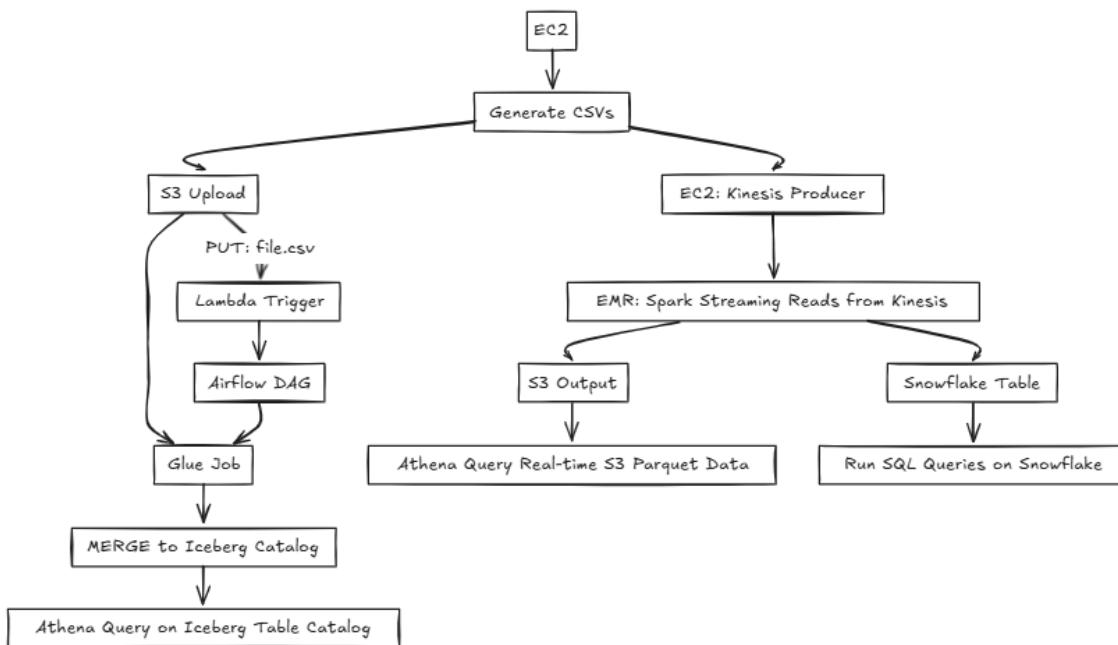


Media Stream Analytics

Our Technical Solution: Architecture Approach

This architecture is modular, scalable, and cloud-native. It separates batch and streaming pipelines while maintaining a unified data lake using Iceberg. Real-time ingestion via Kinesis and EMR ensures low-latency analytics, while Glue and Airflow orchestrate batch updates and CDC operations.



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
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
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)
        logs.append([user_id, channel_id, timestamp, duration, region,
                    subscription, device, platform, is_live, genre,
                    ads_watched, ad_revenue, engagement, buffer_count,
                    completion_pct])
    return logs

```

```

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.strptime('%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 (3) 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](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	ad_revenue.csv	csv	August 1, 2025, 13:18:38 (UTC+05:30)	13.4 KB	Standard
<input type="checkbox"/>	channel_metadata.csv	csv	August 1, 2025, 13:18:39 (UTC+05:30)	2.0 KB	Standard
<input type="checkbox"/>	demographics.csv	csv	August 1, 2025, 13:18:40 (UTC+05:30)	160.5 KB	Standard

```
ec2-user@ip-172-31-10-45:~ x + v
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
 ~~ V~' `->
 ~~~ /
 ~~-.-/_/
 _/_/
 _/m/' 

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

tbsm-ice

Last modified on 1/8/2025, 2:19:06 pm Actions ▾ Save Run

Script Job details Runs Data quality Schedules Version Control

Script Info

```
1 from pyspark.sql import SparkSession
2
3 # Initialize SparkSession with Iceberg support
4 spark = SparkSession.builder \
5     .appName("Glue Iceberg Table CDC") \
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://tbsm-icebergtabler/") \
9     .config("spark.sql.catalog.glue_catalog.io-impl", "org.apache.iceberg.aws.s3.S3FileIO") \
10    .getOrCreate()
11
12 # Create Glue database if not exists
13 spark.sql("CREATE DATABASE IF NOT EXISTS glue_catalog.tbsm_icebergdb")
14
15 # Read demographics CSV from S3
16 df = spark.read.option("header", True).csv("s3://tbsm-calm/ice/demographics.csv")
17 df.show()
18
19 # Create Iceberg table if not exists
20 spark.sql("""
21 CREATE TABLE IF NOT EXISTS glue_catalog.tbsm_icebergdb.demographics (
22     user_id string,
```

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

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
Succeeded	0	08/01/2025 14:19:11	08/01/2025 14:20:31	1 m 13 s	10 DPUs	G.1X	5.0
Failed	0	08/01/2025 14:07:51	08/01/2025 14:08:42	42 s	10 DPUs	G.1X	5.0
Failed	0	08/01/2025 14:05:35	08/01/2025 14:06:39	52 s	10 DPUs	G.1X	5.0

Run details	Input arguments (9)	Logs	Run insights	Metrics	Troubleshooting analysis - preview	Spark UI
Job name tbsm-ice Id jr_4d43596bf6af6a4519a12ff33bc74e6bc7e85fe3 d3cc3ba77a8888593441086e	Start time (Local) 08/01/2025 14:19:11 End time (Local) 08/01/2025 14:20:31	Glue version 5.0 Worker type G.1X	Last modified on (Local) 08/01/2025 14:20:31 Log group name /aws-glue/jobs			
Run status Succeeded	Start-up time 7 seconds	Max capacity 10 DPUs	Number of workers 10			
Retry attempt number Initial run	Execution time 1 minute 13 seconds	Execution class Standard	Timeout 480 minutes			
Trigger name	Security configuration	Cloudwatch logs	Usage profile			

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

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The 'Basic information' section is visible, containing fields for Function name (tbsm-trigger-airflow-dag), Runtime (Python 3.10), and Architecture (x86_64). Other tabs like Code, Test, Monitor, Configuration, Aliases, and Versions are present at the top.

Execution role

Role name: tbsm-trigger-airflow-dag-role-bh7xtm0k

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs

By action | By resource

Resource: arn:aws:logs:ap-southeast-2:008673239246:*

Actions: Allow: logs:CreateLogGroup

tbsm-trigger-airflow-dag-role-bh7xtm0k [Info](#) [Delete](#)

Summary		Edit
Creation date	August 01, 2025, 14:49 (UTC+05:30)	ARN
Last activity	-	arn:aws:iam::008673239246:role/service-role/tbsm-trigger-airflow-dag-role-bh7xtm0k
		Maximum session duration 1 hour

[Permissions](#) [Trust relationships](#) [Tags](#) [Last Accessed](#) [Revoke sessions](#)

Permissions policies (3) [Info](#) [Simulate](#) [Remove](#) [Add permissions](#)

You can attach up to 10 managed policies.

