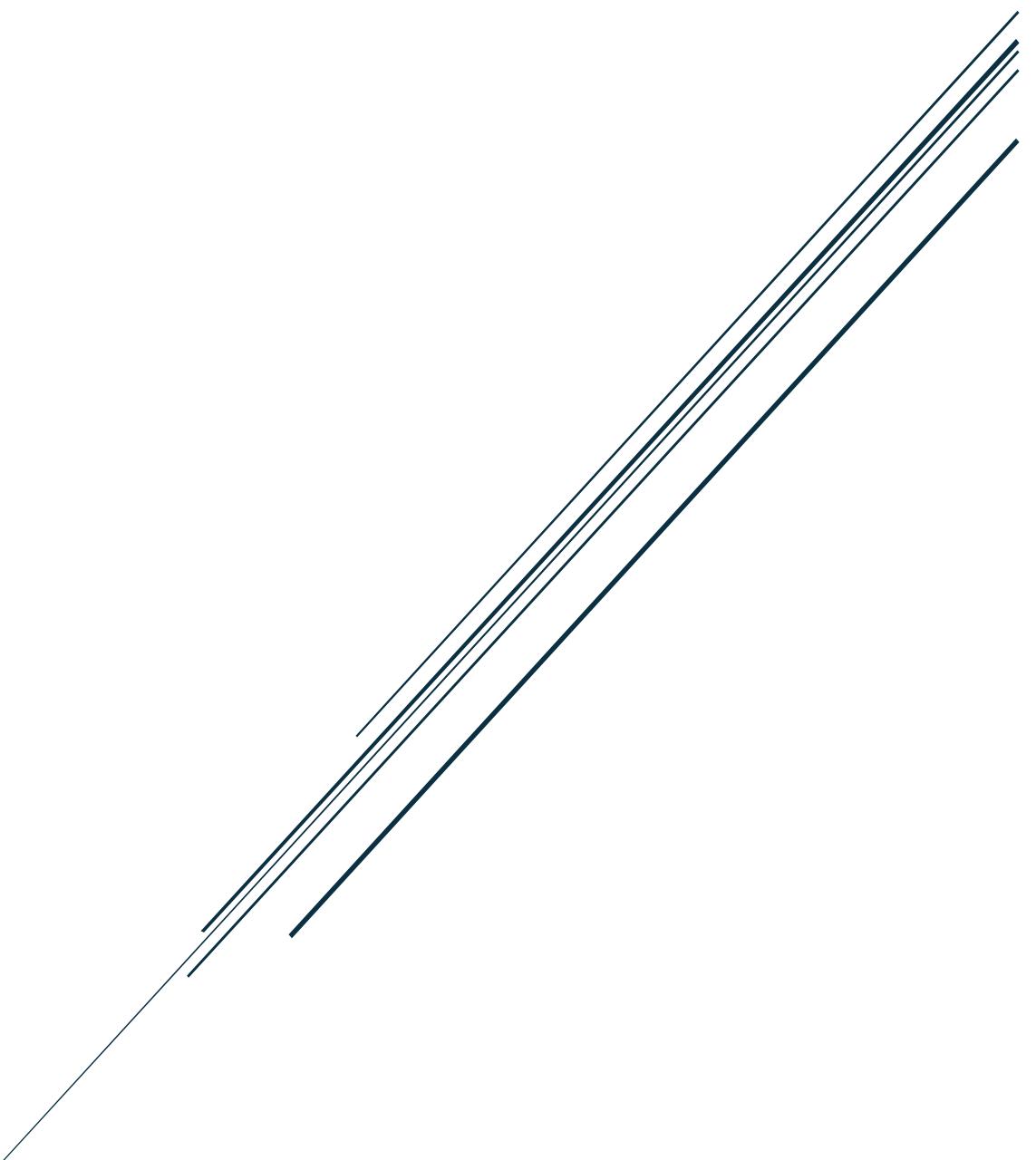


Media Stream Analytics (Real-Time Intelligence For Media Decisions)

Project Report

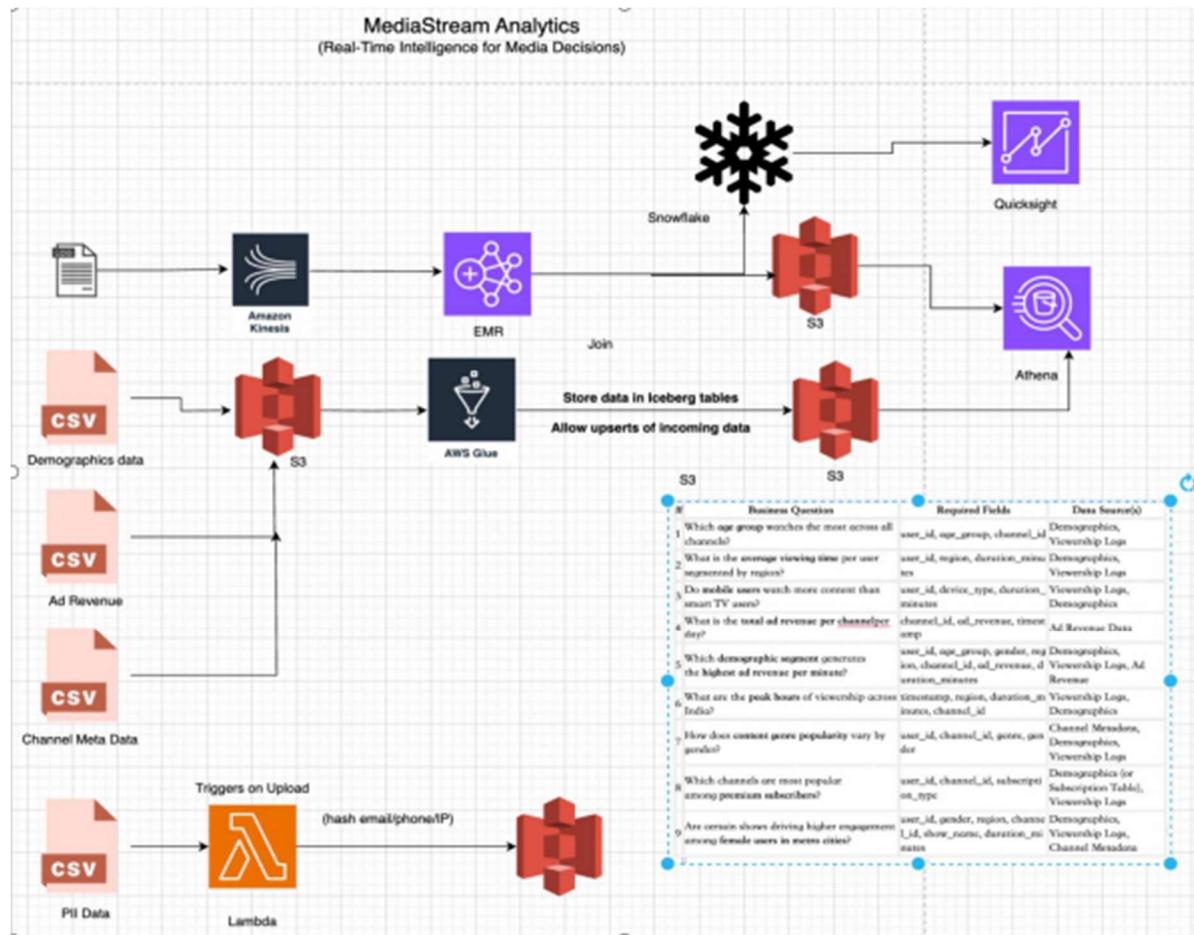


By
Aahash Kamble

Problem Statement:

The Media Stream Analytics project aims to address the growing need for real-time and batch analytics in the media industry by leveraging scalable cloud-based big data technologies. With the increasing volume of viewership data, ad revenue metrics, and user demographics, media companies require a robust solution to extract actionable insights that can drive programming decisions, optimize ad placements, and enhance user engagement.

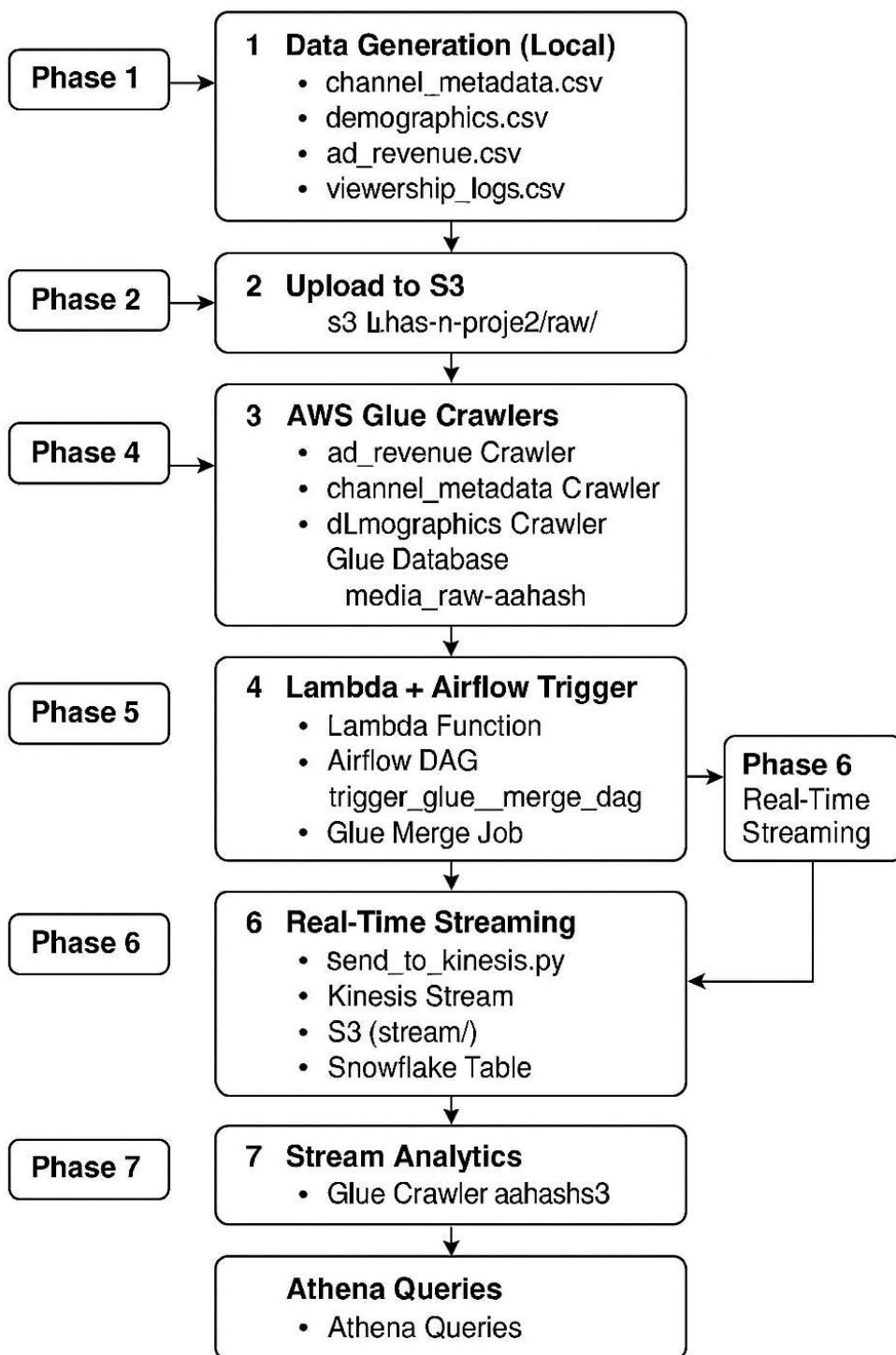
Data Flow Architecture:



