COSC6376 - Final Report
CloudTunes: Automating Spotify ETL Data Pipeline using AWS

Narayan Soni(2301240) **Hitesh Reddy Yerradoddi (2301944)**
nsoni@cougarnet.uh.edu **hyerrado@cougarnet.uh.edu**
Monika Kommineni (2287571)
mkommine@cougarnet.uh.edu

December 12, 2023

CONTENTS

List of Figures:

1. Introduction

In a world where music consumption has transitioned into a digital era, the complexities of managing and understanding one's music preferences continue to evolve. Enter CloudTunes – an innovative project designed to revolutionize how we interact with and derive insights from music data. Our aim is to harness the power of Spotify's vast music database and transform it into a resourceful platform that not only enhances user experience but also offers valuable insights into music consumption patterns.

The main objective of the project is to create a system that extracts, processes, and loads Spotify data into a format that is useful for various purposes.
It has the following goals:

- Gather data from the Spotify Web API, including user-specific information like playlists, saved songs, listening history, and metadata about songs, albums, and artists.
- Process the raw data to make it usable and informative. This can involve cleaning, structuring, and enriching the data to extract valuable insights and facilitate various applications.
- Store the transformed data efficiently and securely, on AWS storage services.
- Use the processed data to derive valuable insights, such as user listening habits, popular songs or genres, and trends in music consumption.

2. Description

CloudTunes is a transformative project focused on harnessing Spotify's wealth of data through AWS services. Its core objectives include extracting personalized information from Spotify's API, processing raw data for enhanced usability, and securely storing this transformed data on AWS. The project employs Lambda functions for data transformation, EventBridge for automated scheduling, and Amazon S3 as the primary data store. Leveraging AWS Glue and Athena, CloudTunes facilitates seamless querying and analysis of Spotify data, offering insights into user listening habits and music trends. Compliance with Spotify API policies and AWS resource limits remains integral to the project's success, ensuring efficient execution and storage management.

3. Architecture Overview

This section provides a high-level view of the overall technical architecture and key components that make up the Spotify ETL pipeline.

3.1 Architectural Diagram

The architectural diagram depicts the end-to-end flow through the pipeline, highlighting the major services and how they integrate.

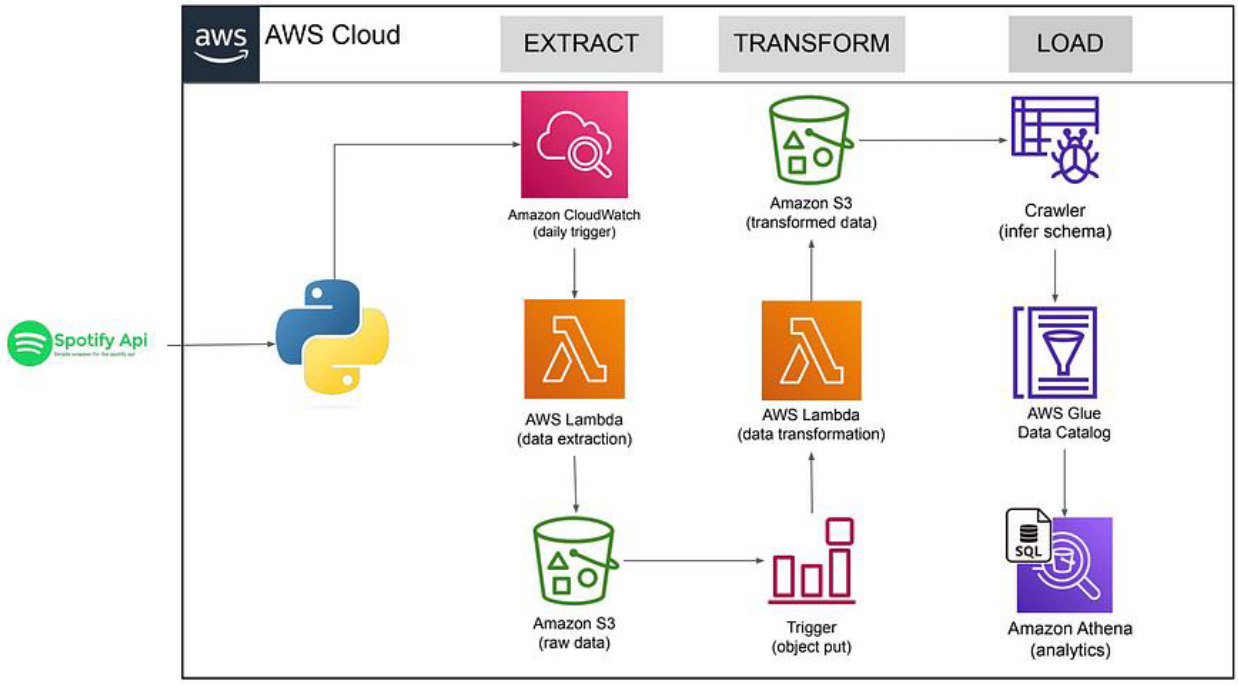


Figure 1: Architecture Diagram

This Spotify ETL pipeline leverages AWS cloud services for automated data extraction, transformation, and loading. A daily crawler retrieves data from the Spotify API and stores it raw in S3. A Lambda function triggered by new data infers the schema and transforms it into a desired format, storing the transformed data in a separate S3 bucket. AWS Glue catalogs the transformed data schema while Athena enables analysis through standard SQL queries. This architecture offers scalability, reliability, cost-effectiveness, and security, making it ideal for managing and analyzing Spotify data.

3.2 Component Details

This subsection explains each component's purpose and function within the overall system:
Data Extraction from Spotify API: The system will feature a robust mechanism to extract personalized song data from Spotify's "Top Songs-Global" playlist using the Spotipy library. This includes obtaining necessary credentials and implementing a Lambda function for periodic data extraction.

- **Automated Triggers with EventBridge:** EventBridge is a serverless scheduling service on AWS, allows cron or pattern-based invocation of other resources. This is used to

trigger the core ETL scripts based on time intervals or data arrival events. In our project the system will utilize AWS EventBridge to schedule and automate the extraction process, ensuring it runs every 12 hours. Additionally, S3 triggers will be implemented for seamless transformation whenever new data is added.

- **Data Transformation with Lambda Functions:** Lambda function is a AWS service providing functions-as-a-service, executes custom code without provisioning servers. This is a house of the Python ETL scripts that handle extract and transform phases of the pipeline. This is stateless and scales automatically. Our project involves the transformation of raw data extracted from Spotify. Lambda functions will be employed to clean and structure the data, generating separate datasets for artist, album, and song information.
- **S3 Buckets:** S3 is a scalable object storage on AWS, used to house all data. This maintains raw input, intermediate working data, and final transformed outputs. This provides data lake capabilities. In our project the system relies on Amazon S3 as the main data store, featuring a well-defined bucket structure. Raw data, intermediate data to process, processed data, and transformed datasets for albums, artists, and songs are stored within the S3 bucket.
- **AWS Glue Crawler for Schema Inference:** Glue Crawler is a AWS service that scans S3 data and infers schema/structure. This Populates metadata in the Glue Data Catalog depicting data tables, columns, formats etc. AWS Glue's crawler functionality will be utilized to automatically scan and analyze the transformed data, inferring its schema. This metadata will be stored in the Glue Data Catalog for reference.
- **Glue Catalog**: Glue Catalog is a fully managed data catalog powered by crawlers, which enables discovery and tracking of datasets hosted in S3 and other data stores.
- **Querying Transformed Data with Athena:** Athena provides SQL querying interface over S3 data, leveraging underlying Glue Catalog metadata. This facilitates analytics without additional servers. In our project the transformed data will be queried using Amazon Athena, allowing users to perform SQL queries for in-depth analysis. This sets up an analytics layer for users to derive insights from the music data.

3.3 Data Flow

The data flow explains the data progression through each stage of the pipeline, from initial extract to final analytics. Key steps include:

- First we extract data from spotify API by registering our application on spotify API so we get the login credentials like client id and secret key which are useful in collecting data from spotify API.
- The collected data is moved into S3 bucket in the raw format which is the same form of the extracted data from spotify API.

- Now we apply the transformation process on the above S3 bucket and store back the transformed data in a different folder in S3 bucket.
- This transformed data is used by the glue crawler to build a glue catalog by inferring the schema.
- Once we have data on the glue catalog we use Amazon Athena to run SQL queries on the data.

4. Implementation

Overview of the process:
Jupyter Notebook is used to develop the initial Python code for data extraction and transformation. We have done the data extraction of raw using spotipy library and pandas library.

ETL (Extract, Transform, Load) Overview:
Extract

- Extract data from Spotify API using the Spotipy library
- Deploy the data extraction code using the Lambda function
- Run trigger using EventBridge to automate data extraction every Tuesday at 4 pm
- Data extract is saved in the spotify-etl-project-06112023/raw_data/to_process folder in the S3 bucket

Transform

- Run S3 trigger when any new data is added into the spotify-etl-project-06112023/raw_data/to_process folder in the S3 bucket. This will run the data transformation code on Lambda
- The transformation code will clean and transform the data to prepare 3 files for the album, artist, and songs. The data will be stored in the 3 subfolders in transformed_data. Lastly, files in the to_process folder will be copied to the processed folder and files in to_process will be deleted. We are just moving data from one folder to another.

Load

- Glue crawler will infer schema when new data arrives in the 3 folders in the transformed_data folder
- Data catalog manage metadata repository
- Query S3 data using Athena

This section dives into the specific technologies and steps taken to construct the Spotify ETL pipeline.

4.1 Spotify API Integration

4.1.1 Overview of Spotify API Capabilities

The Spotify Web API allows developers to access Spotify catalog data including metadata about artists, albums, tracks, playlists, user profiles, and more. Key capabilities offered by the API include:

- Search Spotify catalog entities like tracks, albums, artists using keywords, genre, etc
- Retrieve detailed metadata about artists, albums, tracks, audio features, related artists
- Manage user libraries and playlists - add/remove tracks, create/edit playlists
- Get recommendations for tracks, artists, albums based on seeds
- Follow friends and other Spotify users' public activity and playlists
- Read user's recently played tracks and current playback state
- Where applicable, multiple calls can be batched to improve performance
- The API uses OAuth 2 authentication and provides rich response data in JSON format.
- Rate limiting requires caching and optimization to avoid throttling.

4.1.2 Registering a Developer Account

To use the Spotify API, you first need to register as a Spotify Developer. This is done through the Spotify for Developers dashboard at https://developer.spotify.com. On the dashboard under Developer, choose Register. This will create an account tied to your app.

