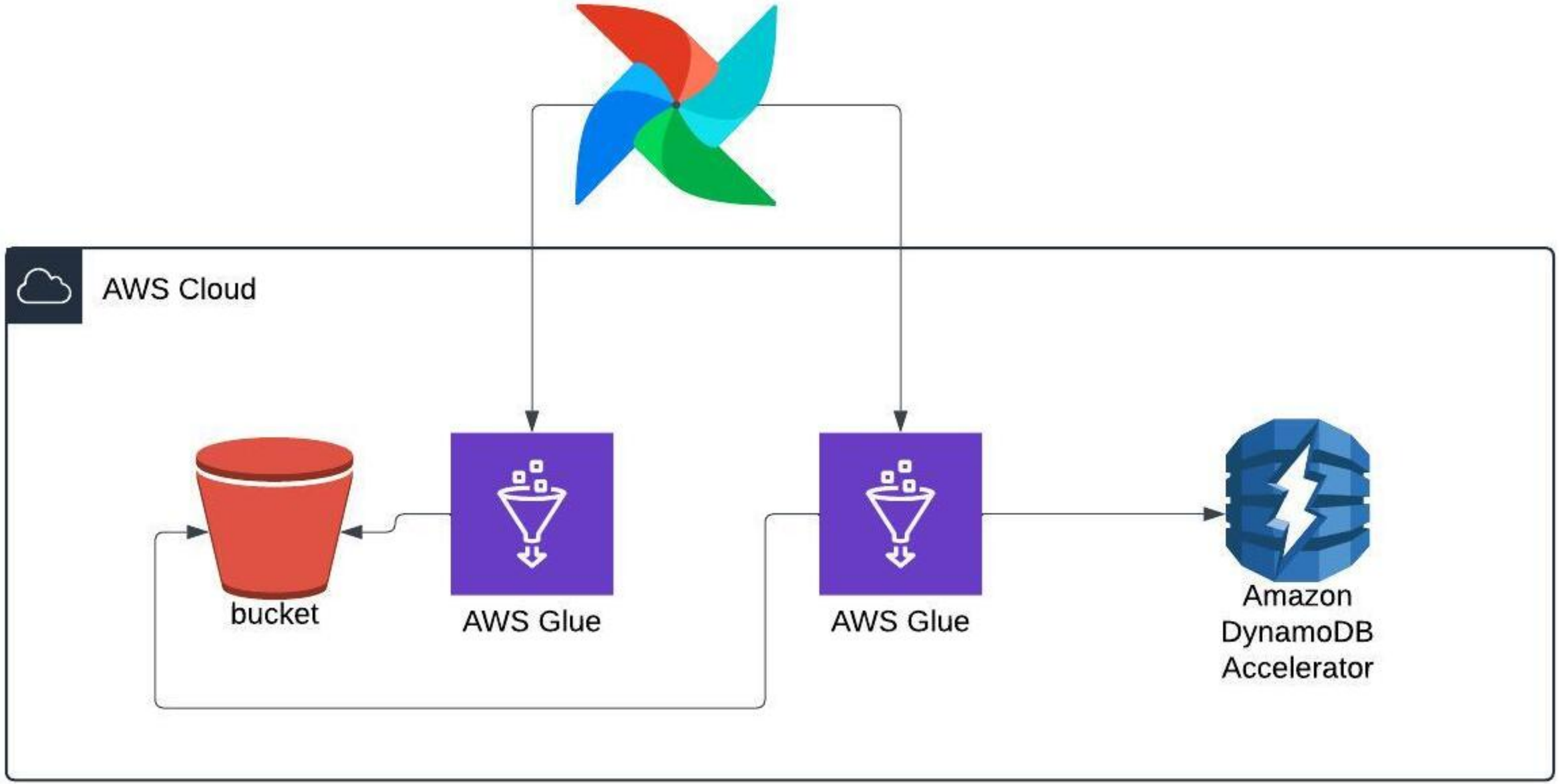
ETL Using Apache Airflow, Aws Glue and Pyspark

Objective

- Trigger: Use Apache Airflow to orchestrate two AWS Glue jobs (PySpark and Python-based) for processing data stored in Amazon S3.
- Transform: Perform data transformations through the AWS Glue jobs as per business requirements.
- Load: Store the processed data into Amazon DynamoDB for real-time or transactional use cases.
- Organize: Move the processed files within Amazon S3 to a designated location for archiving or further processing.

