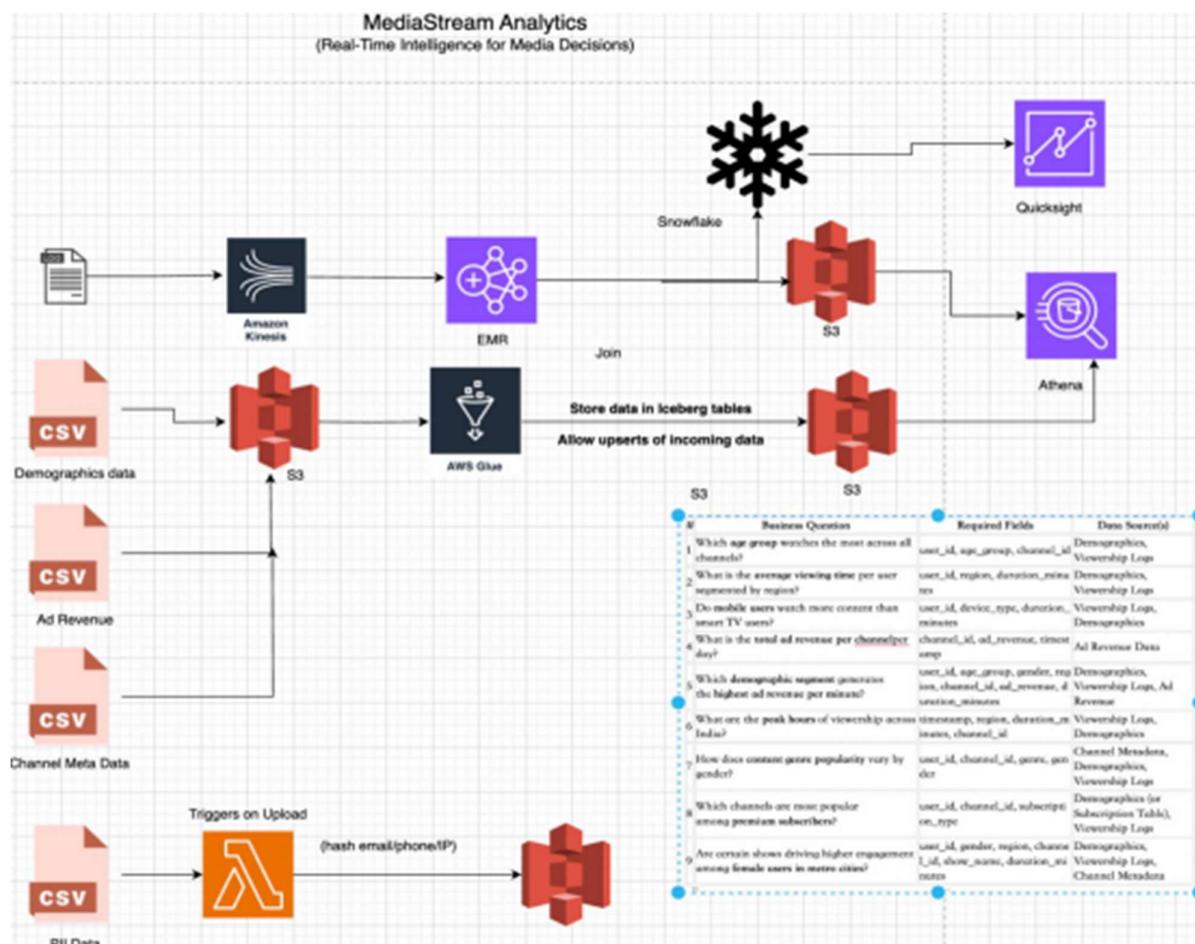


## MediaStream Analytics Project

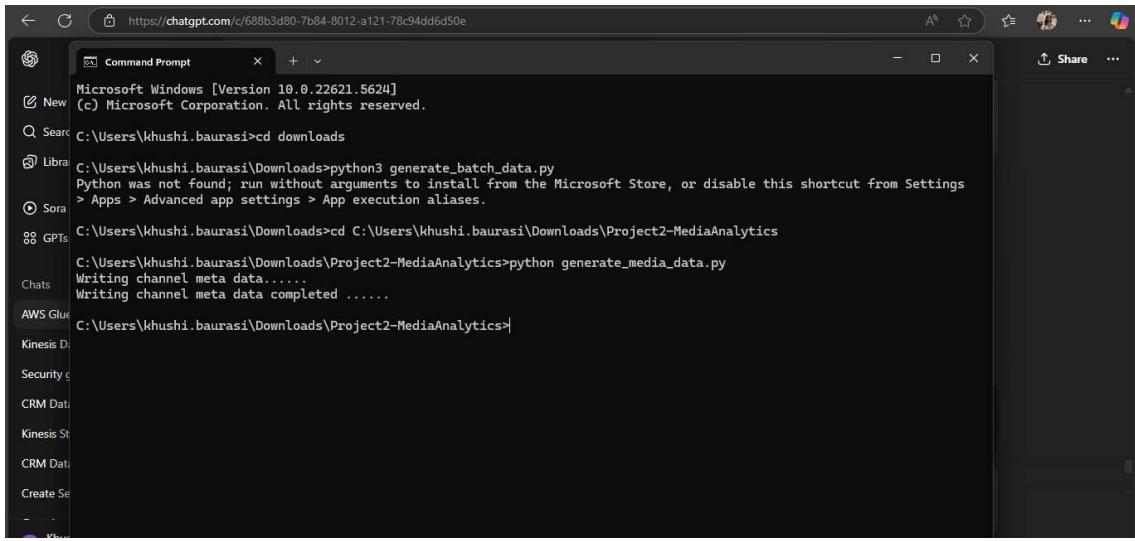
Name: Khushi Baurasi

Data Engineering Bootcamp 2025

Submit Date-04/08/25



locally on my machine then uploaded it on my S3 bucket

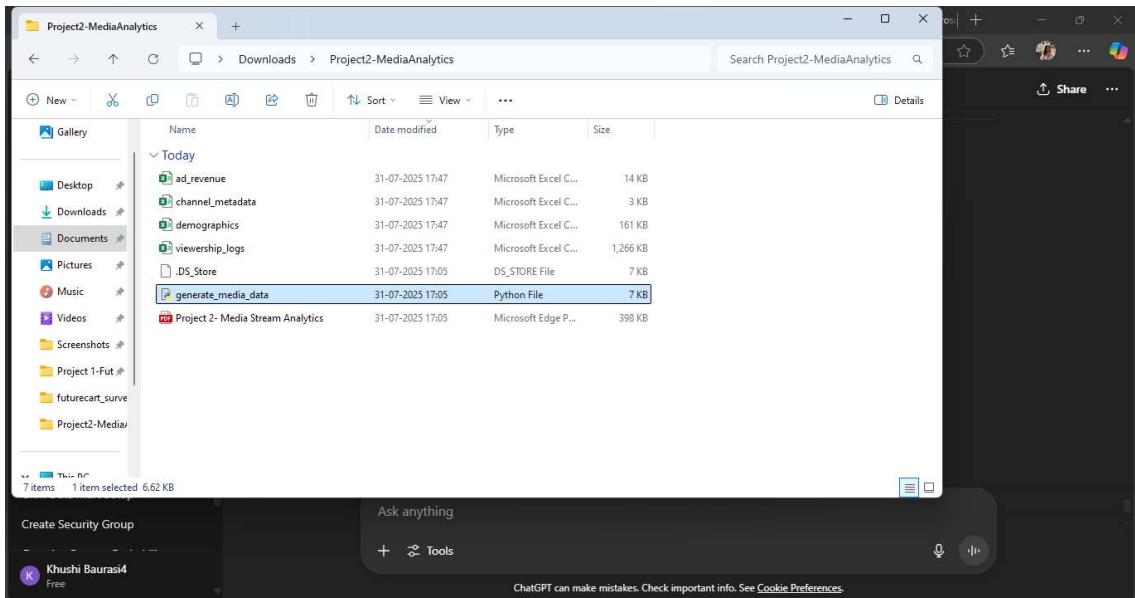


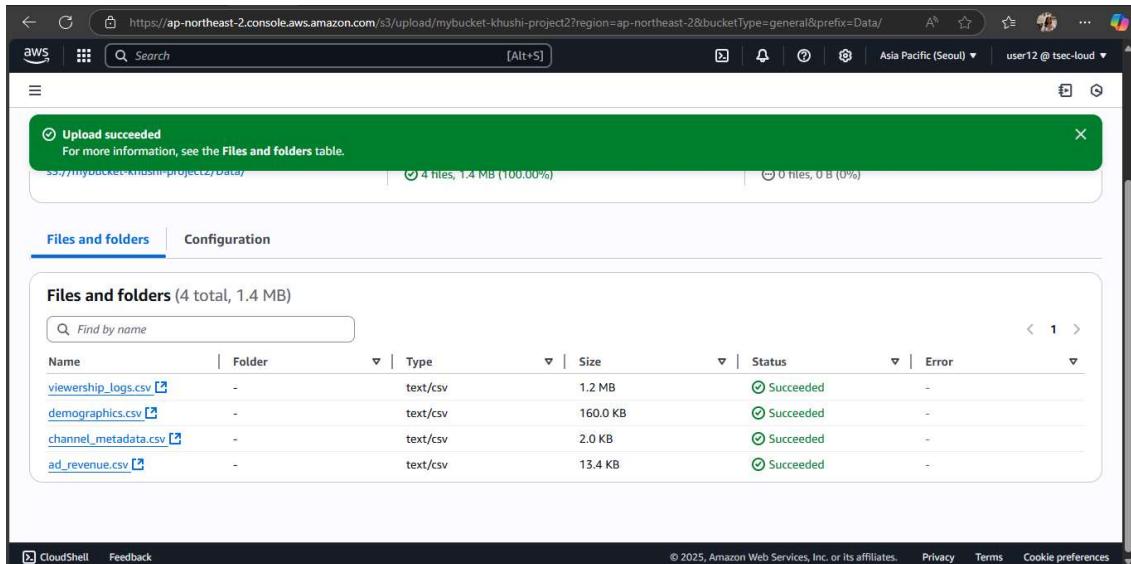
```
Microsoft Windows [Version 10.0.22621.5624]
(c) Microsoft Corporation. All rights reserved.

C:\Users\khushi.baurasi>cd downloads

C:\Users\khushi.baurasi\Downloads>python3 generate_batch_data.py
Python was not found; run without arguments to install from the Microsoft Store, or disable this shortcut from Settings
> Apps > Advanced app settings > App execution aliases.

C:\Users\khushi.baurasi\Downloads>cd C:\Users\khushi.baurasi\Downloads\Project2-MediaAnalytics
C:\Users\khushi.baurasi\Downloads\Project2-MediaAnalytics>python generate_media_data.py
Writing channel meta data.....
Writing channel meta data completed .....
```





## NEXT STEP: Process the CSVs in Glue Job

### 1. Create Tables using a Glue job.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date

spark = SparkSession.builder \
    .appName("Write Iceberg Tables") \
    .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://mybucket-khushi-project2/Output/") \
    .config("spark.sql.catalog.glue_catalog.io-impl", "org.apache.iceberg.aws.s3.S3FileIO") \
    .getOrCreate()

# Drop and recreate to be sure
spark.sql("DROP TABLE IF EXISTS glue_catalog.mediadata_db.ad_revenue")
spark.sql("DROP TABLE IF EXISTS glue_catalog.mediadata_db.channel_metadata")
spark.sql("DROP TABLE IF EXISTS glue_catalog.mediadata_db.demographics")

spark.sql("CREATE DATABASE IF NOT EXISTS glue_catalog.mediadata_db")

spark.sql("""
CREATE TABLE glue_catalog.mediadata_db.ad_revenue (

```

```

    channel_id STRING,
    channel_name STRING,
    date DATE,
    ad_revenue DOUBLE
)
USING ICEBERG
PARTITIONED BY (date)
""")

spark.sql("""
CREATE TABLE glue_catalog.mediadata_db.channel_metadata (
    channel_id STRING,
    channel_name STRING,
    genre STRING,
    language STRING,
    launch_year INT
)
USING ICEBERG
""")