Key Technologies Used

- AWS Services:** S3, Glue, Lambda, EMR, Kinesis, Athena, MWAA
- Data Formats:** CSV, Iceberg, Parquet
- Processing Engines:** Spark (Batch & Streaming)
- Analytics:** Athena SQL, Snowflake
- Automation:** Airflow DAGs, Lambda triggers

Flow of Media Stream Analytics (Real-Time Intelligence for Media Decisions)



PHASE 1: DATA GENERATION (LOCAL)

```
vi generate_media_data.py
```

```
python3 generate_media_data.py
```

This creates the following CSV files:

- channel_metadata.csv
- demographics.csv
- ad_revenue.csv
- viewership_logs.csv

Code:

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

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

GENRES = ["Entertainment", "News", "Sports", "Kids", "Music", "Movies"]
LANGUAGES = ["Hindi", "English", "Tamil", "Telugu", "Malayalam", "Kannada", "Marathi", "Bengali"]
REGIONS = ["North", "South", "East", "West", "Central", "Northeast", "Pan-India"]
DEVICES = ["Mobile", "Tablet", "Smart TV", "Laptop"]
PLATFORMS = ["Android", "iOS", "Web", "FireTV", "Roku"]

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

```

"Times Now": ("News", "English"), "CNN-News18": ("News", "English"),
"Zee News": ("News", "Hindi"), "ABP News": ("News", "Hindi"),
"Star Vijay": ("Entertainment", "Tamil"), "KTV": ("Movies", "Tamil"),
}

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


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

def generate_demographics(num_users=5000):
    demographics = []
    user_ids = set()
    for _ in range(num_users):
        user_id = f"U{10000 + random.randint(0, 49999)}"
        gender = random.choice(["Male", "Female", "Other"])
        age_group = random.choice(["<18", "18-25", "26-35", "36-45", "46-60", "60+"])
        region = random.choice(REGIONS)
        subscription_type = random.choice(["Free", "Basic", "Premium"])
        demographics.append([user_id, gender, age_group, region, subscription_type])
        user_ids.add(user_id)
    with open("demographics.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["user_id", "gender", "age_group", "region", "subscription_type"])
        writer.writerows(demographics)
    return user_ids

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

```

```

writer = csv.writer(f)
writer.writerow(["channel_id", "channel_name", "date", "ad_revenue"])
writer.writerows(ad_revenue)

def generate_viewership_logs(channel_id_map, user_ids, num_records=10000):
    logs = []
    start_time = datetime.now() - timedelta(days=7)
    user_id_list = list(user_ids)
    for _ in range(num_records):
        user_id = random.choice(user_id_list)
        channel_name = random.choice(CHANNEL_NAMES)
        channel_id = channel_id_map[channel_name]
        timestamp = start_time + timedelta(seconds=random.randint(0, 604800))
        duration = random.randint(1, 180)
        region = random.choice(REGIONS)
        subscription = random.choice(["Free", "Basic", "Premium"])
        device = random.choice(DEVICES)
        platform = random.choice(PLATFORMS)
        is_live = random.choice([True, False])
        genre = CHANNEL_ATTRIBUTES.get(channel_name, (random.choice(GENRES),))[0]
        ads_watched = random.randint(0, 5)
        ad_revenue = round(ads_watched * random.uniform(10, 200), 2)
        engagement = round(random.uniform(0.2, 1.0), 2)
        buffer_count = random.randint(0, 3)
        completion_pct = round(random.uniform(20, 100), 2)
        session_id = f"SID {random.randint(100000, 999999)}"
        show_name = f>Show_{random.randint(1, 200)}"
        logs.append([
            session_id, user_id, channel_id, channel_name, show_name, genre, timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            duration, region, subscription, device, platform, is_live, ads_watched, ad_revenue,
            engagement, buffer_count, completion_pct
        ])
    with open("viewership_logs.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([
            "session_id", "user_id", "channel_id", "channel_name", "show_name", "genre", "timestamp",
            "duration_minutes", "region", "subscription_type", "device", "platform", "is_live",
            "ads_watched", "ad_revenue", "engagement_score", "buffer_count", "completion_percentage"
        ])
        writer.writerows(logs)

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

```

PHASE 2: UPLOAD FILES TO S3

Step 2: Create S3 folder structure

Target bucket: s3://Aahash-project2

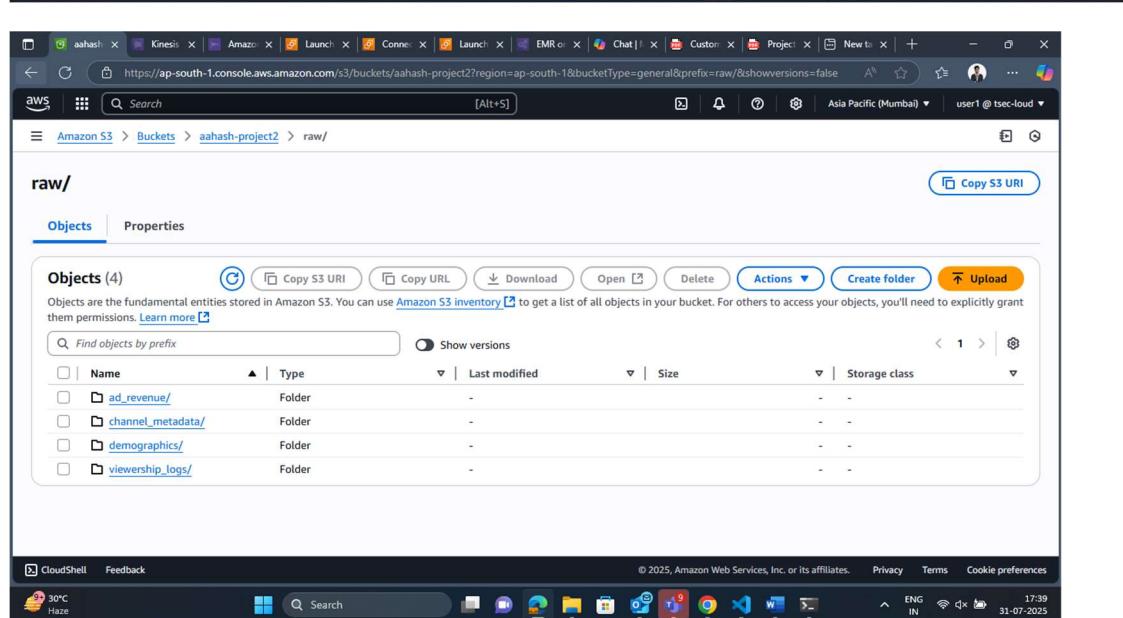
aahash-project2/

```
└── raw/
    ├── channel_metadata/
    ├── demographics/
    ├── ad_revenue/
    └── viewership_logs/
```

Step 3: Upload the files to bucket

```
C:\Users\aaahash.kamble\Downloads>ssh -i "keypair-aahash.pem" ec2-user@ec2-13-201-227-184.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-201-227-184.ap-south-1.compute.amazonaws.com (13.201.227.184)' can't be established.
ED25519 key fingerprint is SHA256:pcCygCFYmnoyLks5+/f0KSXJgo+jeEl0M2DlDeZfic.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-201-227-184.ap-south-1.compute.amazonaws.com' (ED25519) to the list of known hosts.

[ec2-user@ip-172-31-9-15 ~]$ aws configure
AWS Access Key ID [None]: AKIAQEBHVDHAQ2JM00Z
AWS Secret Access Key [None]: RNIhVgqLBwgAEejfYp3PPh4lZjn1DJPpwTL9tKw
Default region name [None]:
Default output format [None]:
[ec2-user@ip-172-31-9-15 ~]$ vi generate_media_data.py
[ec2-user@ip-172-31-9-15 ~]$ python3 generate_media_data.py
Writing channel meta data.....
Writing channel meta data completed .....
[ec2-user@ip-172-31-9-15 ~]$ aws s3 cp channel_metadata.csv s3://aahash-project2/raw/channel_metadata/
upload: ./channel_metadata.csv to s3://aahash-project2/raw/channel_metadata/channel_metadata.csv
[ec2-user@ip-172-31-9-15 ~]$ aws s3 cp demographics.csv s3://aahash-project2/raw/demographics/
upload: ./demographics.csv to s3://aahash-project2/raw/demographics/demographics.csv
[ec2-user@ip-172-31-9-15 ~]$ aws s3 cp ad_revenue.csv s3://aahash-project2/raw/ad_revenue/
upload: ./ad_revenue.csv to s3://aahash-project2/raw/ad_revenue/ad_revenue.csv
[ec2-user@ip-172-31-9-15 ~]$ aws s3 cp viewership_logs.csv s3://aahash-project2/raw/viewership_logs/
upload: ./viewership_logs.csv to s3://aahash-project2/raw/viewership_logs/viewership_logs.csv
[ec2-user@ip-172-31-9-15 ~]$
```



The screenshot shows the AWS CloudShell interface running on a Windows desktop. The terminal window displays the execution of the AWS CLI commands to upload files from the local machine to the S3 bucket. The terminal window has a dark theme with white text. The desktop taskbar at the bottom shows various icons for Microsoft Office applications like Word, Excel, and Powerpoint, as well as browser and system icons.

The screenshot also shows the AWS Management Console in the background, specifically the S3 Buckets page. It lists the 'raw' folder under 'aahash-project2'. The 'Objects' tab is selected, showing four items: 'ad_revenue/' (Folder), 'channel_metadata/' (Folder), 'demographics/' (Folder), and 'viewership_logs/' (Folder). Each item has a checkbox next to it and columns for Name, Type, Last modified, Size, and Storage class. The 'Actions' dropdown menu is visible above the table.

PHASE 3: AWS Glue Crawler Setup – Media Stream Analytics

Created Glue Database:

Glue Database

- Name: media_raw-aahash

Steps to Create Glue Crawlers for 3 csv

For each dataset:

Crawler 1: aahash-crawler-ad-revenue

The screenshot shows the AWS Glue Crawler properties page for 'aahash-crawler-ad-revenue'. The crawler is named 'aahash-crawler-ad-revenue', has an IAM role 'AWSGlueServiceRole-Aahash1', and is associated with the database 'media_raw-aahash'. The state is 'READY'. The 'Crawler runs' section shows 0 runs. The left sidebar includes options like 'Getting started', 'ETL jobs', 'Data Catalog', and 'Crawlers'.

Crawler 2: aahash-crawler-channel-meta

The screenshot shows the AWS Glue Crawler properties page for 'aahash-crawler-channel-meta'. A green banner at the top indicates 'One crawler successfully created' and lists the newly created crawler. The crawler properties are identical to the first one: name 'aahash-crawler-channel-meta', IAM role 'AWSGlueServiceRole-Aahash1', database 'media_raw-aahash', and state 'READY'. The 'Crawler runs' section shows 0 runs. The left sidebar includes options like 'Getting started', 'ETL jobs', 'Data Catalog', and 'Crawlers'.