Tools Used

- Apache Airflow: For workflow orchestration and scheduling of Glue job triggers.
- AWS Glue: For data transformation using PySpark and Python scripts.
- Amazon S3: For storing raw and processed data.
- Amazon DynamoDB: As the target database for storing processed data.
- Python: For scripting and implementing transformation logic within Glue jobs.
- AWS SDK (boto3): For interacting with AWS services programmatically.


AWS Cloud
bucket
AWS Glue
AWS Glue
Amazon
DynamoDB
Accelerator

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 12:14 UTC R

DAGs

All 2 | Active 0 | Paused 2 | Running 0 | Failed 0

Filter DAGs by tag

Search DAGs

Auto-refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
process-songs-metrics	airflow		*/5 * * * *		2025-01-16, 12:05:00			
sample_etl_pipeline	airflow		None					

« ‹ 1 › »

Showing **1-2** of **2** DAGs

Version: v2.10.3
Git Version: .release:c99887ec11ce3e1a43f2794fcf36d27555140f00

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 12:15 UTC R

16/01/2025 12:15 All Run Types All Run States Clear Filters Auto-refresh 25

Press shift + / for Shortcuts

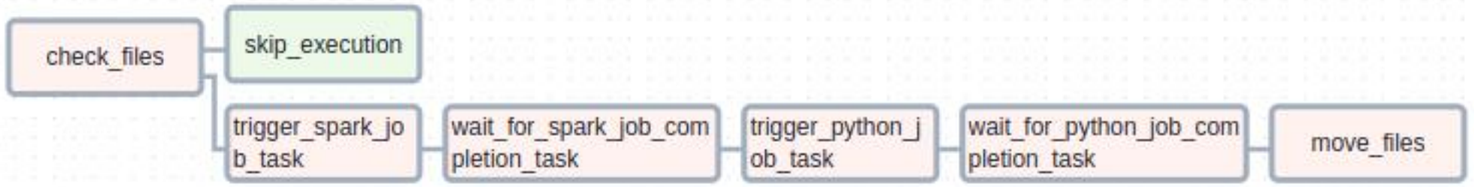
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status

DAG
process-songs-metrics

Details Graph Gantt Code Event Log Run Duration Task Duration Calendar

Layout: Left -> Right

- check_files
- trigger_spark_job_task
- wait_for_spark_job_completion_task
- trigger_python_job_task
- wait_for_python_job_completion_task
- move_files
- skip_execution



dag-glue-workflow.py > ...

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator, BranchPythonOperator
from airflow.operators.dummy_operator import DummyOperator
from datetime import datetime, timedelta
import boto3
import logging
import time

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime.now() - timedelta(days=1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=15)
}

dag = DAG('process-songs-metrics',
          default_args=default_args,
          description='Trigger Glue job when new files are uploaded to S3 and manage output',
          schedule_interval='*/5 * * * *',
          catchup=False)

def check_files_in_s3(prefix):
    s3 = boto3.client('s3')
    bucket_name = 'gd-aws-de-labs'

    response = s3.list_objects_v2(Bucket=bucket_name, Prefix=prefix)
    contents = response.get('Contents', [])
    logging.info(f"Contents in {prefix}: {contents}")
```

dag-glue-workflow.py > ...

```python
            schedule_interval='*/5 * * * *',
            catchup=False)

def check_files_in_s3(prefix):
    s3 = boto3.client('s3')
    bucket_name = 'gd-aws-de-labs'

    response = s3.list_objects_v2(Bucket=bucket_name, Prefix=prefix)
    contents = response.get('Contents', [])
    logging.info(f"Contents in {prefix}: {contents}")

    for obj in contents:
        if obj['Size'] > 0:
            logging.info(f"Non-empty file found in {prefix}: {obj['Key']}")
            return True

    logging.info(f"No non-empty files found in {prefix}")
    return False

def check_all_files(**kwargs):
    logging.info("Checking for files in S3 prefixes")
    user_streams = check_files_in_s3('spotify_data/user-streams/')
    songs = check_files_in_s3('spotify_data/songs/')
    users = check_files_in_s3('spotify_data/users/')

    logging.info(f"user_streams: {user_streams}, songs: {songs}, users: {users}")

    if user_streams and songs and users:
        logging.info("All directories have files, proceeding with Spark job")
        return 'trigger_spark_job_task'
    else:
        logging.info("One or more directories are missing files, skipping execution")
```

dag-glue-workflow.py > ...