spark.sql("""
CREATE TABLE glue_catalog.mediadata_db.demographics (
    user_id STRING,
    gender STRING,
    age_group STRING,
    region STRING,
    subscription_type STRING
)
USING ICEBERG
""")

ad_revenue_df = spark.read.option("header", True).csv("s3://mybucket-khushi-project2/Data/ad_revenue.csv")
ad_revenue_df = ad_revenue_df \
    .withColumn("channel_id", col("channel_id").cast("string")) \
    .withColumn("channel_name", col("channel_name").cast("string")) \
    .withColumn("date", to_date("date", "yyyy-MM-dd"))

```

```

.withColumn("ad_revenue", col("ad_revenue").cast("double"))

channel_metadata_df = spark.read.option("header", True).csv("s3://mybucket-khushi-project2/Data/channel_metadata.csv")

channel_metadata_df = channel_metadata_df \
    .withColumn("channel_id", col("channel_id").cast("string")) \
    .withColumn("channel_name", col("channel_name").cast("string")) \
    .withColumn("genre", col("genre").cast("string")) \
    .withColumn("language", col("language").cast("string")) \
    .withColumn("launch_year", col("launch_year").cast("int"))

demographics_df = spark.read.option("header", True).csv("s3://mybucket-khushi-project2/Data/demographics.csv")

demographics_df = demographics_df \
    .withColumn("user_id", col("user_id").cast("string")) \
    .withColumn("gender", col("gender").cast("string")) \
    .withColumn("age_group", col("age_group").cast("string")) \
    .withColumn("region", col("region").cast("string")) \
    .withColumn("subscription_type", col("subscription_type").cast("string"))

ad_revenue_df.writeTo("glue_catalog.mediadata_db.ad_revenue").append()
channel_metadata_df.writeTo("glue_catalog.mediadata_db.channel_metadata").append()
demographics_df.writeTo("glue_catalog.mediadata_db.demographics").append()

print("✅ All tables written to Iceberg successfully.")

```

The screenshot shows the AWS Glue Studio interface. On the left, there's a sidebar with 'AWS Glue' navigation, including 'ETL jobs', 'Data Catalog', and 'Data Integration and ETL'. The main area is titled 'Ad\_Revenue-Khushi job' and shows a 'Script' tab. The script content is as follows:

```

1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import col, to_date
3
4  spark = SparkSession.builder \
5      .appName("Write Iceberg Tables") \
6      .config("spark.sql.catalog.glue_catalog", "org.apache.iceberg.spark.SparkCatalog") \
7      .config("spark.sql.catalog.glue_catalog.catalog-impl", "org.apache.iceberg.aws.glue.GlueCatalog") \
8      .config("spark.sql.catalog.glue_catalog.warehouse", "s3://mybucket-khushi-project2/Output/") \
9      .config("spark.sql.catalog.glue_catalog.io-impl", "org.apache.iceberg.aws.s3.S3FileIO") \
10     .getOrCreate()
11
12 # Drop and recreate to be sure
13 spark.sql("DROP TABLE IF EXISTS glue_catalog.mediadata_db.ad_revenue")
14 spark.sql("DROP TABLE IF EXISTS glue_catalog.mediadata_db.channel_metadata")

```

At the bottom of the script editor, it says 'Python Ln 1, Col 1 Errors: 0 Warnings: 0'. The browser address bar at the top shows the URL: https://ap-northeast-2.console.aws.amazon.com/gluestudio/home?region=ap-northeast-2#/editor/job/Ad\_Revenue-Khushi%20job/script.

Now my tables are created:

The screenshot shows the AWS Glue Data Catalog Tables page. The left sidebar has sections for Getting started, ETL jobs, Data Catalog tables, Data Catalog, and Data Integration and ETL. The main area displays a table titled 'Tables (3/20)' with columns: Name, Database, Location, Classification, Deprecation, View data, and Data quality. The table contains three rows, each with a checkbox, a name, a database name, a location, and a CSV file type. The rows are: ad\_revenue (mediadata\_db, s3://mybucket-khushi-project2/Outp), channel\_metadata (mediadata\_db, s3://mybucket-khushi-project2/Outp), and demographics (mediadata\_db, s3://mybucket-khushi-project2/Outp). The 'ad\_revenue' row is selected, indicated by a blue border around its row.

Name	Database	Location	Classification	Deprecation	View data	Data quality
ad_revenue	mediadata_db	s3://mybucket-khushi-project2/Outp	-	-	Table data	View data qualit
channel_metadata	mediadata_db	s3://mybucket-khushi-project2/Outp	-	-	Table data	View data qualit
demographics	mediadata_db	s3://mybucket-khushi-project2/Outp	-	-	Table data	View data qualit

### Preview Data in Athena-

1. Go to **Athena Console**
2. Choose database: mediadata\_db

Now I can see my data in Athena-

The screenshot shows the AWS Glue Console Query editor interface. On the left, there's a sidebar for 'Data' with 'Data source' set to 'AwsDataCatalog', 'Catalog' set to 'None', and 'Database' set to 'mediadata\_db'. Under 'Tables and views', there are three tables listed: 'ad\_revenue', 'channel\_metadata', and 'demographics'. The 'ad\_revenue' table has columns: channel\_id (string), channel\_name (string), date (date), and ad\_revenue (double). The 'channel\_metadata' table has columns: channel\_id (string), channel\_name (string), genre (string), language (string), and launch\_year (date). The 'demographics' table has columns: gender (string), age\_group (string), and count (integer). In the main area, three queries are listed:

```

1 select * from ad_revenue limit 5;
2 select * from channel_metadata limit 5;
3 select * from demographics limit 5;

```

The first query, 'select \* from ad\_revenue limit 5;', is marked as 'Completed' with a green status bar. The results table shows two rows of data:

#	channel_id	channel_name	genre	language	launch_year
1	CH001	Star Plus	Entertainment	Hindi	2014
2	CH002	Colors TV	Entertainment	Hindi	2016

SELECT \* FROM ad\_revenue LIMIT 5;

SELECT \* FROM channel\_metadata LIMIT 5;

SELECT \* FROM demographics LIMIT 5;

Lambda → Triggers Airflow DAG → Airflow runs AWS Glue Job

## 1. Created MWAA (Amazon Managed Workflows for Apache Airflow) Environment

- Environment Name: Khushi-MWAA-P2
- Airflow version: (likely default, e.g. 2.9.0)
- Environment class: mw1.small
- Logging: CloudWatch task logs enabled
- Status: Now ACTIVE or STARTING

The screenshot shows the AWS MWAA console with the URL <https://ap-northeast-2.console.aws.amazon.com/mwaa/home?region=ap-northeast-2#environments>. The page title is "Airflow environments". A blue banner at the top indicates "Environment Khushi-MWAA-P2 is updating.". Below is a table titled "Environments (2)" with columns: Name, Status, Created date, Airflow version, and Airflow UI. Two environments are listed: "MyAirflowEnvironmentGP" (Creating, Aug 01, 2025 15:48:52, 2.10.3, Open Airflow UI) and "Khushi-MWAA-P2" (Updating, Aug 01, 2025 14:56:21, 2.10.3, Open Airflow UI).

## 2. Networking: VPC Configuration

- VPC: vpc-02453d5b657dd8785
- Subnets:
  - subnet-01ed01e62ee252289
  - subnet-06a71efafc6461f3f
- Security Group: sg-0d03796aba8449471

The screenshot shows the AWS CloudFormation console with the URL <https://ap-northeast-2.console.aws.amazon.com/cloudformation/home?region=ap-northeast-2#/stacks/events?stackId=arn%3Aaws%3Acloudformatio...>. The left sidebar shows "Stacks (4)" with "MWAA-VPC" selected. The main panel is titled "MWAA-VPC" and shows the "Events" tab. It displays 77 events, with the most recent ones being "CREATE\_COMPLETE" for the stack and its resources. The table includes columns for Timestamp, Logical ID, Status, and Details.

- Access Type: Public (can open Airflow UI via browser)

### **3. IAM Role (Execution Role for MWAA)**

- Role Name: AWSGlueServiceRole-S3-Khushi
  - Trust policy updated to allow MWAA to assume the role
    - Principal: "Service": "airflow-env.amazonaws.com"
- 

### **4. S3 Bucket Setup for MWAA**

- Bucket Name: mybucket-khushi-project2
  - MWAA paths configured:
    - DAGs folder: Dags/
    - requirements.txt: in root of the bucket
- 

### **5. DAG File Created**

I created a DAG file named ad\_revenue\_glue\_dag.py with content like:

```
from airflow import DAG
from airflow.providers.amazon.aws.operators.glue import GlueJobOperator
from datetime import datetime

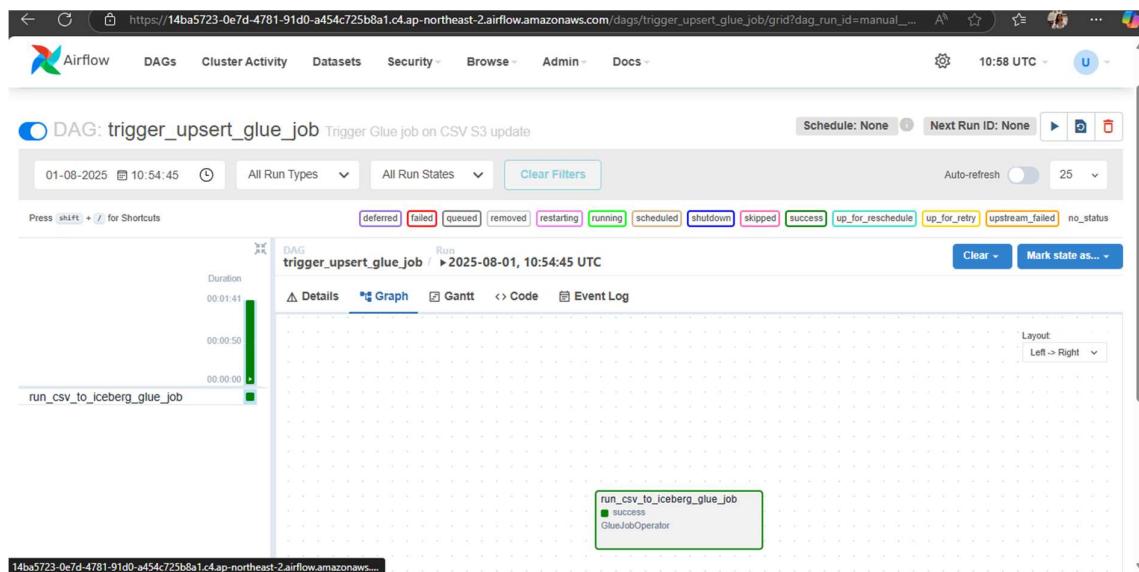
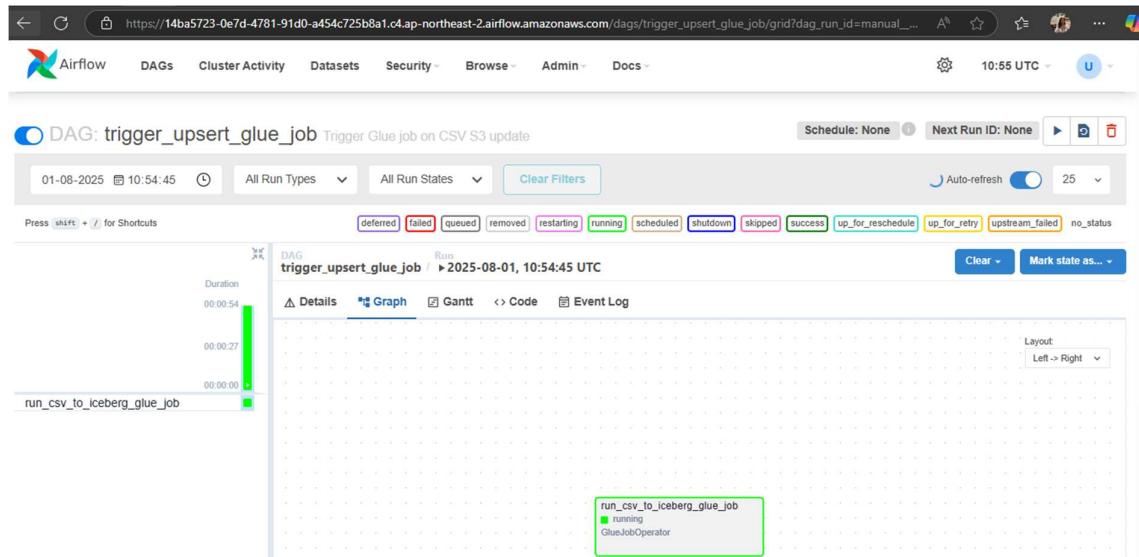
# Replace with your values
GLUE_JOB_NAME = "Khushi-P2-job" # <- Your Glue job name
AWS_REGION = "ap-northeast-2" # e.g., Mumbai
DAG_ID = "trigger_upsert_glue_job"