Filter by Type		
Search	All types	Attached entities
<input type="checkbox"/> Policy name	▲ Type	▼ Attached entities
<input type="checkbox"/> AmazonS3FullAccess	AWS managed	302
<input type="checkbox"/> AWSLambdaBasicExecutionRole-50f47db3-655d-42...	Customer managed	1
<input type="checkbox"/> MWAAFullAccess	Customer managed	3

```

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)})
    }
```

```

lambda_function.py
1 import json
2 import boto3
3 import urllib3
4 from datetime import datetime
5
6 MWAAS_ENV_NAME = "tbsm-MyAirflowEnvironment"
7 DAG_NAME = "trigger_upsert_glue_job"
8
9 def lambda_handler(event, context):
10     print("Received S3 Event:")
11     print(json.dumps(event, indent=2))
12
13     mwaa = boto3.client('mwaa')
14
15     try:
16         # 1. Create MWAA CLI token
17         resp = mwaa.create_cli_token(Name=MWAAS_ENV_NAME)
18         web_token = resp['cliToken']
19         web_server_hostname = resp['WebServerHostname']
20         print(f"CLI token and hostname retrieved successfully.")
21
22         # 2. Generate Airflow CLI trigger command
23         execution_date = datetime.utcnow().isoformat()
24         cli_command = f"dag trigger -e {execution_date} {DAG_NAME}"
25         trigger_url = f"https:///{web_server_hostname}/aws_mwaa/cli"
26
27         print(f"Triggering DAG: {DAG_NAME} at {execution_date}")
28
29         # 3. Send request using urllib3
30         http = urllib3.PoolManager()
31         headers = {
32             "Authorization": f"Bearer {web_token}",
33             "Content-Type": "text/plain"
34         }
35

```

Create new test event

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

```

1 {
2     "Records": [
3         {
4             "eventVersion": "2.0",
5             "eventSource": "aws:s3",
6             "awsRegion": "us-east-1",
7             "eventTime": "1970-01-01T00:00:00.000Z",
8             "eventName": "ObjectCreated:Put",
9             "userIdentity": {
10                 "principalId": "EXAMPLE"
11             },
12             "requestParameters": {
13                 "sourceIPAddress": "127.0.0.1"
14             },
15             "responseElements": {
16                 "x-amz-request-id": "EXAMPLE123456789",
17                 "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnc"
18             },
19             "s3": {

```

```

 7  def lambda_handler(event, context):
12      web_token = response['CliToken']
13      mwaa_webserver = response['WebServerHostname']
14
15      # Build the Airflow CLI URL to trigger the DAG
16      trigger_url = f"https://{{mwaa_webserver}}/aws_mwaa/cli"
17
18      # The airflow command to trigger the DAG (normally sent to the MWAA CLI endpoint)
19      cli_command = f"airflow dags trigger {DAG_NAME}"
20
21      # For demonstration, returning the URL and token,
22      # but actual triggering would need an authenticated HTTP POST with this token

```

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL Execution Results

Status: Succeeded
Test Event Name: test1

Response:

```

{
  "statusCode": 200,
  "body": "{\"message\": \"Trigger command for DAG run_glue_demographics_etl ready!\", \"cli_token\": \"eyJ0exAiOiJKViQiLCJhbGciOiJSUzIiNj9, eyJhdQioiibGkiiC1leH4iojE3NTQmWDyODisInVzXioi3hc3N1bWVklXjBGuvdGjzb510cmLn22yLWFpcmZsb3ctGFnlXjvbGuTymg3eHRTMgsvdGjzb510cmLn22yLWFpcmZsb3ct2GFnIn0, Kd-3Sj901-0$BjMTX7-baAchqejZLEImprn04h0EYsZquZgaCpRInhnIDF88v9uuFimDuBa6pcalWjITmwjpjALEj8gQiYuRR_z294F2d9GP1-CQAK09EEEM-dm3JnuIFBp7QHQ8QAg80BEjEYLVPK0gSP_nsUkQfEtvh87R4680KA0MzKts6zRQJ5fe95EAhScwlv33s5DUzph04df7zurapolRYrdm5AX-NDY0)1YLIVGXCRXMo19yfIKKU4ArjBH_hgjgx_mlVqZ8b0hvavLvpQa6yXntr0rZvglvdpv9BDan2y00au36otwRg7dETt778Gtpxxd7g\", \"cli_url\": \"https://248a24c3-f0de-42e5-bffb-40160ccb0c02.c6.ap-southeast-2.amazonaws.com/aws_mwaa/cli\", \"cli_command\": \"airflow dags trigger run_glue_demographics_etl\"}"
}

```

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

Amazon MWAA > Environments > tbsm-MyAirflowEnvironment > Edit

tbsm-MyAirflowEnvironment is being created. This takes 20-30 minutes.

Specify details

Environment details

Airflow version: 2.10.3 (Latest)

Weekly maintenance window start (UTC): Sunday 00:30

DAG code in Amazon S3

S3 Bucket: s3://tbsm-encore

DAGs folder: s3://tbsm-encore/dag

Plugins file - optional

Need to create new MWAA vpc around it

CloudFormation > Stacks

Stacks (7)

Stack name	Status	Created time	Description
MWAA-VPC-Airflow2-TK	CREATE_IN_PROGRESS	2025-08-01 15:18:45 UTC+0530	This template deploys a VPC, with a pair of public and private subnets spread across two Availability Zones. It deploys an internet gateway, with a default route on the public subnets. It deploys a pair of NAT gateways (one in each AZ), and default routes for them in the private subnets.
sm-MWAA-VPC	CREATE_COMPLETE	2025-08-01 15:03:47 UTC+0530	This template deploys a VPC, with a pair of public and private subnets spread across two Availability Zones. It deploys an internet gateway, with a default route on the public subnets. It deploys a pair of NAT gateways (one in each AZ), and default routes for them in the private subnets.

sm-MWAA-VPC

Resources (22)

Logical ID	Physical ID	Type	Status	Module
DefaultPrivateRoute1	rtb-02e7dc90d3f2b96bc0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
DefaultPrivateRoute2	rtb-08a1e08a9638fde09j0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
DefaultPublicRoute	rtb-089733ac72b19e1f9j0.0.0/0	AWS::EC2::Route	CREATE_COMPLETE	-
InternetGateway	igw-05f0ef293d995c281	AWS::EC2::InternetGateway	CREATE_COMPLETE	-
InternetGatewayAttachment	IGWVpc-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](#)

[Create MWAA VPC](#)

- [vpc-0cf5747025ac8078b](#)
VPC
- [vpc-01fdada2d208280c6](#)
VPC
- [vpc-025cc9b28b367cd7f](#)
MWAA-VPC-Airflow2-TK
- [vpc-0bc3981d59038a53e](#)
Default

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.