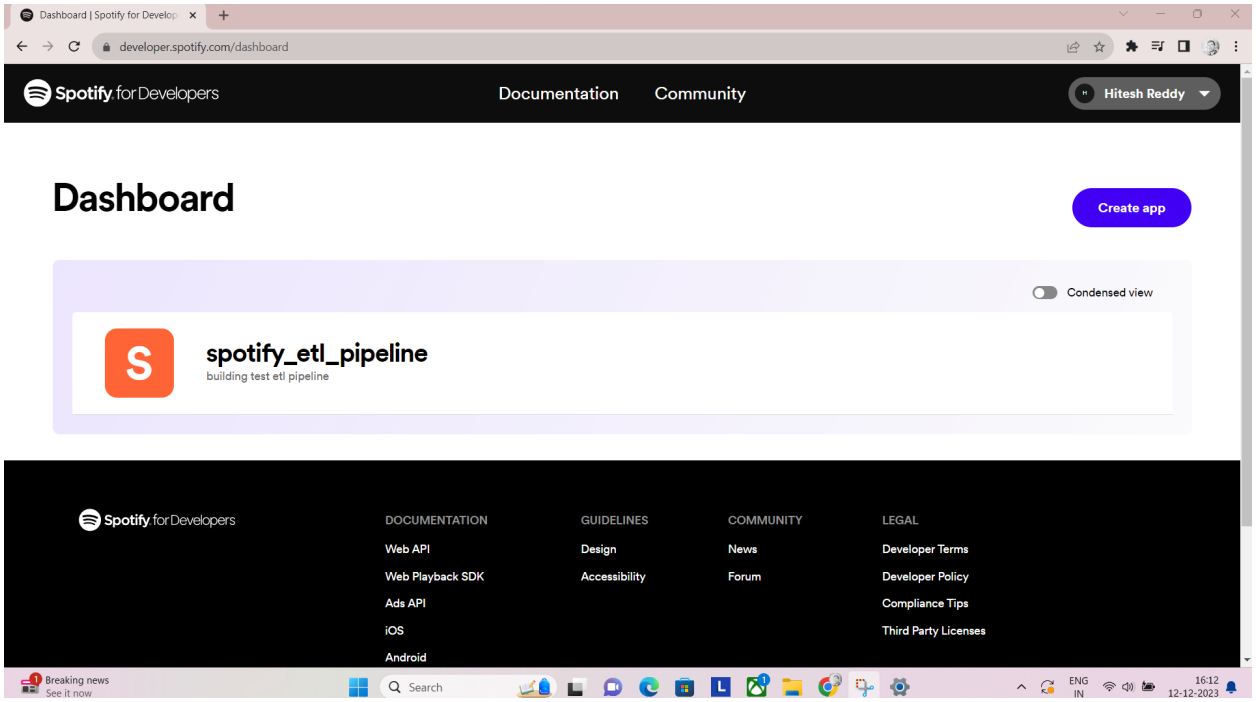


Figure 2: Spotify Developer Account

Once registered, create an app by clicking on CREATE AN APP. Give your app a name, description and set the APP TYPE as Web API. The Signing Secret is used for various authentication purposes. Make a note of the Client ID and Client Secret values on the dashboard, these will be needed later to authenticate API requests.

4.1.3 Obtaining Client Credentials

The Client ID and Client Secret for your app act as client credentials you can use to obtain access tokens for the API. The credentials authenticate your app identity. Treat the values as sensitive credentials.

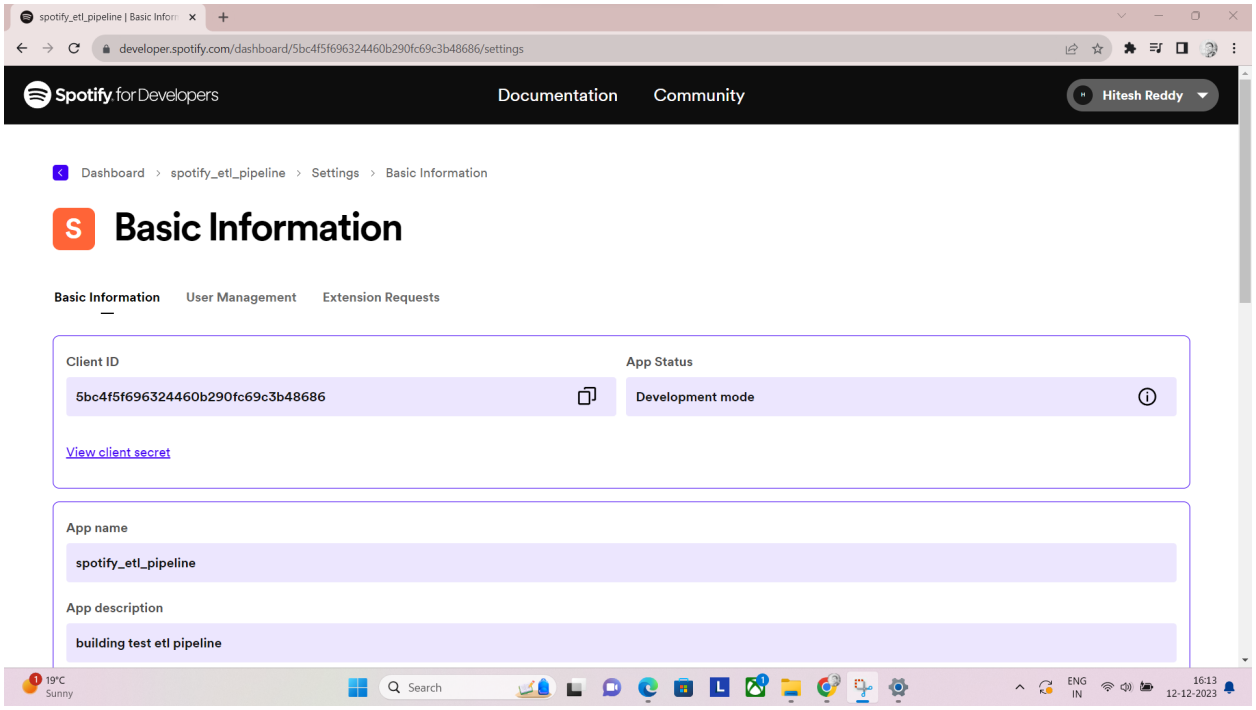


Figure 3: Account Information

The Client ID identifies your application. The Client Secret authorizes access and should be kept secure, similar to a password. Together they grant your app the needed permissions to access specific API resources based on scopes.
When making API calls, you exchange these credentials for an access token. The access token then proves your app's approved identity to access the requested data resources via the API endpoints.

4.1.4 Using Spotipy Library for Access

Spotipy is a convenient Python library for working with the Spotify API. Under the hood it handles the OAuth access token flow using your app's Client ID and Client Secret. It provides helper methods to call various API endpoints, parsing responses into Python objects making development easier.

To use Spotipy, install via pip:

```
pip install spotipy
```

Then in Python code, import and initialize client object:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

client_id = 'YOUR_CLIENT_ID
client_secret = 'YOUR_CLIENT_SECRET'

client_credentials_manager = SpotifyClientCredentials(client_id, client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

The client credentials manager handles access tokens internally. You can now call various methods to search tracks, get playlists etc.

Python Example for Authentication: Here is a simple Python snippet demonstrating getting an access token and using it to search tracks in the Spotify catalog:

```
import json
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

# Spotify credentials
CLIENT_ID = 'abcd1234'
CLIENT_SECRET = 'xyz5678'

# Initialize client credentials manager
client_credentials_manager = SpotifyClientCredentials(CLIENT_ID, CLIENT_SECRET)

# Initialize Spotipy client
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

# Get access token
token = client_credentials_manager.get_access_token()

# Search tracks with access token
results = sp.search(q='Nirvana', type='track')
print(json.dumps(results, indent=4))
```

This covers some key details on getting setup with the Spotify API in Python.

4.2 Create a bucket in S3

The bucket name must be globally unique and we selected US East (N. Virginia) us-east-1 as the AWS region. Below is the folder structure in the S3 bucket we created, we created the main folder name as spotify-etl-project-06112023.