default_args = {
    'owner': 'airflow',
}

with DAG(
    dag_id=DAG_ID,
    default_args=default_args,
    description="Trigger Glue job on CSV S3 update",
    start_date=datetime(2025, 8, 1),
    schedule_interval=None, # DAG is manually triggered
    catchup=False,
    tags=["glue", "csv", "iceberg"]
) as dag:

    run_glue = GlueJobOperator(
        task_id="run_csv_to_iceberg_glue_job",
        job_name=GLUE_JOB_NAME,
        region_name=AWS_REGION
    )

    run_glue
```



## ✓ GOAL: Lambda → Trigger Airflow DAG

### ✓ 3. Create Lambda Function (Python)

Go to Lambda > Create function > Author from scratch:

- Name: `'trigger_upsert_glue_job'`
- Runtime: Python 3.11
- Execution role: use the IAM role you just created

```
• import boto3
• import hashlib
• import json
• from datetime import datetime
•
•
• s3 = boto3.client('s3')
• airflow = boto3.client('mwaa') # or use requests if using webserver
• endpoint
•
• BUCKET = 'mybucket-khushi-project2'
• KEY = 'Data/ad_revenue.csv'
• ARCHIVE_KEY = 'archive/ad_revenue_previous.csv'
• MWAA_ENV = 'Khushi-MWAA-P2'
• DAG_NAME = 'trigger_upsert_glue_job'
•
•
• def get_file_hash(bucket, key):
•     try:
•         obj = s3.get_object(Bucket=bucket, Key=key)
•         return hashlib.md5(obj['Body'].read()).hexdigest()
•     except s3.exceptions.NoSuchKey:
•         # If archive file not found, return None or empty string
•         return None
•
•
• def lambda_handler(event, context):
•     try:
•         new_hash = get_file_hash(BUCKET, KEY)
•         old_hash = get_file_hash(BUCKET, ARCHIVE_KEY)
•
•         if new_hash != old_hash:
•             print("Data has changed. Triggering Airflow DAG...")
•
•             # Archive current as previous
•             s3.copy_object(Bucket=BUCKET, CopySource=f"{BUCKET}/{KEY}",
• Key=ARCHIVE_KEY)
•
•             response = airflow.create_cli_token(Name=MWAA_ENV)
•             web_token = response['CliToken']
•             web_server = response['WebServerHostname']
•
•             # Airflow DAG Trigger
•             import requests
•             url = f"https:///{web_server}/aws_mwaa/cli"
•             trigger_cmd = f"dags trigger {DAG_NAME}"
•             resp = requests.post(url, headers={"Authorization":
• f"Bearer {web_token}"}, data=trigger_cmd)
```

```

        return {
            'statusCode': 200,
            'body': f'DAG {DAG_NAME} triggered successfully.'
        }
    else:
        print("No change detected.")
        return {
            'statusCode': 200,
            'body': 'No update in ad_revenue.csv.'
        }

except Exception as e:
    print(f"Error: {str(e)}")
    return {
        'statusCode': 500,
        'body': str(e)
    }

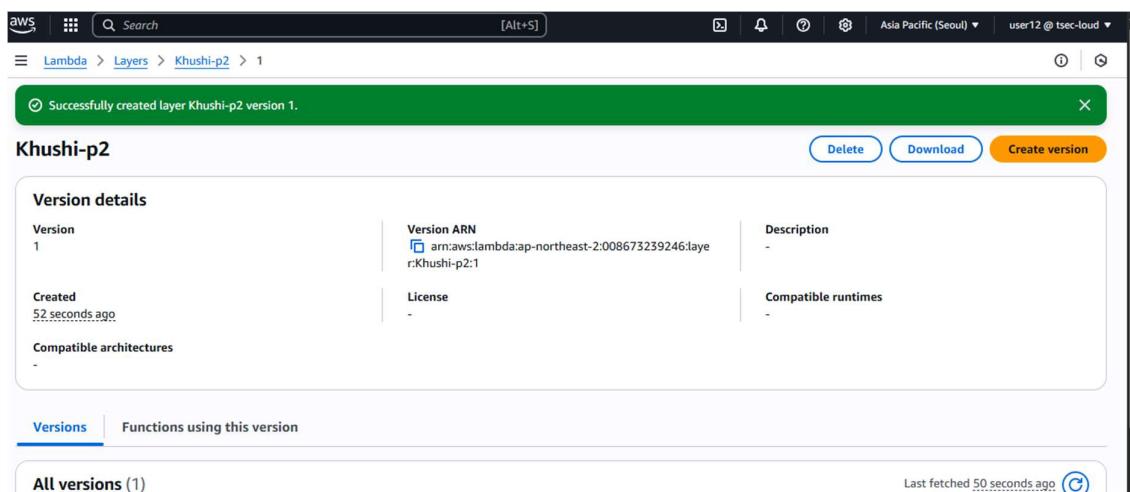
```

---

#### 4. Deploy and Test Lambda

### Use AWS Lambda Layer (Simplest Way)

1. Go to the AWS Console → **Lambda Layers**
2. Click "**Create Layer**"
3. Set name: requests-layer
4. Upload a .zip file that contains the requests module



The screenshot shows the AWS Lambda code editor interface. The left sidebar displays the project structure: EXPLORER (UPDATE\_AD\_REVENUE\_LAMBDA-KHUSHI), DEPLOY (Deploy, Test), and TEST EVENTS [SELECTED: KHUSHI-LAMDA]. The main area shows the code for `lambda_function.py`:

```

1 import json
2 from datetime import datetime
3
4 s3 = boto3.client('s3')
5 airflow = boto3.client('mwaa') # or use requests if using webserver endpoint
6
7 BUCKET = 'mybucket-khushi-project2'
8 KEY = 'Data/ad_revenue.csv'
9 ARCHIVE_KEY = "archive/ad_revenue_previous.csv"
10 MWAAS_ENV = 'Khushi-MWAA-P2'
11 DAG_NAME = 'trigger_upsert_glue_job'
12
13
14
    