- [sg-00eff3a3165d63ffd](#)
Security Group for Amazon MWAA Environment tbsm-MyAirflowEnvironment airflow-security-group-tbsm-MyAirflowEnvironment-4n13c5

Max 5 security groups

AIRLFOW DAG CODE

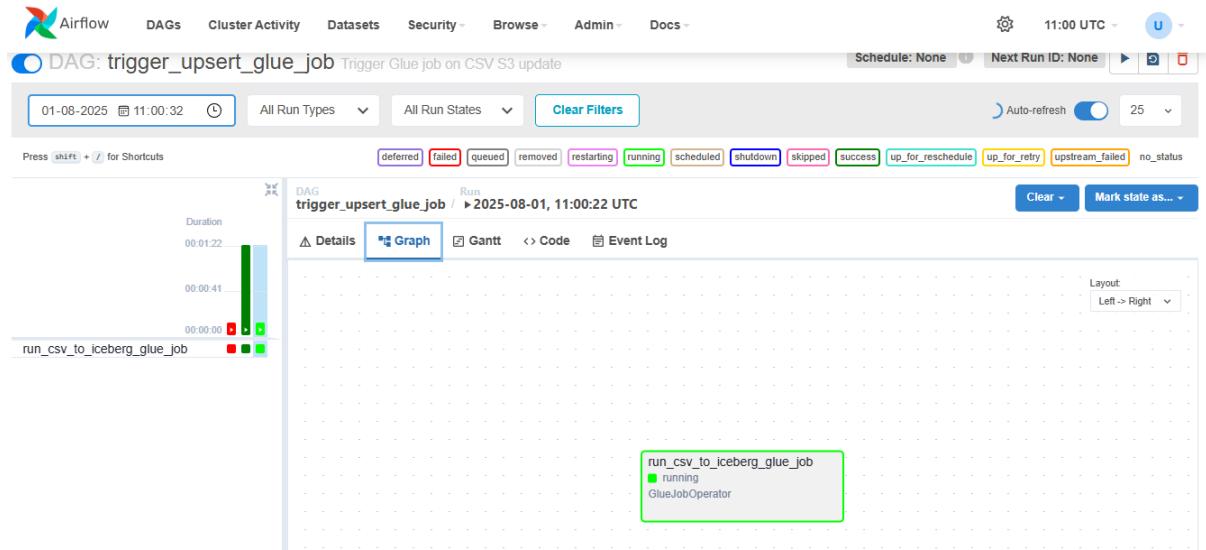
```
GNU nano 8.3      run_glue_demographics_etl.py          from airflow import DAG
from airflow.providers.amazon.aws.operators.glue import AwsGlueJobOperator>from
datetime import datetime

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:

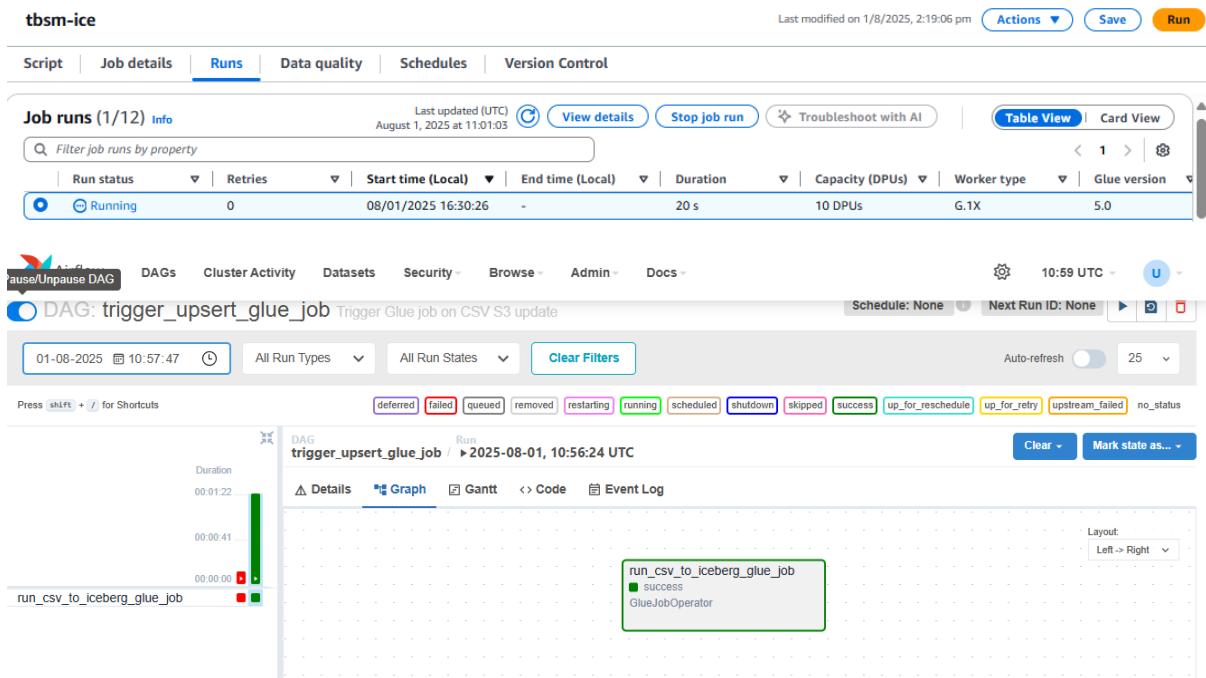
    run_glue_job = AwsGlueJobOperator(
        task_id='run_iceberg_merge_job',
        job_name='tbsm-ice',
        script_location='s3://aws-glue-assets-008673239246-ap-southeast>
aws_conn_id='aws_default',
        region_name='ap-southeast-2'
    )
```

The screenshot shows the AWS MWAH (Amazon Managed Workflows for Apache Airflow) environment details page. The top navigation bar includes the AWS logo, search bar, and user information (Asia Pacific (Sydney) and user28@tac-loud). The main content area displays the environment configuration:

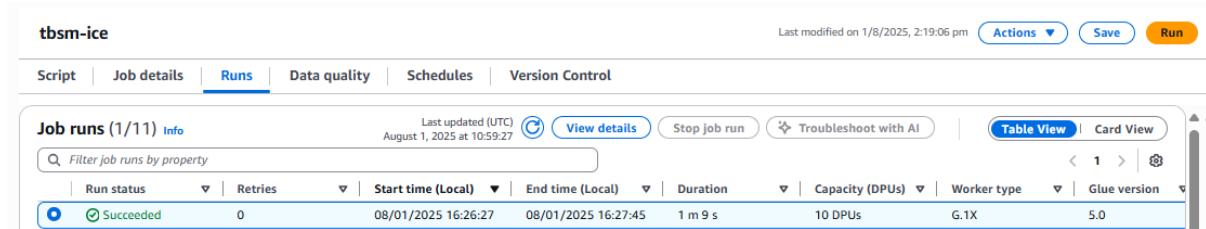
- Details**:
 - Status: Available
 - ARN: arn:aws:airflow:ap-southeast-2:008673239246:environment/tbsm-MyAirflowEnvironment
- Airflow UI**: A link to the Airflow UI at 248a24c3-f0de-42e5-bffb-40160ccb0c02-vpce.c6.ap-southeast-2.airflow.amazonaws.com.
- Weekly maintenance window start (UTC)**: Sunday 00:30
- Last update**:
 - Status: SUCCESS
 - Created at: 2025-08-01T09:42:53.000Z
- Worker replacement strategy**: Forced
- DAG code in Amazon S3**:
 - S3 Bucket: tbsm-encore
 - Plugins file: -
- DAGs folder**: dag
- Requirements file**: -