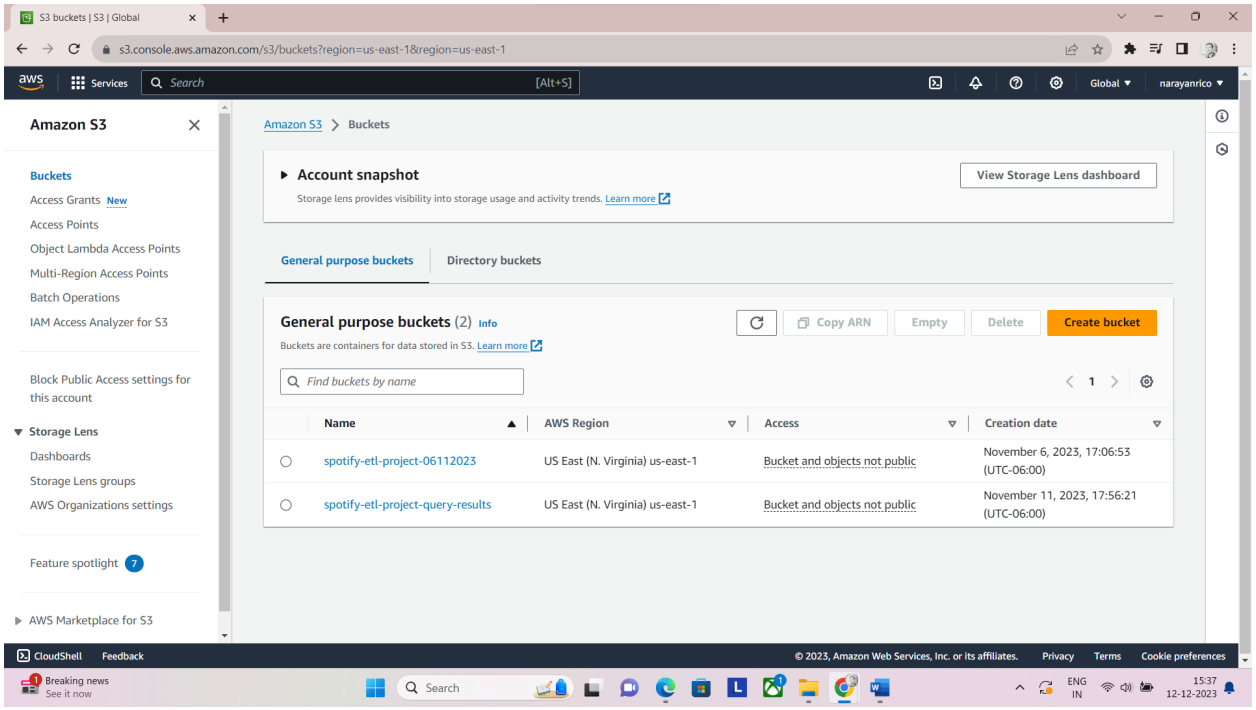


Figure 4: Amazon S3

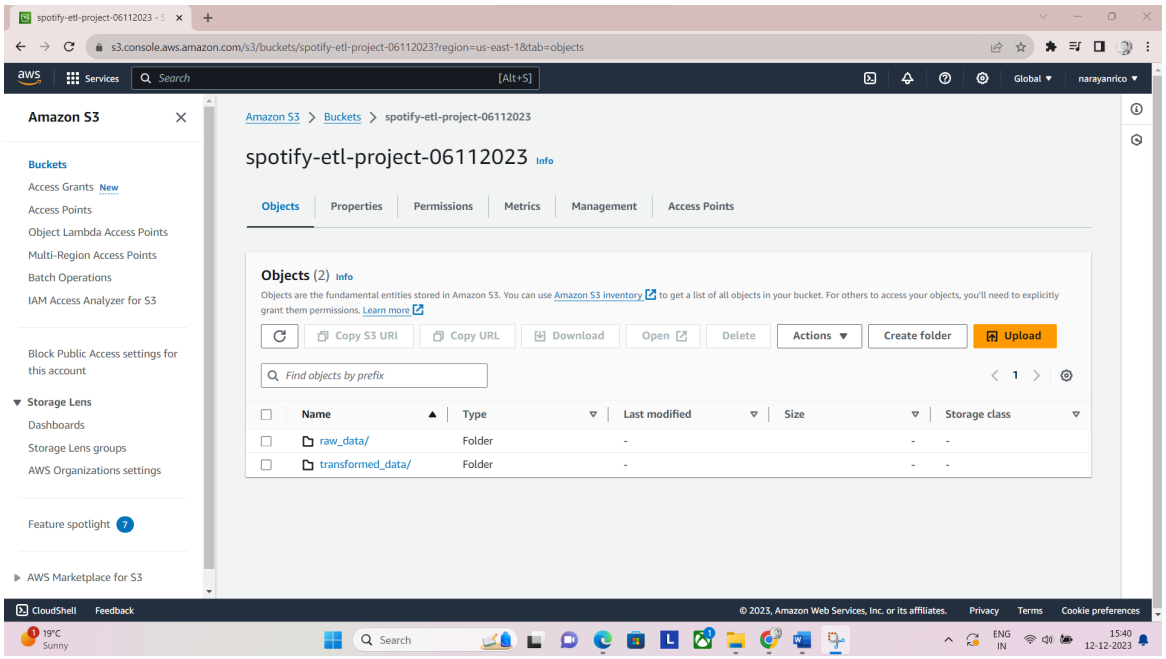


Figure 5: S3 bucket(/soptiy-etl-project-06112023 - Main folder)

/raw_data: Raw data is stored here

- ***to_process:*** When the data extraction function is invoked, data extracted from the API will be stored here
- ***processed:*** When the transformation function is invoked, files in to_process folder will be copied to this folder and the file in to_process will be deleted. We are just moving data from one folder to another. It is demonstrated in the below image

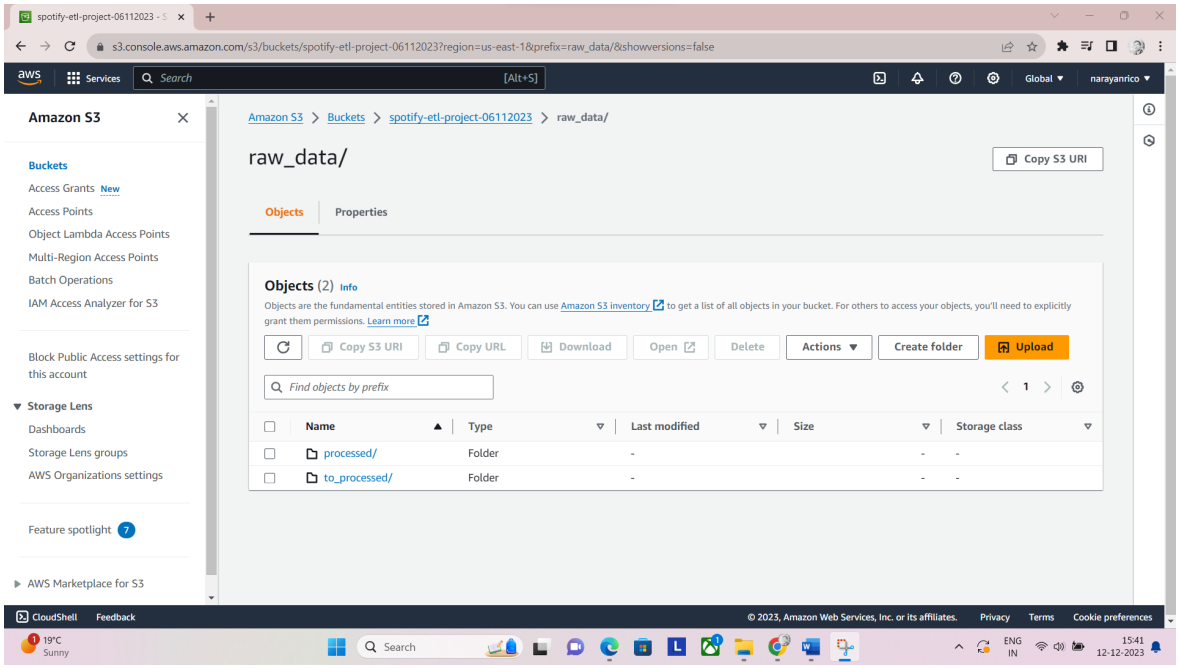


Figure 6: Raw Data

transformed_data: These 3 folders will contain the transformed dataset where basic cleaning and transformation have been applied.

- */album_data*
- */artist_data*
- */song_data*

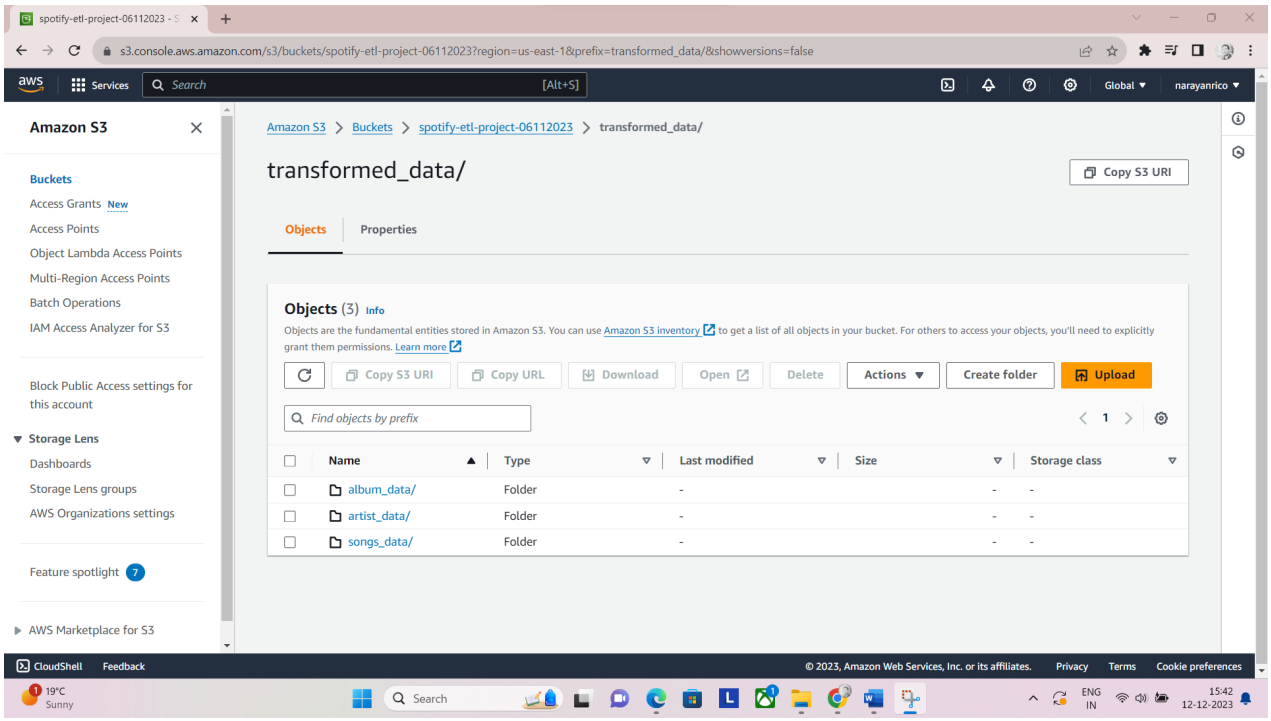


Figure 7: Transformed Data

4.3 Create a layer in Lambda

We will use the Spotipy library for the Spotify web API. A layer can package libraries and other dependencies that you can use with Lambda functions.

4.4 Create a Lambda function (spotify-api-data-function) to extract the raw data from the Spotify API

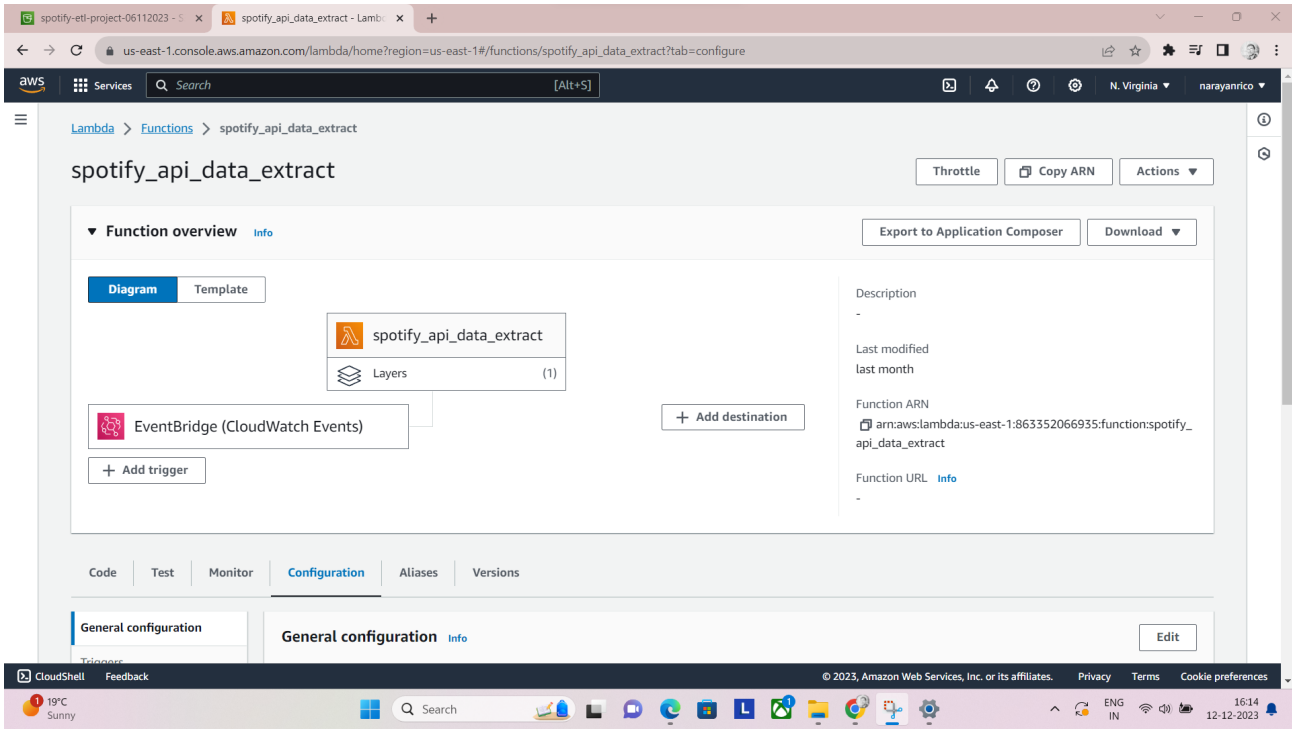


Figure 8: Raw Data Extraction(Lambda function)