```python
def check_all_files(**kwargs):
    logging.info("Checking for files in S3 prefixes")
    user_streams = check_files_in_s3('spotify_data/user-streams/')
    songs = check_files_in_s3('spotify_data/songs/')
    users = check_files_in_s3('spotify_data/users/')

    logging.info(f"user_streams: {user_streams}, songs: {songs}, users: {users}")

    if user_streams and songs and users:
        logging.info("All directories have files, proceeding with Spark job")
        return 'trigger_spark_job_task'
    else:
        logging.info("One or more directories are missing files, skipping execution")
        return 'skip_execution'

def wait_for_glue_job_completion(job_name, client, poll_interval=60):
    while True:
        response = client.get_job_runs(JobName=job_name, MaxResults=1)
        job_runs = response.get('JobRuns', [])

        if job_runs and job_runs[0]['JobRunState'] in ['RUNNING', 'STARTING', 'STOPPING']:
            logging.info(f"Glue job {job_name} is still running. Waiting for it to finish...")
            time.sleep(poll_interval)
        else:
            logging.info(f"Glue job {job_name} has finished.")
            break

def trigger_glue_job(job_name, **kwargs):
    client = boto3.client('glue', region_name='us-east-2')
    logging.info(f"Checking if Glue job {job_name} is running...")
    wait_for_glue_job_completion(job_name, client)
```

dag-glue-workflow.py > ...

```python
def wait_for_glue_job_completion(job_name, client, poll_interval=60):
            logging.info(f"Glue job {job_name} has finished.")
            break

def trigger_glue_job(job_name, **kwargs):
    client = boto3.client('glue', region_name='us-east-2')
    logging.info(f"Checking if Glue job {job_name} is running...")
    wait_for_glue_job_completion(job_name, client)
    logging.info(f"Triggering Glue job: {job_name}")
    response = client.start_job_run(JobName=job_name)

def wait_for_spark_job_completion(**kwargs):
    glue_job_name = 'calculate_metrics_etl'
    client = boto3.client('glue', region_name='us-east-2')
    wait_for_glue_job_completion(glue_job_name, client)

def wait_for_python_job_completion(**kwargs):
    glue_job_name = 'insert_metrics_dynamo'
    client = boto3.client('glue', region_name='us-east-1')
    wait_for_glue_job_completion(glue_job_name, client)

def move_files_to_archived(**kwargs):
    s3 = boto3.client('s3')
    bucket = 'gd-aws-de-labs'
    source_prefix = 'spotify_data/user-streams/'
    dest_prefix = 'spotify_data/user-streams-archived/'

    response = s3.list_objects_v2(Bucket=bucket, Prefix=source_prefix)
    for obj in response.get('Contents', []):
        source_key = obj['Key']
        dest_key = source_key.replace(source_prefix, dest_prefix)
```

dag-glue-workflow.py > ...

```python
    wait_for_glue_job_completion(glue_job_name, client)

def move_files_to_archived(**kwargs):
    s3 = boto3.client('s3')
    bucket = 'gd-aws-de-labs'
    source_prefix = 'spotify_data/user-streams/'
    dest_prefix = 'spotify_data/user-streams-archived/'

    response = s3.list_objects_v2(Bucket=bucket, Prefix=source_prefix)
    for obj in response.get('Contents', []):
        source_key = obj['Key']
        dest_key = source_key.replace(source_prefix, dest_prefix)

        s3.copy_object(Bucket=bucket, CopySource={'Bucket': bucket, 'Key': source_key}, Key=dest_key)
        s3.delete_object(Bucket=bucket, Key=source_key)

check_files = BranchPythonOperator(
    task_id='check_files',
    python_callable=check_all_files,
    provide_context=True,
    dag=dag
)

trigger_spark_job_task = PythonOperator(
    task_id='trigger_spark_job_task',
    python_callable=trigger_glue_job,
    op_args=['calculate_metrics_etl'],
    provide_context=True,
    dag=dag
)
```

dag-glue-workflow.py > ...

```
    dag=dag
)

wait_for_spark_job_completion_task = PythonOperator(
    task_id='wait_for_spark_job_completion_task',
    python_callable=wait_for_spark_job_completion,
    provide_context=True,
    dag=dag
)
trigger_python_job_task = PythonOperator(
    task_id='trigger_python_job_task',
    python_callable=trigger_glue_job,
    op_args=['insert_metrics_dynamo'],
    provide_context=True,
    dag=dag
)

wait_for_python_job_completion_task = PythonOperator(
    task_id='wait_for_python_job_completion_task',
    python_callable=wait_for_python_job_completion,
    provide_context=True,
    dag=dag
)

move_files = PythonOperator(
    task_id='move_files',
    python_callable=move_files_to_archived,
    provide_context=True,
    dag=dag
)

skip_execution = DummyOperator(
```

dag-glue-workflow.py > ...

