

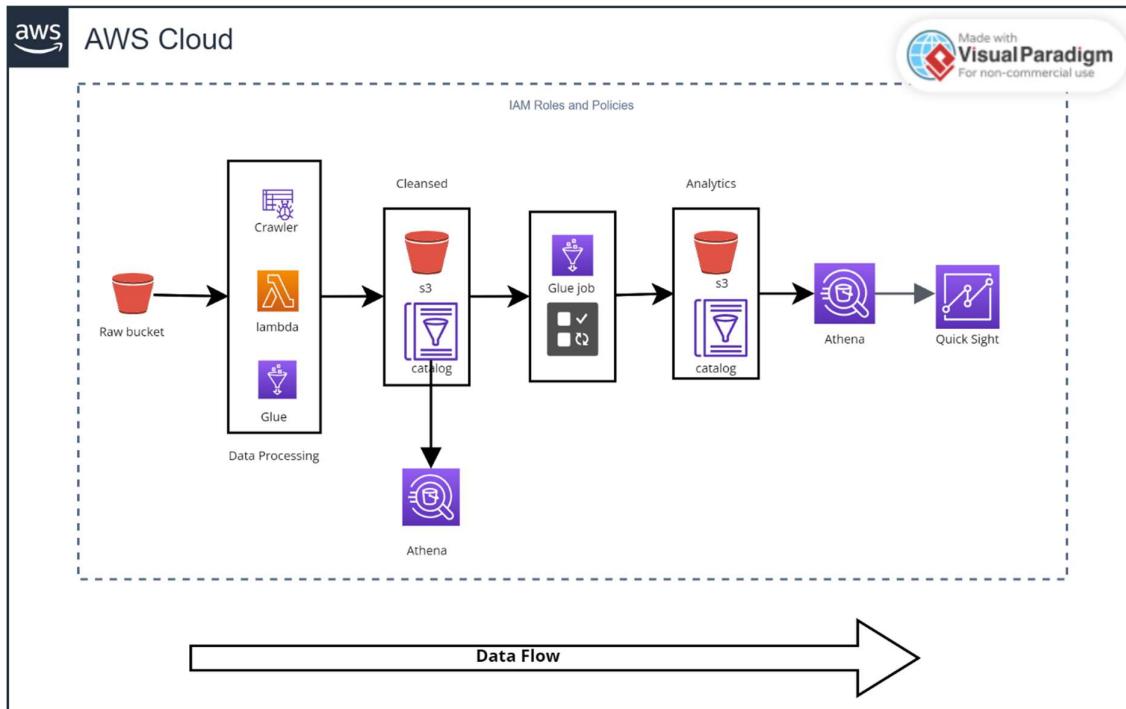
# Data Analysis and Visualisation on Trending YouTube Data using AWS Glue and Quick Sight

**Objective :** To perform Data Analysis and Visualisation on the trending youtube data using various AWS Services.

**Services used :** AWS Glue, AWS Lambda, Amazon S3, Quick Sight, Amazon Athena, IAM.

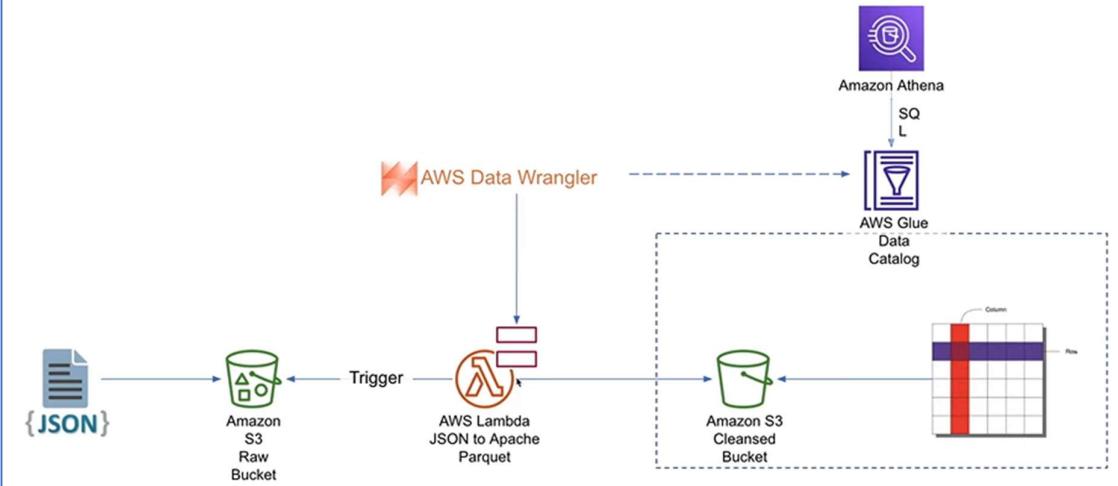
**Dataset :** The data is about 211MB, in which it consists of two types of files one is json files and other is csv files. There are total 6 files in which 3 are json files and 3 are csv files. This is a real world data and it consists of 3 regions data. The json files consists of metadata whereas the csv files consists of the actual data.