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

```
U30279,Male,46-60,West,Premium
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

DAG: trigger_upsert_glue_job Trigger Glue job on CSV S3 update

Schedule: None | Next Run ID: None | Auto-refresh | 25

01-08-2025 11:40:05 | All Run Types | All Run States | Clear Filters

Press shift + / for Shortcuts

deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

DAG: trigger_upsert_glue_job Run: 2025-08-01, 11:40:05 UTC / run.csv_to_iceberg_glue.job

Task: run.csv_to_iceberg_glue.job

Task Duration

Duration: 00:01:52 (Aug 01, 11:00)

Details Graph Gantt Code Events

Task ID: run.csv_to_iceberg_glue.job

Status: running

Started: 2025-08-01, 11:40:00 UTC

Duration: 00:00:08

Trigger Rule: all_success

run.csv_to_iceberg_glue.job

running GlueJobOperator

Which will trigger glue in running the job

Job runs (1/16) Info		Last updated (UTC) August 1, 2025 at 11:40:40	 View details	 Stop job run	 Troubleshoot with AI	Table View	Card View
Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
 Running	0	08/01/2025 17:10:10	-	21 s	10 DPUs	G.1X	5.0

It will be successfully run

tbsm-ice								Last modified on 1/8/2025, 2:19:06 pm	Actions	Save	Run
Script	Job details	Runs	Data quality	Schedules	Version Control						
Job runs (1/16) Info											
Last updated (UTC)	August 1, 2025 at 11:41:41	 View details	Stop job run	 Troubleshoot with AI	Table View	Card View	<	1	>		
Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version				
 Succeeded	0	08/01/2025 17:10:10	08/01/2025 17:11:38	1 m 19 s	10 DPUs	G.1X	5.0				

Then you can see table database created

Databases (12)				Last updated (UTC) August 1, 2025 at 11:42:30	Edit	Delete	Add database
A database is a set of associated table definitions, organized into a logical group.							
	Filter databases						
	Name	Description	Location URI			Created on (UTC)	
<input type="checkbox"/>	tbsm_icbeerbdb	-	-			August 1, 2025 at 11:40:59	

And iceberg s3 is created wih folder named

Amazon S3

General purpose buckets

Directory buckets

Table buckets

Vector buckets [Preview](#)

Access Grants

Access Points (General Purpose Buckets, FSx file systems)

Access Points (Directory Buckets)

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

IAM Access Analyzer for S3

demographics/

Objects (2)

Actions ▾ Create folder Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
data/	Folder	-	-	-
metadata/	Folder	-	-	-

And in that parquet

Amazon S3

General purpose buckets

Directory buckets

Table buckets

Vector buckets [Preview](#)

Access Grants

Access Points (General Purpose Buckets, FSx file systems)

Access Points (Directory Buckets)

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens

Dashboards

Storage Lens groups

AWS Organizations settings

Feature spotlight 11

data/

Objects (5)

Actions ▾ Create folder Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
00000-4-08dd959c-19dd-4489-9543-7f71cabd969b-0-00001.parquet	parquet	August 1, 2025, 16:44:48 (UTC+05:30)	21.7 KB	Standard
00000-4-a0fa2e8a-f477-435b-80cf-088096cb775-0-00001.parquet	parquet	August 1, 2025, 16:24:20 (UTC+05:30)	21.7 KB	Standard
00000-4-a5c8d748-564f-434a-a6f2-5de231f2cc2-0-00001.parquet	parquet	August 1, 2025, 16:49:30 (UTC+05:30)	21.7 KB	Standard
00000-4-c19254d1-b259-475a-84be-5227d9ae1a0c-0-00001.parquet	parquet	August 1, 2025, 16:27:33 (UTC+05:30)	21.7 KB	Standard
00000-4-ce81db3a-2063-4846-bcf4-d338c96aaece-0-00001.parquet	parquet	August 1, 2025, 17:11:26 (UTC+05:30)	21.7 KB	Standard

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)
```

```
{}
✓ Sent to Kinesis. SequenceNumber: 49665723011096496375117208002099357694618995169282752530
🕒 Streaming Record → {
  "session_id": "SID480667",
  "user_id": "U58068",
  "channel_id": "CH020",
  "channel_name": "Eurosport India",
  "show_name": "Show_60",
  "genre": "Music",
  "timestamp": "2025-07-29 07:31:07",
  "duration_minutes": "93",
  "region": "North",
  "subscription_type": "Basic",
  "device": "Smart TV",
  "platform": "iOS",
  "is_live": "False",
  "ads_watched": "4",
  "ad_revenue": "718.82",
  "engagement_score": "0.87",
  "buffer_count": "1",
  "completion_percentage": "90.01"
}
✓ Sent to Kinesis. SequenceNumber: 49665723011118797120315738625249355893628945935011676194
```

viewership-stream [Info](#) [Delete](#)

Data stream summary			
Status Active	Capacity mode On-demand	ARN arn:aws:kinesis:ap-southeast-2:008673239246:stream/viewership-stream	Creation time August 01, 2025 at 17:25 GMT+5:30
Data retention period 1 day			

[Applications](#) [Monitoring](#) [Configuration](#) [Enhanced fan-out \(0\)](#) [Data viewer](#) [Data analytics - new](#) [Data stream sharing](#) [EventBridge Pipes](#)

Producers [Info](#)

Producers put records into Kinesis Data Streams.

Amazon Kinesis Agent
Use a stand-alone Java software application to send data to the stream. [Learn more](#)

AWS SDK
Use AWS SDK for Java to develop producers. [Learn more](#)

Amazon Kinesis Producer Library (KPL)
Use KPL to develop producers. [Learn more](#)

[View in GitHub](#) [View in GitHub](#) [View in GitHub](#)

Partition key	Data	Approximate arrival timestamp	Sequence number
U29953	{"session_id": "SID572805", "user_id": "U29953", "...	August 01, 2025 at 17:39:57 GMT+5:30	4966572301107419562991867741422460598614...
U56062	{"session_id": "SID983604", "user_id": "U56062", "...	August 01, 2025 at 17:39:57 GMT+5:30	4966572301107419562991867741422944168942...
U40976	{"session_id": "SID234699", "user_id": "U40976", "...	August 01, 2025 at 17:39:58 GMT+5:30	4966572301107419562991867741423306846688...