```python
    python_callable=trigger_glue_job,
    op_args=['insert_metrics_dynamo'],
    provide_context=True,
    dag=dag
)

wait_for_python_job_completion_task = PythonOperator(
    task_id='wait_for_python_job_completion_task',
    python_callable=wait_for_python_job_completion,
    provide_context=True,
    dag=dag
)

move_files = PythonOperator(
    task_id='move_files',
    python_callable=move_files_to_archived,
    provide_context=True,
    dag=dag
)

skip_execution = DummyOperator(
    task_id='skip_execution',
    dag=dag
)

# Setup the task dependencies
check_files >> [trigger_spark_job_task, skip_execution]
trigger_spark_job_task >> wait_for_spark_job_completion_task >> trigger_python_job_task >> wait_for_python_job_completion_
```

(import) DummyOperator: Any

0 △ 2 ⚠ 0 Ln 11, Col 30 Spaces: 4 UTF-8 LF

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
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

spotify_data/

Copy S3 URI

Objects | Properties

Objects (3) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
songs/	Folder	-	-	-
user-streams/	Folder	-	-	-
users/	Folder	-	-	-

AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Data Integration and ETL
Legacy pages

What's New
Documentation
AWS Marketplace

Enable compact mode
Enable new navigation

AWS Glue Studio Info

Create job Info

Author in a visual interface focused on data flow.
Visual ETL

Author using an interactive code notebook.
Notebook

Author code with a script editor.
Script editor

Example jobs Info

Create example job

Your jobs (2) Info

Actions | Run job

Filter jobs by property

1

Job name	Type	Created by	Last modified	AWS Glue version
insert_metrics_dynamo	Python shell	Script	16/01/2025, 16:45:59	
calculate_metrics_etl	Glue ETL	Script	16/01/2025, 16:41:18	5.0

glue pyspark.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, count, sum as _sum, avg, approx_count_distinct, expr, rank, desc
from pyspark.sql.window import Window

spark = SparkSession.builder \
    .appName("Spotify Advanced KPI Processing") \
    .getOrCreate()

# Define S3 bucket and file paths
bucket_name = 'nl-aws-de-labs'
songs_file_path = f's3a://{bucket_name}/spotify_data/songs/'
users_file_path = f's3a://{bucket_name}/spotify_data/users/'
user_streams_path = f's3a://{bucket_name}/spotify_data/user-streams/'

# Read the CSV files into DataFrames
songs_df = spark.read.csv(songs_file_path, header=True, inferSchema=True)
users_df = spark.read.csv(users_file_path, header=True, inferSchema=True)
user_streams_df = spark.read.csv(user_streams_path + "*", header=True, inferSchema=True)

# Extract the date from listen_time to use as report_date
user_streams_df = user_streams_df.withColumn("report_date", to_date(col("listen_time")))

# Ensure there are no null keys where joins are going to happen
user_streams_df = user_streams_df.filter(user_streams_df["track_id"].isNotNull())
songs_df = songs_df.filter(songs_df["track_id"].isNotNull())

# Calculate KPIs for each song on a daily basis
song_kpis_df = user_streams_df.groupBy("track_id", "report_date").agg(
    count("*").alias("total_listens"),
    approx_count_distinct("user_id").alias("unique_users"),
    _sum(expr("unix_timestamp(listen_time)")).alias("total_listening_time"),
    avg(expr("unix_timestamp(listen_time)")).alias("avg_listening_time_per_user")
```