## Architecture

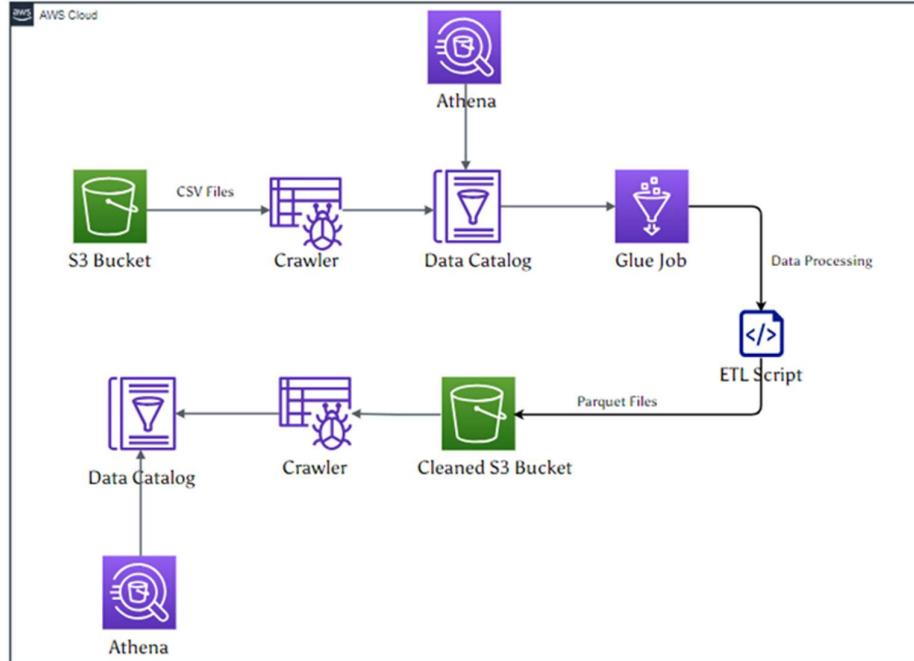


## Data Cleansing

### Semi-structured data to Structured pipeline



JSON Data Processing Architecture



CSV Data Processing Architecture

# IMPLEMENTATION

1. Download the trending Youtube data from Kaggle

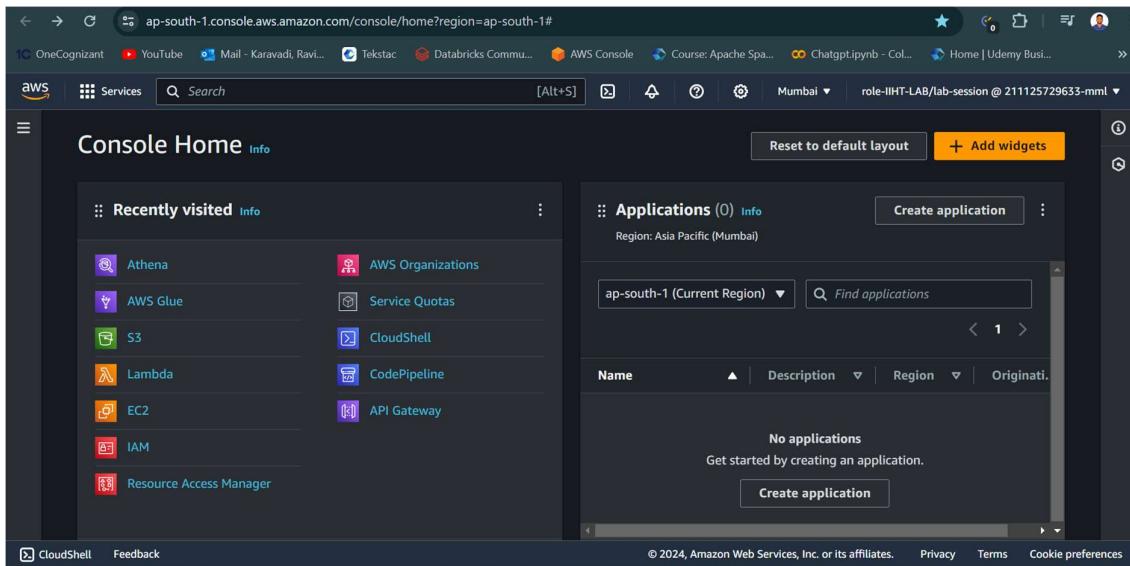
The screenshot shows the Kaggle interface. On the left, there's a sidebar with options like 'Create', 'Home', 'Competitions', 'Datasets' (which is selected), 'Models', 'Code', 'Discussions', 'Learn', and 'More'. The main content area is titled 'Trending YouTube Video Statistics' with a subtitle 'Daily statistics for trending YouTube videos'. It features a large 'YouTube' logo. Below the title, there are links for 'Data Card', 'Code (3080)', 'Discussion (44)', and 'Suggestions (1)'. A section titled 'About Dataset' includes a 'Usability' rating of 7.94 and a 'License' link. At the bottom, there's a note about source code and a 'License' link.

2. Now, extract the downloaded file, It contains the data files of json and csv

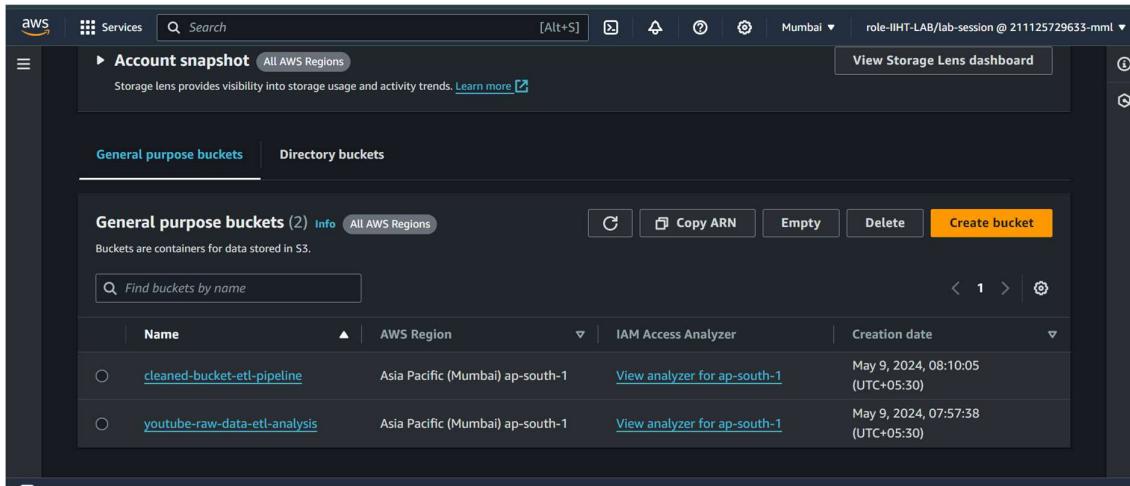
The screenshot shows a file explorer interface with a dark theme. On the left, there's a sidebar with 'Home', 'Gallery', and links to 'Desktop', 'Downloads', 'Documents', 'Pictures', 'Music', and 'Videos'. The main area displays a table of files:

Name	Date modified	Type	Size
US_category_id	10-05-2024 12:21	JSON Source File	9 KB
GB_category_id	10-05-2024 12:21	JSON Source File	9 KB
CA_category_id	10-05-2024 12:21	JSON Source File	8 KB
USvideos	10-05-2024 12:21	Microsoft Excel Com...	61,286 KB
GBvideos	10-05-2024 12:21	Microsoft Excel Com...	51,967 KB
CAvideos	10-05-2024 12:21	Microsoft Excel Com...	62,567 KB

3. Login to your AWS account



4. Navigate to S3 service, create an S3 bucket for storing the data and created another bucket for storing the cleaned data.



5. Now create two folders inside the S3 bucket, one for json data and one for csv files

youtube-raw-data-etl-analysis [Info](#)

Objects Properties Permissions Metrics Management Access Points

Objects (2) [Info](#)

C 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

Name	Type	Last modified	Size	Storage class
<a href="#">csv_dat/</a>	Folder	-	-	-
<a href="#">json_dat/</a>	Folder	-	-	-

6. Navigate inside the json\_dat folder and uploaded all json files present in the downloaded file

Amazon S3 > Buckets > youtube-data-storage-raw-1729 > json\_files/

json\_files/ [Copy S3 URI](#)

Objects Properties

Objects (3) [Info](#)

C 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

Name	Type	Last modified	Size	Storage class
<a href="#">CA_category_id.json</a>	json	May 9, 2024, 15:46:25 (UTC+05:30)	7.7 KB	Standard
<a href="#">GB_category_id.json</a>	json	May 9, 2024, 15:46:25 (UTC+05:30)	8.0 KB	Standard
<a href="#">US_category_id.json</a>	json	May 9, 2024, 15:46:25 (UTC+05:30)	8.3 KB	Standard

7. Now navigate to csv\_dat folder and create folder for every region present in the csv\_file name, and then upload the csv file specific to region, and given the folder name as region=<region\_name>

csv\_files/

[Copy S3 URI](#)

[Objects](#) [Properties](#)

**Objects (3) [Info](#)**

[C](#) [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

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">region=ca/</a>	Folder	-	-	-
<input type="checkbox"/>	<a href="#">region=gb/</a>	Folder	-	-	-
<input type="checkbox"/>	<a href="#">region=us/</a>	Folder	-	-	-

region=ca/

[Copy S3 URI](#)

[Objects](#) [Properties](#)

**Objects (1) [Info](#)**

[C](#) [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

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">CAvideos.csv</a>	csv	May 9, 2024, 15:52:01 (UTC+05:30)	61.1 MB	Standard

## 8. Create a Crawler to process JSON data

- Crawler is glue service used to crawl data from different sources(s3) and stores the data in the glue catalog which further used for the analytical purpose.
- Give a name to the Crawler : yt-json-crawler
- Provide the Data Source location of json files for the crawler : s3location
- Create IAM role for the Crawler with necessary policies(S3FullAccess, GlueService) and assign the role to the Crawler

IAM > Roles > yt-glue-role

**yt-glue-role** [Info](#)

Allows Glue to call AWS services on your behalf.

**Summary**

Creation date: May 09, 2024, 22:44 (UTC+05:30)

Last activity: 12 hours ago

ARN: arn:aws:iam::637423175804:role/yt-glue-role

Maximum session duration: 1 hour

**Permissions** [Edit](#)

**Permissions policies (2)** [Info](#)

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	2
AWSGlueServiceRole	AWS managed	2

- Create a Database for the Crawler to create Table and store the data, In short the Data Catalog.