Crawler 3: aahash-crawler-demographics

Crawler properties

Name aahash-crawler-demographics	IAM role AWSGlueServiceRole-Aahash1	Database media_raw-aahash	State READY
Description -	Security configuration -	Lake Formation configuration -	Table prefix demographics
Maximum table threshold -			

Crawler runs (0)

After creating all 3 crawlers:

1. I have Run each crawler manually from the Glue Console.

Crawlers (41) Info

Name	State	Last run	Log
aahash-crawler-ad-revenue	Ready	Succeeded	View log
aahash-crawler-channel-meta	Ready	Succeeded	View log
aahash-crawler-demographics	Ready	Succeeded	View log

2. Verified the tables are created in the **media_raw-aahash** Glue database.

media_raw-aahash

Database properties

Name media_raw-aahash	Description -	Location -	Created on (UTC) July 31, 2025 at 12:11:32
--------------------------	------------------	---------------	---

Tables (3)

Name	Database	Location	Classification	Deprecation	Action
ad_revenue	media_raw-aahash	s3://aahash-pro	CSV	-	Table data
channel_metadata	media_raw-aahash	s3://aahash-pro	CSV	-	Table data
demographics	media_raw-aahash	s3://aahash-pro	CSV	-	Table data

PHASE 4: Transform & Write to Iceberg Format (Ad Revenue, Channel Metadata, Demographics)

Transform raw CSV data from S3 and write it to Iceberg-format tables stored in S3 using AWS Glue, with schema evolution and ACID support.

Raw Data Location (Input):

All from bucket: s3://aahash-project2/media/raw/

- ad_revenue/
- channel_metadata/
- demographics/

AWS Glue Studio (ETL jobs in Spark)

Apache Iceberg (table format)

AWS Glue Data Catalog (catalog for Iceberg tables)

S3 (aahash-project2/media/iceberg/) as Iceberg storage

Target Folder Structure

Output Iceberg tables go here:

s3://aahash-project2/media/iceberg/ad_revenue/

s3://aahash-project2/media/iceberg/channel_metadata/

s3://aahash-project2/media/iceberg/demographics/

Step 1: Created a New Glue ETL Job for all ad_revenue, channel_metadata/ demographics/

The screenshot shows the AWS Glue Studio interface for creating a new ETL job. The job name is "aahash-transform-ad-revenue-to-iceberg". The "Basic properties" section includes fields for Name, Description (which is empty), IAM Role (set to "AWSGlueServiceRole-Aahash"), and Type (set to "ETL job"). The "Job details" tab is currently selected. Other tabs include Script, Runs, Data quality, Schedules, and Version Control. The top right corner shows the last modified date (31/7/2025, 7:48:45 pm) and buttons for Actions, Save, and Run.

Step 2: Script for Iceberg Transformation

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from awsglue.job import Job
from pyspark.sql import SparkSession

# Accept JOB_NAME as argument
```

```

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

# Initialize contexts ONCE
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Set Iceberg-specific configurations
spark.conf.set("spark.sql.catalog.glue_catalog", "org.apache.iceberg.spark.SparkCatalog")
spark.conf.set("spark.sql.catalog.glue_catalog.catalog-impl", "org.apache.iceberg.aws.glue.GlueCatalog")
spark.conf.set("spark.sql.catalog.glue_catalog.warehouse", "s3://aahash-project2/media/iceberg/")
spark.conf.set("spark.sql.catalog.glue_catalog.io-impl", "org.apache.iceberg.aws.s3.S3FileIO")

# Load raw tables from Glue Data Catalog
try:
    # Load and transform ad_revenue data
    ad_revenue_raw = spark.table("`media_raw-aahash`.ad_revenue")
    ad_revenue_df = ad_revenue_raw.withColumn("date", ad_revenue_raw["date"].cast("date"))
    print(f"Ad revenue columns: {ad_revenue_df.columns}")

    # Load channel metadata
    channel_metadata_df = spark.table("`media_raw-aahash`.channel_metadata")
    print(f"Channel metadata columns: {channel_metadata_df.columns}")

    # Load demographics and fix column names
    demographics_raw = spark.table("media_raw-aahash`.demographics")
    print(f"Demographics raw columns: {demographics_raw.columns}")

    # Map generic column names to proper names
    # Assuming order: user_id, gender, age_group, region, subscription_type
    demographics_df = demographics_raw \
        .withColumnRenamed("col0", "user_id") \
        .withColumnRenamed("col1", "gender") \
        .withColumnRenamed("col2", "age_group") \
        .withColumnRenamed("col3", "region") \
        .withColumnRenamed("col4", "subscription_type")

    print(f"Demographics after renaming: {demographics_df.columns}")

    # Show sample to verify the mapping is correct
    print("Demographics sample data:")
    demographics_df.show(5)

    print("Successfully loaded and transformed source tables")
except Exception as e:
    print(f"Error loading source tables: {e}")
    raise
except Exception as e:
    print(f"Error loading source tables: {e}")
    raise

# Create Iceberg Database and Tables
try:
    # Create the database first
    print("Creating media_iceberg database...")
    create_db_sql = "CREATE DATABASE IF NOT EXISTS `glue_catalog`.`media_iceberg` LOCATION 's3://aahash-project2/media/iceberg/'"
    spark.sql(create_db_sql)
    print("Database media_iceberg created/verified")

    # Create ad_revenue table
    print("Creating ad_revenue table...")
    ad_revenue_sql = """CREATE TABLE IF NOT EXISTS `glue_catalog`.`media_iceberg`.`ad_revenue` (

```

```

channel_id STRING,
channel_name STRING,
date DATE,
ad_revenue FLOAT
) USING iceberg LOCATION 's3://aahash-project2/media/iceberg/ad_revenue'"""
spark.sql(ad_revenue_sql)
print("Ad revenue table created/verified")

# Create channel_metadata table
print("Creating channel_metadata table...")
channel_sql = """CREATE TABLE IF NOT EXISTS `glue_catalog`.`media_iceberg`.`channel_metadata` (
    channel_id STRING,
    channel_name STRING,
    genre STRING,
    language STRING,
    launch_year INT
) USING iceberg LOCATION 's3://aahash-project2/media/iceberg/channel_metadata'"""
spark.sql(channel_sql)
print("Channel metadata table created/verified")

# Create demographics table
print("Creating demographics table...")
demographics_sql = """CREATE TABLE IF NOT EXISTS `glue_catalog`.`media_iceberg`.`demographics` (
    user_id STRING,
    gender STRING,
    age_group STRING,
    region STRING,
    subscription_type STRING
) USING iceberg LOCATION 's3://aahash-project2/media/iceberg/demographics'"""
spark.sql(demographics_sql)
print("Demographics table created/verified")
print("All Iceberg tables created/verified successfully")

except Exception as e:
    print(f"Error creating Iceberg tables: {e}")
    raise

# Write DataFrames to Iceberg Tables
try:
    # Write ad_revenue data
    ad_revenue_df.writeTo("glue_catalog.media_iceberg.ad_revenue").using("iceberg").option("overwrite-mode",
"dynamic").tableProperty("format-version", "2").append()
    print("Ad revenue data written successfully")

    # Write channel_metadata data
    channel_metadata_df.writeTo("glue_catalog.media_iceberg.channel_metadata").using("iceberg").option("overwrite-mode",
"dynamic").tableProperty("format-version", "2").append()
    print("Channel metadata written successfully")

    # Write demographics data
    demographics_df.writeTo("glue_catalog.media_iceberg.demographics").using("iceberg").option("overwrite-mode",
"dynamic").tableProperty("format-version", "2").append()
    print("Demographics data written successfully")

except Exception as e:
    print(f"Error writing to Iceberg tables: {e}")
    raise

# Job Commit
print("Job completed successfully")
job.commit()

```

The screenshot shows the AWS Glue Job History page. The job name is 'aahash-transform-ad-revenue-to-iceberg'. There are 13 job runs listed, with one marked as 'Succeeded' and four marked as 'Failed'. The 'Runs' tab is selected. The table includes columns for Run status, Retries, Start time (Local), End time (Local), Duration, Capacity (DPUs), Worker type, and Glue version.

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
SUCCEEDED	0	07/31/2025 19:48:45	07/31/2025 19:50:10	1 m 17 s	5 DPUs	G.1X	5.0
FAILED	0	07/31/2025 19:03:10	07/31/2025 19:04:17	59 s	5 DPUs	G.1X	5.0
FAILED	0	07/31/2025 19:00:33	07/31/2025 19:01:56	1 m 15 s	5 DPUs	G.1X	5.0
FAILED	0	07/31/2025 18:58:48	07/31/2025 18:59:53	56 s	5 DPUs	G.1X	5.0
FAILED	0	07/31/2025 18:56:21	07/31/2025 18:57:42	1 m 13 s	5 DPUs	G.1X	5.0

Data is generated and send to the bucket successfully

The screenshot shows the Amazon S3 console. The bucket path is 'Amazon S3 > Buckets > aahash-project2 > media_ > iceberg_ > ad_revenue_ > data/'. The 'Objects' tab is selected, showing 48 objects. All objects are of type 'parquet' and have a size of 1.5 KB. The storage class is 'Standard'. The objects are named with a timestamp and a unique identifier.

Name	Type	Last modified	Size	Storage class
00000-0-959a9919-8b34-427b-b067-af8300a745b7-0-00001.parquet	parquet	July 31, 2025, 18:57:15 (UTC+05:30)	1.5 KB	Standard
00000-0-a9928cb5-004e-4372-9893-2d7a228fa3fa-0-00001.parquet	parquet	July 31, 2025, 19:01:36 (UTC+05:30)	1.5 KB	Standard
00000-1-b8d9f55f-b059-46c5-94cc-c021a72bd899-0-00001.parquet	parquet	July 31, 2025, 19:49:50 (UTC+05:30)	1.5 KB	Standard

Using Athena visualise the data of Iceberg

The screenshot shows the Amazon Athena Query editor. The URL is 'https://ap-south-1.console.aws.amazon.com/athena/home?region=ap-south-1#/query-editor/history/76ba238e-a7c4-4997-9398-510195ceea9'. The query editor interface shows a table named 'ad_revenuedata' with three rows of data. The results are displayed in a table format with columns: #, user_id, gender, age_group, region, and subscription_type.

#	user_id	gender	age_group	region	subscription_type
1	U48181	Female	46-60	North	Premium
2	U36431	Other	60+	East	Premium
3	U29227	Female	<18	East	Basic

Phase 5: Iceberg Table Update Trigger via Lambda & Airflow

Automatically trigger an **Airflow DAG** via **Lambda** when **Iceberg tables** (stored in S3 using Glue Catalog) are updated. The DAG will run a **Glue Job** to **merge or insert** the updated records.

Step 1: Prepare S3 Bucket Folder for DAGs, Plugins, Requirements

- **S3 Bucket:**
Use your existing bucket aahash-project2.
- **Create folders:**
s3://aahash-project2/airflow/
 - └── dags/
 - └── plugins/
 - └── requirements.txt (optional)

Uploaded empty.py file in dags/ because it required for MWAA to start

Step 2: Create MWAA Environment Setup

Environment Name: Aahash-iceberg

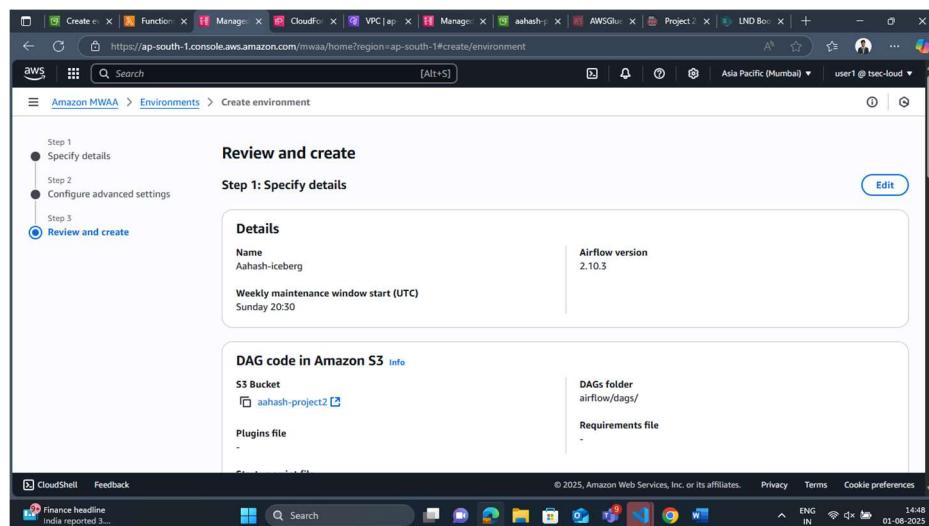
S3 Bucket: aahash-project2

DAGs folder path: airflow/dags/

Airflow Version: 2.10.3

Execution Role: MWAAExecutionRole-Aahash

VPC and Subnets configured



MWAA-VPC-Aahash

Events (94)

Timestamp	Logical ID	Status	Details
2025-08-01 15:49:07 UTC+0530	MWAA-VPC-Aahash	UPDATE_COMPLETE	-
2025-08-01 15:49:06 UTC+0530	MWAA-VPC-Aahash	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	-
2025-08-01 15:49:05 UTC+0530	PrivateRouteTable2	UPDATE_COMPLETE	-

Aahash-iceberg

Details

- Status: Available
- ARN: arn:aws:airflow:ap-south-1:008673239246:environment/Aahash-iceberg

Last update

- Status: SUCCESS
- Created at: 2025-08-01T09:18:32.000Z

Airflow UI

bed2e1b1-cf20-4979-83ba-5d7ca13a17e2.c1.ap-south-1.airflow.amazonaws.com

Worker replacement strategy

Forced

DAG code in Amazon S3

S3 Bucket: aahash-project2

DAGs folder: airflow/dags/

Step 2: Prepare DAG for Glue Trigger

from airflow import DAG

from airflow.providers.amazon.aws.operators.glue import GlueJobOperator

from datetime import datetime