glue pyspark.py > ...

```python
# Calculate KPIs for each song on a daily basis
song_kpis_df = user_streams_df.groupBy("track_id", "report_date").agg(
    count("*").alias("total_listens"),
    approx_count_distinct("user_id").alias("unique_users"),
    _sum(expr("unix_timestamp(listen_time)")).alias("total_listening_time"),
    avg(expr("unix_timestamp(listen_time)")).alias("avg_listening_time_per_user")
)

# Join with songs data to get genre
song_kpis_with_details_df = song_kpis_df.join(songs_df, "track_id")

# Window specification for ranking songs within each genre by total_listens
windowSpec = Window.partitionBy("report_date", "track_genre").orderBy(desc("total_listens"))
ranked_songs_df = song_kpis_with_details_df.withColumn("rank", rank().over(windowSpec))

# Filter for top 3 songs per genre per day
top_songs_per_genre = ranked_songs_df.filter(ranked_songs_df.rank <= 3)

# Find the top 5 genres based on total listens across all songs and all days
genre_window = Window.partitionBy("report_date").orderBy(desc("total_listens"))
top_genres = top_songs_per_genre.withColumn("genre_rank", rank().over(genre_window)) \
                .filter(col("genre_rank") <= 5)

final_df = top_genres.select(
    col("report_date"),
    col("track_id"),
    col("track_name"),
    col("artists"),
    col("track_genre"),
    col("total_listens"),
    col("unique_users"),
    col("total_listening_time"),
```

glue pyspark.py > ...

```python
top_songs_per_genre = ranked_songs_df.filter(ranked_songs_df.rank <= 3)

# Find the top 5 genres based on total listens across all songs and all days
genre_window = Window.partitionBy("report_date").orderBy(desc("total_listens"))
top_genres = top_songs_per_genre.withColumn("genre_rank", rank().over(genre_window)) \
                .filter(col("genre_rank") <= 5)

final_df = top_genres.select(
    col("report_date"),
    col("track_id"),
    col("track_name"),
    col("artists"),
    col("track_genre"),
    col("total_listens"),
    col("unique_users"),
    col("total_listening_time"),
    col("avg_listening_time_per_user")
)

# Write the final DataFrame to the same S3 bucket as a CSV file
output_path = f's3a://{bucket_name}/spotify_data/output/song_kpis/'
final_df.write.mode("overwrite").csv(output_path, header=True)
spark.stop()
```

glue_dynamo.py > upsert_item

```python
import boto3
import csv
from io import StringIO
from decimal import Decimal
import logging

# Setting up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Initialize a boto3 client
s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('track_level_reports')

# S3 bucket and file path
bucket_name = 'gd-aws-de-labs'
file_path = 'spotify_data/output/song_kpis/'

# Get the latest file from the output folder
response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=file_path)
all_files = response['Contents']
latest_file = max(all_files, key=lambda x: x['LastModified'])['Key']

# Download the latest CSV file content
csv_obj = s3_client.get_object(Bucket=bucket_name, Key=latest_file)
body = csv_obj['Body'].read().decode('utf-8')
csv_data = StringIO(body)
csv_reader = csv.DictReader(csv_data)

# Function to upsert into DynamoDB
def upsert_item(row):
    try:
```

(variable) bucket_name: Literal['gd-aws-de-labs']

glue_dynamo.py > upsert_item

```python
# Function to upsert into DynamoDB
def upsert_item(row):
    try:
        # Attempt to parse and insert/update data in DynamoDB
        track_id = row['track_id']
        report_date = row['report_date']
        total_listens = int(row['total_listens'])
        unique_users = int(row['unique_users'])
        total_listening_time = Decimal(str(row['total_listening_time']))
        avg_listening_time_per_user = Decimal(str(row['avg_listening_time_per_user']))

        table.update_item(
            Key={
                'track_id': track_id,
                'report_date': report_date
            },
            UpdateExpression='SET total_listens = :tl, unique_users = :uu, total_listening_time = :tlt, avg_listening_time_p
            ExpressionAttributeValues={
                ':tl': total_listens,
                ':uu': unique_users,
                ':tlt': total_listening_time,
                ':alt': avg_listening_time_per_user
            },
            ReturnValues="UPDATED_NEW"
        )
        logging.info(f"Successfully processed record for track_id={track_id}, report_date={report_date}")
    except ValueError as e:
        logging.error(f"Error processing row {row}: {e}")

# Process each row in the CSV file
for row in csv_reader:
    upsert_item(row)
```

(variable) avg_listening_time_per_user: Decimal

Ln 43, Col 33 Spaces: 4 UTF-8 LF

glue_dynamo.py > upsert_item

```python
def upsert_item(row):
            table.update_item(
                Key={
                    'track_id': track_id,
                    'report_date': report_date
                },
                UpdateExpression='SET total_listens = :tl, unique_users = :uu, total_listening_time = :tlt, avg_listening_time_p
                ExpressionAttributeValues={
                    ':tl': total_listens,
                    ':uu': unique_users,
                    ':tlt': total_listening_time,
                    ':alt': avg_listening_time_per_user
                },
                ReturnValues="UPDATED_NEW"
            )
            logging.info(f"Successfully processed record for track_id={track_id}, report_date={report_date}")
        except ValueError as e:
            logging.error(f"Error processing row {row}: {e}")

# Process each row in the CSV file
for row in csv_reader:
    upsert_item(row)

# Close the StringIO object
csv_data.close()
```

DynamoDB

Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings

DAX
Clusters
Subnet groups
Parameter groups
Events

The track_level_reports table was created successfully.