AWS Glue > Databases > yt-raw-db

**yt-raw-db**

Last updated (UTC): May 10, 2024 at 07:17:40

**Database properties**

Name	Description	Location	Created on (UTC)
yt-raw-db	-	-	May 10, 2024 at 07:17:31

**Tables (0)**

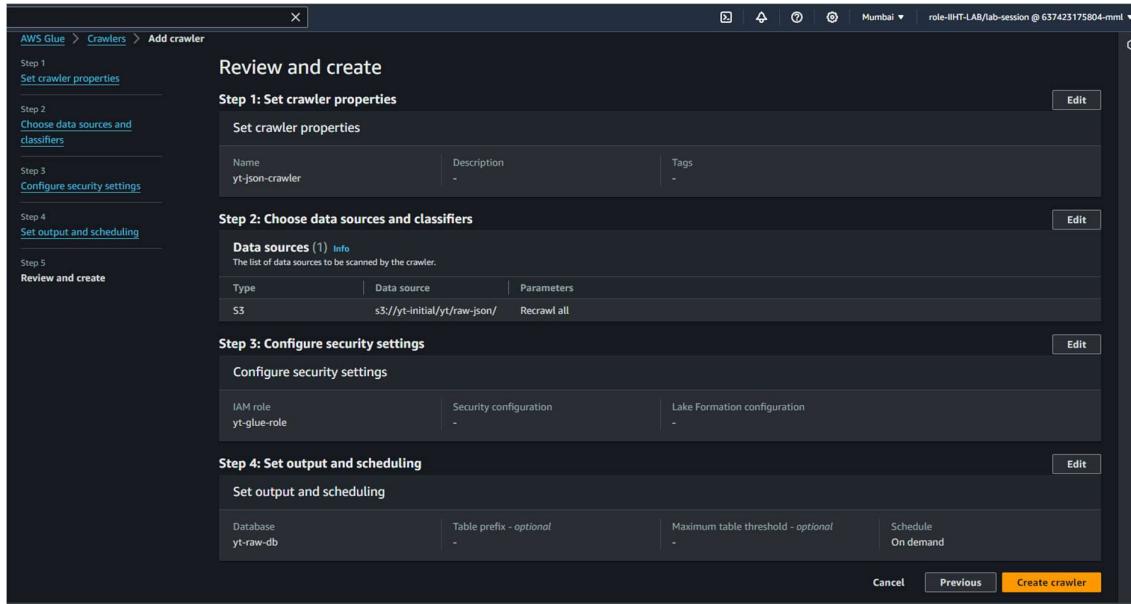
Last updated (UTC): May 10, 2024 at 07:17:42

[Add tables using crawler](#) [Add table](#)

View and manage all available tables.

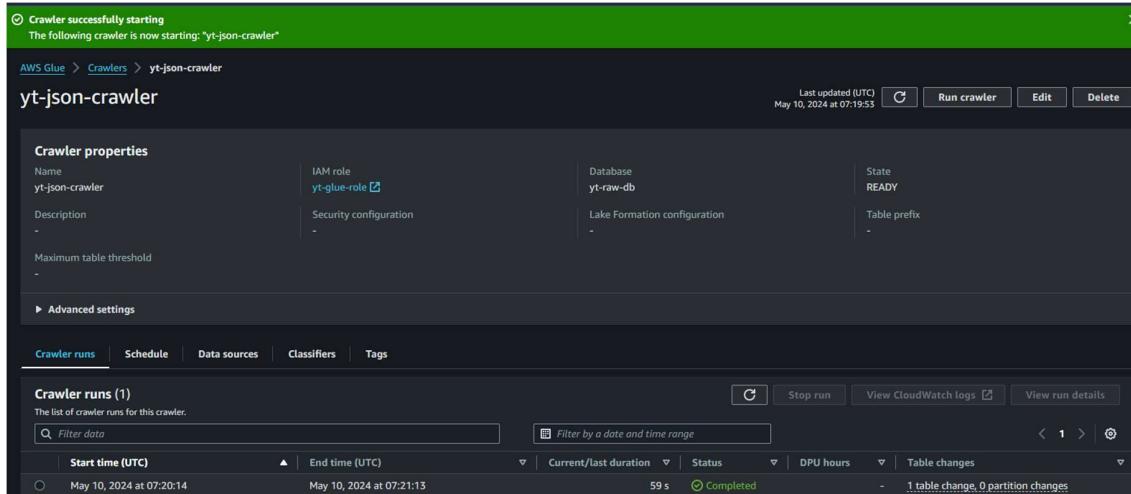
Name	Database	Location	Classification	Deletion Policy	View data	Data quality
------	----------	----------	----------------	-----------------	-----------	--------------

- Review and Create the Crawler



## 9. Run the Crawler

- Run the Crawler and wait for the data catalog to be formed i.e. the Table with the crawled data.
- There create a table named raw\_json



## 10. Review the Table and Perform the Query analysis.

- The Crawler will create a table with the json data in the specified database location.

The screenshot shows the AWS Glue Table details for 'raw\_json'. It includes sections for Table overview, Data quality, Table details, Advanced properties, Schema, Partitions, Indexes, and Column statistics. The Schema section displays the following table:

#	Column name	Data type	Partition key	Comment
1	kind	string	-	-
2	etag	string	-	-
3	items	array	-	-

- View the data in the table through the Amazon Athena and Run the Query for the retrieval of the data.
- Here comes an error message (JSON Exception) stating that improper json format, so that unable to fetch the data while Querying.
- To Overcome the above Error. We create a Lambda for extraction of specified data.

The screenshot shows the AWS Athena console with a failed query. The query is:

```
1 SELECT * FROM "AwsDataCatalog"."yt-raw-db"."raw_json" limit 10;
```

The results pane shows the following error message:

**HIVE\_CURSOR\_ERROR: Row is not a valid JSON Object - JSONException: A JSONObject text must end with ')' at 2 [character 3 line 1]**  
 This query ran against the "" database, unless qualified by the query. Please post the error message on our [forum](#) or contact [customer support](#) with Query Id: 5983dd4d-9164-4b78-bbe9-cec527a9042c

## 11. Lambda Function Creation

- We create the lambda function for the extraction of only desired data from the json file that convert the json to parquet format which can be readability for the glue and Lambda.
- Give a name to Lambda Function: yt-json-par-lambda.
- Select the runtime Engine: Python 3.8
- Create and attach the IAM Role for the Lambda with necessary permissions and Policies (S3FullAccess, GlueService)

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use when writing your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
 ▼

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
 x86\_64  
 arm64

**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.

**Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
 Create a new role with basic Lambda permissions  
 Use an existing role  
 Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
 ▼

[View the yt-lambda-role role](#)

- Create a new Database(yt-cleaned-db) for the storage of the cleaned json table for analytical purposes.

AWS Glue > Databases > yt-cleaned-db

### yt-cleaned-db

Last updated (UTC)  
May 10, 2024 at 07:34:30  Edit Delete

**Database properties**

Name	Description	Location	Created on (UTC)
yt-cleaned-db	-	-	May 10, 2024 at 07:34:25

**Tables (0)**  
View and manage all available tables.

Name	Database	Location	Classification	Deprecated	View data	Data quality
No available tables						

Last updated (UTC)  
May 10, 2024 at 07:34:32  Delete Add tables using crawler Add table

- Create a New S3 Bucket for the storage of the all cleaned data. here in this case all cleaned data parquets of both csv and json file will be stored

Amazon S3 > Buckets > yt-clean

### yt-clean [Info](#)

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

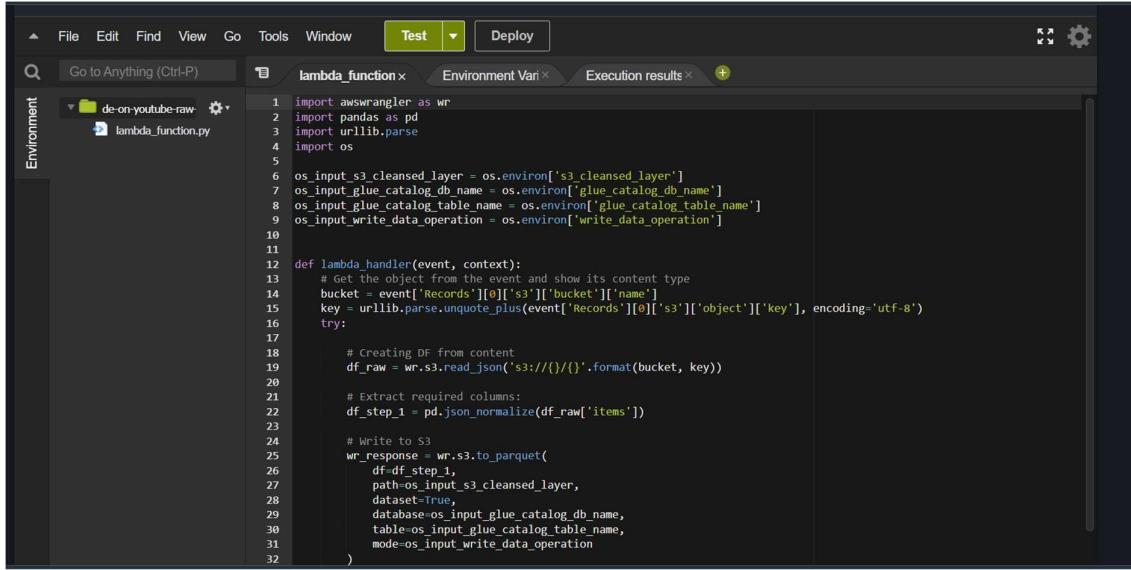
**Objects (0)**  Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

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](#)

◀ 1 ▶ ⌂

Name	Type	Last modified	Size	Storage class
No objects You don't have any objects in this bucket.				