```
default_args = {
    'owner': 'aahash',
    'start_date': datetime(2025, 1, 1),
    'retries': 0
}
```

```

with DAG(
    dag_id='trigger_glue_merge_dag',
    default_args=default_args,
    schedule_interval=None,
    catchup=False,
    description='Trigger Glue job for Iceberg table updates'
) as dag:
```

```

trigger_glue_job = GlueJobOperator(
    task_id='run_glue_merge',
    job_name='iceberg_merge_job-Aahash',
    script_args={
        '--JOB_NAME': "iceberg_merge_job-Aahash"
    },
    region_name='ap-south-1',
    wait_for_completion=True
)
```

Step3: Glue job run script

```

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

# Step 1: Initialize SparkSession with Iceberg configs
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://aahash-project2/media/iceberg/") \
    .config("spark.sql.catalog.glue_catalog.io-impl", "org.apache.iceberg.aws.s3.S3FileIO") \
    .getOrCreate()

# Step 2: Create Iceberg table (only first time)
spark.sql("""
CREATE TABLE IF NOT EXISTS glue_catalog.media_iceberg.ad_revenuedata (
    channel_id STRING,
    channel_name STRING,
    date DATE,
    ad_revenue DOUBLE
)
USING ICEBERG
PARTITIONED BY (date)
""")
```

```
# Step 3: Read source CSV
df = spark.read.option("header", True).option("inferSchema", True).csv("s3://aahash-project2/raw/ad_revenue/ad_revenue.csv")
```

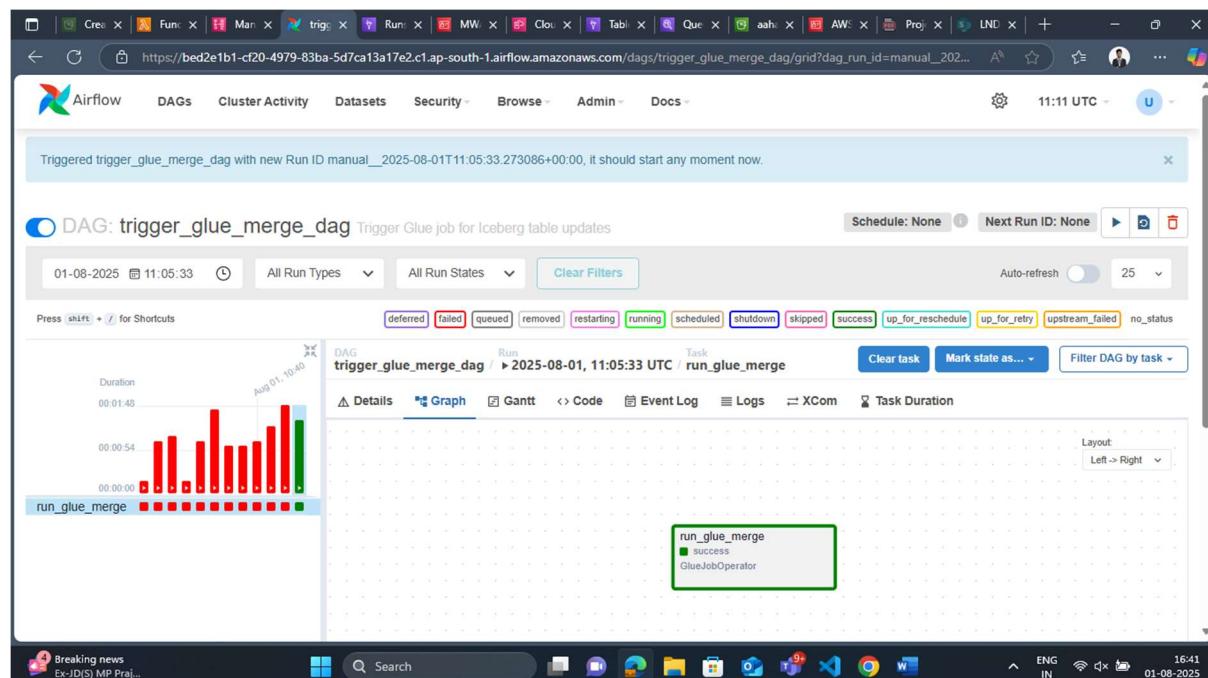
```
# Step 4: Transform
df = df.withColumn("date", to_date("date", "dd-MM-yyyy")) \
    .withColumn("ad_revenue", col("ad_revenue").cast("double"))
```

```
# Step 5: Register temporary view
df.createOrReplaceTempView("updates")
```

```
# Step 6: Perform MERGE into Iceberg table
spark.sql("""
MERGE INTO glue_catalog.media_iceberg.ad_revenuedata AS target
USING updates AS source
ON target.channel_id = source.channel_id AND target.date = source.date
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
""")
```

Step 4: Run the Airflow

Glue job name: iceberg_merge_job-Aahash



Step 5: Create the Lambda Function

🔧 Configuration:

- **Name:** TriggerMWAADAGOnIcebergUpdate
- **Runtime:** Python 3.10

- **IAM Role:** Create or attach a role with:
 - AmazonMWAFFullAccess
 - AmazonS3ReadOnlyAccess
 - CloudWatchLogsFullAccess

Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
TriggerMWAADAGOnIcebergUpdate-Aahash

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.13

Architecture Info
Choose the instruction set architecture you want for your function code.
 arm64
 x86_64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Environment Variables:

MWAA_ENV_NAME : Aahash-iceberg

REGION ap-south-1

DAG_NAME trigger_glue_on_iceberg_update

TriggerMWAADAGOnIcebergUpdate-Aahash

Successfully updated the function TriggerMWAADAGOnIcebergUpdate-Aahash.

Code | **Test** | **Monitor** | **Configuration** | **Aliases** | **Versions**

Environment variables (3)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
DAG_NAME	trigger_glue_merge_dag
MWAA_ENV_NAME	Aahash-iceberg
REGION	ap-south-1

Lambda Code (Python):

```
import json
import boto3
import requests
from urllib.parse import urljoin

def lambda_handler(event, context):
    mwaa_env_name = "Aahash-iceberg"
    dag_name = "trigger_glue_merge_dag"

    # Step 1: Create MWAA client
    client = boto3.client('mwaa')

    # Step 2: Get CLI token & webserver hostname
    resp = client.create_cli_token(Name=mwaa_env_name)
    token = resp['CliToken']
    webserver_hostname = resp['WebServerHostname']

    # Step 3: Trigger DAG using MWAA CLI
    dag_trigger_cmd = f"dags trigger {dag_name}"

    url = f"https://{{webserver_hostname}}/aws_mwaa/cli"
    headers = {
        'Authorization': f'Bearer {{token}}',
        'Content-Type': 'text/plain'
    }

    mwaa_response = requests.post(
        url,
        headers=headers,
        data=dag_trigger_cmd.encode('utf-8')
    )

    print("Lambda triggered with the following event:")
    print(json.dumps(event, indent=4))
    print("DAG trigger response:")
    print(mwaa_response.text)

    return {
        'statusCode': 200,
        'body': 'Triggered MWAA DAG from Lambda'
    }
```

Step 6: Add S3 Event Notification

Go to the **S3 bucket:** aahash-project2

► Configure an event like this:

- **Name:** iceberg-put-trigger
- **Event type:** PUT
- **Prefix:** iceberg/
- **Send to:** Lambda function
- **Select function:** TriggerMWAADAGOnIcebergUpdate-Aahash

📌 This will trigger the Lambda whenever a file is added/updated inside the iceberg/ folder.

The screenshot shows the 'Create event notification' configuration page in the Amazon S3 console. The path in the navigation bar is: Amazon S3 > Buckets > aahash-project2 > Create event notification. The main section is titled 'Create event notification' with a 'Info' link. A note below says: 'To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications.' The 'General configuration' section contains fields for 'Event name' (triggericebergUpdate), 'Prefix - optional' (iceberg/), and 'Suffix - optional' (.jpg). The 'Event types' section specifies 'All object create events' (s3:ObjectCreated:*). The 'Object creation' section lists two options: 'Put' (s3:ObjectCreated:Put) and 'Post' (s3:ObjectCreated:Post), with 'Put' checked.

Now let's test the Iceberg Table Update Trigger via Lambda & Airflow:

- Uploading a new version of the file (overwrite/append),
- Or uploading a new partition/file to the same S3 path.

Upload a new file (or overwrite an existing one) to the S3 path of your Iceberg table, e.g.:

```
aws s3 cp updated_ad_revenue.csv s3://aahash-project2/ad_revenuedata/
```

This should trigger the **S3 event notification**, which invokes your **Lambda function**.

Modify the Parquet File: created file.py for making changes

```

ec2-user@ip-172-31-19-204:~ X + ~
import pandas as pd

# Load original file
df = pd.read_parquet("00000-6-4244a253-125e-4207-82fc-9d9e7a4c0f32-0-00001.parquet")

# Modify something - Example: Add 1 to ad_revenue
if "ad_revenue" in df.columns:
    df["ad_revenue"] += 1
else:
    df["ad_revenue"] = 999.99 # dummy column if not found