We have written the code for the extraction in the code section and setup appropriate environment as follows:

- Select Environment variables and click on the Edit button. Add client_id and client_secret as the keys and add in the respective values. Environment variables can be configured for Lambda function to keep sensitive information separate from the code.
- Select General configuration and click on the Edit button. Set the timeout to 6 min. This can be adjusted accordingly based on the expected runtime and nature of the task. The default timeout is set to 3s. It ensures functions do not run indefinitely and prevents potential excessive costs.
- Select Permissions. Check that the role is set to a role which has these permissions: AmazonS3FullAccess and AWSLambdaRole.
- Otherwise, click on Add permission and attach the 2 policies which grant specific permissions to perform actions within AWS services.
- Add a spotipy layer to the lambda function in order to use the spotify library.

Python code:

```
import json
import os
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import boto3
from datetime import datetime

def lambda_handler(event, context):
    client_id=os.environ.get('client_id')
    client_secret=os.environ.get('client_secret')

    client_credentials_manager=SpotifyClientCredentials(client_id=client_id,
client_secret=client_secret)
    sp = spotipy.Spotify(client_credentials_manager= client_credentials_manager)
    playlists=sp.user_playlists('spotify')

    playlist_link = "https://open.spotify.com/playlist/37i9dQZEVXbNG2KDcFcKOF"
    playlist_URI=playlist_link.split("/")[-1]

    spotify_data=sp.playlist_tracks(playlist_URI)

    client=boto3.client('s3')

    filename="spotify_raw_" + str(datetime.now())+".json"

    client.put_object(
        Bucket="spotify-etl-project-06112023",
        Key="raw_data/to_processed/" + filename,
        Body=json.dumps(spotify_data)
        )
```

Code explanation:

- lambda_handler function is the entry point for the Lambda function.
- It retrieves the Spotify API credentials from the environment variables and playlist data
- Filename is based on the current timestamp and data extracted is saved in the to_process folder
- Configure test event. Use any event name and use the default setting Click on Deploy.
- After deployment, you can invoke the Lambda function manually (Test event) or through triggers like CloudWatch events, S3 events or other AWS services

4.5 Create a Lambda function (transform_spotify_data) to transform the data

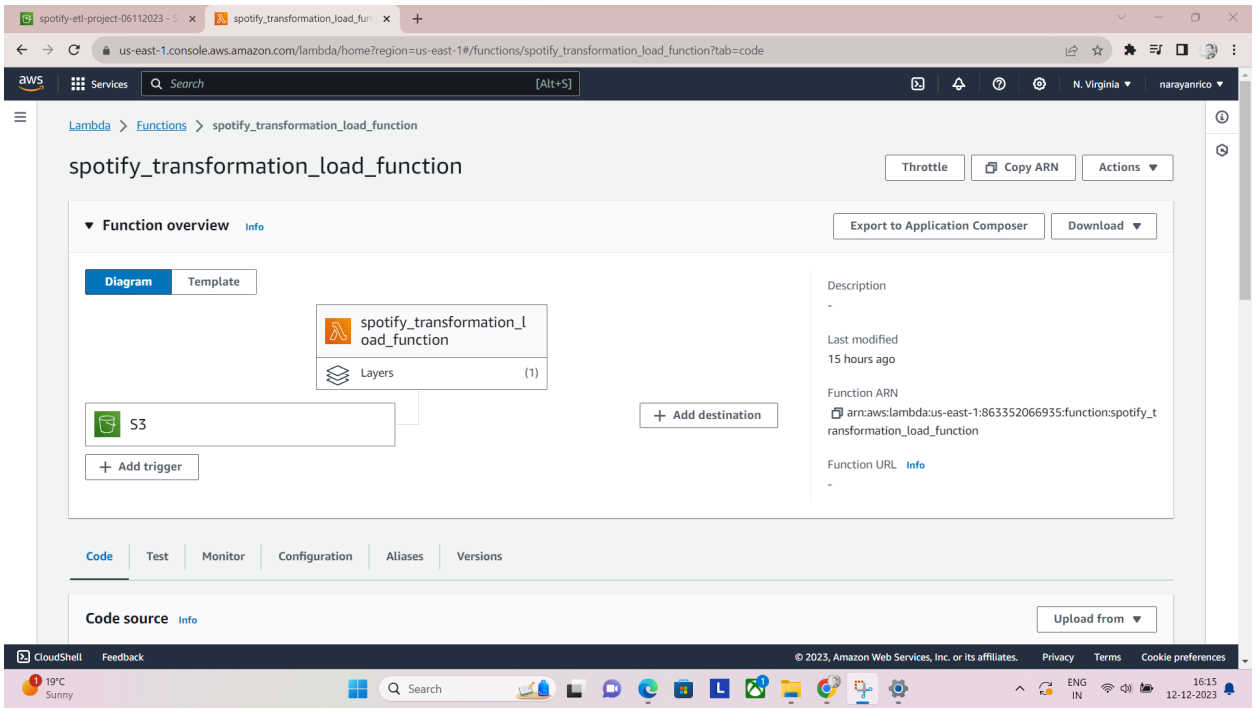


Figure 9: Data Transformation(Lambda function)

1. We can select the same role used in the previous function.
2. We create transform_spotify_data function which is used to convert the json files from the raw_data bucket to the csv files in a structured format.
3. Configuration tab (timeout):
    a. Select General configuration and click on the Edit button.
    b. Set the timeout to 5 min.
4. Add the Pandas layer: AWSSDKPandas-Python38 to the lambda function.

Python code:

```
import json
import boto3
from datetime import datetime
from io import StringIO
import pandas as pd
```

```
def album(data):
    album_list = []
    for row in data['items']:
        album_id=row['track']['album']['id']
        album_name=row['track']['album']['name']
        album_release_date=row['track']['album']['release_date']
        album_total_tracks=row['track']['album']['total_tracks']
        album_url=row['track']['album']['external_urls']['spotify']
        album_element={'album_id' : album_id , 'name' : album_name , 'release_date' :
album_release_date , 'total_tracks' : album_total_tracks , 'url' : album_url }
        album_list.append(album_element)
    return album_list

def artist(data):
    artist_list = []
    for row in data['items']:
        for key,value in row.items():
            if key == "track":
                for artist in value['artists']:
                    artist_dict = {'artist_id': artist['id'], 'artist_name': artist['name'], 'external_url':
artist['href']}
                    artist_list.append(artist_dict)
    return artist_list

def songs(data):
    song_list=[]
    for row in data['items']:
        song_id=row['track']['id']
        song_name=row['track']['name']
        song_duration=row['track']['duration_ms']
        song_url=row['track']['external_urls']['spotify']
        song_popularity=row['track']['popularity']
        song_added=row['added_at']
        album_id=row['track']['album']['id']
        artist_id=row['track']['album']['artists'][0]['id']
        song_element={'sond_id':song_id , 'song_name':song_name,
'duration_ms':song_duration, 'url':song_url, 'popularity':song_popularity,
'song_added':song_added, 'album_id': album_id, 'artist_id': artist_id }
        song_list.append(song_element)
    return song_list

def lambda_handler(event, context):
    s3=boto3.client('s3')
    Bucket="spotify-etl-project-06112023"
    Key="raw_data/to_processed/"
```

```
    spotify_data=[]
    spotify_keys=[]
    for file in s3.list_objects(Bucket=Bucket, Prefix=Key)['Contents']:
        file_key=file['Key']
        if file_key.split(".")[-1]=="json":
            response=s3.get_object(Bucket=Bucket, Key=file_key)
            content=response['Body']
            jsonObject=json.loads(content.read())
            spotify_data.append(jsonObject)
            spotify_keys.append(file_key)
    for data in spotify_data:
        album_list=album(data)
        artist_list=artist(data)
        song_list=songs(data)

        album_df=pd.DataFrame.from_dict(album_list)
        album_df=album_df.drop_duplicates(subset=['album_id'])

        artist_df=pd.DataFrame.from_dict(artist_list)
        artist_df=artist_df.drop_duplicates(subset=['artist_id'])
        song_df=pd.DataFrame.from_dict(song_list)

        album_df['release_date']=pd.to_datetime(album_df['release_date'], errors='coerce')
        song_df['song_added']=pd.to_datetime(song_df['song_added'], errors='coerce')

        songs_key = "transformed_data/songs_data/song_transformed_"+ str(datetime.now())+
".csv"
        song_buffer=StringIO()
        song_df.to_csv(song_buffer, index=False)
        song_content=song_buffer.getvalue()
        s3.put_object(Bucket=Bucket, Key=songs_key, Body=song_content)

        album_key = "transformed_data/album_data/album_transformed_"+ str(datetime.now())+
".csv"
        album_buffer=StringIO()
        album_df.to_csv(album_buffer, index=False)
        album_content=album_buffer.getvalue()
        s3.put_object(Bucket=Bucket, Key=album_key, Body=album_content)
        artist_key = "transformed_data/artist_data/artist_transformed_"+ str(datetime.now())+
".csv"
        artist_buffer=StringIO()
        artist_df.to_csv(artist_buffer, index=False)
        artist_content=artist_buffer.getvalue()
        s3.put_object(Bucket=Bucket, Key=artist_key, Body=artist_content)
    s3_resource=boto3.resource('s3')
    for key in spotify_keys:
```

```
    copy_source={
        'Bucket' : Bucket,
        'Key' : key
    }
    s3_resource.meta.client.copy(copy_source, Bucket, 'raw_data/processed/' +
key.split("/")[-1])
    s3_resource.Object(Bucket, key).delete()
```

Code explanation:

1. 3 functions (artist, album and songs) to extract specific information from the Spotify API data
2. Lists and retrieves JSON files from the specified S3 bucket and prefix, storing the data in spotify_data and the file keys in spotify_keys.
3. For each JSON object, the 3 functions are called and results are stored in Pandas dataframes. Duplicate rows are removed and the date columns are converted to datetime format.
4. Export the transformed data to CSV files in the respective subfolders
5. Move the files in to_process to processed folder and delete files in to_process
6. Similarly, you can set the test event and deploy the function.