- Customise the lambda function, so that it extracts only needed data from the json file and the store it in a new file format Parquet and stores these parquets in new bucket and Also a table is created in a database.

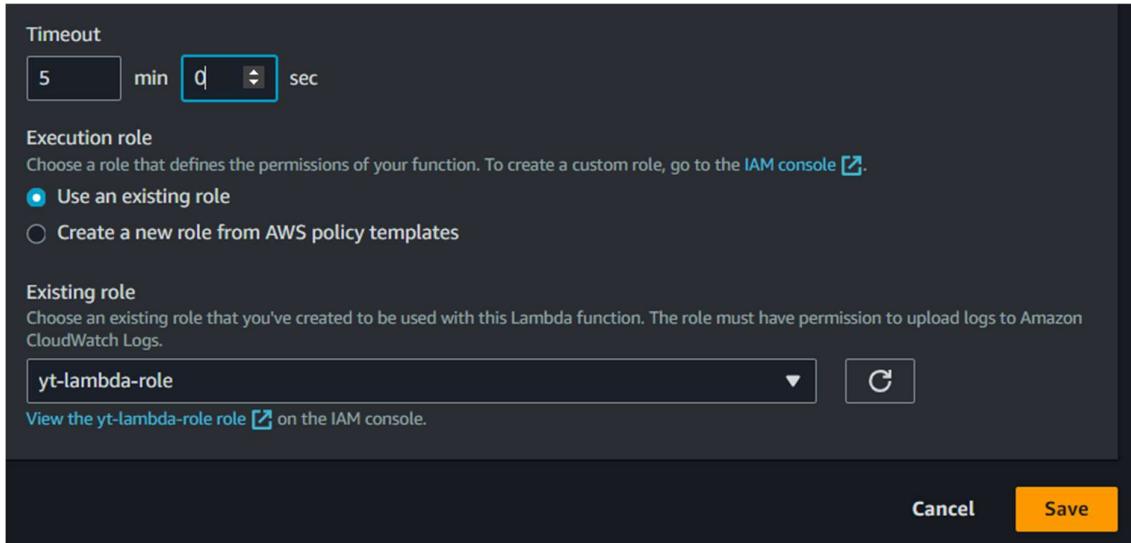


```

1 import awswrangler as wr
2 import pandas as pd
3 import urllib.parse
4 import os
5
6 os_input_s3_cleansed_layer = os.environ['s3_cleansed_layer']
7 os_input_glue_catalog_db_name = os.environ['glue_catalog_db_name']
8 os_input_glue_catalog_table_name = os.environ['glue_catalog_table_name']
9 os_input_write_data_operation = os.environ['write_data_operation']
10
11
12 def lambda_handler(event, context):
13     # Get the object from the event and show its content type
14     bucket = event['Records'][0]['s3']['bucket']['name']
15     key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
16     try:
17
18         # Creating DF from content
19         df_raw = wr.s3.read_json('s3://{}{}'.format(bucket, key))
20
21         # Extract required columns:
22         df_step_1 = pd.json_normalize(df_raw['items'])
23
24         # Write to S3
25         wr_response = wr.s3.to_parquet(
26             df=df_step_1,
27             path=os_input_s3_cleansed_layer,
28             dataset=True,
29             database=os_input_glue_catalog_db_name,
30             table=os_input_glue_catalog_table_name,
31             mode=os_input_write_data_operation
32         )

```

- Configure the default timeout time and set it to 5 mins in the General Configuration tab



**Timeout**  
5 min 0 sec

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role  
 Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

yt-lambda-role [View the yt-lambda-role role](#) on the IAM console.

**Cancel** **Save**

- Configure the Environment variables such S3 target location, Database location, Table name, Write operation the lambda.

**Environment variables**

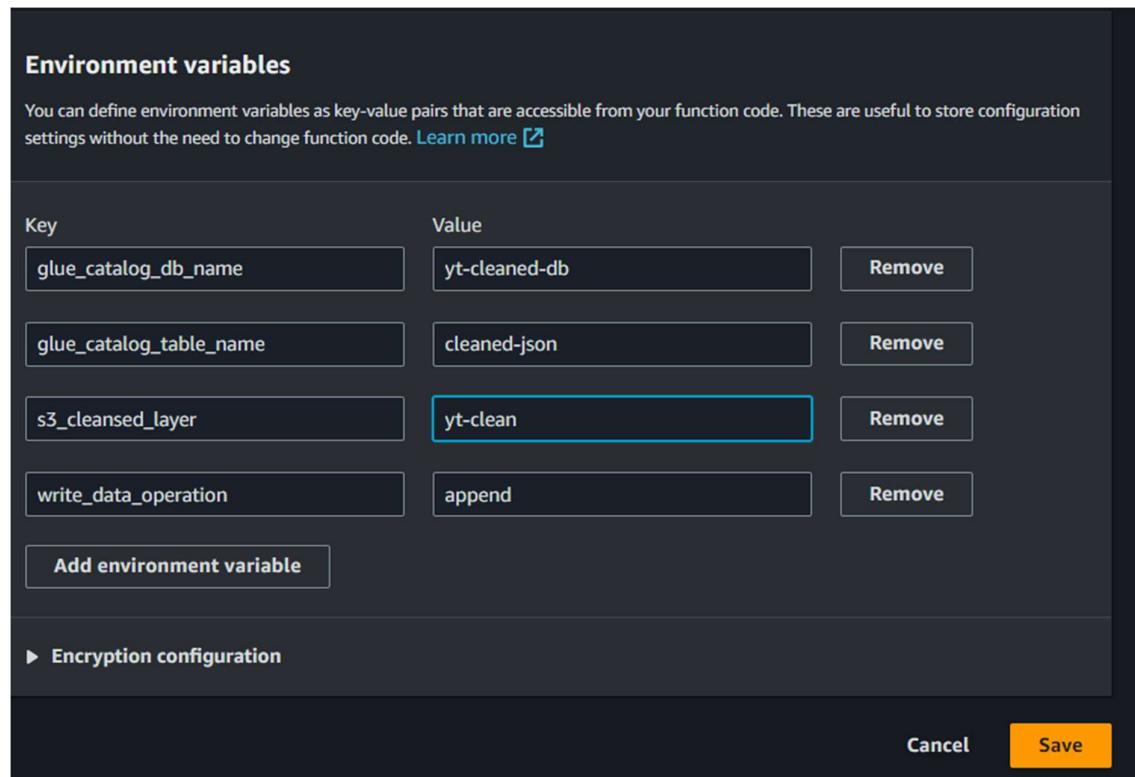
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
glue_catalog_db_name	yt-cleaned-db	<a href="#">Remove</a>
glue_catalog_table_name	cleaned-json	<a href="#">Remove</a>
s3_cleansed_layer	yt-clean	<a href="#">Remove</a>
write_data_operation	append	<a href="#">Remove</a>

[Add environment variable](#)

▶ [Encryption configuration](#)

[Cancel](#) [Save](#)



- Add AWSDataWrangler Layer which helps to provide compute environment for the lambda executions uninterrupted.

**aws**

Home > Applications > Application details

**aws-data-wrangler-layer-py3-8**  
arn:aws:serverlessrepo:us-east-1:336392948345:applications/aws-data-wrangler-layer-py3-8

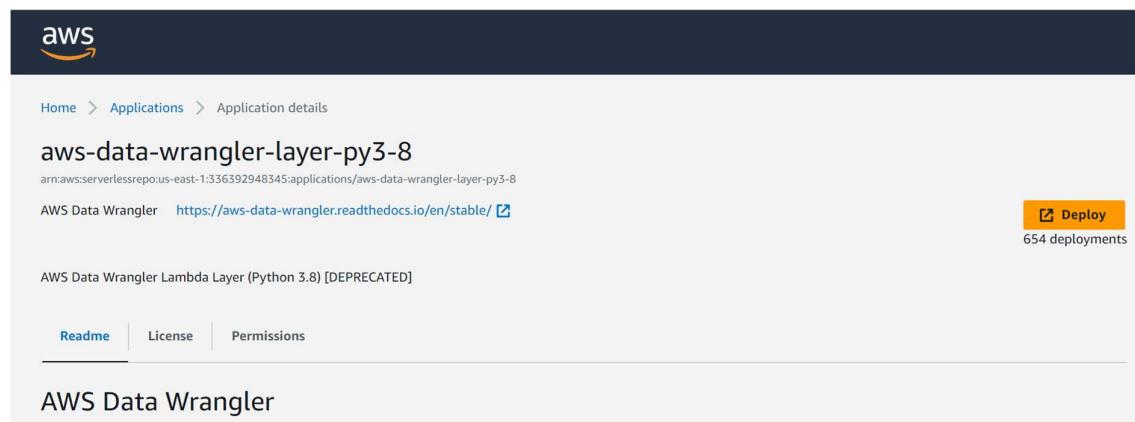
AWS Data Wrangler <https://aws-data-wrangler.readthedocs.io/en/stable/>

[Deploy](#) 654 deployments

AWS Data Wrangler Lambda Layer (Python 3.8) [DEPRECATED]

[Readme](#) | [License](#) | [Permissions](#)

**AWS Data Wrangler**



## Choose a layer

### Layer source [Info](#)

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also [create a new layer](#).