# Save modified version
df.to_parquet("modified_trigger_file.parquet", index=False)

print("Parquet file modified and saved as modified_trigger_file.parquet")

```

Upload Modified File with New Name

aws s3 cp modified_trigger_file.parquet s3://Aahash
project2/media/iceberg/ad_revenue/data/trigger_file.parquet

Uploaded modified .parquet to trigger path

Verified CloudWatch logs:

Lambda execution started

File name and bucket printed

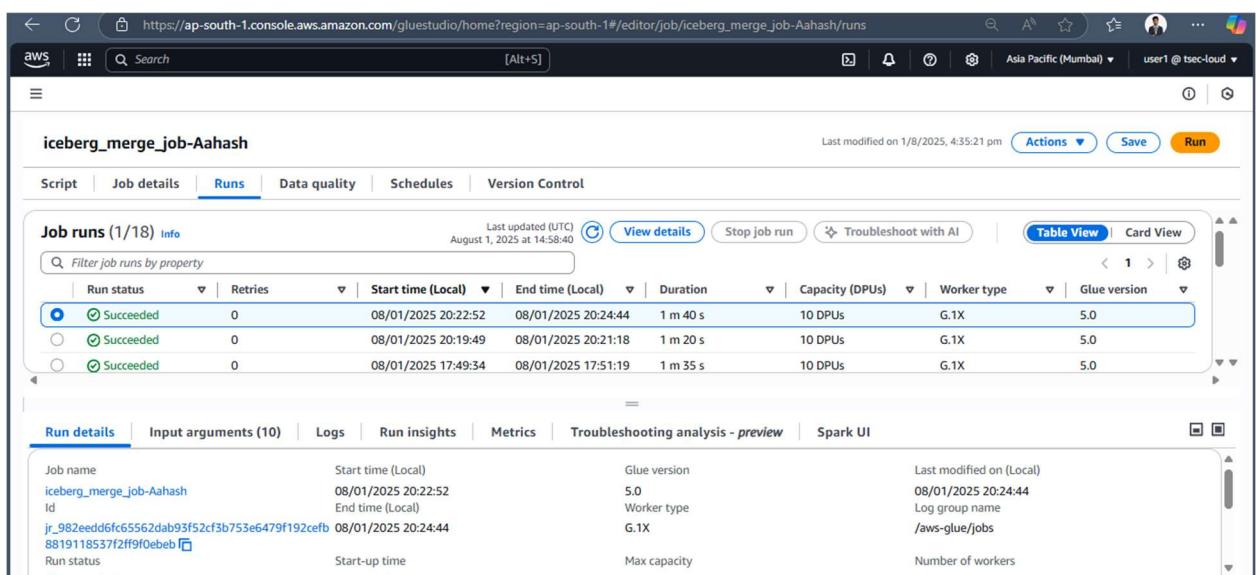
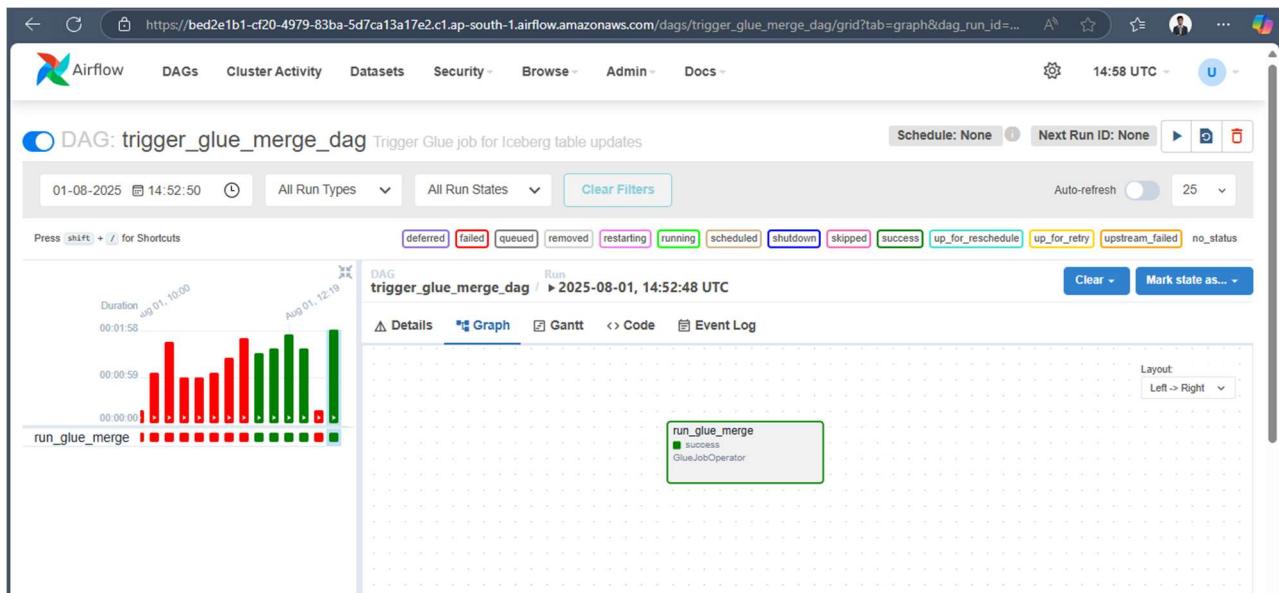
MWAA CLI token retrieved

DAG trigger initiated

Checked Airflow UI → DAG was successfully triggered

The screenshot shows the AWS CloudWatch Logs interface. The left sidebar navigation includes 'CloudWatch' (selected), 'Log groups' (selected), 'Logs' (selected), 'Log events', 'Log anomalies', 'Live tail', 'Logs insights', 'Contributor insights', 'Metrics', 'Application signals (APM)', and 'Network monitoring'. At the bottom of the sidebar are links for 'CloudShell' and 'Feedback'.

The main content area displays 'Log events' for the log group '/aws/lambda/TriggerMWAAADAGOnIcebergUpdate-Aahash'. It shows log entries from August 1, 2025, at 01T20:21:18.836+05:30. The first entry is 'INIT_START Runtime Version: python:3.13.v50 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:83a0b29e480e14176225231a6e561282a87732824063ebab771b15e4c1a2c71c'. Subsequent entries show 'START RequestId: 240b5f5d-cc27-47a8-98a4-00c9db874c5 Version: \$LATEST' and 'Lambda triggered with the following event:'. The last entry shows a JSON object: { "key1": "value1" }.



Phase 6 – Real-Time Streaming: Kinesis to Spark to S3 & Snowflake

Ingest live viewership logs through Kinesis → Process with Spark Streaming on EMR → Write output to **Snowflake** and **S3 (raw layer)**.

Create EMR:

The screenshot shows the AWS EMR console with the cluster 'Aahash-cluster' successfully created. The 'Summary' tab is selected, displaying cluster info (Cluster ID: j-1Y1CKQJTEEIC, Cluster ARN: arn:aws:elasticmapreduce:ap-south-1:008673239246:cluster/j-1Y1CKQJTEEIC), applications (Amazon EMR version emr-6.15.0, installed applications Hadoop 3.5.6, Hive 3.1.3, Spark 3.4.1), cluster management (Log destination in Amazon S3 aws-logs-008673239246-ap-south-1/elasticmapreduce, Primary node public DNS ec2-13-235-54-173.ap-south-1.compute.amazonaws.com, Connect to the Primary node using SSM), and status and time (Status: Starting, Creation time: August 01, 2025, 17:30 (UTC+05:30), Elapsed time: 2 minutes). Below the summary, there are tabs for Properties, Bootstrap actions, Instances (Hardware), Steps, Applications, Configurations, Monitoring, Events, and Tags (2). Under Properties, sections include Operating system (Amazon Linux release 2.0.20250623.0), Cluster logs (Archive log files to Amazon S3 Turned on, Encryption for logs Turned off), and Cluster termination and node replacement (Edit, Termination option, Idle time).

Step 6.1 – Generate and Send Real-Time Logs to Kinesis

We used a Python script to simulate live streaming of media viewership logs to Amazon Kinesis from a pre-prepared CSV.

Script: send_to_kinesis.py

```
import boto3
import time
import csv
import io

# Initialize Kinesis client
kinesis = boto3.client('kinesis', region_name='ap-south-1')
stream_name = 'aahash-newstream' # Replace with your actual stream name

# Open and read the CSV
with open('viewership_logs.csv', 'r') as file:
    reader = csv.reader(file)
    headers = next(reader) # Read header row and skip

    for row in reader:
        # Convert the row back into a single CSV string
        output = io.StringIO()
        writer = csv.writer(output)
        writer.writerow(row)
        csv_data = output.getvalue().strip()
```

```

# Send to Kinesis
response = kinesis.put_record(
    StreamName=stream_name,
    Data=csv_data.encode('utf-8'),
    PartitionKey=row[0] if row[0] else 'default' # Use session_id or fallback
)

print(f"Sent CSV row: {csv_data} | Response: {response}")
time.sleep(0.5) # Optional: simulate streaming by throttling

```

Download CSV from S3:

```

aws s3 cp s3://aahash-project2/raw/viewership_logs/viewership_logs.csv
python3 send_to_kinesis.py

```

```

esponse: {'ShardId': 'shardId-000000000001', 'SequenceNumber': '49665806910781676325439118801048439418840032829237100562', 'ResponseMetadata': {'RequestId': 'd571c5db-7d62-bda6-b47b-8a407a8d6760', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'd571c5db-7d62-bda6-b47b-8a407a8d6760', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:11 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID558123,UAH361,COLOR_TAMIL,Show_135,Entertainment,2025-07-30 09:30:13,3,Central,Premium,Laptop,Web, False, 4, 85.18, 0.98, 2, 85.11 | Response: {'ShardId': 'shardId-000000000001', 'SequenceNumber': '49665806910781676325439118801137899929491515456884834322', 'ResponseMetadata': {'RequestId': 'c84d2d40-1ca7-a287-a947-62dc1b487841', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'c84d2d40-1ca7-a287-a947-62dc1b487841', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:11 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID676387,US2397,CH021,Aaj Tak,Show_63,News,2025-07-27 20:34:53,36,Central,Premium,Smart TV,FireTV, True, 4, 159.78, 0.26, 0, 45.99 | Response: {'ShardId': 'shardId-000000000001', 'SequenceNumber': '496658069108083977070637649240032829725626735861432354', 'ResponseMetadata': {'RequestId': 'f9b25321-5ef8-3511-98b8-1cbd5917ef7d', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'f9b25321-5ef8-3511-98b8-1cbd5917ef7d', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:12 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID753243,U29394,CH035,KTV,Show_108,Movies,2025-07-28 02:16:04,57,North,Free,Tablet,Roku,True,5,867.8,0.63,3,78.67 | Response: {'ShardId': 'shardId-000000000001', 'SequenceNumber': '4966580691078167632543911880138693864833212913593783314', 'ResponseMetadata': {'RequestId': 'f9850840-1794-7ee5-988f-47dd107ba423', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'f9850840-1794-7ee5-988f-47dd107ba423', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:13 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID734023,US5352,CH025,Indian Bharat,Show_23,News,2025-07-29 19:08:36,139,Central,Premium,Smart TV,iOS,False,2,318.57,0.31,2,96.63 | Response: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49665806910808397707063764924003282972562673586143410', 'ResponseMetadata': {'RequestId': 'e80c6118-304b-8c26-8906-2b8537a56e0', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'e80c6118-304b-8c26-8906-2b8537a56e0', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:13 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID808244,US8129,CH007,Dangal TV,Show_172,Kids,2025-07-30 21:28:37,23,Pan-India,Premium,Smart TV,iOS,True,5,346.34,0.66,0,29.46 | Response: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '4966580691088397707063764924074728993995502544641058338', 'ResponseMetadata': {'RequestId': 'd4dfa538-fad2-d9e6-b5d5-eaa6fd3d0320', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'd4dfa538-fad2-d9e6-b5d5-eaa6fd3d0320', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:14 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}
Sent CSV row: SID240308,US2282,CH012,Star Sports 2,Show_62,Music,2025-07-30 19:53:24,148,East,Free,Laptop,FireTV, True, 3, 395.71, 0.71, 0, 45.69 | Response: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '4966580691088397707063764924869391447736182993086382114', 'ResponseMetadata': {'RequestId': 'd7b880ee-33c8-725f-b6b2-cf783427a899', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'd7b880ee-33c8-725f-b6b2-cf783427a899', 'x-amz-id-2': 'r3MPRzBCQ2CF74M5Xp2p3W4jhCtVyyEcx20n8cbP08DtrwBktGkPxvUT2rtBwnbrChn/HRB2HZL2T+Q8+Nkhkw5AZP2', 'date': 'Mon, 04 Aug 2025 06:04:14 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive', 'RetryAttempts': 0}}

```

Step 6.2 – Launch Spark Structured Streaming on EMR

Created a PySpark script that reads real-time data from Kinesis, transforms the JSON, and writes:

- To S3 in JSON format
- To Snowflake for analytical use

Script: spark_to_s3_snowflake.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", "Aahash-newstream") \
    .option("endpointUrl", "https://kinesis.ap-south-1.amazonaws.com") \
    .option("region", "ap-south-1") \
    .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": "XZBTYSG-TQB56893.snowflakecomputing.com",
            "sfUser": "AAHASH06",
            "sfPassword": "Aahash@snowflake06",
            "sfDatabase": "STREAMDATA",
            "sfSchema": "PUBLIC",
            "sfWarehouse": "COMPUTE_WH",
            "sfRole": "ACCOUNTADMIN"
        }

        print("Writing to Snowflake...")
        batch_df.write \
            .format("net.snowflake.spark.snowflake") \
            .options(**snowflake_options) \
            .option("dbtable", "REALTIME_VIEWERSHIP") \
            .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://aahash-project2/stream/")
    
```

```

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://aahash-project2/temp/") \
    .trigger(processingTime="10 seconds") \
    .start()

print("Streaming started.")
query.awaitTermination()