4.6 Add trigger in the extract_data function to automate data extraction

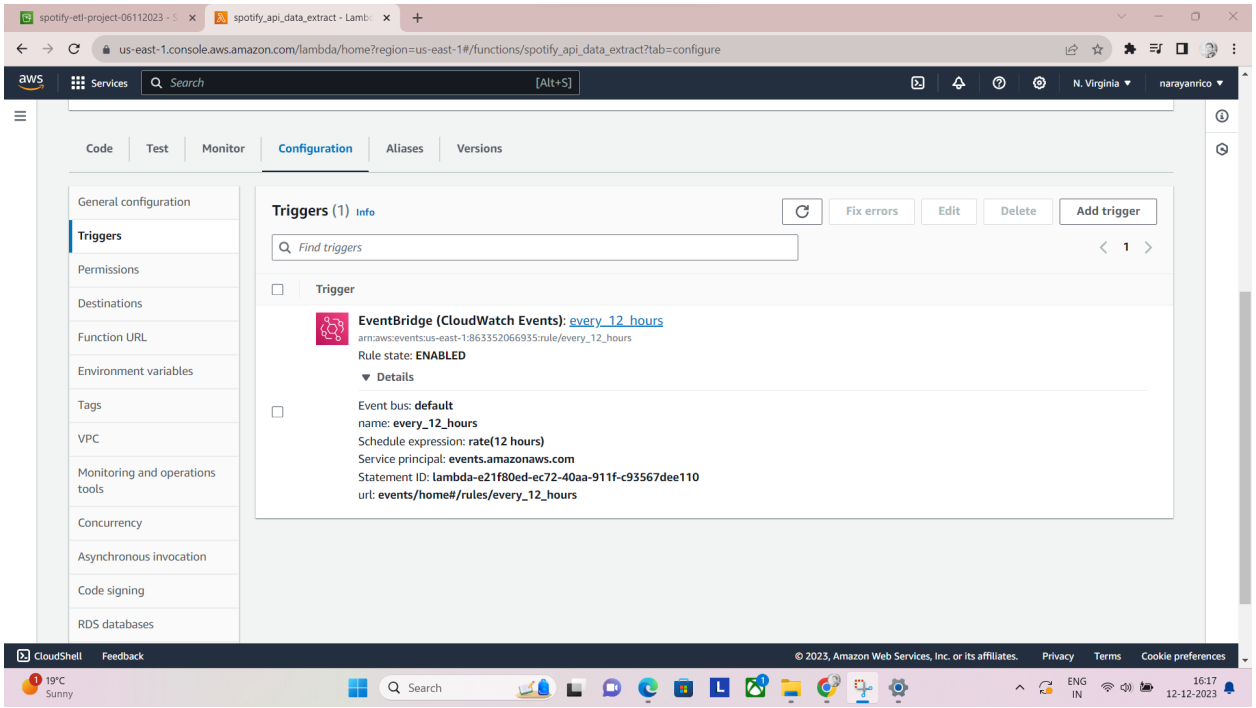


Figure 10: Extract Data Function Trigger

1. By configuring an EventBridge rule with a cron expression, it will invoke the function to extract data from the Spotify API based on the given schedule.
2. In the extract_data function, click on Add trigger.
3. Add trigger with EventBridge as the source
4. Set the source to EventBridge (CloudWatch Events). Create a new rule.
5. Add trigger to the function

4.7 Add trigger in the transform_spotify_data function to automate data transformation

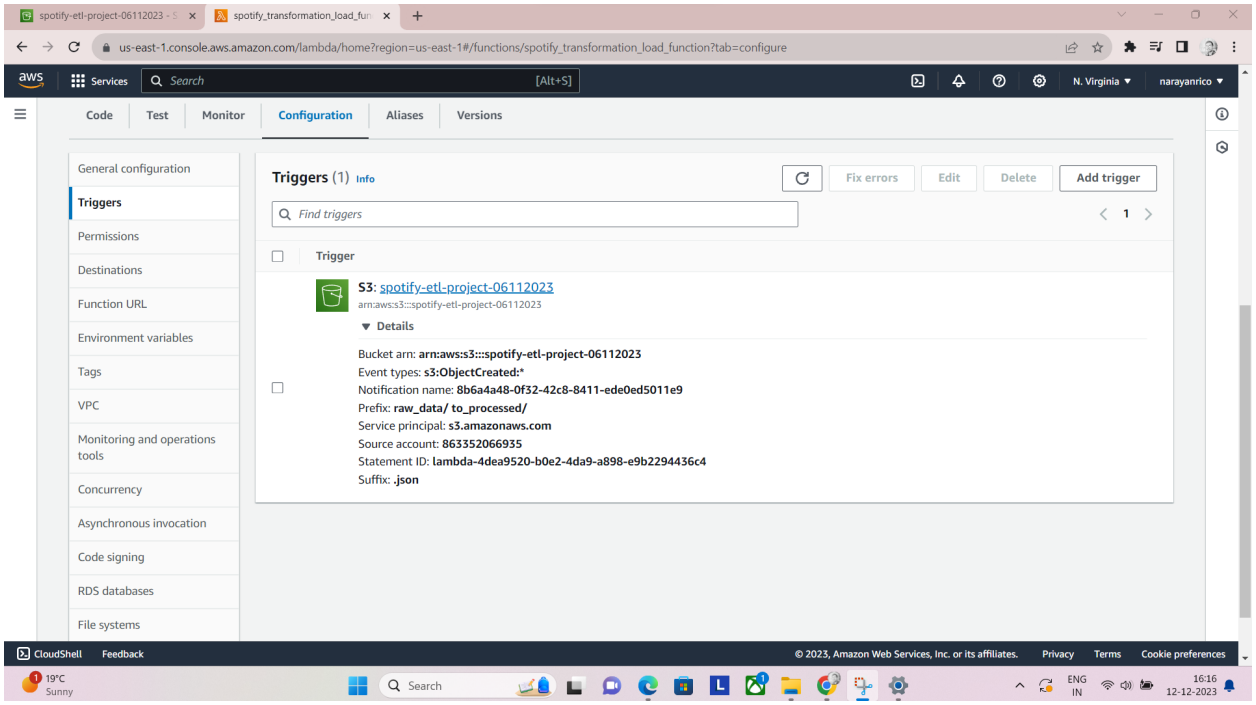


Figure 11: Transform Data Function Trigger

1. Create a trigger to invoke the transform_spotify_data function whenever new objects are added to the folder (spotify-etl-project-06112023/raw_data/to_process)
2. In the transform_spotify_data function, click on Add trigger.
3. Select the bucket.
4. Use “All object create events” as the event type.
5. Set the prefix to the folder where we want to monitor for new objects.
6. Set the suffix to .json to limit the trigger’s activation to specific types of objects within the bucket.
7. Create trigger with S3 as the source

4.8 Create a crawler

1. Create the 3 crawler: Spotify_album, Spotify_artist and Spotify_song.
2. Add S3 as the data source.
3. Configure security settings. By creating new IAM role.
4. Set output and scheduling.
5. Review and create crawler. Click on Create Crawler and Run Crawler

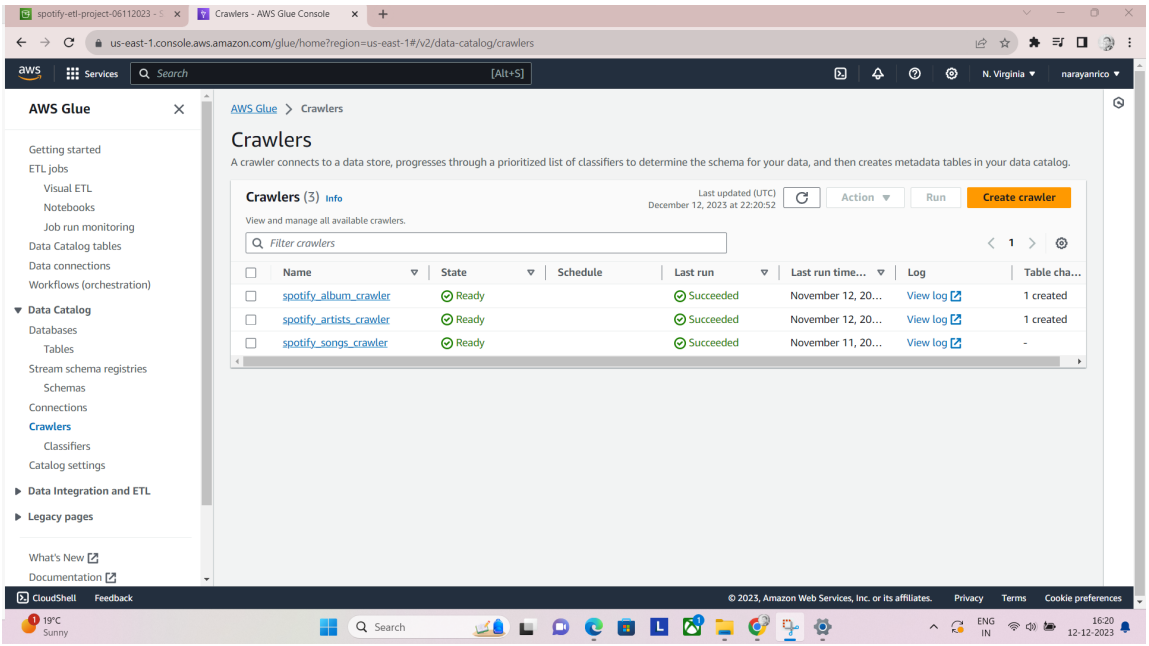


Figure 12: Crawlers

6. In AWS Glue > Databases, select spotify_db. You can see the respective tables here.

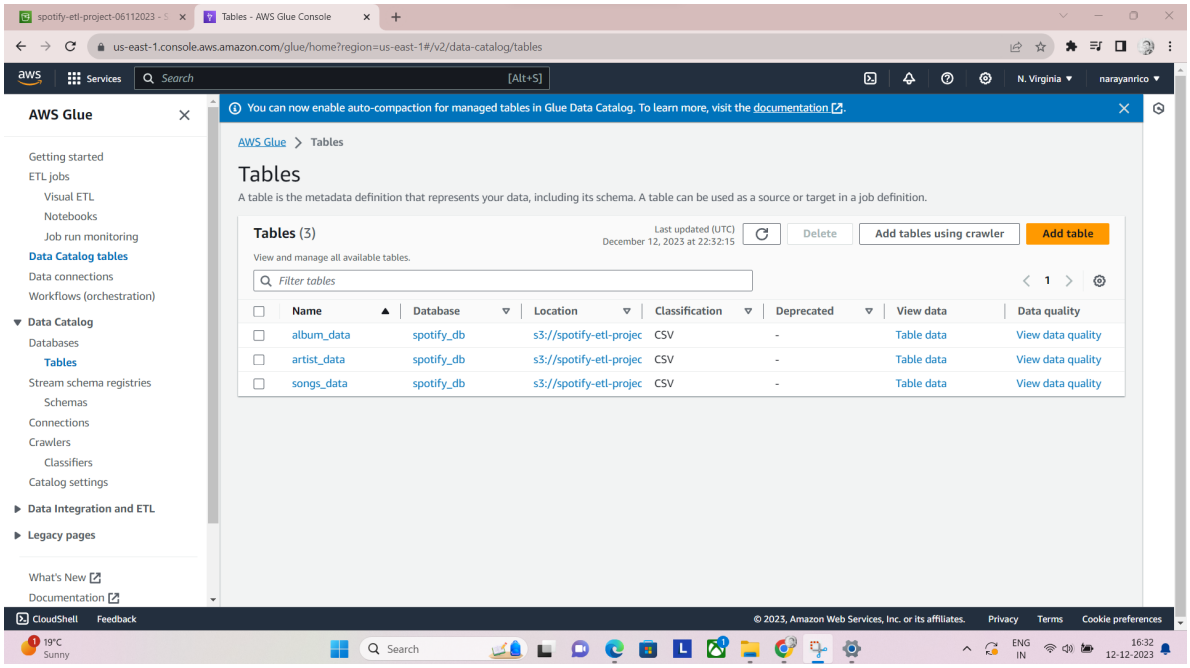


Figure 13: Database Tables

To edit schema, click on a table. You can click on edit the schema as JSON or edit schema to make any modifications.