Tables (1) Info

Actions | Delete | Create table

Find tables | Any tag key | Any tag value

1

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion pro
track_level_reports	Active	track_id (S)	report_date (S)	0	0	Off

DynamoDB

Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings

DAX
Clusters
Subnet groups
Parameter groups
Events

Any tag value

Find tables

1

track_level_reports

Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. Learn more

Edit PITR

General information Info

Partition key
track_id (String)

Sort key
report_date (String)

Capacity mode
On-demand

Table status
Active

Alarms
No active alarms

Point-in-time recovery (PITR) Info
Off

Resource-based policy Info
Not active

Additional info

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 12:24 UTC R

16/01/2025 12:23 | All Run Types | All Run States | Clear Filters | Auto-refresh | 25

Press shift + / for Shortcuts

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

Duration

00:01:06

00:00:33

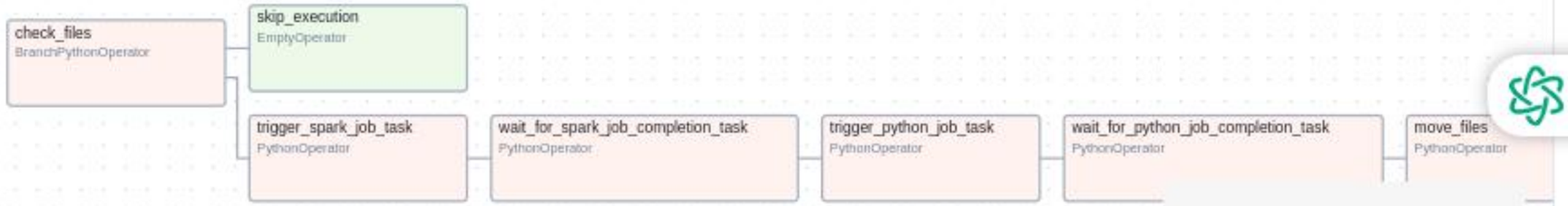
00:00:00

check_files
trigger_spark_job_task
wait_for_spark_job_completion_task
trigger_python_job_task
wait_for_python_job_completion_task
move_files
skip_execution

DAG
process-songs-metrics

Details | Graph | Gantt | Code | Event Log | Run Duration | Task Duration | Calendar

Layout: Left -> Right



AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Data Integration and ETL
Legacy pages

What's New
Documentation
AWS Marketplace

Enable compact mode
Enable new navigation

calculate_metrics_etl

Last modified on 16/01/2025, 16:41:18 | Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job runs (1/1) Info