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

```
[ec2-user@ip-172-31-0-233 ~]$ nano kinesis_to_snowflake_s3.py
[ec2-user@ip-172-31-0-233 ~]$ 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
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy2-5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/ec2-user/.ivy2/cache
The jars for the packages stored in: /home/ec2-user/.ivy2/jars
com.qubole.spark#spark-sql-kinesis_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-491385e2-7042-4928-b954-faae6c6b3dd;1.0
  confs: [default]
    found com.qubole.spark#spark-sql-kinesis_2.12;1.2.0_spark-3.0 in central
:: resolution report :: resolve 130ms :: artifacts dl 4ms
  :: modules in use:
    com.qubole.spark#spark-sql-kinesis_2.12;1.2.0_spark-3.0 from central in [default]
    +---+
    |   conf      |   modules   ||   artifacts  | | | | |
    |           |   number| search|dwvnlded|evicted||   number|dwvnlded|
    |   default   |   1   |   0   |   0   |   0   ||   1   |   0   |
    +---+
:: retrieving :: org.apache.spark#spark-submit-parent-491385e2-7042-4928-b954-faae6c6b3dd
  confs: [default]
  0 artifacts copied, 1 already retrieved (0kB/6ms)
25/08/01 13:36:05 WARN DependencyUtils: Local jar /usr/lib/spark/jars/spark-snowflake_2.12-2.9.0-spark_3.1.jar does not exist, skipping.
Initializing Spark session...
25/08/01 13:36:06 INFO SparkContext: Running Spark version 3.4.1-amzn-2
25/08/01 13:36:06 INFO ResourceUtils: =====
25/08/01 13:36:06 INFO ResourceUtils: No custom resources configured for spark.driver.
25/08/01 13:36:06 INFO ResourceUtils: =====
one)
25/08/01 13:36:07 INFO BlockManagerMasterEndpoint: Registering block manager ip-172-31-0-233.ap-southeast-2.compute.internal:43095 with 912.3 MiB RAM, BlockManagerId(driver, ip-172-31-0-233.ap-southeast-2.compute.internal, 43095, None)
25/08/01 13:36:07 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, ip-172-31-0-233.ap-southeast-2.compute.internal, 43095, None)
25/08/01 13:36:07 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, ip-172-31-0-233.ap-southeast-2.compute.internal, 43095, None)
25/08/01 13:36:08 INFO SingleEventLogFileWriter: Logging events to hdfs:/var/log/spark/apps/local-1754055367195.inprogress
✓ Spark session initialized.
Connecting to Kinesis...
25/08/01 13:36:08 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.
25/08/01 13:36:08 INFO SharedState: Warehouse path is 'hdfs://ip-172-31-0-233.ap-southeast-2.compute.internal:8020/user/spark/warehouse'.
Connected to Kinesis.
Starting streaming query...
25/08/01 13:36:12 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint
25/08/01 13:36:13 INFO ClientConfigurationFactory: Set initial getObject socket timeout to 2000 ms.
25/08/01 13:36:14 INFO ResolveWriteToStream: Checkpoint root s3://tbsm-encore/dag resolved to s3://tbsm-encore/dag.
25/08/01 13:36:14 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
25/08/01 13:36:14 INFO ResolveWriteToStream: Using adaptive enabled = false for streaming DataFrames/Datasets.
```

```

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@0d81617d,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,EndingHashKey: 85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber: 496657230110741956299137173692589280939436444581368168
000000000001,AT_TIMESTAMP,1754055375802}, ShardInfo(shardId-000000000002,AT_TIMESTAMP,1754055375802), ShardInfo(shardId-000000000003,AT_TIMESTAMP,1754055375802)}
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_TIMESTAMP,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 NativeMethodAccessoImpl.java:0
25/08/01 13:36:18 INFO DAGScheduler: Registering RDD 5 (start at NativeMethodAccessoImpl.java:0) as input to shuffle 0
25/08/01 13:36:18 INFO DAGScheduler: Got job 0 (start at NativeMethodAccessoImpl.java:0) with 1 output partitions
25/08/01 13:36:18 INFO DAGScheduler: Final stage: ResultStage 1 (start at NativeMethodAccessoImpl.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 NativeMethodAccessoImpl.java:0), which has no missing 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 NativeMethodAccessoImpl.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 NativeMethodAccessoImpl.java:0, took 2.162825 s
💡 Records in batch: 163
💡 Writing to Snowflake...
25/08/01 13:36:21 WARN SnowflakeConnectorUtils$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of Spark.
25/08/01 13:36:21 INFO CodeGenerator: Code generated in 21.067161 ms
25/08/01 13:36:21 WARN DefaultJDBCWapper$: 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 KiB, 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 SnowflakeConnectorUtil$: Query pushdown is not supported because you are using Spark 3.4.1-amzn-2 with a connector designed to support Spark 3.3. Either use the version of Spark supported by the connector or install a version of the connector that supports your version of Spark.
25/08/01 13:36:30 WARN DefaultJDBCWapper$: 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.FileSystemOptimizedOutputCommitterFactory
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 DAGScheduler: Got job 2 (start at NativeMethodAccessoImpl.java:0) with 1 output partitions
25/08/01 13:36:32 INFO FileFormatWriter: Write Job dbc3aaaf3-f142-4f3f-a019-fb275b78b40e committed. Elapsed time: 64 ms.
25/08/01 13:36:32 INFO FileFormatWriter: Finished processing stats for write job dbc3aaaf3-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/.0.607a15f1-0202-4d99-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/commits/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://tbsm-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' , timestamp
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 identifi
er(?)
25/08/01 13:36:36 INFO CloudStorageOperations$: Spark Connector Master: Begin to process and upload data for 4 partitions: directory=WvLCnThFI CSV
true
25/08/01 13:36:36 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
25/08/01 13:36:36 INFO DAGScheduler: Got job 4 (start at NativeMethodAccessorImpl.java:0) with 4 output partitions
25/08/01 13:36:36 INFO DAGScheduler: Final stage: ResultStage 6 (start at NativeMethodAccessorImpl.java:0)
25/08/01 13:36:39 INFO StageWriter$: 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
25/08/01 13:36:40 INFO DAGScheduler: Got job 5 (start at NativeMethodAccessorImpl.java:0) with 4 output partitions

```

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

	SESSION_ID	USER_ID	CHANNEL_ID	CHANNEL_NAME	SHOW_NAME	GENRE
1	"session_id": "SID535571"	"user_id": "U19750"	"channel_id": "CH038"	"channel_name": "ETV Telugu"	"show_name": "Show_160"	"genre": "
2	"session_id": "SID416211"	"user_id": "U34570"	"channel_id": "CH005"	"channel_name": "Star Bharat"	"show_name": "Show_43"	"genre": "
3	"session_id": "SID410579"	"user_id": "U58333"	"channel_id": "CH046"	"channel_name": "ETV Marathi"	"show_name": "Show_66"	"genre": "
4	"session_id": "SID490667"	"user_id": "U56672"	"channel_id": "CH044"	"channel_name": "Zee Marathi"	"show_name": "Show_171"	"genre": "
5	"session_id": "SID432075"	"user_id": "U15636"	"channel_id": "CH045"	"channel_name": "Zee Bangla"	"show_name": "Show_188"	"genre": "
6	"session_id": "SID633914"	"user_id": "U47032"	"channel_id": "CH034"	"channel_name": "Zee Tamil"	"show_name": "Show_75"	"genre": "
7	"session_id": "SID345951"	"user_id": "U31663"	"channel_id": "CH022"	"channel_name": "ABP News"	"show_name": "Show_119"	"genre": "

Query Details: 501ms
Rows: 835
Query ID: 01be1450-0106-1ea-0...
SESSION_ID: 100% filled
Ask Copilot

Some Queries to run and visualize

1. Total viewership duration per channel

```
SELECT
    channel_name,
    SUM(duration_minutes) AS total_duration
FROM "view"
GROUP BY channel_name
ORDER BY total_duration DESC;
```

The screenshot shows a database query editor with the following details:

- SQL code (lines 9-16):

```
9  -- Some Queries to run and visualize
10 -- 1. Total viewership duration per channel
11 SELECT
12     channel_name,
13     SUM(duration_minutes) AS total_duration
14 FROM "view"
15 GROUP BY channel_name
16 ORDER BY total_duration DESC;
```
- Run button.
- Explain button.
- Cancel button.
- Clear button.
- Create dropdown.
- Reuse query results checkbox (unchecked).
- Query results tab (selected).
- Query stats tab.
- Completed status.
- Time in queue: 100 ms.
- Run time: 585 ms.
- Data scanned: 1.23 MB.
- Results table:

#	channel_name	total_duration
1	Dangal TV	20802
2	ABP News	20604
3	ETV Telugu	20049
4	Zee Anmol	19975

2. Average engagement by device

```
SELECT
    device,
    AVG(engagement_score) AS avg_engagement
FROM "view"
GROUP BY device
ORDER BY avg_engagement DESC;
```

The screenshot shows a database query editor with the following details:

- SQL code (lines 18-29):

```
18  -- 2. Average engagement by device
19  SELECT
20      device,
21      AVG(engagement_score) AS avg_engagement
22  FROM "view"
23  GROUP BY device
24  ORDER BY avg_engagement DESC;
25  -- 3. Daily ad revenue per channel
26  -- 4. Gender-wise average completion percentage
27  -- 5. Most watched genres in each region
28  -- 6. Top 5 channels with highest ad revenue in the past 7 days
29  -- 7. Peak viewership hours by region
```
- Run again button.
- Explain button.
- Cancel button.
- Clear button.
- Create dropdown.
- Reuse query results checkbox (unchecked).
- Query results tab (selected).
- Query stats tab.
- Completed status.
- Time in queue: 115 ms.
- Run time: 646 ms.
- Data scanned: 1.23 MB.
- Results table:

#	device	avg_engagement
1	Tablet	0.606807419100238
2	Mobile	0.6059832802547772
3	Laptop	0.6032895260852263
4	Smart TV	0.6016864510847327

3. Daily ad revenue per channel

```
SELECT
    timestamp AS date,
    channel_name,
    SUM(ad_revenue) AS daily_revenue
FROM "view"
GROUP BY timestamp, channel_name
ORDER BY date, daily_revenue DESC;
```

The screenshot shows a database query interface with the following details:

- Query Editor:** Contains the SQL code for selecting daily ad revenue per channel from a view.
- Run Again:** A button to re-execute the query.
- Explain:** A button to view the query execution plan.
- Cancel:** A button to cancel the query.
- Clear:** A button to clear the query text.
- Create:** A button to create a new query.
- Reuse query results:** An option to reuse results up to 60 minutes ago.
- Query results:** Tab selected, showing the completed status of the query.
- Query stats:** Tab showing run time (1.473 sec), data scanned (1.23 MB), and time in queue (96 ms).
- Results (9,998):** Shows a table with columns: #, date, channel_name, and daily_revenue.
- Search rows:** A search bar for the results table.
- Copy:** A button to copy the results.
- Download results CSV:** A button to download the results as a CSV file.

#	date	channel_name	daily_revenue
1	2025-07-28 10:45:05	Sun TV	502.23
2	2025-07-28 10:46:04	Star Plus	224.85
3	2025-07-28 10:47:55	Star Sports 2	55.53
4	2025-07-28 10:49:18	Star Sports 1	363.19

4. Gender-wise average completion percentage

```
SELECT
    d.gender,
    AVG(v.completion_percentage) AS avg_completion
FROM "view" v
JOIN demo d ON v.user_id = d.user_id
GROUP BY d.gender
ORDER BY avg_completion DESC;
```

The screenshot shows a database query interface with the following details:

- Query Editor:** Contains the SQL code for selecting gender-wise average completion percentage from a view.
- Run again:** A button to re-execute the query.
- Explain:** A button to view the query execution plan.
- Cancel:** A button to cancel the query.
- Clear:** A button to clear the query text.
- Create:** A button to create a new query.
- Reuse query results:** An option to reuse results up to 60 minutes ago.
- Query results:** Tab selected, showing the completed status of the query.
- Query stats:** Tab showing run time (947 ms), data scanned (1.25 MB), and time in queue (67 ms).
- Results (3):** Shows a table with columns: #, gender, and avg_completion.
- Search rows:** A search bar for the results table.
- Copy:** A button to copy the results.
- Download results CSV:** A button to download the results as a CSV file.

#	gender	avg_completion
1	Other	60.09751946607331
2	Male	59.99292634560919
3	Female	59.23642295368489

5. Most watched genres in each region

```
SELECT
    region,
    genre,
    SUM(duration_minutes) AS total_watch_time
FROM "view"
GROUP BY region, genre
ORDER BY region, total_watch_time DESC;
```

The screenshot shows a database query interface with the following details:

- SQL Editor:** The code is identical to the one in the previous block.
- Run Status:** The query has been completed successfully.
- Performance Metrics:** Time in queue: 67 ms, Run time: 587 ms, Data scanned: 1.23 MB.
- Results:** A table titled "Results (42)" showing the top 4 genres in the Central region. The data is as follows:

#	region	genre	total_watch_time
1	Central	Entertainment	34009
2	Central	News	30814
3	Central	Sports	20381
4	Central	Music	14250

6. Top 5 channels with highest ad revenue in the past 7 days

```
SELECT
    channel_name,
    SUM(ad_revenue) AS total_revenue
FROM "view"
WHERE cast("timestamp" as timestamp) >= DATE_ADD('day', -7, CURRENT_DATE)
GROUP BY channel_name
ORDER BY total_revenue DESC
LIMIT 5;
```

The screenshot shows a database query interface with the following details:

- SQL Editor:** The code is identical to the one in the previous block.
- Run Status:** The query has been completed successfully.
- Performance Metrics:** Time in queue: 101 ms, Run time: 581 ms, Data scanned: 1.23 MB.
- Results:** A table titled "Results (5)" showing the top 5 channels with their total ad revenue over the past 7 days. The data is as follows:

#	channel_name	total_revenue
1	Zee Anmol	63740.91000000001
2	Zee Tamil	63614.24
3	ABP News	61879.90000000004
4	Gemini TV	60727.36999999999

7. Peak viewership hours by region

```
SELECT
    region,
    HOUR(CAST("timestamp" AS timestamp)) AS hour_of_day,
    SUM(duration_minutes) AS total_viewing_time
FROM "view"
GROUP BY region, HOUR(CAST("timestamp" AS timestamp))
ORDER BY region, total_viewing_time DESC;
```

The screenshot shows a SQL query editor with the following details:

- SQL code (Ln 62 to Ln 73):

```
62 -- 7. Peak viewership hours by region
63 SELECT
64     region,
65     HOUR(CAST("timestamp" AS timestamp)) AS hour_of_day,
66     SUM(duration_minutes) AS total_viewing_time
67 FROM "view"
68 GROUP BY region, HOUR(CAST("timestamp" AS timestamp))
69 ORDER BY region, total_viewing_time DESC;
70
71 -- 8. Which age group watches the most live content?
72 -- 9. Subscription type driving most revenue per channel
73 -- 10. High engagement sessions (more than 90% completion and >1 min)
```
- Run status: Completed
- Time in queue: 101 ms
- Run time: 448 ms
- Data scanned: 1.23 MB
- Results (168 rows):

#	region	hour_of_day	total_viewing_time
1	Central	19	6962
2	Central	21	6542
3	Central	5	6334
4	Central	3	6271

8. Which age group watches the most live content?

```
SELECT
    d.age_group,
    SUM(v.duration_minutes) AS total_live_viewing
FROM "view" v
JOIN "demo" d ON v.user_id = d.user_id
WHERE v.is_live = true
GROUP BY d.age_group
ORDER BY total_live_viewing DESC;
```

The screenshot shows a SQL query editor with the following details:

- SQL code (Ln 71 to Ln 80):

```
71 -- 8. Which age group watches the most live content?
72 SELECT
73     d.age_group,
74     SUM(v.duration_minutes) AS total_live_viewing
75     FROM "view" v
76     JOIN "demo" d ON v.user_id = d.user_id
77     WHERE v.is_live = true
78     GROUP BY d.age_group
79     ORDER BY total_live_viewing DESC;
80
```
- Run status: Completed
- Time in queue: 76 ms
- Run time: 1.418 sec
- Data scanned: 1.25 MB
- Results (6 rows):