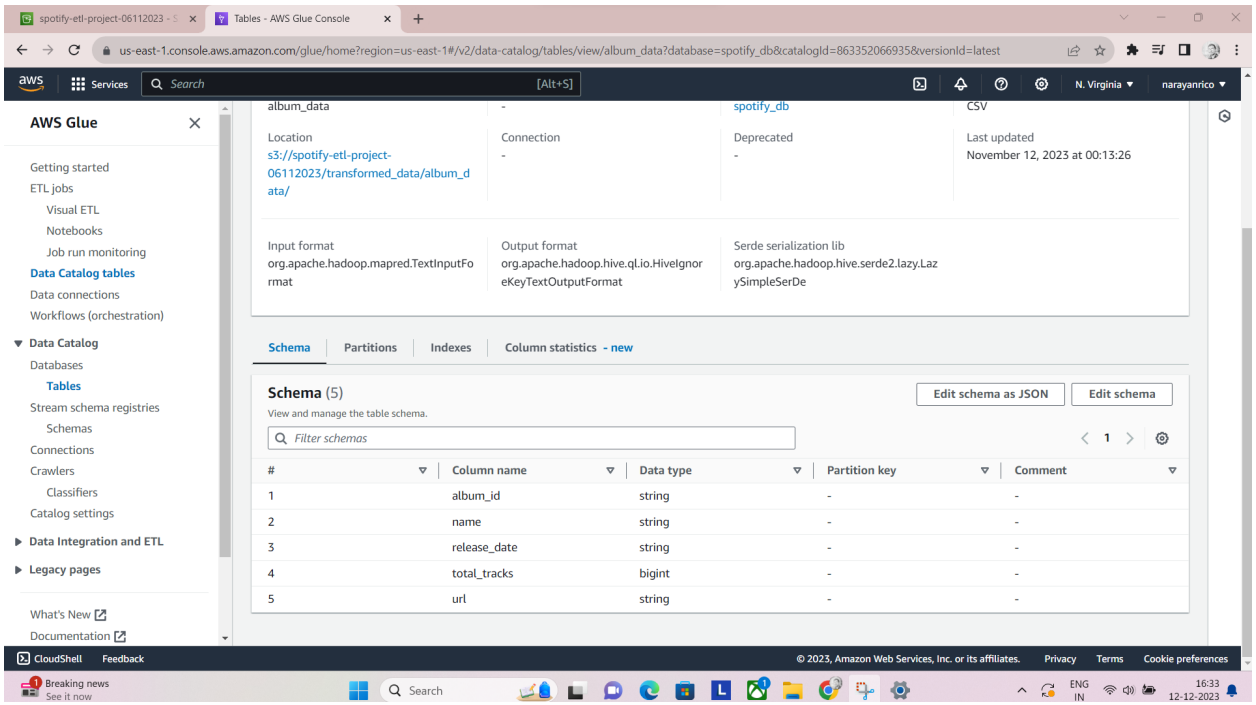


Figure 14: Schema

4.9 Querying with Athena

1. Open Athena.
2. Data source is AwsDataCatalog.
3. Select spotify_db as the database. You can view the tables along with their respective columns.
4. In the Query Editor, you might notice a message at the top. Before running your first query, it is necessary to configure the query result location. This step involves setting up the location where query results will be stored. This allows you to retrieve and analyze the results at a later time.
5. Click on Edit Settings. I created a new folder called queryresults.
6. Click on Browse S3 and select this path.
7. You can query the data now

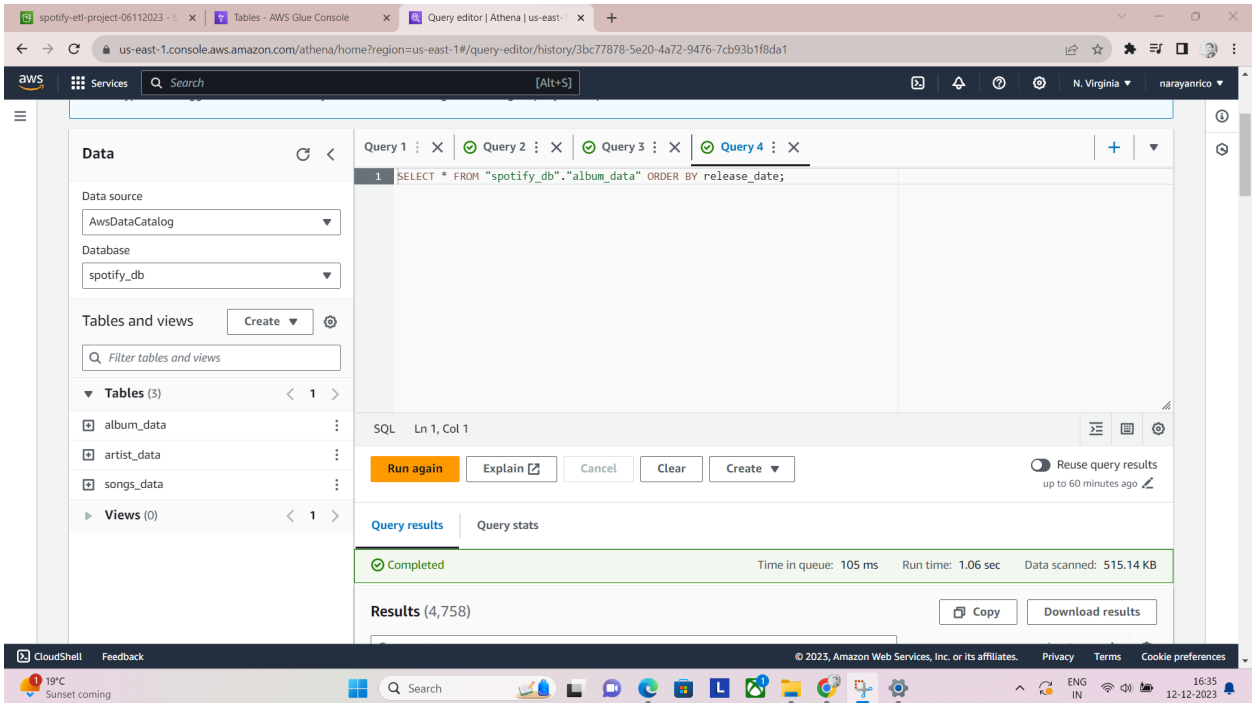


Figure 15: Data Query

5. Result

- We collected data from the spotify playlist “Top Songs - Global” for every 12 hours.
- In total we collected 65 objects through the playlist each of the average size of 216kb. So, in total we got the data of 140Mb.
- After transformation, we got “album_data” of 110 objects each of average size of 5kb. So, in total it would be 550kb.

- Similarly, “artist_data” consists of 110 objects each of average size 4.5kb. So, in total it would be 495kb and “ songs_data” of 110 objects each of average size 8.6kb and total size would be 946kb.