AWS layers

Choose a layer from a list of layers provided by AWS.

Custom layers

Choose a layer from a list of layers created by your AWS account or organization.

Specify an ARN

Specify a layer by providing the ARN.

### Custom layers

Layers created by your AWS account or organization that are compatible with your function's runtime.

AWSDataWrangler-Python38



### Version

1



[Cancel](#)

[Add](#)

- Create a Test event for Deploying and Testing the code that we have written, in the script of the event provide some details like (template : S3PUT) and source bucket name and the key of the files stored in the bucket

THIS EVENT IS AVAILABLE TO IAM USERS WITHIN THE SAME ACCOUNT WHO HAVE PERMISSIONS TO ACCESS AND USE SHAREABLE EVENTS. [LEARN MORE](#)

**Template - optional**

s3-put

**Event JSON**

```

4   "eventversion": "2.0",
5   "eventSource": "aws:s3",
6   "awsRegion": "us-east-1",
7   "eventTime": "1970-01-01T00:00:00.000Z",
8   "eventName": "ObjectCreated:Put",
9   "userIdentity": {
10     "principalId": "EXAMPLE"
11   },
12   "requestParameters": {
13     "sourceIPAddress": "127.0.0.1"
14   },
15   "responseElements": {
16     "x-amz-request-id": "EXAMPLE123456789",
17     "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmabaisawesome/mnopqrstuvwxyzABCDEFGH"
18   },
19   "s3": {
20     "s3SchemaVersion": "1.0",
21     "configurationId": "testConfigRule",
22     "bucket": {
23       "name": "yt-initial",
24       "ownerIdentity": {
25         "principalId": "EXAMPLE"
26       },
27       "arn": "arn:aws:s3:::yt-initial"
28     },
29     "object": {
30       "key": "|t/raw-json/CA_category_id.json",
31       "size": 1024,
32       "eTag": "0123456789abcdef0123456789abcdef",
33       "sequencer": "0A1B2C3D4E5F678901"
34     }
35   }

```

**Format JSON**

**Cancel** **Invoke** **Save**

- Test the Lambda Function and Check for the Results
- It should create the parquet files in the bucket and the Tables in the database.

Successfully updated the function yt-json-par-lambda.

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

**Code source** **Info**

**Test** **Deploy**

**Execution results**

Status: Succeeded | Max memory used: 128 MB | Time: 10684.75 ms

Execution result

```

{
  "paths": [
    "s3://yt-clean/13545b218d1648382d8f8d0c84cf280.snappy.parquet"
  ],
  "partitions_values": {}
}

```

**Function Logs**

```

START RequestId: 9c3b402a-5a9b-4a62-bc8c-b0fe993890ca Version: $LATEST
REPORT RequestId: 9c3b402a-5a9b-4a62-bc8c-b0fe993890ca Duration: 10664.75 ms Billed Duration: 10665 ms Memory Size: 128 MB Max Memory Used: 128 MB Init Duration: 3793.30 ms
RequestID
9c3b402a-5a9b-4a62-bc8c-b0fe993890ca

```

- Check for the Confirmation of Parquet file generation.

The screenshot shows the Amazon S3 console interface. The path is 'Amazon S3 > Buckets > yt-clean'. The 'Objects' tab is selected, showing one object: '15545b218d16483882d8f8d0c8\_4df280.snappy.parquet'. The file is a Parquet file, 6.1 KB in size, last modified on May 10, 2024, at 14:35:56 (UTC+05:30).

- Check whether the table is created or not in the cleaned database.

The screenshot shows the AWS Glue Table Overview for the 'json\_cleaned' table. The table details are as follows:

Name	Description	Database	Classification
json_cleaned	-	db_cleaned	Parquet
Location	Connection	Deprecated	Last updated
s3://yt-cleansed	-	-	May 10, 2024 at 09:06:24
Input format	Output format	Serde serialization lib	
org.apache.hadoop.hive.ql.io.parquet.MappedParquetInputFormat	org.apache.hadoop.hive.ql.io.parquet.MappedParquetOutputFormat	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	

- View the table data through the Athena Query Editor and perform a simply query and check for the proper execution.
- Here, It proves that the Error we encountered earlier was rectified and can proceed with this new data.

The screenshot shows the Athena Query Editor interface. The query being run is:

```
1 SELECT * FROM "AwsDataCatalog"."yt-cleaned-db"."cleaned_json" limit 10;
```

The results pane shows the output of the query. The status bar at the bottom indicates 'SQL Ln 1, Col 1'.

Query results
Query stats

Completed
Time in queue: 63 ms
Run time: 328 ms
Data scanned: 2.11 KB

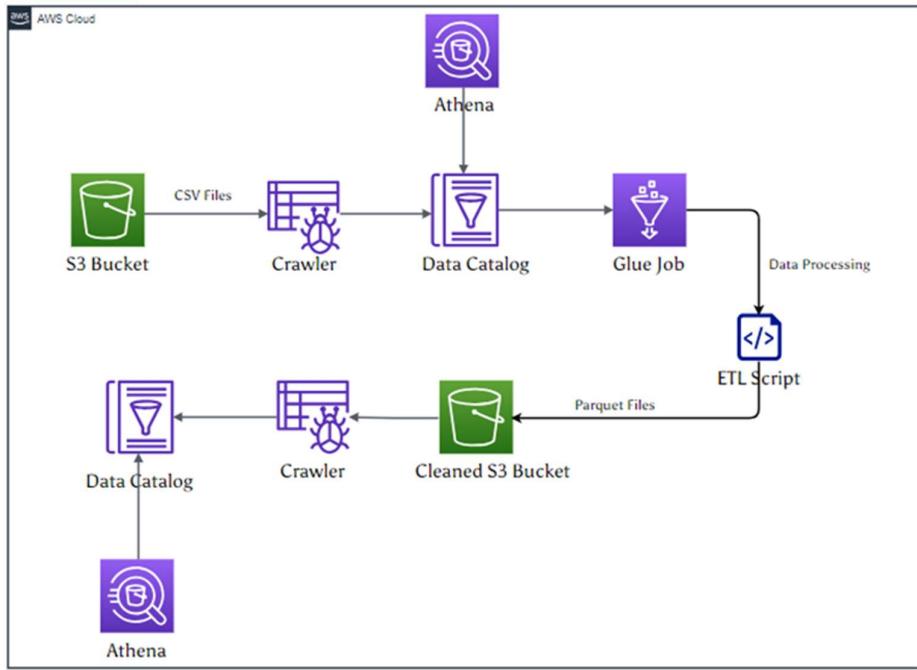
Results (10)
Copy
Download results

Q

Search rows

<
1
>
↻

#	kind	etag		id	snippet_channelId		snippet_title
1	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/Xy1mB4_yLrHy_BmKmPBggty2mZQ"		1	UCBR8-60-B28hp2BmDPdntcQ		Film & Animation
2	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/UZ1oLlz2dxlhO45ZTFR3a3NyTA"		2	UCBR8-60-B28hp2BmDPdntcQ		Autos & Vehicles
3	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/nqRiq97-xe5XRZTxbkxFV5Lmg"		10	UCBR8-60-B28hp2BmDPdntcQ		Music
4	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/HwXKamM1Q20q9BN-oBJavSGkfDI"		15	UCBR8-60-B28hp2BmDPdntcQ		Pets & Animals
5	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/9GQMSRjrZdHeb1OEM1XVQ9zbGec"		17	UCBR8-60-B28hp2BmDPdntcQ		Sports
6	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/FJwVpGCV1yJrqZbpqe68Sy_OE"		18	UCBR8-60-B28hp2BmDPdntcQ		Short Movies
7	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/M-3ID9dwK7YJCafRf_DkLN8couA"		19	UCBR8-60-B28hp2BmDPdntcQ		Travel & Events
8	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/WmA0qYFjWsAoyJFSw2zinhr2wM"		20	UCBR8-60-B28hp2BmDPdntcQ		Gaming
9	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/EapFaGYG7K0StIXvf8aba249tdM"		21	UCBR8-60-B28hp2BmDPdntcQ		Videoblogging
10	youtube#videoCategory	"ld9biNPkAjgjV7EZ4EKeEGrhao/xld8RX7vRN8rqkbYZbNlytUQDRo"		22	UCBR8-60-B28hp2BmDPdntcQ		People & Blogs



## 12. Create a Crawler to process csv Files.

- Configure the crawler with the details like, Set the crawler properties, Data source locations, Security Settings, Output location and scheduling.