```

The status bar at the bottom indicates Status: Succeeded and Test Event Name: khushi-lambda.

The screenshot shows the AWS Lambda function overview page for `update_ad_revenue_lambda-Khushi`. The left sidebar includes tabs for Diagram, Template, Code (selected), Test, Monitor, Configuration, Aliases, and Versions. The main area displays the function's configuration:

- Function Overview:** Shows the function icon, name, and layers (1).
- Destinations:** An S3 destination is listed with a + Add destination button.
- Triggers:** A + Add trigger button.
- Description:** Last modified 6 minutes ago.
- Function ARN:** arn:aws:lambda:ap-northeast-2:008673239246:function:update\_ad\_revenue\_lambda-Khushi
- Function URL:** Info

I change my CSV file to check if it is working or not-

The screenshot shows the AWS Lambda deployment summary and files/folders table. The summary indicates an upload was successful to the destination `s3://mybucket-khushi-project2/Data/`.

Name	Folder	Type	Size	Status	Error
ad_revenue.csv	-	text/csv	13.4 KB	Succeeded	-

As I uploaded –

The screenshot shows the AWS Glue Job runs page for the job "Khushi-P2-job". The interface includes a sidebar with options like Getting started, ETL jobs, Data Catalog, and Data Integration and ETL. The main area displays a table of job runs with the following data:

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity...	Worker t
Running	0	08/01/2025 17:31:31	-	0 s	10 DPUs	G.1X
Succeeded	0	08/01/2025 16:59:36	08/01/2025 17:01:07	1 m 22 s	10 DPUs	G.1X
Succeeded	0	08/01/2025 16:24:48	08/01/2025 16:26:23	1 m 27 s	10 DPUs	G.1X
Succeeded	0	08/01/2025 16:11:52	08/01/2025 16:13:21	1 m 22 s	10 DPUs	G.1X
Succeeded	0	08/01/2025 15:35:26	08/01/2025 15:36:58	1 m 25 s	10 DPUs	G.1X

My Airflow Running-

The screenshot shows the Airflow DAG details page for the DAG "trigger\_upsert\_glue\_job". The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The main area displays the DAG run summary with the following data:

Total Runs Displayed	3
Total success	2
Total running	1

Below the summary, it shows the first run start at 2025-08-01, 10:54:45 UTC, the last run start at 2025-08-01, 12:01:28 UTC, and the max run duration of 00:01:41.

Real Time: AWS Console → Kinesis → Data Streams → EMR →(Snowflake & S3)

## Kinesis Setup

## **1. Created Kinesis Data Stream:**

- Stream name: ad-revenue-stream-Khushi-P2
  - Step 1: Go to AWS Console → Kinesis → Data Streams
  - Click Create Data Stream
  - Region: ap-northeast-2

## 2. IAM Role for EC2:

- Policies attached:
    - AmazonKinesisFullAccess
    - AmazonS3FullAccess
    - CloudWatchLogsFullAccess

## Step 5: EC2 Setup

1. Launched EC2 (Amazon Linux 2)
  2. Security Group: Opened port 22 for SSH
  3. Attached IAM Role (created in Step 4)
  4. SSH into EC2:

```
ssh -i "khushi-seoul.pem" ec2-user@ec2-43-203-139-103.ap-northeast-2.compute.amazonaws.com
```

```
[ec2-user@ip-172-31-33-204 ~]$ pip install boto3
[ec2-user@ip-172-31-33-204 ~]$ command not found
[ec2-user@ip-172-31-33-204 ~]$ sudo yum update -y
[sudo] password for ec2-user:
Amazon Linux 2023 Kernel Livepatch repository
```

## **5. Install Dependencies:**

```
sudo yum update -y  
sudo yum install python3 -y  
pip3 install boto3
```

---

### **Step 6: CSV → Kinesis**

#### **1. From EC2 to Kinesis Data Stream:**

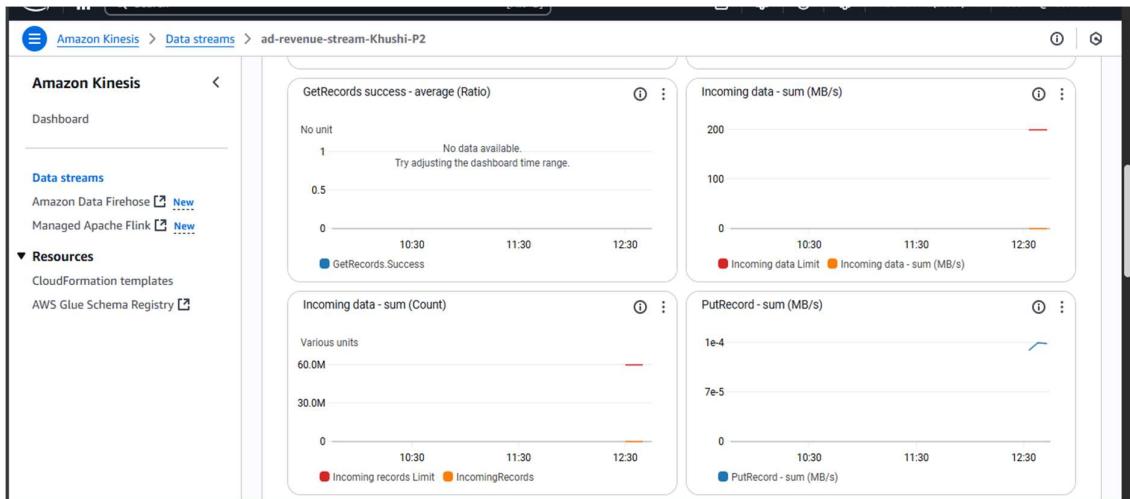
```
nano stream_to_kinesis.py
```

```
import boto3  
import time  
  
# Kinesis setup  
stream_name = 'nano stream_to_kinesis.py'  
region_name = 'ap-northeast-2'  
  
kinesis = boto3.client('kinesis', region_name=region_name)  
s3 = boto3.client('s3')  
  
# S3 path info  
bucket_name = 'mybucket-khushi-project2'  
s3_key = 'Data/viewership_logs.csv'  
  
# Download from S3  
response = s3.get_object(Bucket=bucket_name, Key=s3_key)  
lines = response['Body'].read().decode('utf-8').splitlines()  
  
# Send each line to Kinesis  
for line in lines:  
    line = line.strip()  
    if line:  
        kinesis.put_record(  
            StreamName=stream_name,  
            Data=line,  
            PartitionKey='partitionKey'  
        )  
        print(f"Sent: {line}")  
        time.sleep(1) # Simulate real-time stream
```