Last updated (UTC)
January 16, 2025 at 12:24:19

View details | Stop job run | Troubleshoot with AI

Table View | Card View

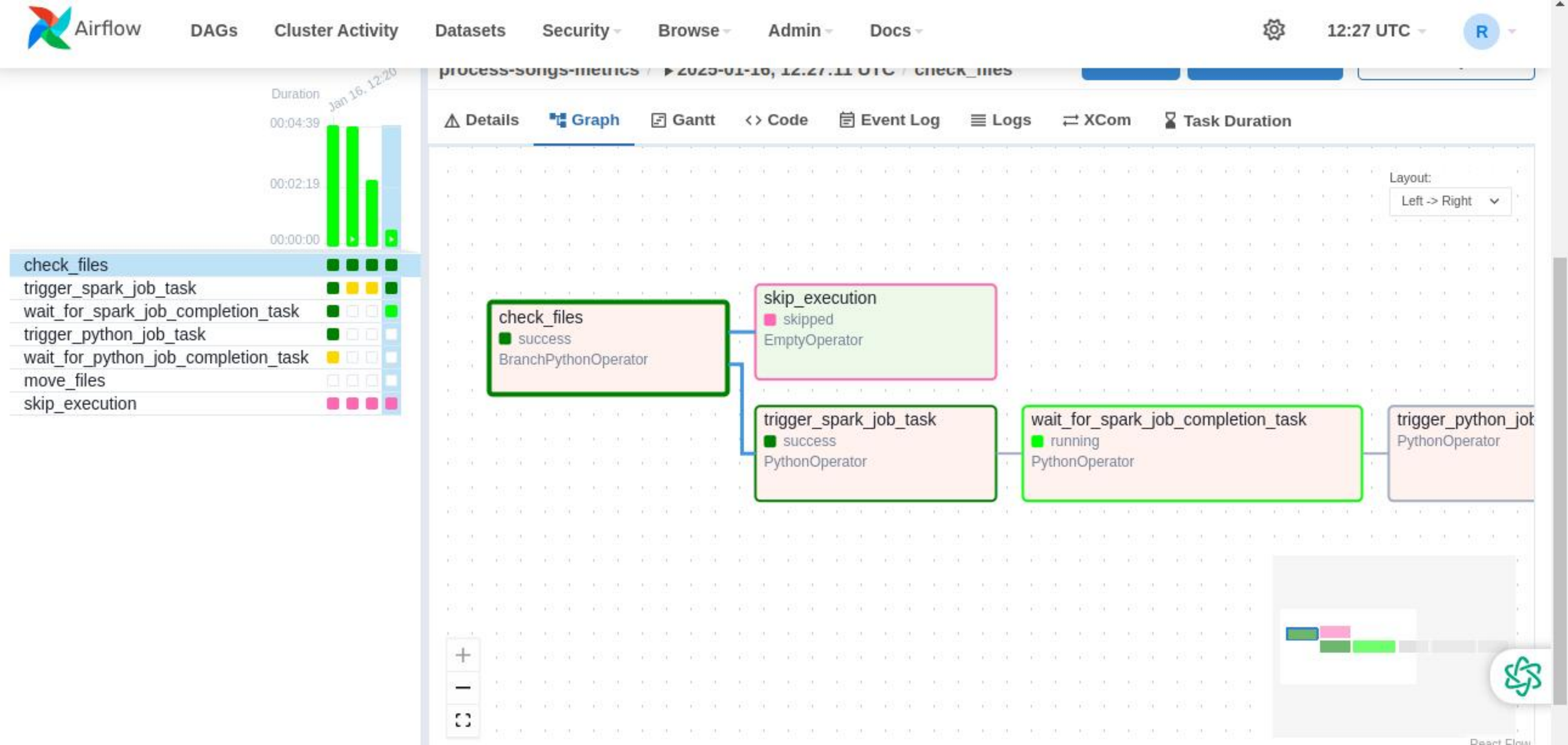
Filter job runs by property

1

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacit...	Wo
Running	0	01/16/2025 17:53:16	-	54 s	3 DPUs	G.1

Run details | Input arguments (10) | Continuous logs | Run insights | Metrics | Stop job run

Job name	Start time (Local)	Glue version	Last modified on (Local)


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
12:27 UTC
R
process-songs-metrics / ▶2025-01-16, 12:27:11 UTC / check_files
Duration
Jan 16, 12:20
00:04:39
00:02:19
00:00:00
check_files
trigger_spark_job_task
wait_for_spark_job_completion_task
trigger_python_job_task
wait_for_python_job_completion_task
move_files
skip_execution
Details
Graph
Gantt
Code
Event Log
Logs
XCom
Task Duration
Layout:
Left -> Right
check_files
success
BranchPythonOperator
skip_execution
skipped
EmptyOperator
trigger_spark_job_task
success
PythonOperator
wait_for_spark_job_completion_task
running
PythonOperator
trigger_python_job
PythonOperator
React Flow

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 12:32 UTC R

process-songs-metrics / ▶2025-01-16, 12:27:11 UTC / wait_for_spark_job_completion_task

Clear task | Mark state as... | Filter DAG by task

Duration

Jan 16, 12:20

00:09:47

00:04:53

00:00:00

- check_files
- trigger_spark_job_task
- wait_for_spark_job_completion_task
- trigger_python_job_task
- wait_for_python_job_completion_task
- move_files
- skip_execution

Details | Graph | Gantt | Code | Event Log | Logs | XCom | Task Duration

Layout: Left -> Right

ask

wait_for_spark_job_completion_task
success
PythonOperator

trigger_python_job_task
success
PythonOperator

wait_for_python_job_completion_task
up_for_retry
PythonOperator

React Flow

AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
Data Catalog
Data Integration and ETL
Legacy pages