#	age_group	total_live_viewing
1	36-45	85226
2	26-35	83474
3	<18	80959
4	60+	80708

9. Subscription type driving most revenue per channel

```
SELECT
    channel_name,
    subscription_type,
    SUM(ad_revenue) AS revenue
FROM "view"
GROUP BY channel_name, subscription_type
ORDER BY revenue DESC;
```

The screenshot shows a database query interface with the following details:

- Code Area:** Displays the SQL code for step 9.
- Execution Buttons:** Includes "Run again", "Explain", "Cancel", "Clear", and "Create".
- Reuse Query Results:** A button to reuse results up to 60 minutes ago.
- Results Tab:** Active tab, showing "Completed" status.
- Query Stats:** Time in queue: 100 ms, Run time: 420 ms, Data scanned: 1.23 MB.
- Results Table:**

#	channel_name	subscription_type	revenue
1	Zee Tamil	Basic	27020.290000000008
2	Sony Sports Ten 1	Basic	24386.55
3	Zee News	Basic	24348.050000000003
4	Star Vijay	Free	24085.420000000006

10. High engagement sessions (more than 90% completion and >1 min)

```
SELECT *
FROM "view"
WHERE completion_percentage > 90
AND duration_minutes > 1;
```

The screenshot shows a database query interface with the following details:

- Code Area:** Displays the SQL code for step 10.
- Execution Buttons:** Includes "Run again", "Explain", "Cancel", "Clear", and "Create".
- Reuse Query Results:** A button to reuse results up to 60 minutes ago.
- Results Tab:** Active tab, showing "Completed" status.
- Query Stats:** Time in queue: 60 ms, Run time: 445 ms, Data scanned: 1.23 MB.
- Results Table:**

#	session_id	user_id	channel_id	channel_name	show_name	genre	timestamp	duration_minutes	region
1	SID885804	U50474	CH036	Star Maa	Show_117	Kids	2025-08-03 12:49:08	69	Northeast
2	SID443946	U30164	CH009	Colors Rishtey	Show_62	Sports	2025-07-29 14:40:36	38	Northeast
3	SID679454	U50785	CH047	DD Sahyadri	Show_35	Movies	2025-07-29 19:12:52	71	South
4	SID959109	U54883	CH029	Zee News	Show_180	News	2025-07-30 12:58:15	172	North

11. Channels with above-average ad revenue per day

```
WITH avg_revenue AS (
    SELECT AVG(ad_revenue) AS avg_rev FROM "view"
)
SELECT *
FROM "view", avg_revenue
WHERE ad_revenue > avg_rev;
```

The screenshot shows a database query interface with the following details:

- SQL Editor:** The code for query 11 is pasted into the editor.
- Run Status:** The status bar indicates the query is completed.
- Query Results:** The results table has 4 rows and the following data:

#	session_id	user_id	channel_id	channel_name	show_name	genre	timestamp	duration_minutes	region
1	SID224399	U16570	CH044	Zee Marathi	Show_88	Sports	2025-08-04 01:04:29	155	Pan-India
2	SID885804	U50474	CH036	Star Maa	Show_117	Kids	2025-08-03 12:49:08	69	Northeast
3	SID690117	U53655	CH035	Colors Tamil	Show_122	News	2025-08-03 03:52:47	118	South
4	SID280239	U45958	CH046	ETV Marathi	Show_114	Entertainment	2025-07-28 13:17:58	153	South

12. Most engaging show for female users in metro regions

```
SELECT
    v.show_name,
    AVG(v.engagement_score) AS avg_engagement
FROM "view" v
JOIN "demo" d ON v.user_id = d.user_id
WHERE d.gender = 'Female' AND d.region = 'Metro'
GROUP BY v.show_name
ORDER BY avg_engagement DESC;
```

The screenshot shows a database query interface with the following details:

- SQL Editor:** The code for query 12 is pasted into the editor.
- Run Status:** The status bar indicates the query is completed.
- Query Results:** The results table has 0 rows, indicating no results found.

#	show_name	avg_engagement
No results		

Message: Run a query to view results

13. Genre-wise ad revenue and completion comparison

```
SELECT
    genre,
    AVG(ad_revenue) AS avg_revenue,
    AVG(completion_percentage) AS avg_completion
FROM "view"
GROUP BY genre;
```

```
114 -- 13. Genre-wise ad revenue and completion comparison
115 SELECT
116     genre,
117     AVG(ad_revenue) AS avg_revenue,
118     AVG(completion_percentage) AS avg_completion
119     FROM "view"
120     GROUP BY genre;
121
```

SQL Ln 120, Col 16

[Run again](#) [Explain](#) [Cancel](#) [Clear](#) [Create](#)

Reuse query results
up to 60 minutes ago

[Query results](#) [Query stats](#)

[Completed](#)

Time in queue: 55 ms Run time: 751 ms Data scanned: 1.23 MB

[Copy](#)

[Download results CSV](#)

Results (6)

[Search rows](#)

#	genre	avg_revenue	avg_completion
1	News	262.52720576131685	59.0150452674898
2	Entertainment	266.1333906194058	59.96267096312217
3	Sports	259.96539790252905	59.83132634176424
4	Music	266.365811789038	60.036349534643186

14. Channels with highest buffer counts but good engagement

```
SELECT
    channel_name,
    buffer_count,
    engagement_score
FROM "view"
WHERE engagement_score > 0.5
ORDER BY buffer_count DESC;
```

```
122 -- 14. Channels with highest buffer counts but good engagement
123 SELECT
124     channel_name,
125     buffer_count,
126     engagement_score
127     FROM "view"
128     WHERE engagement_score > 0.5
129     ORDER BY buffer_count DESC;
130
```

SQL Ln 129, Col 28

[Run again](#) [Explain](#) [Cancel](#) [Clear](#) [Create](#)

Reuse query results
up to 60 minutes ago

[Query results](#) [Query stats](#)

[Completed](#)

Time in queue: 108 ms Run time: 472 ms Data scanned: 1.23 MB

[Copy](#)

[Download results CSV](#)

Results (6,288)

[Search rows](#)

#	channel_name	buffer_count	engagement_score
1	Sony Entertainment Television	3	0.69
2	Dangal TV	3	0.9
3	Star Sports 3	3	0.51
4	Colors Kannada	3	0.81

15. which age group watches the most across all the channels

```
SELECT
    d.age_group,
    SUM(v.duration_minutes) AS total_watch_time
FROM "view" v
JOIN "demo" d ON v.user_id = d.user_id
GROUP BY d.age_group
ORDER BY total_watch_time DESC;
```

The screenshot shows a SQL query editor with the following details:

- SQL code (131-139):

```
131 -- 15. which age group watches the most across all the channels
132 SELECT
133     d.age_group,
134     SUM(v.duration_minutes) AS total_watch_time
135 FROM "view" v
136 JOIN "demo" d ON v.user_id = d.user_id
137 GROUP BY d.age_group
138 ORDER BY total_watch_time DESC;
139
```
- Run status: Completed
- Query stats: Time in queue: 103 ms, Run time: 895 ms, Data scanned: 1.25 MB
- Results (6 rows):

#	age_group	total_watch_time
1	26-35	164337
2	<18	164150
3	60+	161140
4	36-45	157145

16. what is the average viewing time per user segmented by region

```
SELECT
    region,
    AVG(duration_minutes) AS avg_viewing_time
FROM "view"
GROUP BY region
ORDER BY avg_viewing_time DESC;
```