```

ec2-user@ip-172-31-33-204:~ % + -
    return self._emitter.emit(aliasated_event_name, **kwargs)
File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/hooks.py", line 256, in emit
    return self._emit(event_name, kwargs)
File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/hooks.py", line 239, in _emit
    response = handler(**kwargs)
File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/signers.py", line 108, in handler
    return self.sign(operation_name, request)
File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/signers.py", line 200, in sign
    auth.add_auth(request)
File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/crt/auth.py", line 56, in add_auth
    raise NoCredentialsError()
botocore.exceptions.NoCredentialsError: Unable to locate credentials
[ec2-user@ip-172-31-33-204 ~]$ nano stream_to_kinesis.py
[ec2-user@ip-172-31-33-204 ~]$ python3 stream_to_kinesis.py
Sent: 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
Sent: SID652253,U56366,CH008,Zee Anmol,Show_48,Entertainment,2025-07-30 15:18:14,98,Central,Premium,Smart TV/Web,False,1,.86,1,0,36,2,80,09
Sent: SID369996,U27555,CH005,Star Bharat,Show_111,Movies,2025-07-28 04:21:02,70,East,Basic,Mobile,Android,True,5,122.71,0.49,1,70,83
Sent: SID591350,U51120,CH032,Star Vijay,Show_77,Entertainment,2025-07-25 01:01:24,174,Pan-India,Basic,Smart TV,Roku,True,4,437.73,0.76,1,79,11
Sent: SID462921,U10309,CH003,Sony Entertainment Television,Show_155,Entertainment,2025-07-30 14:18:33,162,West,Basic,Smart TV,Roku,False,3,41.61,0.84,1,53.54
Sent: SID195719,U49174,CH047,DD Sahyadri,Show_90,News,2025-07-27 03:38:56,19,Pan-India,Free,Laptop,Roku,False,1,20.05,0.93,2,35,48
Sent: SID907005,U31682,CH035,KTV,Show_65,Movies,2025-07-26 20:24:32,134,Central,Premium,Mobile,FireTV,False,3,236.31,0.85,0,68.74
Sent: SID348132,U23434,CH005,Star Bharat,Show_125,Kids,2025-07-25 18:38:11,115,East,Free,Tablet,Android,True,2,99.9,0.21,0,70,67
Sent: SID663714,U24447,CH015,Star Sports Select 2,Show_15,Sports,2025-07-27 19:54:25,95,Central,Free,Smart TV,FireTV,True,2,263.73,0.8,1,45,04
Sent: SID491012,U55582,CH049,News18 Kerala,Show_63,Movies,2025-07-30 15:29:26,174,Northeast,Basic,Tablet,Web,True,1,133.67,0.99,0,50,9

```



As you can see, I can see my data coming in my data stream

## Step 7: EMR Spark Streaming (Kinesis → S3 + Snowflake)

### 1. EMR Cluster Launched (with Spark)

### 2. Uploaded PySpark job khushi.py:

- Reads from Kinesis stream viewership-stream
- Writes to:
  - S3 for raw/processed logs
  - Snowflake for transformed data

### 3. Required JARs:

- snowflake-jdbc-\*jar
- spark-snowflake\_2.12-\*jar
- spark-sql-kinesis\_2.12-1.2.0\_spark-3.0.jar

### 4. Run Spark Submit:

```
nano Khushi.py

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", "ad-revenue-stream-Khushi-P2") \
    .option("endpointUrl", "https://kinesis.ap-northeast-2.amazonaws.com") \
    .option("region", "ap-northeast-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": "snowflake-khushiiinfocepts.snowflakecomputing.com",
            "sfUser": "khushiiinfocepts",
            "sfPassword": "Khushi9407489099",
            "sfDatabase": "media_stream_db",
            "sfSchema": "media_data",

```

```

        "sfWarehouse": "media_warehouse",
        "sfRole": "ACCOUNTADMIN"
    }
    print("📝 Writing to Snowflake...")

    batch_df.write \
        .format("net.snowflake.spark.snowflake") \
        .options(**snowflake_options) \
        .option("dbtable", "channel_logs") \
        .mode("append") \
        .save()

    print("✅ Write to Snowflake succeeded.")

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

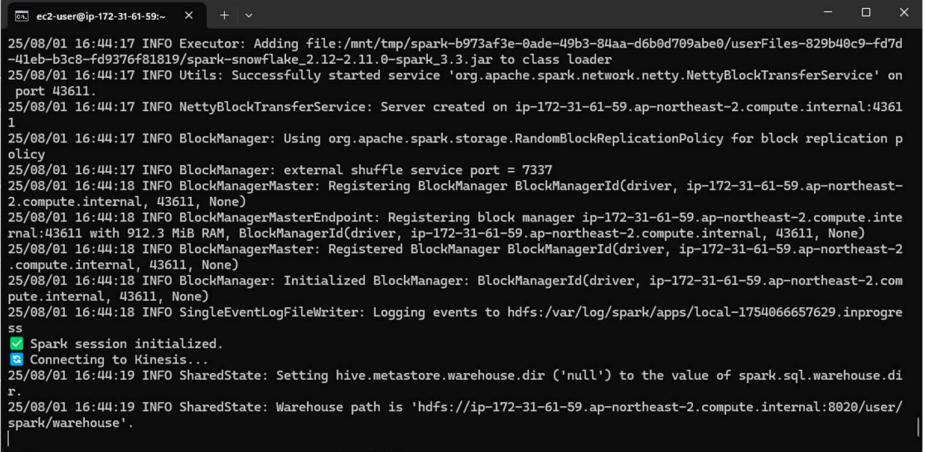
try:
    print("📤 Writing to S3...")

    batch_df.write \
        .format("csv") \
        .mode("append") \
        .save("s3://mybucket-khushi-project2/checkpoints/log")

    print("✅ Write to S3 succeeded.")

spark-submit khushi.py

```



The terminal window shows the following log output:

```

25/08/01 16:44:17 INFO Executor: Adding file:/mnt/tmp/spark-b973af3e-0ade-49b3-84aa-d6b0d709abe0/userFiles-829b40c9-fd7d-41eb-b3c8-fd9376f81819/spark-snowflake_2.12-2.11.0-spark_3.3.jar to class loader
25/08/01 16:44:17 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 43611.
25/08/01 16:44:17 INFO NettyBlockTransferService: Server created on ip-172-31-61-59.ap-northeast-2.compute.internal:43611
25/08/01 16:44:17 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
25/08/01 16:44:17 INFO BlockManager: external shuffle service port = 7337
25/08/01 16:44:18 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)
25/08/01 16:44:18 INFO BlockManagerMasterEndpoint: Registering block manager ip-172-31-61-59.ap-northeast-2.compute.internal:43611 with 912.3 MiB RAM, BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)
25/08/01 16:44:18 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)
25/08/01 16:44:18 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)
25/08/01 16:44:18 INFO SingleEventLogFileWriter: Logging events to hdfs:/var/log/spark/apps/local-1754066657629.inprogress
ss
✅ Spark session initialized.
🕒 Connecting to Kinesis...
25/08/01 16:44:19 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.
25/08/01 16:44:19 INFO SharedState: Warehouse path is 'hdfs://ip-172-31-61-59.ap-northeast-2.compute.internal:8020/user/spark/warehouse'.

```

```
ec2-user@ip-172-31-61-59:~
```

25/08/01 16:44:18 INFO BlockManagerMasterEndpoint: Registering block manager ip-172-31-61-59.ap-northeast-2.compute.internal:43611 with 912.3 MB RAM, BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)  
25/08/01 16:44:18 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)  
25/08/01 16:44:18 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, ip-172-31-61-59.ap-northeast-2.compute.internal, 43611, None)  
25/08/01 16:44:18 INFO SingleEventLogFileWriter: Logging events to hdfs:/var/log/spark/apps/local-1754066657629.inprogress  
✓ Spark session initialized.  
Connecting to Kinesis...  
25/08/01 16:44:19 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.  
25/08/01 16:44:19 INFO SharedState: Warehouse path is 'hdfs://ip-172-31-61-59.ap-northeast-2.compute.internal:8020/user/spark/warehouse'.  
Connected to Kinesis.  
Starting streaming query...  
25/08/01 16:44:22 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint  
25/08/01 16:44:23 INFO ClientConfigurationFactory: Set initial getObject socket timeout to 2000 ms.  
25/08/01 16:44:24 INFO ResolveWriteToStream: Checkpoint root s3://mybucket-khushi-project2/checkpoints/log resolved to s3://mybucket-khushi-project2/checkpoints/log.  
25/08/01 16:44:24 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.  
25/08/01 16:44:24 INFO deprecation: org.apache.hadoop.shaded.io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum  
25/08/01 16:44:24 INFO S3NativeFileSystem: Opening 's3://mybucket-khushi-project2/checkpoints/log/metadata' for reading