```

After running script, you can see the file has been saved to s3

Name	Type	Last modified	Size	Storage class
airflow/	Folder	-	-	-
checkpoints/	Folder	-	-	-
media/	Folder	-	-	-
processed/	Folder	-	-	-
raw/	Folder	-	-	-
realtime_checkpoint/	Folder	-	-	-
scripts/	Folder	-	-	-
stream/	Folder	-	-	-
streaming/	Folder	-	-	-
temp/	Folder	-	-	-

```

25/08/01 16:46:26 INFO DAGScheduler: Job "W" finished; start at NativeMethodAccessImpl.java:0, took 4.065577 s
25/08/01 16:46:26 WARN DefaultIOManager: JDBC 3.13.3 is being used. But the certified JDBC version 3.13.3 is recommended.
25/08/01 16:46:26 INFO StageWriter: Spark Connector Master: execute query with bind variable: desc table identifier(?)
25/08/01 16:46:26 INFO StageWriter: Begin to write at 2025-08-01T16:46:25.458 (Coordinated Universal Time)
25/08/01 16:46:26 INFO StageWriter: Total file count is 4, non-empty files count is 2, total file size is 1.35 KB.
25/08/01 16:46:28 INFO StageWriter: First COPY command is: copy into viewership_stream FROM @spark_connector_load_stage_RzldzGNVnp/Phm7ObIfiB/
FILE_FORMAT =
  TYPECSV
  FIELD_DELIMITER='|'
  NULL_IFC ''
  FIELD_OPTIONALLY_ENCLOSED_BY=''''
  TIMESTAMP_FORMAT='TZTZZ YYYY-MM-DD HH24:MI:SS.FF9'
  DATE_FORMAT='YYYY-MM-DD HH24:MI:SS.FF9'
  BINARY_FORMAT=BASE64
)

25/08/01 16:46:28 INFO SnowFlakeSQLStatement: Spark Connector Master: execute query with bind variable: copy into viewership_stream FROM @spark_connector_load_stage_RzldzGNVnp/Phm7ObIfiB/
FILE_FORMAT = (
  TYPECSV
  FIELD_DELIMITER='|'
  NULL_IFC ''
  FIELD_OPTIONALLY_ENCLOSED_BY=''''
  TIMESTAMP_FORMAT='TZTZZ YYYY-MM-DD HH24:MI:SS.FF9'
  DATE_FORMAT='YYYY-MM-DD HH24:MI:SS.FF9'
  BINARY_FORMAT=BASE64
)

25/08/01 16:46:29 INFO StageWriter: First COPY command is done in 1.32 seconds at 2025-08-01T16:46:26.781, queryID is 01be150e-0306-1f03-000a-7dcf001582ee
25/08/01 16:46:29 INFO StageWriter: Success in 1.32 seconds at 2025-08-01T16:46:26.781
25/08/01 16:46:27 INFO StageWriter: Spark Connector Master: Total job time is 4.57 seconds including read & upload time: 1.51 seconds and COPY time: 3.06 seconds.
25/08/01 16:46:27 WARN SnowFlakeConnectorUtils: Query pushdown is not supported because you are using Spark 3.4.1+ - even-2 with a connector designed to support Spark 3.1. 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 16:46:27 WARN DefaultIOManager: JDBC 3.13.30 is being used. But the certified JDBC version 3.13.3 is recommended.
25/08/01 16:46:27 INFO SnowFlakeSQLStatement: Spark Connector Master: execute query with bind variable: select * from viewership_stream where l = 0
  ✓ Write to Snowflake succeeded.

25/08/01 16:46:29 INFO PathOutputCommitterFactory: No output committer factory defined, defaulting to class org.apache.hadoop.mapreduce.lib.output.FileSystemOptimizedOutputCommitterFactory
25/08/01 16:46:29 INFO FileOutputCommitterFactory: ENABLED
25/08/01 16:46:29 INFO FileOutputCommitter: File Output Committer Algorithm version is 7
25/08/01 16:46:29 INFO FileOutputCommitter: File Output Committer will skip cleanup of temporary folders under output directory>false, ignore cleanup failures: true
25/08/01 16:46:29 INFO SQLCommitterFactory: Using output committer class is org.apache.hadoop.mapreduce.lib.output.FileSystemOptimizedCommitter
25/08/01 16:46:29 INFO FileSystemOptimizedCommitter: Nothing to setup as successful task attempt outputs are written directly
25/08/01 16:46:29 INFO SparkContext: Starting job: start at NativeMethodAccessImpl.java:0
25/08/01 16:46:29 INFO DAGScheduler: Submitting 1 tasks so far at NativeMethodAccessImpl.java:0 with 4 output partitions
25/08/01 16:46:29 INFO DAGScheduler: Final stage: ResultStage 98 (start at NativeMethodAccessImpl.java:0)
25/08/01 16:46:29 INFO DAGScheduler: Parents of final stage: list()
25/08/01 16:46:29 INFO DAGScheduler: Missing parents: list()
25/08/01 16:46:29 INFO DAGScheduler: Submitting 95 (MapPartitionsRDD[0]@6097) at start at NativeMethodAccessImpl.java:0, which has no missing parents
25/08/01 16:46:29 INFO MemoryStore: Block broadcast_95 stored as bytes in memory (estimated size 411.4 KB, free 910.2 MB)
25/08/01 16:46:29 INFO MemoryStore: Block broadcast_95_piece0 stored as bytes in memory (estimated size 151.3 KB, free 910.2 MB)
25/08/01 16:46:29 INFO BlockManagerInfo: Added broadcast_95_piece0 in memory on ip-172-31-19-204.ap-south-1.compute.internal:45787 (size: 151.3 KB, free: 910.2 MB)
25/08/01 16:46:29 INFO TaskSchedulerImpl: Submitting 4 missing tasks from ResultStage 95 (MapPartitionsRDD[0]@6097) at start at NativeMethodAccessImpl.java:0 (First 15 tasks are for partitions Vector(0, 1, 2, 3))
25/08/01 16:46:29 INFO TaskSchedulerImpl: Adding task set 95.0 with 4 tasks resource profile 0
25/08/01 16:46:29 INFO TaskSetManager: Starting task 0.0 in stage 95 (TID 388) (ip-172-31-19-204.ap-south-1.compute.internal, executor driver, partition 0, PROCESS_LOCAL, 7400 bytes)
25/08/01 16:46:29 INFO TaskSetManager: Starting task 1.0 in stage 95.0 (TID 389) (ip-172-31-19-204.ap-south-1.compute.internal, executor driver, partition 1, PROCESS_LOCAL, 7400 bytes)
25/08/01 16:46:29 INFO TaskSetManager: Starting task 2.0 in stage 95.0 (TID 390) (ip-172-31-19-204.ap-south-1.compute.internal, executor driver, partition 2, PROCESS_LOCAL, 7400 bytes)
25/08/01 16:46:29 INFO TaskSetManager: Starting task 3.0 in stage 95.0 (TID 391) (ip-172-31-19-204.ap-south-1.compute.internal, executor driver, partition 3, PROCESS_LOCAL, 7400 bytes)
25/08/01 16:46:29 INFO TaskSetManager: Starting task 0.0 in stage 95.0 (TID 388)
25/08/01 16:46:29 INFO Executor: Running task 0.0 in stage 95.0 (TID 388)
25/08/01 16:46:29 INFO Executor: Running task 1.0 in stage 95.0 (TID 389)
25/08/01 16:46:29 INFO Executor: Running task 2.0 in stage 95.0 (TID 390)
25/08/01 16:46:29 INFO Executor: Running task 3.0 in stage 95.0 (TID 391)

```

Data received by the snowflake:

The screenshot shows the Snowflake web interface. On the left, the sidebar includes 'Home', 'Search', 'Projects', 'Workbooks' (selected), 'Notebooks', 'Streamlit', 'Dashboards', 'App Packages', 'Templates', 'Services and Jobs'. A message indicates '\$336 credits left' and 'Trial ends in 58 days'. On the right, the main area shows a query history with a recent run on '2025-08-01 10:33pm'. Below it, a query is running in a worksheet titled 'STREAMDATA.PUBLIC'. The query is:

```

SELECT * FROM "STREAMDATA"."PUBLIC"."REALTIME_VIEWERSHIP" LIMIT 100;

```

The results pane displays the data from the 'REALTIME_VIEWERSHIP' table, showing 11 rows with columns: SESSION_ID, USER_ID, CHANNEL_ID, CHANNEL_NAME, SHOW_NAME, and GENRE. The data includes various TV shows and genres like 'DD Bangla', 'Sun TV', 'DD Sports', etc. The 'Query Details' panel on the right shows a duration of 49ms, 100 rows, and a query ID of 01be2316-0306-26db-0... The 'SESSION_ID' and 'USER_ID' filters are set to '100% filled'.

Phase 7: Stream Analytics Using Athena

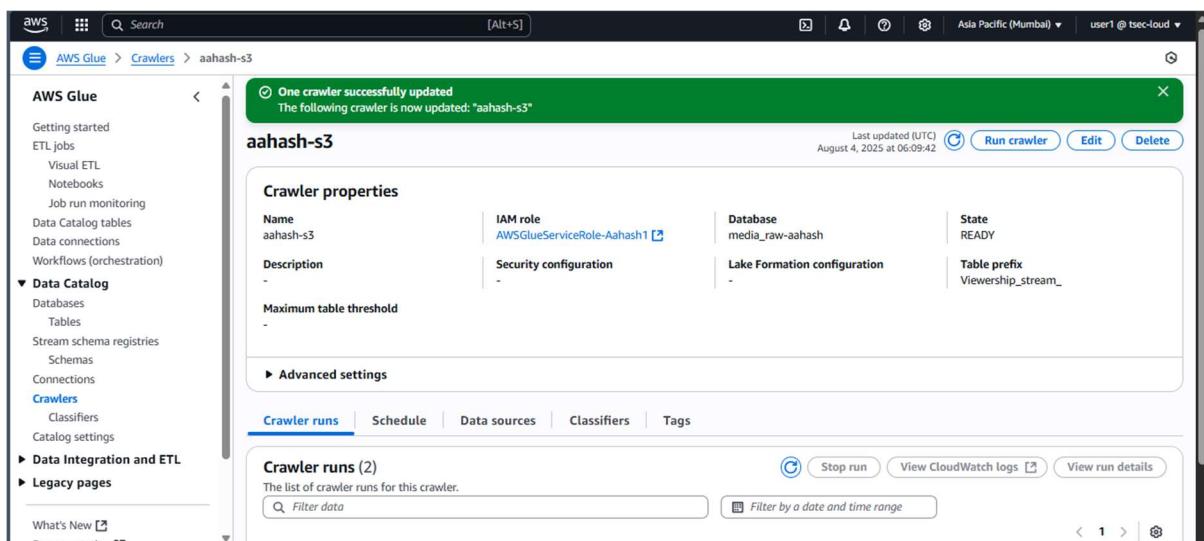
Objective: Analyze real-time viewership data ingested into S3 (stream1 folder) using Athena for interactive querying.