**Step 1: Set crawler properties**

Name	Description	Tags
yt-csv-crawler	-	-

**Step 2: Choose data sources and classifiers**

Type	Data source	Parameters
S3	s3://yt-initial/yt/raw-csv/	Recrawl all

**Step 3: Configure security settings**

IAM role yt-glue-role	Security configuration -	Lake Formation configuration -
--------------------------	-----------------------------	-----------------------------------

**Step 4: Set output and scheduling**

Database yt-raw-db	Table prefix - optional -	Maximum table threshold - optional -	Schedule On demand
-----------------------	------------------------------	---	-----------------------

**Review and create**

Cancel Previous Create crawler

## 13. Run the Crawler

- Same as previous, It creates data catalog table.

The screenshot shows the 'Crawler runs' section of the AWS Glue console. It displays a single crawler run named 'raw\_csv'. The run was completed successfully in 1 minute and 48 seconds, with 1 table change and 3 partition changes. The status is 'Completed'.

- Here it creates a partitioned table with region as partition key.

The screenshot shows the 'Table details' section for the 'raw\_csv' table. It provides metadata such as Name: raw\_csv, Description: -, Database: yt-raw-db, Classification: CSV, Location: s3://yt-initial/yt/raw-csv/, Input format: org.apache.hadoop.mapred.TextInputFormat, Output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, Serde serialization lib: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, and Last updated: May 10, 2024 at 09:32:48.

The 'Schema' tab is selected, showing 17 columns: video\_id, trending\_date, title, channel\_title, category\_id, publish\_time, and tags. The schema table has the following structure:

#	Column name	Data type	Partition key	Comment
1	video_id	string	-	-
2	trending_date	string	-	-
3	title	string	-	-
4	channel_title	string	-	-
5	category_id	bigint	-	-
6	publish_time	string	-	-
7	tags	string	-	-

#### 14. View the Table data and Perform some Query operations through Athena.

- Try joining the two table(json, csv) on a joining condition and query the data, But here we need TypeCast datatype of column in joining condition, inorder to overcome the mismatch error.

The screenshot shows the AWS Athena Query Editor. The sidebar on the left lists the data source as 'AwsDataCatalog' and the database as 'yt-cleaned-db'. Under 'Tables and views', there is one table named 'cleaned\_json'. The main area shows a query editor with the following SQL code:

```

1 SELECT * FROM "AwsDataCatalog"."yt-raw-db"."raw_csv" as r
2 join "yt-cleaned-db"."cleaned_json" as c
3 on r.category_id = cast(c.id as int)
4 limit 10
    
```

The status bar at the bottom indicates 'SQL Ln 3, Col 36'.

#	video_id	trending_date	title	channel_title	count
1	Kf2-86o5S1o	18.05.03	"Film Theory: The Bee Movie LIED To You!"	"The Film Theorists"	1
2	Juad74hE6rs	18.05.03	"Crazy Frosting Recipe: The Best Buttercream Frosting with Endless Flavor Variations!"	"Gemma Stafford"	2
3	MgMLdq9DnNc	18.05.03	"February Favorites 2018"	"Jenn Im"	2
4	xwXP7vB4mlY	18.05.03	"She Ruined The Surprise Gender Reveal"	"KKandbabyJ"	2
5	9baIXJoNdy0	18.05.03	"One thing that makes you a better friend"	"Anna Akana"	2
6	n9JVbbRBqfY	18.05.03	"LEGENDARY All You Can Eat Buffet in Manila Philippines - Spiral Buffet Review"	"Strictly Dumpling"	1
7	n_S8d_1KVhU	18.05.03	"How Satellites Capture 400 Megapixel Images Of Earth's Globe - Himawari 8 & GOES-16"	"Scott Manley"	2

- But, Typecasting the data is not a good practice, So we change the schema of the table Instead.

**Schema (1/6)**  
View and manage the table schema.

#	Column name	Data type	Partition key	Comment
1	kind	string	-	-
2	etag	string	-	-
3	id	bigint	-	-
4	snippet_channelid	string	-	-
5	snippet_title	string	-	-
6	snippet_assigna...	boolean	-	-

Cancel      Save as new table version

- Now after changing the schema and try to query data, Even then we encounter an Error message (Hive Data Error)
- This means changing schema on local table itself does not reflect in the parquet file that is generated previously.

```

Query 2 : X | Query 3 : X | Query 4 : X | Query 5 : X | Query 6 : X | Query 7 : X | + | ▾

1 SELECT * FROM "AwsDataCatalog"."yt-raw-db"."raw_csv" as r
2 join "yt-cleaned-db"."cleaned_json" as c
3 on r.category_id = c.id
4 limit 10

```

**Query results**    **Query stats**

**Failed**    Time in queue: 60 ms    Run time: 905 ms    Data scanned: -

**HIVE\_BAD\_DATA:** Field id's type BINARY in parquet file s3://yt-clean/yt/555cd0d1cd264696850f9bca1cd8af14.snappy.parquet is incompatible with type bigint defined in table schema

This query ran against the "yt-cleaned-db" database, unless qualified by the query. Please post the error message on our [forum](#) or contact [customer support](#) with Query Id: 30844e07-5a7d-4f85-8cad-8164cc18c2ee

- To overcome this, we need to delete the previously generated parquet file and rerun the lambda to generate a new parquet with all changes in the schema reflected in it.

Successfully updated the function yt-json-par-lambda.

Code source Info

File Edit Find View Go Tools Window Test Configuration Aliases Versions

Code source Info

Environment Environment

lambda\_function

Execution result

Status: Succeeded Max memory used: 128 MB Time: 10684.75 ms

Response

```
{
  "paths": [
    "s3://yt-clean/13545b218d16483882d8f8d0c84df280.snappy.parquet"
  ],
  "partitions_values": {}
}
```

Function Logs

```
START RequestId: 8c3bd02a-6acb-4a62-bc8c-b0fe993800ca Version: $LATEST
REPORT RequestId: 9c3bd02a-6acb-4a62-bc8c-b0fe993800ca Duration: 10664.75 ms Billed Duration: 10665 ms Memory Size: 128 MB Max Memory Used: 128 MB Init Duration: 3703.30 ms
RequestId: 9c3bd02a-6acb-4a62-bc8c-b0fe993800ca
```

- Now, Run the Query again and we see that there are no errors while running it.

Data

Data source: AwsDataCatalog

Database: db\_cleaned

Tables and views: json\_cleaned

Tables (1)

json\_cleaned

kind	string
etag	string
id	bigint
snippet_channelid	string
snippet_title	string
snippet_assignable	boolean

Query 17 : | Query 18 : | Query 20 : | **Query 21 :**

```
1 SELECT * FROM "db_cleaned"."csv_raw" a
  2 inner join "db_cleaned"."json_cleaned" b
  3 on a.category_id=b.id
  4 limit 10;
```

Run again Explain Cancel Clear Create

Completed

Time in queue: 55 ms    Run time: 1.216 sec    Data scanned: 10.06 MB

## 15. Create a Lambda Trigger so as to generate all parquets for json file

- Initially we generated a single parquet file for single json file manually, to ensure the process that we follow is correct.
- So, we need generate all parquet files for all json data. In order to do this we need to add a trigger to the lambda function,

- The purpose of Adding trigger to lambda is to automate the generation of the parquets for all json files
- so in order to trigger the lambda we delete all the uploaded json files and previously generated single parquet file, and reupload the files, now it triggers and generate the parquets.
- Make sure provide all the details in the trigger details.