```
ec2-user@ip-172-31-61-59:~
```

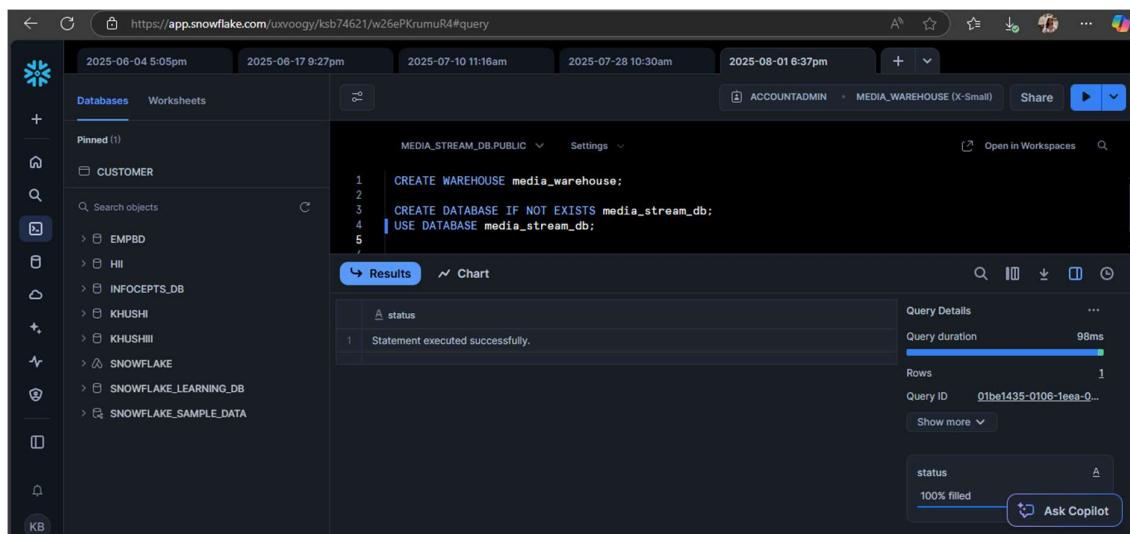
25/08/01 16:44:40 INFO KinesisSource: End Offset is {"shardId-000000000002":{"iteratorType":"AFTER\_SEQUENCE\_NUMBER","iteratorPosition":"49665728807528489122396556292537783389138034220381241378"}, "metadata": [{"streamName": "ad-revenue-stream-Khushi-P2", "batchId": "36"}, {"shardId-000000000001": {"iteratorType": "AT\_TIMESTAMP", "iteratorPosition": "1754066106928"}, "shardId-000000000000": {"iteratorType": "AT\_TIMESTAMP", "iteratorPosition": "1754066106928"}, "shardId-000000000003": {"iteratorType": "AT\_TIMESTAMP", "iteratorPosition": "1754066106928"}]}  
25/08/01 16:44:40 INFO KinesisSource: Processing 4 shards from Stream(ShardInfo(shardId-000000000000, AT\_TIMESTAMP, 1754066106928), ShardInfo(shardId-000000000001, AT\_TIMESTAMP, 1754066106928), ShardInfo(shardId-000000000002, AFTER\_SEQUENCE\_NUMBER, 49665728807528489122396556292537783389138034220381241378), ShardInfo(shardId-000000000003, AT\_TIMESTAMP, 1754066106928))  
25/08/01 16:44:40 INFO KinesisSource: GetBatch generating RDD of offset range: ShardInfo(shardId-000000000000, AT\_TIMESTAMP, 1754066106928), ShardInfo(shardId-000000000001, AT\_TIMESTAMP, 1754066106928), ShardInfo(shardId-000000000002, AFTER\_SEQUENCE\_NUMBER, 49665728807528489122396556292537783389138034220381241378), ShardInfo(shardId-000000000003, AT\_TIMESTAMP, 1754066106928)  
Processing batch 36...  
25/08/01 16:44:40 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0  
25/08/01 16:44:40 INFO DAGScheduler: Registering RDD 25 (start at NativeMethodAccessorImpl.java:0) as input to shuffle 2  
25/08/01 16:44:40 INFO DAGScheduler: Got job 2 (start at NativeMethodAccessorImpl.java:0) with 1 output partitions  
25/08/01 16:44:40 INFO DAGScheduler: Final stage: ResultStage 5 (start at NativeMethodAccessorImpl.java:0)  
25/08/01 16:44:40 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 4)  
25/08/01 16:44:40 INFO DAGScheduler: Submitting ShuffleMapStage 4 (MapPartitionsRDD[25] at start at NativeMethodAccessorImpl.java:0), which has no missing parents  
25/08/01 16:44:40 INFO MemoryStore: Block broadcast\_4 stored as values in memory (estimated size 158.3 KiB, free 911.9 MiB)  
25/08/01 16:44:40 INFO MemoryStore: Block broadcast\_4\_piece0 stored as bytes in memory (estimated size 58.6 KiB, free 911.9 MiB)