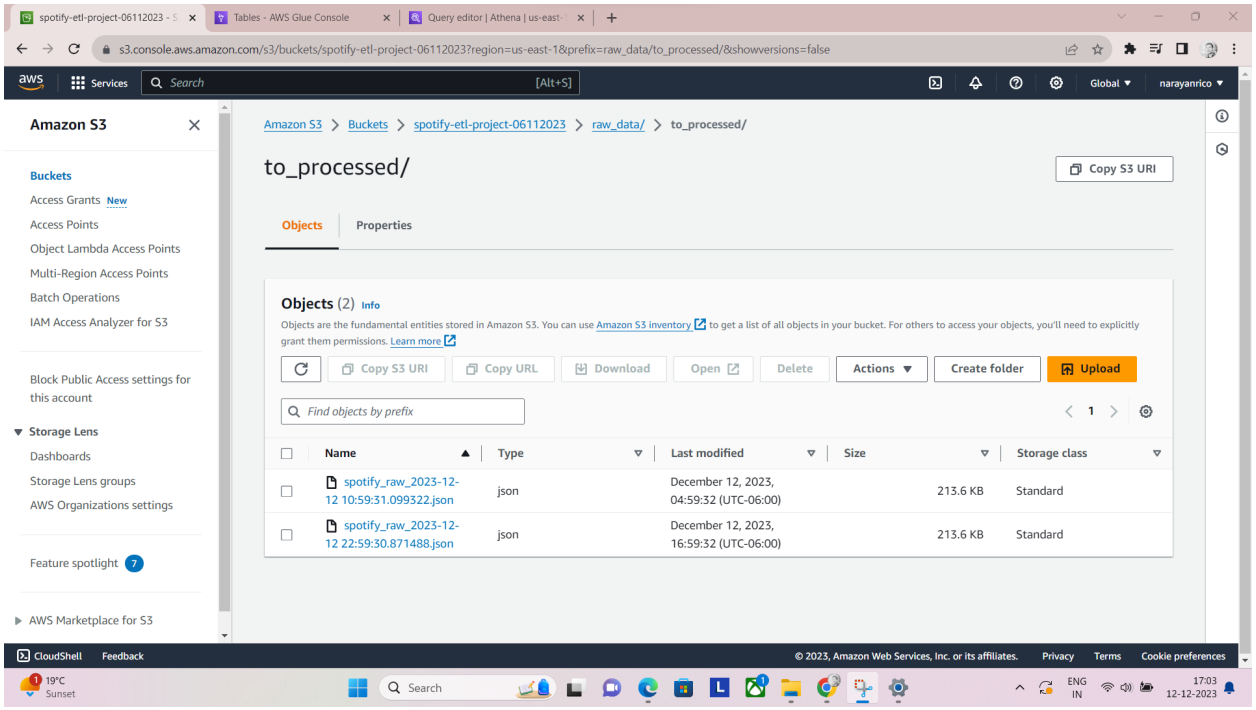


Figure 16: JSON files

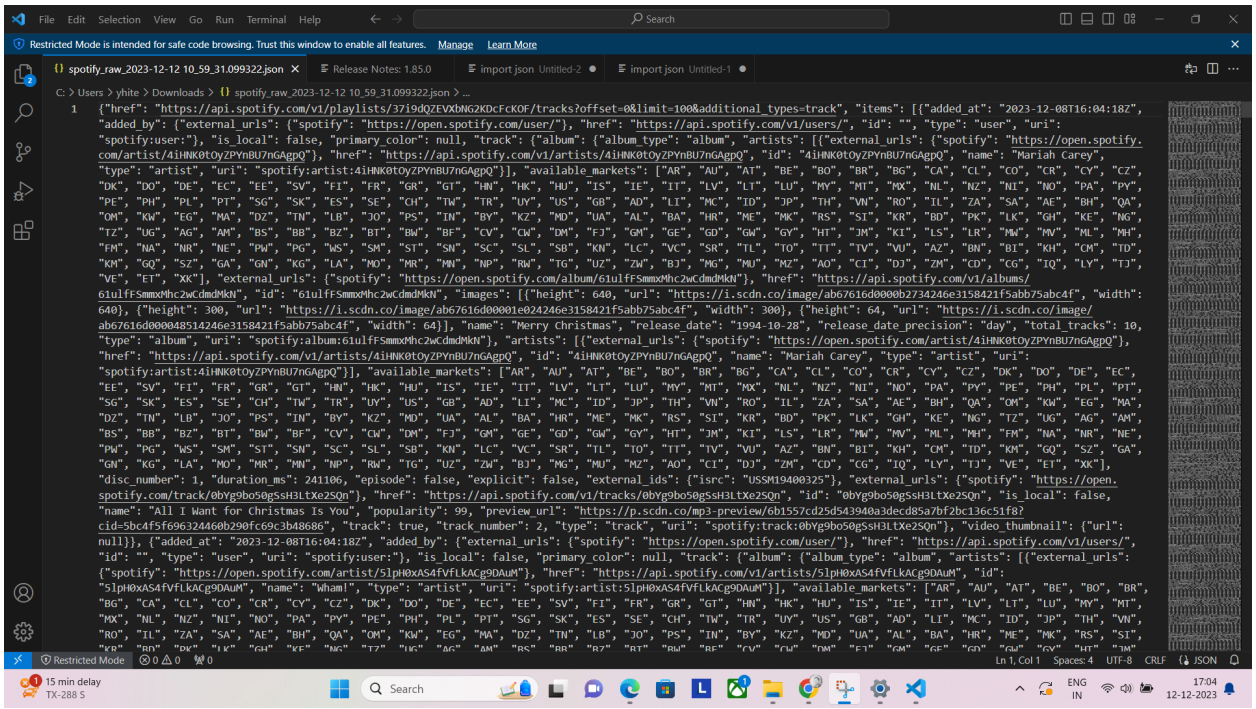
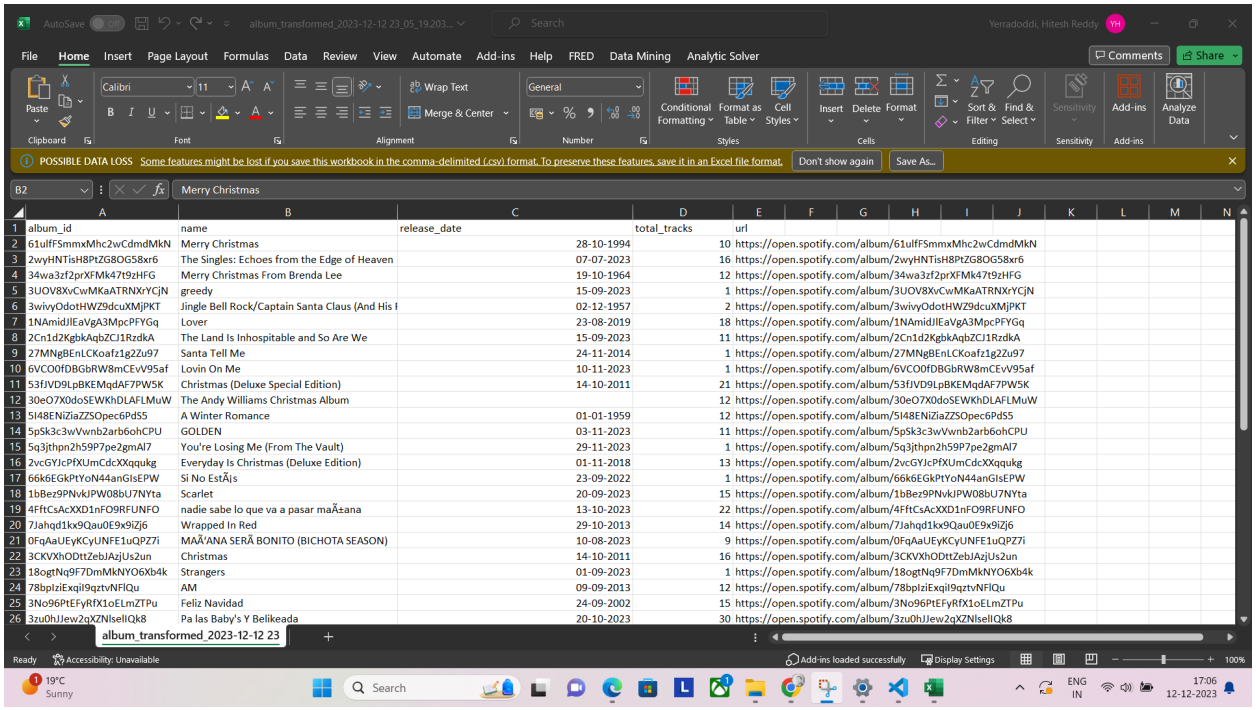
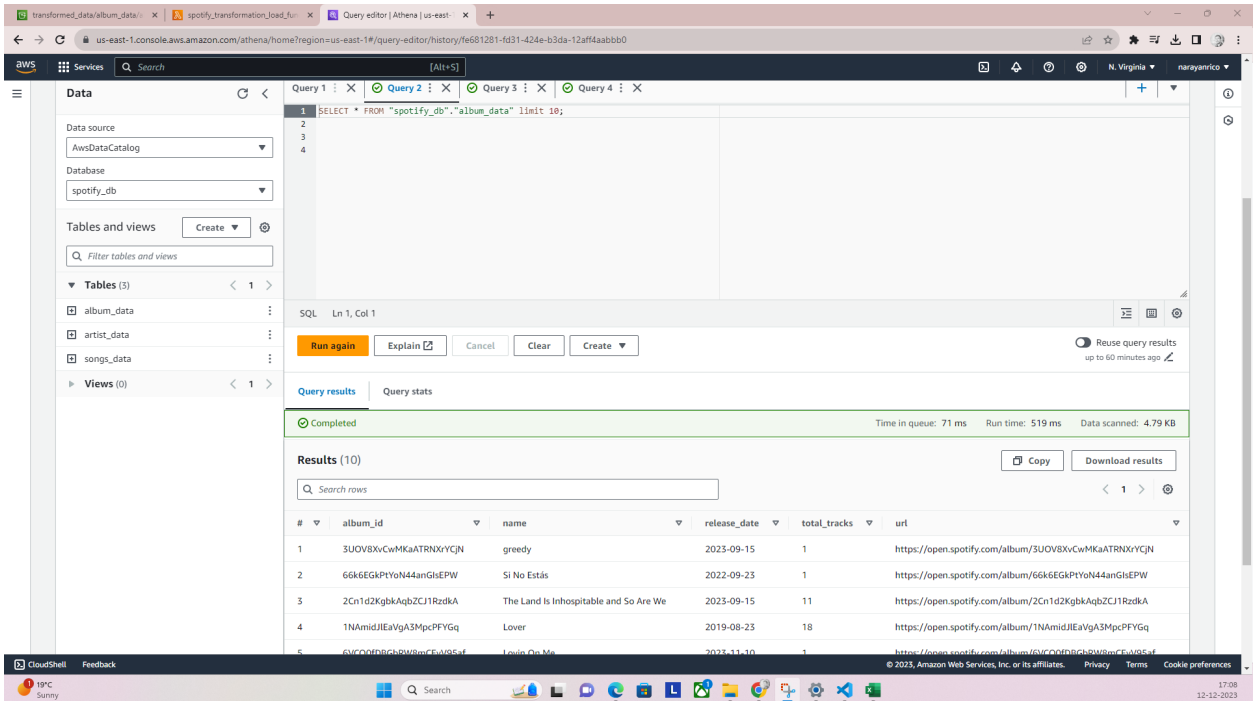