**Trigger configuration** [Info](#)

S3 aws asynchronous storage ▾

**Bucket**  
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

[X](#) [C](#)

Bucket region: ap-south-1

**Event types**  
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

▾

[All object create events](#) [X](#)

**Prefix - optional**  
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

**Suffix - optional**  
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

**Recursive invocation**  
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#) ↗

I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

- deleting json files

Successfully deleted objects  
View details below.

**Delete objects: status**

The information below will no longer be available after you navigate away from this page.

Summary		
Source <a href="s3://yt-clean/yt/">s3://yt-clean/yt/</a>	Successfully deleted 1 object, 6.1 KB	Failed to delete 0 objects

- Reupload the json files

The screenshot shows the AWS S3 console with a green success message banner at the top: "Upload succeeded" with a link to "View details below.". Below the banner, there is a table titled "Files and folders (6 Total, 171.7 MB)" showing the following data:

Name	Folder	Type	Size	Status	Error
CA_categor...	yt/raw-json/	application/...	7.7 KB	✓ Succeeded	-
GB_categor...	yt/raw-json/	application/...	8.0 KB	✓ Succeeded	-
US_categor...	yt/raw-json/	application/...	8.3 KB	✓ Succeeded	-

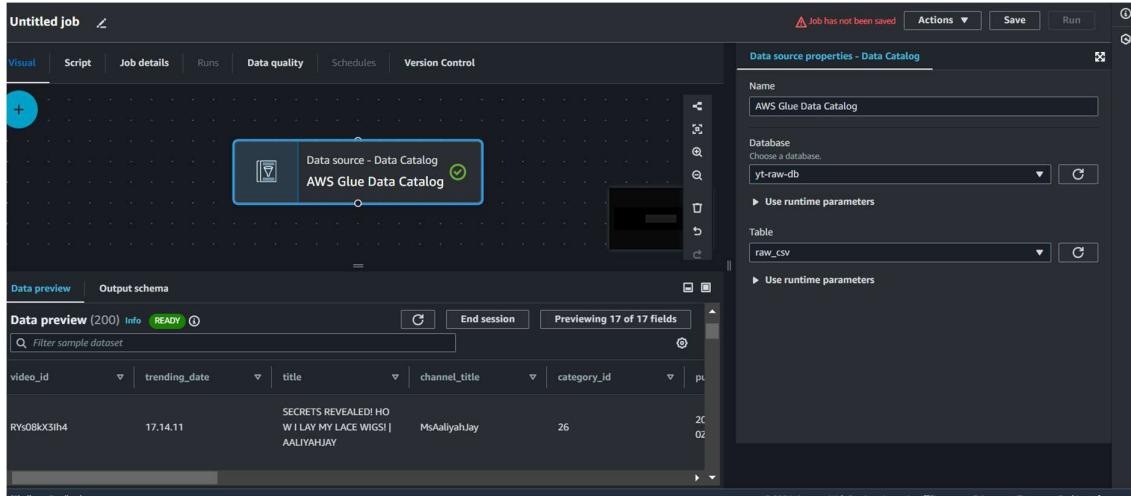
- Confirmation for the parquet files generated.

The screenshot shows the AWS S3 console displaying a list of objects in a bucket. The top navigation bar includes "Objects (4) Info" and various actions like Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload. The main table lists the following objects:

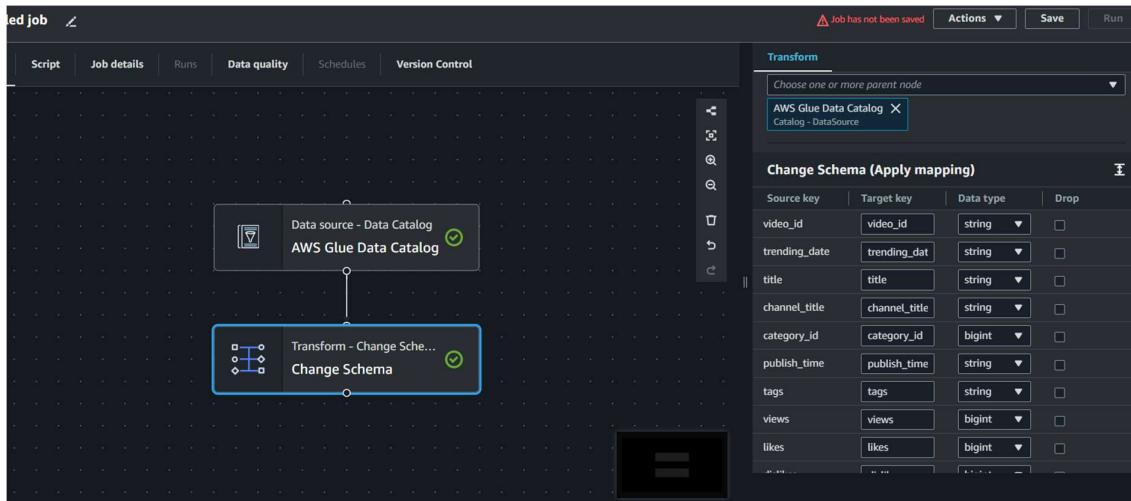
Name	Type	Last modified	Size	Storage class
06c22480eafa4deba6db164af4dd7eea.s nappy.parquet	parquet	May 10, 2024, 17:55:43 (UTC+05:30)	6.1 KB	Standard
c59ac5530a1c44ed85ba52f76f8fa499.s nappy.parquet	parquet	May 10, 2024, 17:55:42 (UTC+05:30)	6.1 KB	Standard
cleaned-csv/ f43b7eea79014a849c95c7773a1ea50. snappy.parquet	Folder	-	-	-

16. Create a ETL job for the csv to parquet conversion.

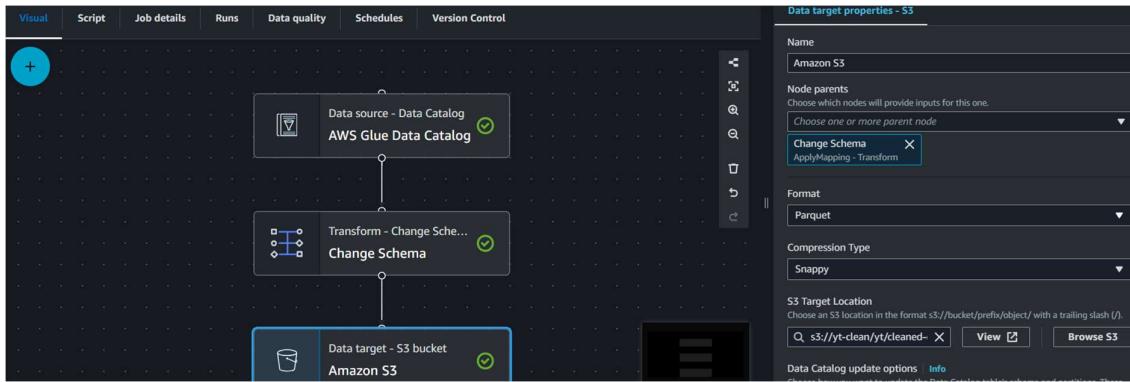
- Create a Glue ETL job that converts the raw csv files into new cleansed data format i.e. into the parquet format. We use Visual ETL tool for the designing the job and provide all the necessary details.
- Select the Data source, in this case, AWS Glue Data Catalog
- Choose the database and raw csv table.
- Provide the IAM role for the Glue Job



- Choose the Transformation that we need to apply, Here we are changing the schema of the table



- Choose the Data Target, here we choose the cleansed S3 bucket for the parquets and cleansed database for the tables .
- Also choose the format(parquet) for the conversion, Partition key (region : partition\_0) and create the job