```
ec2-user@ip-172-31-61-59:~
```

rces/0/shard-commit/37/shardId-000000000000  
25/08/01 16:44:54 INFO MultipartUploadOutputStream: close closed:true s3://mybucket-khushi-project2/checkpoints/log/sources/0/shard-commit/37/shardId-000000000000  
25/08/01 16:44:54 INFO Executor: Finished task 0.0 in stage 8.0 (TID 20). 2467 bytes result sent to driver  
25/08/01 16:44:54 INFO TaskSetManager: Finished task 0.0 in stage 8.0 (TID 20) in 631 ms on ip-172-31-61-59.ap-northeast-2.compute.internal (executor driver) (4/4)  
25/08/01 16:44:54 INFO TaskSchedulerImpl: Removed TaskSet 8.0, whose tasks have all completed, from pool  
25/08/01 16:44:54 INFO DAGScheduler: ResultStage 8 (start at NativeMethodAccessorImpl.java:0) finished in 0.784 s  
25/08/01 16:44:54 INFO DAGScheduler: Job 4 is finished. Cancelling potential speculative or zombie tasks for this job  
25/08/01 16:44:54 INFO TaskSchedulerImpl: Killing all running tasks in stage 8: Stage finished  
25/08/01 16:44:54 INFO DAGScheduler: Job 4 finished: start at NativeMethodAccessorImpl.java:0, took 0.787036 s  
25/08/01 16:44:54 INFO FileFormatWriter: Start to commit write Job e5fb0d7b-a4c0-4bf4-bb5d-a57d5de4d45f.  
25/08/01 16:44:54 INFO MultipartUploadOutputStream: close closed:false s3://mybucket-khushi-project2/Dags/\_SUCCESS  
25/08/01 16:44:54 INFO FileFormatWriter: Write Job e5fb0d7b-a4c0-4bf4-bb5d-a57d5de4d45f committed. Elapsed time: 68 ms.  
25/08/01 16:44:54 INFO FileFormatWriter: Finished processing stats for write job e5fb0d7b-a4c0-4bf4-bb5d-a57d5de4d45f.  
✓ Write to S3 succeeded.  
25/08/01 16:44:54 INFO CheckpointFileManager: Writing atomically to s3://mybucket-khushi-project2/checkpoints/log/commits/s/37 using temp file s3://mybucket-khushi-project2/checkpoints/log/commits/.37.74bff9e5-e4c0-498a-b1f2-2318f6d50b3b.tmp  
25/08/01 16:44:54 INFO MultipartUploadOutputStream: close closed:false s3://mybucket-khushi-project2/checkpoints/log/commits/.37.74bff9e5-e4c0-498a-b1f2-2318f6d50b3b.tmp to s3://mybucket-khushi-project2/checkpoints/log/commits/37  
25/08/01 16:44:54 INFO S3NativeFileSystem: rename s3://mybucket-khushi-project2/checkpoints/log/commits/.37.74bff9e5-e4c0-498a-b1f2-2318f6d50b3b.tmp s3://mybucket-khushi-project2/checkpoints/log/commits/37 using algorithm version 1  
25/08/01 16:44:54 INFO CheckpointFileManager: Renamed temp file s3://mybucket-khushi-project2/checkpoints/log/commits/.37.74bff9e5-e4c0-498a-b1f2-2318f6d50b3b.tmp to s3://mybucket-khushi-project2/checkpoints/log/commits/37  
25/08/01 16:44:54 INFO MicroBatchExecution: Streaming query made progress: {  
 "id" : "8a498332-2e2d-41d8-a502-ca8599227adb",

## ✓ Step 8: Snowflake Setup

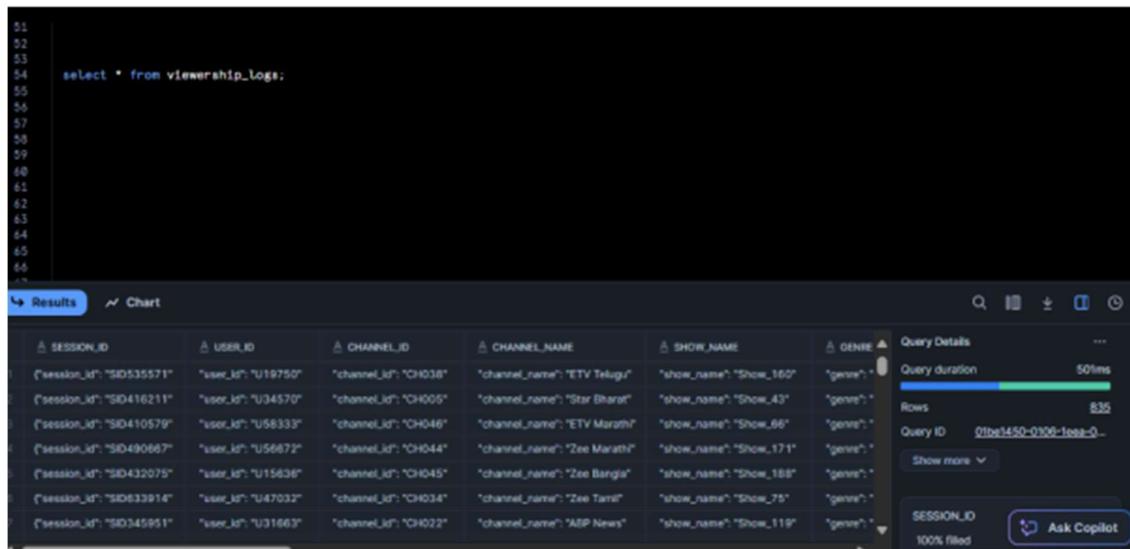
1. Used Snowflake Web UI:  
<https://app.snowflake.com/uxvoogy/ksb74621/w26ePKruMuR4>
2. Database: MEDIA\_STREAM\_DB
3. Schema: MEDIA\_DATA
4. Table: viewership\_logs



The screenshot shows the Snowflake Web UI interface. On the left, there's a sidebar with icons for databases, worksheets, and other account details. The main area shows a timeline at the top with dates from 2025-06-04 to 2025-08-01. Below the timeline, under the 'Databases' tab, a query window is open with the following SQL code:

```
CREATE WAREHOUSE media_warehouse;
CREATE DATABASE IF NOT EXISTS media_stream_db;
USE DATABASE media_stream_db;
```

The results section shows a single row: "Statement executed successfully." To the right, there are 'Query Details' such as duration (98ms), rows (1), and a query ID.



The screenshot shows the same Snowflake Web UI interface. A new query window is open with the following SQL code:

```
select * from viewership_logs;
```

The results section displays a table with columns: SESSION\_ID, USER\_ID, CHANNEL\_ID, CHANNEL\_NAME, SHOW\_NAME, and GENRE. There are 835 rows returned. The table data is as follows:

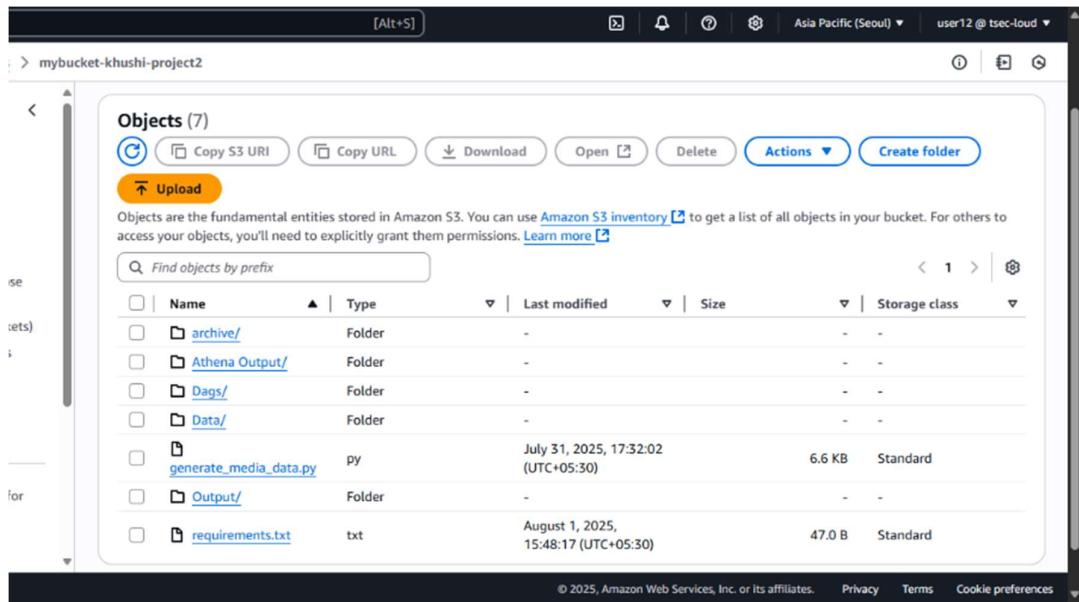
SESSION_ID	USER_ID	CHANNEL_ID	CHANNEL_NAME	SHOW_NAME	GENRE
5035571	U19750	CH038	ETV Telugu	Show_160	...
50416211	U34570	CH005	Star Bharat	Show_43	...
50410579	U58333	CH046	ETV Marathi	Show_66	...
50490667	U56872	CH044	Zee Marathi	Show_171	...
50432075	U15626	CH045	Zee Bangla	Show_168	...
50633914	U47032	CH034	Zee Tamil	Show_75	...
50345951	U31663	CH022	YAH News	Show_119	...

On the right, 'Query Details' show a duration of 501ms and 835 rows. A 'Show more' button is visible.

## Step 9: Issues Faced

- Snowflake JDBC write failed due to locked account – needs to be unlocked by admin.
- Missing JAR error for Kinesis connector.
- Permissions and path corrections done using chmod and checking /usr/lib/spark/jars.

Data Send Sucessfully on S3-



The screenshot shows the AWS S3 console interface. The top navigation bar includes the region "Asia Pacific (Seoul)" and the user "user12 @ tsec-loud". The main area displays the contents of the bucket "mybucket-khushi-project2". The "Objects (7)" section lists the following items:

Name	Type	Last modified	Size	Storage class
archive/	Folder	-	-	-
Athena Output/	Folder	-	-	-
Dags/	Folder	-	-	-
Data/	Folder	-	-	-
generate_media_data.py	py	July 31, 2025, 17:32:02 (UTC+05:30)	6.6 KB	Standard
Output/	Folder	-	-	-
requirements.txt	txt	August 1, 2025, 15:48:17 (UTC+05:30)	47.0 B	Standard

At the bottom of the page, there are links for "Privacy", "Terms", and "Cookie preferences".