Figure 17: Raw data



album_id	name	release_date	total_tracks	url
61ulfFSmmxMhc2wCdmdMkN	Merry Christmas	28-10-1994	10	https://open.spotify.com/album/61ulfFSmmxMhc2wCdmdMkN
2wyHNTisH8PtZG8OG58xr6	The Singles: Echoes from the Edge of Heaven	07-07-2023	16	https://open.spotify.com/album/2wyHNTisH8PtZG8OG58xr6
34wa3zf2prXFMk47t9zHFG	Merry Christmas From Brenda Lee	19-10-1964	12	https://open.spotify.com/album/34wa3zf2prXFMk47t9zHFG
3UOV8XvCwMKaATRNXrYCjN	greedy	15-09-2023	1	https://open.spotify.com/album/3UOV8XvCwMKaATRNXrYCjN
3wivyOdotHWZ9dcuXMjPKT	Jingle Bell Rock/Captain Santa Claus (And His	02-12-1957	2	https://open.spotify.com/album/3wivyOdotHWZ9dcuXMjPKT
1NAmidJlEaVgA3MpcPFYGq	Lover	23-08-2019	18	https://open.spotify.com/album/1NAmidJlEaVgA3MpcPFYGq
2Cn1d2KgbkAqbZCJ1RzdkA	The Land Is Inhospitable and So Are We	15-09-2023	11	https://open.spotify.com/album/2Cn1d2KgbkAqbZCJ1RzdkA
27MNgBEnLCKoafz1g2Zu97	Santa Tell Me	24-11-2014	1	https://open.spotify.com/album/27MNgBEnLCKoafz1g2Zu97
6VCO0fDBGbRW8mCEvV95af	Lovin On Me	10-11-2023	1	https://open.spotify.com/album/6VCO0fDBGbRW8mCEvV95af
53fJVD9LpBKEMqdAF7PW5K	Christmas (Deluxe Special Edition)	14-10-2011	21	https://open.spotify.com/album/53fJVD9LpBKEMqdAF7PW5K
30eO7X0doSEWKhDLAFLMuW	The Andy Williams Christmas Album		12	https://open.spotify.com/album/30eO7X0doSEWKhDLAFLMuW
5I48ENiZiaZZSOpec6PdS5	A Winter Romance	01-01-1959	12	https://open.spotify.com/album/5I48ENiZiaZZSOpec6PdS5
5pSk3c3wVwnb2arb6ohCPU	GOLDEN	03-11-2023	11	https://open.spotify.com/album/5pSk3c3wVwnb2arb6ohCPU
5q3jthpn2h59P7pe2gmAI7	You're Losing Me (From The Vault)	29-11-2023	1	https://open.spotify.com/album/5q3jthpn2h59P7pe2gmAI7
2vcGYJcPfXUmCdcXXqqukg	Everyday Is Christmas (Deluxe Edition)	01-11-2018	13	https://open.spotify.com/album/2vcGYJcPfXUmCdcXXqqukg
66k6EGkPtYoN44anGlsEPW	Si No EstÃ¡s	23-09-2022	1	https://open.spotify.com/album/66k6EGkPtYoN44anGlsEPW
1bBez9PNvkJPW08bU7NYta	Scarlet	20-09-2023	15	https://open.spotify.com/album/1bBez9PNvkJPW08bU7NYta
4FftCsAcXXD1nFO9RFUNFO	nadie sabe lo que va a pasar maÃ±ana	13-10-2023	22	https://open.spotify.com/album/4FftCsAcXXD1nFO9RFUNFO
7Jahqd1kx9Qau0E9x9iZj6	Wrapped In Red	29-10-2013	14	https://open.spotify.com/album/7Jahqd1kx9Qau0E9x9iZj6
0FqAaUEyKCyUNFE1uQPZ7i	MAÃ'ANA SERÃ BONITO (BICHOTA SEASON)	10-08-2023	9	https://open.spotify.com/album/0FqAaUEyKCyUNFE1uQPZ7i
3CKVXhODttZebJAzjUs2un	Christmas	14-10-2011	16	https://open.spotify.com/album/3CKVXhODttZebJAzjUs2un
18ogtNq9F7DmMkNYO6Xb4k	Strangers	01-09-2023	1	https://open.spotify.com/album/18ogtNq9F7DmMkNYO6Xb4k
78bpIziExqil9qztvNFlQu	AM	09-09-2013	12	https://open.spotify.com/album/78bpIziExqil9qztvNFlQu
3No96PtEFyRfX1oELmZTPu	Feliz Navidad	24-09-2002	15	https://open.spotify.com/album/3No96PtEFyRfX1oELmZTPu
3zu0hJJew2qXZNlseIlQk8	Pa las Baby's Y Belikeada	20-10-2023	30	https://open.spotify.com/album/3zu0hJJew2qXZNlseIlQk8

Figure 18: Transformed data



#	album_id	name	release_date	total_tracks	url
1	3UOV8XvCwMKaATRNXrYCjN	greedy	2023-09-15	1	https://open.spotify.com/album/3UOV8XvCwMKaATRNXrYCjN
2	66k6EGkPtYoN44anGlsEPW	Si No Estás	2022-09-23	1	https://open.spotify.com/album/66k6EGkPtYoN44anGlsEPW
3	2Cn1d2KgbkAqbZCJ1RzdkA	The Land Is Inhospitable and So Are We	2023-09-15	11	https://open.spotify.com/album/2Cn1d2KgbkAqbZCJ1RzdkA
4	1NAmidJlEaVgA3MpcPFYGq	Lover	2019-08-23	18	https://open.spotify.com/album/1NAmidJlEaVgA3MpcPFYGq

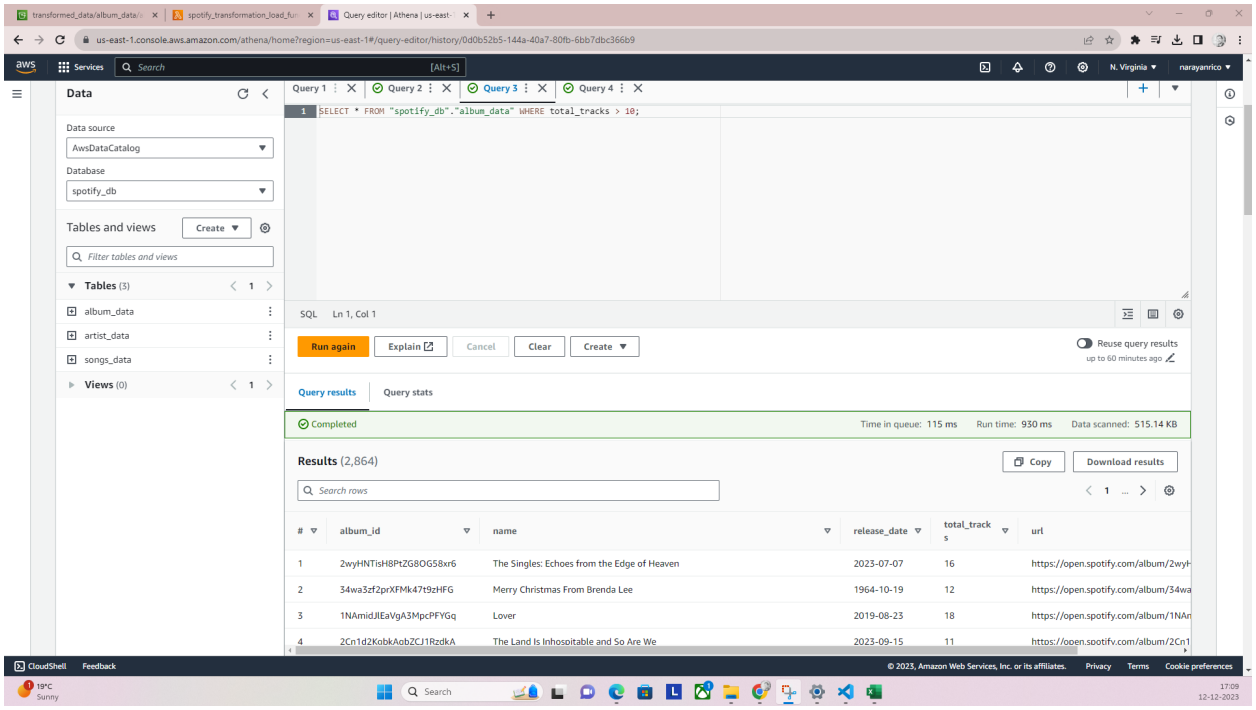


Figure 19: Athena analysis

6. Milestones

Over the course of this project, we progressed through several key milestones centered on developing an automated ETL pipeline to extract, transform and load Spotify playlist data for analytics. The first milestone focused on designing the architecture, choosing the appropriate AWS services and Python libraries. This was followed by developing the Extract and Load modules to pull raw data from the Spotify API and manage loading it into cloud storage. With the raw data pipeline established, We shifted to transforming the extracted data by cleaning and processing it into analysis-ready datasets structured around songs, albums and artists. The next milestone targeted establishing automated workflows for regular data refresh through triggers and handling metadata management. After confirming the full workflow, we focused on enabling interactive SQL-based analytics via AWS Glue catalog and Athena queries. Additional milestones included comprehensive testing, debugging, documentation and gathering community feedback to improve the pipeline. Through the progression of milestones, we were able to build an end-to-end solution demonstrating core skills like Python programming, cloud services orchestration, data processing, metadata management and database querying within a practical data engineering environment using industry-standard tools.

7. Conclusion

The completed project involved designing, developing and deploying a cloud-based ETL pipeline to ingest, transform and load personalized Spotify playlist data for analytical purposes. Through the project execution spanning architecture design, software development, testing and maintenance, the automated pipeline was proven to effectively extract weekly raw data, handle scalable processing in the cloud, and make the cleaned datasets easily available for SQL analysis.

Numerous skills were demonstrated including architecting systems leveraging cloud platform services, Python programming for ETL workflows, configuration of fault-tolerant data pipelines and enabling BI analytics. Real-world exposure was provided into the intricacies when moving data from diverse sources into final repositories in a reusable way. Additional soft skills around documenting complex architectures, liaising with cloud platform technical teams and aligning to project milestones also played a key role.

The solution achieves the core objective of maintaining always up-to-date Spotify listener data that is prepared for querying to unlock unique user insights. Traveling through the project journey necessitated methodically navigating challenges stemming from opaque APIs, unstructured data and complexity of integrating multiple processes. Learnings around tightly focusing on deliverables and reaching small milestones were invaluable for making gradual progress.

With the fundamental automated pipeline established, enhancements can readily be incorporated through supplementary datasets, analytical dashboards, recommendation engines and more. The project forms a robust foundation for further data engineering explorations using real-world repositories.

8 Future Scope And Atudy

This ETL pipeline can be enhanced and built upon in several ways:

- **Enhanced Data Modeling:** The data model can be expanded by incorporating additional attributes from the Spotify API such as audio features, metrics, and cultural metadata. This would enable more diverse analysis.
- **Visual Analytics Dashboard:** Interactive visual dashboards can be developed in Amazon QuickSight or other BI tools by utilizing the transformed datasets. This would provide richer insights into music preferences, listening trends etc.

- **Monitoring and Logging:** Robust monitoring, alarms, notifications can be implemented in Amazon CloudWatch around key metrics like data volumes, job failures. Detailed logging of pipeline activity can also help pinpoint issues.
- **Workflow Orchestration:** Workflow managers like Apache Airflow or Prefect can be evaluated to orchestrate the pipeline stages. These could simplify scheduling, reduce dependencies, and provide visibility.
- **Containerization:** Dockerizing the Python ETL scripts would ease portability by eliminating environment inconsistencies across execution contexts. Containers represent a modern standard for shipping code.
- **Version Control Integration:** Migrating to a Git-based development workflow where code is version controlled and deployed through CI/CD pipelines brings benefits like change tracking, collaboration and code quality.

The processed Spotify dataset presents opportunities for diverse analytics use cases - ranging from understanding listening preferences to building music recommendation systems using machine learning. Expanding the capabilities of the pipeline represents the logical next step.

9. References

[1] Gordon, Rachel. "ETL Best Practices: Measure Twice, Load Once". Matillion Blog. 2022.
[2] Krishnan, Karthik. "Building an ETL Pipeline with AWS Glue and Quicksight". Medium. 2021.
[3] "Processing Spotify Datasets using AWS Glue and Amazon Athena". AWS Samples GitHub. 2020.
[4] "Orchestrating Spotify ETL pipelines with Apache Airflow". Astronomer.io Blog. 2022.
[5] Reis, Joe and Housley, Matt. "Fundamentals of Data Engineering". Databooks.ai. 2022.
[6] https://docs.aws.amazon.com/
[7] https://developer.spotify.com/documentation/web-api
[8] https://aws.amazon.com/what-is/etl/
[9] https://ieeexplore.ieee.org/document/9441984