The screenshot shows a SQL query editor with the following details:

- SQL code (140-147):

```
140 -- 16. what is the average viewing time per user segmented by region
141 SELECT
142     region,
143     AVG(duration_minutes) AS avg_viewing_time
144 FROM "view"
145 GROUP BY region
146 ORDER BY avg_viewing_time DESC;
147
```
- Run status: Completed
- Query stats: Time in queue: 62 ms, Run time: 538 ms, Data scanned: 1.23 MB
- Results (7 rows):

#	region	avg_viewing_time
1	West	91.09045936395759
2	Pan-India	90.75818303273213
3	Northeast	90.41439552760308
4	Central	90.2373123659757

17. Do mobile users watch more content than Smart tv users

```
SELECT
    device,
    SUM(duration_minutes) AS total_watch_time
FROM "view"
WHERE device IN ('Mobile', 'Smart TV')
GROUP BY device
ORDER BY total_watch_time DESC;
```

The screenshot shows a database interface with the following details:

- Query ID: 148 -- 17. Do mobile users watch more content than Smart tv users
- SQL Statement:

```
149   SELECT
150     device,
151     SUM(duration_minutes) AS total_watch_time
152   FROM "view"
153 WHERE device IN ('Mobile', 'Smart TV')
154 GROUP BY device
155 ORDER BY total_watch_time DESC;
```

- Run Status: Completed
- Time in queue: 101 ms
- Run time: 405 ms
- Data scanned: 1.23 MB
- Results (2):

#	device	total_watch_time
1	Mobile	224484
2	Smart TV	219785

18. What is the total ad revenue per channel per day

```
SELECT
    channel_name,
    CAST(parse_datetime("timestamp", 'yyyy-MM-dd HH:mm:ss') AS date) AS date,
    SUM(ad_revenue) AS daily_revenue
FROM "view"
GROUP BY
    channel_name,
    CAST(parse_datetime("timestamp", 'yyyy-MM-dd HH:mm:ss') AS date)
ORDER BY date, daily_revenue DESC;
```

The screenshot shows a database interface with the following details:

- Query ID: 157 -- 18. what is the total ad revenue per channel per day
- SQL Statement:

```
158   SELECT
159     channel_name,
160     CAST(parse_datetime("timestamp", 'yyyy-MM-dd HH:mm:ss') AS date) AS date,
161     SUM(ad_revenue) AS daily_revenue
162   FROM "view"
163   GROUP BY
164     channel_name,
165     CAST(parse_datetime("timestamp", 'yyyy-MM-dd HH:mm:ss') AS date)
166   ORDER BY date, daily_revenue DESC;
```

- Run Status: Completed
- Time in queue: 61 ms
- Run time: 495 ms
- Data scanned: 1.23 MB
- Results (400):

#	channel_name	date	daily_revenue
1	Colors Rishtey	2025-07-28	7214.97
2	Sony Sports Ten 1	2025-07-28	6898.12
3	Aaj Tak	2025-07-28	6465.460000000001
4	DD Sahyadri	2025-07-28	6317.139999999985

19. what are the peak hours of viewership across India

```
SELECT
    HOUR(CAST("timestamp" AS timestamp)) AS hour_of_day,
    SUM(duration_minutes) AS total_viewing_time
FROM "view"
GROUP BY HOUR(CAST("timestamp" AS timestamp))
ORDER BY total_viewing_time DESC;
```

The screenshot shows a database query interface with the following details:

- Query text:

```
169 -- 19. what are the peak hours of viewership across India
170   SELECT
171     HOUR(CAST("timestamp" AS timestamp)) AS hour_of_day,
172     SUM(duration_minutes) AS total_viewing_time
173   FROM "view"
174   GROUP BY HOUR(CAST("timestamp" AS timestamp))
175   ORDER BY total_viewing_time DESC;
176
177 -- 20. how does content genre popularity vary by gender
178 -- 21. which channels are more popular among premium subscribers
179 -- 22. are certain shows driving higher engagement among female users in metro cities
180
```
- Run status: Completed (24 rows)
- Time in queue: 99 ms, Run time: 503 ms, Data scanned: 1.23 MB
- Results table:

#	hour_of_day	total_viewing_time
1	21	42743
2	16	42139
3	17	40502
4	12	40457

20. how does content genre popularity vary by gender

```
SELECT
    d.gender,
    v.genre,
    SUM(v.duration_minutes) AS total_watch_time
FROM "view" v
JOIN "demo" d ON v.user_id = d.user_id
GROUP BY d.gender, v.genre
ORDER BY d.gender, total_watch_time DESC;
```

The screenshot shows a database query interface with the following details:

- Query text:

```
177 -- 20. how does content genre popularity vary by gender
178   SELECT
179     d.gender,
180     v.genre,
181     SUM(v.duration_minutes) AS total_watch_time
182   FROM "view" v
183   JOIN "demo" d ON v.user_id = d.user_id
184   GROUP BY d.gender, v.genre
185   ORDER BY d.gender, total_watch_time DESC;
186
```
- Run status: Completed (18 rows)
- Time in queue: 61 ms, Run time: 3.437 sec, Data scanned: 1.25 MB
- Results table:

#	gender	genre	total_watch_time
1	Female	Entertainment	85259
2	Female	News	79696
3	Female	Sports	50541
4	Female	Movies	37626

21. which channels are more popular among premium subscribers

```
SELECT
    channel_name,
    SUM(duration_minutes) AS total_watch_time
FROM "view"
WHERE subscription_type = 'Premium'
GROUP BY channel_name
ORDER BY total_watch_time DESC;
```

The screenshot shows a SQL query editor with the following details:

- Query text:

```
187 -- 21. which channels are more popular among premium subscribers
188 SELECT
189     channel_name,
190     SUM(duration_minutes) AS total_watch_time
191 FROM "view"
192 WHERE subscription_type = 'Premium'
193 GROUP BY channel_name
194 ORDER BY total_watch_time DESC;
195
```
- Execution status: Completed (green bar)
- Time metrics: Time in queue: 64 ms, Run time: 457 ms, Data scanned: 1.23 MB
- Results table:

#	channel_name	total_watch_time
1	DD National	7915
2	Zee Anmol	7519
3	Zee News	7242
4	Star Sports 1	7232

22. are certain shows driving higher engagement among female users in metro cities

```
SELECT
    show_name,
    AVG(engagement_score) AS avg_engagement
FROM "view" v
JOIN "demo" d ON v.user_id = d.user_id
WHERE d.gender = 'Female' AND d.region = 'Metro'
GROUP BY show_name
ORDER BY avg_engagement DESC;
```

The screenshot shows a SQL query editor with the following details:

- Query text:

```
196 -- 22. are certain shows driving higher engagement among female users in metro cities
197 SELECT
198     show_name,
199     AVG(engagement_score) AS avg_engagement
200     FROM "view" v
201     JOIN "demo" d ON v.user_id = d.user_id
202     WHERE d.gender = 'Female' AND d.region = 'Metro'
203     GROUP BY show_name
204     ORDER BY avg_engagement DESC;
205
```
- Execution status: Completed (green bar)
- Time metrics: Time in queue: 102 ms, Run time: 1.057 sec, Data scanned: 0.15 KB
- Results table:

#	show_name	avg_engagement
No results		

Summary