Now we will fetch the data using crawler from s3

Created crawler Aahash-s3

Created a Glue Crawler: aahashs3

- Crawls data from: s3://aahash-project2/stream1/
- Output table: viewership_stream_stream1 under database: media_raw-aahash



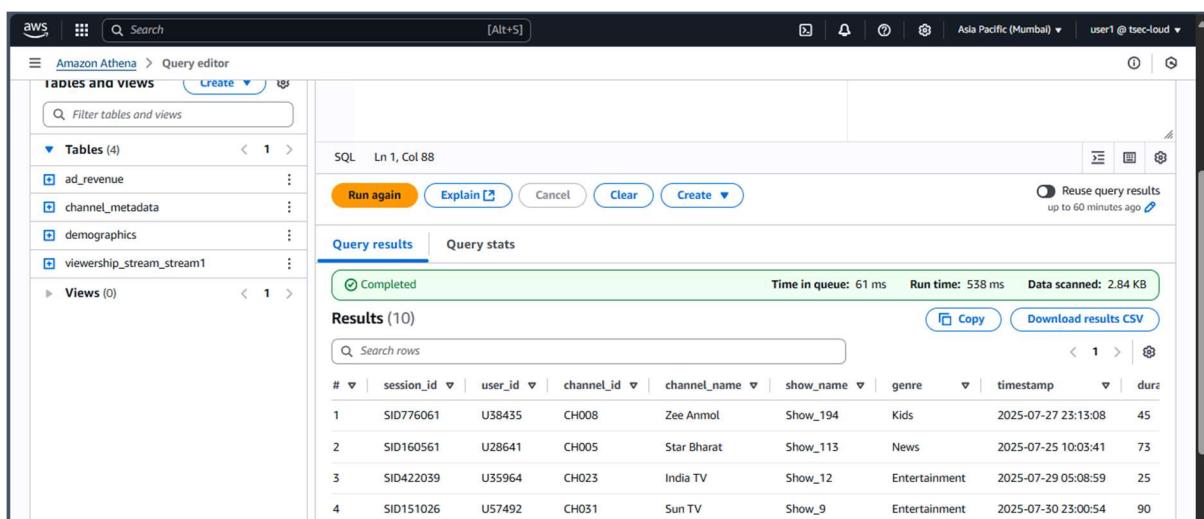
The screenshot shows the AWS Glue Crawler configuration page for 'aahash-s3'. The crawler properties section displays the following details:

- Name: aahash-s3
- IAM role: AWSGlueServiceRole-Aahash1
- Description: -
- Database: media_raw-aahash
- State: READY
- Table prefix: Viewership_stream_

The 'Crawler runs' tab shows two runs, with options to Stop run, View CloudWatch logs, and View run details.

Verified table availability in Athena console.

After running crawler, we can see the data:



The screenshot shows the Amazon Athena Query editor results for the 'viewership_stream_stream1' table. The results table contains the following data:

#	session_id	user_id	channel_id	channel_name	show_name	genre	timestamp	duration
1	SID776061	U38435	CH008	Zee Anmol	Show_194	Kids	2025-07-27 23:13:08	45
2	SID160561	U28641	CH005	Star Bharat	Show_113	News	2025-07-25 10:03:41	73
3	SID422039	U35964	CH023	India TV	Show_12	Entertainment	2025-07-29 05:08:59	25
4	SID151026	U57492	CH031	Sun TV	Show_9	Entertainment	2025-07-30 23:00:54	90

Run the all 22 query:

1. Total viewership duration per channel

```
SELECT channel_id, channel_name, SUM(duration_minutes) AS total_duration
FROM viewership_stream_stream1
GROUP BY channel_id, channel_name
ORDER BY total_duration DESC;
```

Query results | Query stats

Completed

Time in queue: 57 ms Run time: 1.927 sec Data scanned: 289.76 KB

Results (50)

Copy Download results CSV

#	channel_id	channel_name	total_duration
1	CH033	Colors Tamil	6099
2	CH045	Zee Bangla	5724
3	CH009	Colors Rishtey	5718
4	CH048	DD Bangla	5532
5	CH008	Zee Anmol	5295
6	CH047	DD Sahyadri	5249
7	CH039	Gemini TV	5170

2. Average engagement by device

```
SELECT device, AVG(engagement_score) AS avg_engagement
FROM viewership_stream_stream1
GROUP BY device
ORDER BY avg_engagement DESC;
```

Completed

Time in queue: 59 ms Run time: 2.179 sec Data scanned: 289.76 KB

Results (4)

Copy Download results CSV

#	device	avg_engagement
1	Laptop	0.6162234042553189
2	Smart TV	0.6097395833333337
3	Mobile	0.5997986577181211
4	Tablet	0.5944677137870854

3. Daily ad revenue per channel

```
SELECT
date_trunc('day', CAST("timestamp" AS timestamp)) AS view_date,
channel_id,
channel_name,
SUM(ad_revenue) AS total_revenue
FROM viewership_stream_stream1
GROUP BY
date_trunc('day', CAST("timestamp" AS timestamp)),
channel_id,
channel_name
```

```

ORDER BY
date_trunc('day', CAST("timestamp" AS timestamp)),
total_revenue DESC;

```

Completed Time in queue: 55 ms Run time: 954 ms Data scanned: 289.76 KB

Results (395)

Copy Download results CSV

Search rows

#	view_date	channel_id	channel_name	total_revenue
1	2025-07-24 00:00:00.000	CH026	Times Now	2747.74
2	2025-07-24 00:00:00.000	CH007	Dangal TV	2150.85
3	2025-07-24 00:00:00.000	CH010	DD National	2099.31
4	2025-07-24 00:00:00.000	CH013	Star Sports 3	1807.77
5	2025-07-24 00:00:00.000	CH021	Aaj Tak	1806.4699999999998
6	2025-07-24 00:00:00.000	CH012	Star Sports 2	1678.01
7	2025-07-24 00:00:00.000	CH034	Zee Tamil	1584.9199999999998
8	2025-07-24 00:00:00.000	CH038	ETV Telugu	1522.55

4. Gender-wise average completion percentage

```

SELECT d.gender, AVG(v.completion_percentage) AS avg_completion
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
GROUP BY d.gender;

```

Query results **Query stats**

Completed Time in queue: 143 ms Run time: 1.494 sec Data scanned: 449.92 KB

Results (3)

Copy Download results CSV

Search rows

#	gender	avg_completion
1	Male	60.475220125786215
2	Other	59.26628571428562
3	Female	60.287831325301205

5. Most watched genres in each region

```

WITH genre_ranking AS (
SELECT
region,
genre,
SUM(duration_minutes) AS total_duration,
ROW_NUMBER() OVER (PARTITION BY region ORDER BY SUM(duration_minutes) DESC)
AS rn
FROM viewership_stream_stream1
GROUP BY region, genre
)
SELECT region, genre, total_duration
FROM genre_ranking
WHERE rn = 1;

```

Query results | Query stats

Completed Time in queue: 54 ms Run time: 1.067 sec Data scanned: 289.76 KB

Results (7)

Search rows

#	region	genre	total_duration
1	South	Entertainment	9027
2	Central	Entertainment	8345
3	Pan-India	Entertainment	8786
4	North	News	7669
5	West	Entertainment	8744
6	East	Entertainment	7657

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

```

SELECT
    channel_id,
    channel_name,
    SUM(ad_revenue) AS total_revenue
FROM viewership_stream_stream1
WHERE CAST("timestamp" AS timestamp) >= date_add('day', -7, current_date)
GROUP BY channel_id, channel_name
ORDER BY total_revenue DESC
LIMIT 5;

```

Completed Time in queue: 60 ms Run time: 931 ms Data scanned: 289.76 KB

Results (5)

Search rows

#	channel_id	channel_name	total_revenue
1	CH034	Zee Tamil	9623.039999999999
2	CH023	India TV	8649.21
3	CH033	Colors Tamil	8367.439999999997
4	CH009	Colors Rishtey	8341.859999999999
5	CH008	Zee Anmol	7947.169999999999

7. Peak viewership hours by region

```

WITH hourly_views AS (
    SELECT
        region,
        EXTRACT(HOUR FROM CAST("timestamp" AS timestamp)) AS view_hour,
        COUNT(*) AS views,
        ROW_NUMBER() OVER (
            PARTITION BY region
            ORDER BY COUNT(*) DESC
        )
)
```

```

) AS rn
FROM viewership_stream_stream1
GROUP BY
region,
EXTRACT(HOUR FROM CAST("timestamp" AS timestamp))
)
SELECT
region,
view_hour AS hour,
views
FROM hourly_views
WHERE rn = 1;

```

Completed Time in queue: 56 ms Run time: 1.134 sec Data scanned: 289.76 KB

Results (7)

Copy Download results CSV

Search rows

#	region	hour	views
1	South	6	18
2	Central	9	24
3	North	2	21
4	West	2	27
5	Northeast	3	19
6	East	17	18
7	Pan-India	22	18

8. Which age group watches the most live content?

```

SELECT d.age_group, SUM(duration_minutes) AS total_live_duration
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
WHERE is_live = TRUE
GROUP BY d.age_group
ORDER BY total_live_duration DESC;

```

Completed Time in queue: 56 ms Run time: 1.37 sec Data scanned: 449.92 KB

Results (6)

Copy Download results CSV

Search rows

#	age_group	total_live_duration
1	<18	19494
2	26-35	18746
3	60+	18464
4	18-25	18155
5	46-60	17236
6	36-45	16875

9. Subscription type driving most revenue per channel

WITH revenue_by_channel_sub AS (

```

SELECT
    v.channel_id,
    v.channel_name,
    d.subscription_type,
    SUM(v.ad_revenue) AS total_revenue,
    ROW_NUMBER() OVER (
        PARTITION BY v.channel_id
        ORDER BY SUM(v.ad_revenue) DESC
    ) AS rn
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
GROUP BY v.channel_id, v.channel_name, d.subscription_type
)
SELECT
    channel_id,
    channel_name,
    subscription_type,
    total_revenue
FROM revenue_by_channel_sub
WHERE rn = 1;

```

Results (50)

Copy
Download results CSV

Search rows
1

#	channel_id	channel_name	subscription_type	total_revenue
1	CH002	Colors TV	Basic	4047.670000000001
2	CH042	Zee Kannada	Premium	6049.230000000001
3	CH045	Zee Bangla	Premium	7251.11
4	CH028	NDTV India	Basic	5192.27
5	CH004	Zee TV	Premium	5895.459999999999
6	CH049	News18 Kerala	Premium	4593.97
7	CH034	Zee Tamil	Free	5938.500000000002
8	CH041	Surya TV	Premium	6434.150000000001
9	CH038	ETV Telugu	Premium	6189.53
10	CH005	Star Bharat	Premium	4028.720000000003