What's New
Documentation
AWS Marketplace

Enable compact mode
Enable new navigation

insert_metrics_dynamo

Last modified on 16/01/2025, 16:45:59

Actions | Save | Run

Script | Job details | Runs | Data quality | Schedules | Version Control | Upgrade analysis - *preview*

Job runs (1/3) Info

Last updated (UTC)
January 16, 2025 at 12:32:59

View details | Stop job run | Troubleshoot with AI

Table View | Card View

Filter job runs by property

1

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacit...	Wo
Succeeded	0	01/16/2025 18:02:24	01/16/2025 18:02:40	12 s	0.0625 DPUs	-
Succeeded	0	01/16/2025 17:59:31	01/16/2025 17:59:56	16 s	0.0625 DPUs	-
Failed	0	01/16/2025 17:55:26	01/16/2025 17:55:56	20 s	0.0625 DPUs	-

Run details | Input arguments (6) | Continuous logs | Run insights | Metrics | Troubleshooting

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
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

spotify_data/

Copy S3 URI

Objects | Properties

Objects (4) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
output/	Folder	-	-	-
songs/	Folder	-	-	-
user-streams/	Folder	-	-	-
users/	Folder	-	-	-

song_kpis/

Copy S3 URI

Objects | Properties

Objects (2) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions

Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
_SUCCESS	-	January 16, 2025, 18:02:02 (UTC+05:30)	0 B	Standard
part-00000-0a07deac-256e-4e8f-b33a-55c62205e7cc-c000.csv	csv	January 16, 2025, 18:02:02 (UTC+05:30)	2.4 KB	Standard

Airflow DAGs Cluster Activity Datasets Security Browse Admin Docs 06:42 UTC R

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
process-songs-metrics2 / Run ▸2025-01-17, 06:34:36 UTC

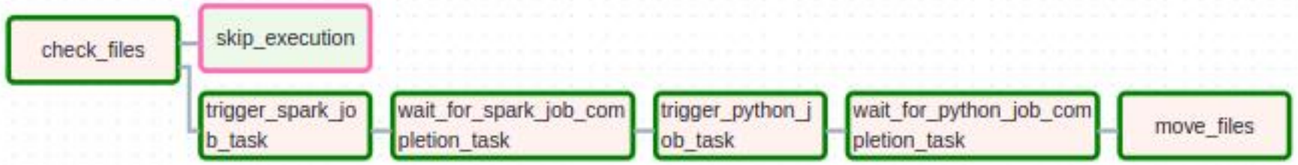
Clear Mark state as...

Duration
00:02:34
00:01:17
00:00:00

Details Graph Gantt Code Event Log

Layout:
Left -> Right

check_files
trigger_spark_job_task
wait_for_spark_job_completion_task
trigger_python_job_task
wait_for_python_job_completion_task
move_files
skip_execution



DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings

DAX
Clusters
Subnet groups
Parameter groups
Events

PartiQL editor

Operations performed using the PartiQL editor might incur charges. Learn more

Tables (1)

Find tables

1

track_level_reports

Query 1 | Query 2

```
1 SELECT * FROM track_level_reports;
```

Run Clear

Table view JSON view

DynamoDB

Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings

DAX
Clusters
Subnet groups
Parameter groups
Events

Find items

1

avg_listening_time_per_user	report_date	total_listening_time
1719314040	2024-06-25	5157942120
1719324855.6666667	2024-06-25	5157974567
1719325085.3333333	2024-06-25	5157975256
1719311941	2024-06-25	5157935823
1719319292	2024-06-25	5157957876
1719294071	2024-06-25	5157882213
1719337041	2024-06-25	5158011123
1719307529	2024-06-25	5157922587
1719339753	2024-06-25	5158019259
1719298982.25	2024-06-25	6877195929
1719340525.75	2024-06-25	6877362103

DynamoDB

Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations New
Reserved capacity
Settings

DAX
Clusters
Subnet groups
Parameter groups
Events

Find items

1

total_listening_time	unique_users	track_id	total_lis
5157942120	3	7wcaieaDTt...	3
5157974567	3	1067ZkQkZ...	3
5157975256	3	3a2jFwnts4...	3
5157935823	3	1mGO8rwC...	3
5157957876	3	5JyEUS5Ew...	3
5157882213	3	4i3xoVyj8W...	3
5158011123	3	2u1EtHkbp...	3
5157922587	3	0pBgoq3dz...	3
5158019259	3	499SvLeXf...	3
6877195929	4	4686EWn7t...	4
6877362103	4	19coiw9dD	4