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

```

SELECT *
FROM viewership_stream_stream1
WHERE completion_percentage > 90

```

AND duration_minutes > 1;

#	session_id	user_id	channel_id	channel_name	show_name	genre	timestamp
1	SID530249	U36362	CH016	Sony Sports Ten 1	Show_157	Sports	2025-07-24 18:36:04
2	SID193732	U15187	CH028	NDTV India	Show_110	News	2025-07-29 12:49:46
3	SID885032	U48924	CH007	Dangal TV	Show_92	Sports	2025-07-24 14:27:10
4	SID476466	U29845	CH045	Zee Bangla	Show_182	Entertainment	2025-07-26 02:22:10
5	SID686217	U13831	CH031	Sun TV	Show_59	Entertainment	2025-07-25 02:15:24
6	SID791008	U13418	CH026	Times Now	Show_53	News	2025-07-24 17:46:46
7	SID476739	U11608	CH029	Zee News	Show_125	News	2025-07-25 18:17:07
8	SID710536	U11611	CH049	News18 Kerala	Show_115	Music	2025-07-26 21:53:18
9	SID873462	U36533	CH027	CNN-News18	Show_94	News	2025-07-27 12:12:51
10	SID892942	U49841	CH037	Zee Telugu	Show_143	Kids	2025-07-25 01:45:45
11	SID762018	U18688	CH039	Gemini TV	Show_182	Sports	2025-07-26 12:36:09

11. Channels with above-average ad revenue per day

```
WITH daily_avg AS (
    SELECT
        DATE(CAST("timestamp" AS timestamp)) AS view_date,
        AVG(ad_revenue) AS avg_revenue
    FROM viewership_stream_stream1
    GROUP BY DATE(CAST("timestamp" AS timestamp)))
),
channel_daily_revenue AS (
    SELECT
        DATE(CAST("timestamp" AS timestamp)) AS view_date,
        channel_id,
        channel_name,
        SUM(ad_revenue) AS total_channel_revenue
    FROM viewership_stream_stream1
    GROUP BY DATE(CAST("timestamp" AS timestamp)), channel_id, channel_name
)
SELECT
    cdr.channel_id,
    cdr.channel_name,
    cdr.view_date,
    cdr.total_channel_revenue
FROM channel_daily_revenue cdr
JOIN daily_avg da ON cdr.view_date = da.view_date
WHERE cdr.total_channel_revenue > da.avg_revenue
ORDER BY cdr.view_date, total_channel_revenue DESC;
```

Completed Time in queue: 58 ms Run time: 1.435 sec Data scanned: 579.52 KB

Results (366)

Search rows

Copy Download results CSV

#	channel_id	channel_name	view_date	total_channel_revenue
1	CH026	Times Now	2025-07-24	2747.74
2	CH007	Dangal TV	2025-07-24	2150.8500000000004
3	CH010	DD National	2025-07-24	2099.31
4	CH013	Star Sports 3	2025-07-24	1807.77
5	CH021	Aaj Tak	2025-07-24	1806.47
6	CH012	Star Sports 2	2025-07-24	1678.01
7	CH034	Zee Tamil	2025-07-24	1584.92
8	CH038	ETV Telugu	2025-07-24	1522.55
9	CH009	Colors Rishtey	2025-07-24	1474.41
10	CH044	Colors TV	2025-07-24	1465.46

12. Most engaging show for female users in metro regions

```
SELECT
    show_name,
    AVG(engagement_score) AS avg_engagement
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
WHERE d.gender = 'Female'
AND d.region IN ('Mumbai', 'Delhi', 'Bangalore', 'Hyderabad', 'Chennai')
GROUP BY show_name
ORDER BY avg_engagement DESC
LIMIT 1;
```

Query results | Query stats

Completed Time in queue: 54 ms Run time: 1.246 sec Data scanned: 160.17 KB

Results (0)

Search rows

Copy Download results CSV

#	show_name	avg_engagement
No results		
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 viewership_stream_stream1
GROUP BY genre
ORDER BY avg_revenue DESC;
```

Query results | **Query stats**

Completed Time in queue: 63 ms Run time: 1.056 sec Data scanned: 289.76 KB

Results (6) [Copy](#) [Download results CSV](#)

Search rows

#	genre	avg_revenue	avg_completion
1	Entertainment	276.58230529595045	59.569361370716486
2	Sports	269.7936612021859	59.535546448087494
3	Movies	258.0136666666667	60.45800000000003
4	Kids	253.59887218045114	59.15748120300751
5	Music	249.95051181102363	60.84488188976379
6	News	249.92268101761275	60.52017612524459

14. Channels with highest buffer counts but good engagement

```
SELECT
    channel_id,
    channel_name,
    AVG(buffer_count) AS avg_buffer,
    AVG(engagement_score) AS avg_engagement
FROM viewership_stream_stream1
GROUP BY channel_id, channel_name
HAVING
    AVG(buffer_count) > 2
    AND AVG(engagement_score) > 70
ORDER BY avg_buffer DESC;
```

[Run again](#) [Explain](#) [Cancel](#) [Clear](#) [Create](#) Reuse query results up to 60 minutes ago

Query results | **Query stats**

Completed Time in queue: 58 ms Run time: 877 ms Data scanned: 289.76 KB

Results (0) [Copy](#) [Download results CSV](#)

Search rows

#	channel_id	channel_name	avg_buffer	avg_engagement
No results Run a query to view results				

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

```
SELECT
    d.age_group,
    SUM(v.duration_minutes) AS total_duration
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
GROUP BY d.age_group
ORDER BY total_duration DESC;
```

Query results | Query stats

Completed Time in queue: 81 ms Run time: 1.136 sec Data scanned: 449.92 KB

Results (6)

Search rows

#	age_group	total_duration
1	60+	38675
2	<18	37686
3	18-25	37224
4	46-60	36728
5	26-35	34895
6	36-45	33328

Copy | Download results CSV | 1 | ⚙️

16. Average viewing time per user segmented by region

WITH user_durations AS (

```
SELECT
    v.user_id,
    SUM(v.duration_minutes) AS total_duration
FROM viewership_stream_stream1 v
GROUP BY v.user_id
)
```

```
SELECT
    d.region,
    AVG(ud.total_duration) AS avg_viewing_time
FROM user_durations ud
JOIN demographics d ON ud.user_id = d.user_id
GROUP BY d.region
ORDER BY avg_viewing_time DESC;
```

Query results | Query stats

Completed Time in queue: 57 ms Run time: 1.061 sec Data scanned: 449.92 KB

Results (7)

Search rows

#	region	avg_viewing_time
1	North	120.36015325670498
2	West	119.29323308270676
3	East	118.86440677966101
4	Pan-India	116.16929133858268
5	Northeast	111.87889273356402
6	South	109.95578231292517
7	Central	108.0909090909091

Copy | Download results CSV | 1 | ⚙️

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

SELECT

device,

```

SUM(duration_minutes) AS total_duration
FROM viewership_stream_stream1
WHERE device IN ('Mobile', 'Smart TV')
GROUP BY device
ORDER BY total_duration DESC;

```

Completed Time in queue: 55 ms Run time: 875 ms Data scanned: 289.76 KB

Results (2)

Search rows

Copy Download results CSV

#	device	total_duration
1	Mobile	52974
2	Smart TV	50359

18. Total ad revenue per channel per day

```

SELECT
    DATE(CAST("timestamp" AS timestamp)) AS view_date,
    channel_id,
    channel_name,
    SUM(ad_revenue) AS total_revenue
FROM viewership_stream_stream1
GROUP BY
    DATE(CAST("timestamp" AS timestamp)),
    channel_id,
    channel_name
ORDER BY view_date, total_revenue DESC;

```

Results (395)

Search rows

Copy Download results CSV

#	view_date	channel_id	channel_name	total_revenue
1	2025-07-24	CH026	Times Now	2747.7400000000002
2	2025-07-24	CH007	Dangal TV	2150.85
3	2025-07-24	CH010	DD National	2099.31
4	2025-07-24	CH013	Star Sports 3	1807.77
5	2025-07-24	CH021	Aaj Tak	1806.469999999998
6	2025-07-24	CH012	Star Sports 2	1678.01
7	2025-07-24	CH034	Zee Tamil	1584.92
8	2025-07-24	CH038	ETV Telugu	1522.5500000000002
9	2025-07-24	CH009	Colors Rishtey	1474.41
10	2025-07-24	CH041	Surya TV	1446.46
11	2025-07-24	CH045	Zee Bangla	1390.67

19. Peak hours of viewership across India

```

WITH hourly_view_counts AS (
    SELECT
        EXTRACT(HOUR FROM CAST("timestamp" AS timestamp)) AS view_hour,

```

```

        COUNT(*) AS total_views,
        ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) AS rn
    FROM viewership_stream_stream1
    GROUP BY EXTRACT(HOUR FROM CAST("timestamp" AS timestamp))
)
SELECT view_hour, total_views
FROM hourly_view_counts
WHERE rn <= 3;

```

Completed Time in queue: 62 ms Run time: 990 ms Data scanned: 289.76 KB

Results (3)

Search rows

Copy Download results CSV

#	view_hour	total_views
1	2	119
2	22	109
3	12	108

20. How does content genre popularity vary by gender

```

SELECT
    d.gender,
    v.genre,
    SUM(v.duration_minutes) AS total_watch_time
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
GROUP BY d.gender, v.genre
ORDER BY d.gender, total_watch_time DESC;

```

Results (18)

Search rows

Copy Download results CSV

#	gender	genre	total_watch_time
1	Female	Entertainment	20075
2	Female	News	15863
3	Female	Sports	11536
4	Female	Kids	10266
5	Female	Movies	8543
6	Female	Music	7361
7	Male	Entertainment	21638
8	Male	News	14030
9	Male	Sports	12177
10	Male	Movies	9333
11	Male	Music	8222

21. Which channels are more popular among premium subscribers

```

SELECT
    v.channel_id,
    v.channel_name,

```

```

SUM(v.duration_minutes) AS total_watch_time
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
WHERE d.subscription_type = 'Premium'
GROUP BY v.channel_id, v.channel_name
ORDER BY total_watch_time DESC;

```

[Query results](#)

[Query stats](#)

Completed

Time in queue: 56 ms Run time: 1.276 sec Data scanned: 449.92 KB

Results (50)

Copy Download results CSV

Search rows

#	channel_id	channel_name	total_watch_time
1	CH045	Zee Bangla	2762
2	CH009	Colors Rishtey	2598
3	CH041	Surya TV	2577
4	CH033	Colors Tamil	2416
5	CH018	Sony Sports Ten 3	2109
6	CH020	Eurosport India	2059
7	CH042	Zee Kannada	1985
8	CH008	Zee Anmol	1953
9	CH017	Sony Sports Ten 2	1859

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

```

SELECT
v.show_name,
AVG(v.engagement_score) AS avg_engagement
FROM viewership_stream_stream1 v
JOIN demographics d ON v.user_id = d.user_id
WHERE d.gender = 'Female'
AND d.region IN ('Central', 'East')
GROUP BY v.show_name
ORDER BY avg_engagement DESC
LIMIT 5;

```

[Results \(5\)](#)

Copy Download results CSV

Search rows

#	show_name	avg_engagement
1	Show_118	0.99
2	Show_8	0.97
3	Show_6	0.97
4	Show_23	0.965
5	Show_169	0.96