### Data target properties - S3

Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions

**Database**  
Choose the database from the AWS Glue Data Catalog.  
yt-cleaned-db

**Table name**  
Enter a table name for the AWS Glue Data Catalog.  
clean-csv

**Partition keys - optional**  
Add partition keys.

**Partition (0)**  
partition\_0

**Add a partition key**

Last modified on 5/10/2024, 5:12:28 PM Actions ▾ Save Run

**Script (Locked) Info**

```

1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7
8 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
9 sc = SparkContext()
10 glueContext = GlueContext(sc)
11 spark = glueContext.spark_session
12 job = Job(glueContext)
13 job.init(args['JOB_NAME'], args)
14
15 # Script generated for node AWS Glue Data Catalog
16 AWSGlueDataCatalog_node1715341147410 = glueContext.create_dynamic_frame.from_catalog(database="yt-cleaned-db", table_name="raw_csv", transformation_ctx="AWSGlueDataCatalog_node1715341147410")
17
18 # Script generated for node Change Schema
19 ChangeSchema_node1715341212976 = ApplyMapping.apply(frame=AWSGlueDataCatalog_node1715341147410, mappings=[("video_id", "string", "video_id", "string"), ("trending_date", "string", "trending_date", "string")]
20
21 # Script generated for node Amazon S3
22 AmazonS3_node17153412158708 = glueContext.getSink(path="s3://yt-cleaned/yt-cleaned-csv/", connection_type="s3", updateBehavior="UPDATE_IN_DATABASE", partitionKeys=["partition_0"], enableUpdateCatalog=True, t
23 AmazonS3_node17153412158708.setCatalogInfo(catalogDatabase="yt-cleaned-db", catalogTableName="clean-csv")
24 AmazonS3_node17153412158708.setFormat("glueparquet", compression="snappy")
25 AmazonS3_node17153412158708.writeFrame(ChangeSchema_node1715341212976)
26 job.commit()

```

## 17. Run the ETL job

- It creates the parquets file in the s3 and tables in the cleaned database

The screenshot shows the AWS Glue Job Runner interface for a job named 'yt-csv-par-job'. The 'Runs' tab is selected, displaying one run entry. The run status is 'Succeeded', with a duration of '1 m 7 s'. The run was started at '05/10/2024 17:14:25' and ended at '05/10/2024 17:15:42'. The interface includes tabs for 'Run details', 'Input arguments (10)', 'Continuous logs', 'Run insights', 'Metrics', and 'Spark UI'.

- We can verify that parquets are generated according to the partition key specified in the cleansed s3 bucket.

The screenshot shows the AWS S3 console listing objects in a bucket. The objects are organized into three nested folders under a parent folder: 'partition\_0=CA-csv/' (Folder), 'partition\_0=GB-csv/' (Folder), and 'partition\_0=US-csv/' (Folder). Each of these contains a single CSV file. The columns shown are Name, Type, Last modified, Size, and Storage class.

- Now, I have created another crawler to create a catalog like a wrapper around the newly created parquet files.

The screenshot shows the AWS Glue Crawler properties page for a crawler named 'wrapper-around-parquet-catalog'. The crawler is in 'READY' state. The 'Crawler properties' section displays the following configuration:

Name	IAM role	Database	State
wrapper-around-parquet-catalog	glue-s3-full-access	cleaned-csv-database	READY

Other properties listed include:

- Description: -
- Security configuration: -
- Lake Formation configuration: -
- Table prefix: -
- Maximum table threshold: -

A 'Advanced settings' button is visible at the bottom.

- Then, click on run crawler. Then it will create a catalog table inside cleaned-csv-database

The screenshot shows the 'Crawler runs' section of the AWS Glue console. It displays a single crawler run named 'statistics-cleaned'. The run was completed successfully in 52 seconds. The interface includes filters for date and time, and navigation buttons for the list of runs.

- Now, the table will be created in the data catalog(cleaned\_csv\_dat), this table contains the data of the csv files for the 3 regions we specified earlier

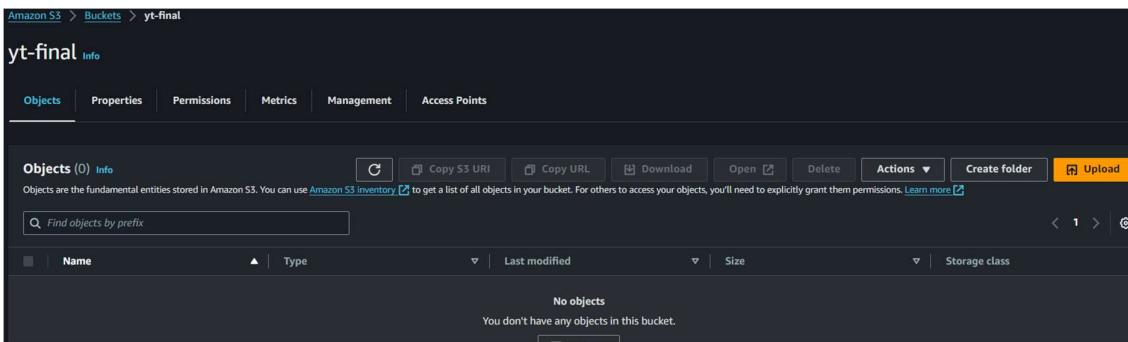
The screenshot shows the 'Databases' section of the AWS Glue console. A new database named 'statistics-cleaned' has been created. It contains two tables: 'cleaned\_csv\_dat' and 'crawl\_processed', both of which are Parquet format and located in the 's3://s3-cleaned-' bucket.

- Also, we can verify that the partition table is generated, and we now query it for further confirmation.

The screenshot shows the AWS Athena console. A query is being run against the 'cleaned\_csv\_dat' table in the 'yt-cleaned-db' database. The query is: 'SELECT \* FROM "AwsDataCatalog"."yt-cleaned-db"."clean-csv" limit 10;'. The results pane is currently empty.

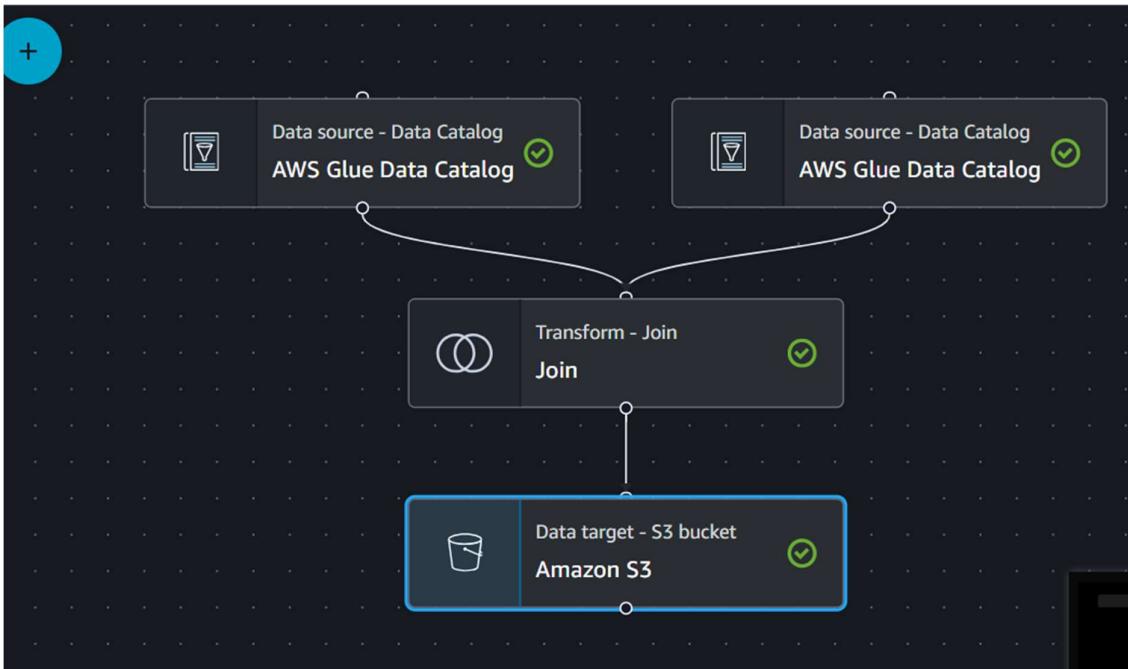
Query results		Query stats	
Completed		Time in queue: 53 ms	Run time: 894 ms
Results (10)			
<input type="text"/> Search rows			<a href="#">Copy</a> <a href="#">Download results</a>
#	video_id	trending_date	title
1	rHwDegptbI4	17.14.11	Dashcam captures truck's near miss with child in Norway
2	J_QGZspO4gg	17.14.11	Sia - Snowman
3	RYS08kX3lh4	17.14.11	SECRETS REVEALED! HOW I LAY MY LACE WIGS!   ALIYAHJAY
4	4FDpjKdltxA	17.14.11	Waking Up with Sam Harris #103 - American Fantasies (with Kurt Andersen)
5	kGOmPmlLndU	17.14.11	The reputation Secret Sessions
6	_wM_jY_rass	17.14.11	Bone on Labour HQ
7	Jj0uBQ7j5c4	17.14.11	Ex engineer leaks how marketing works in the bike industry
8	WqMOHepSpkU	17.14.11	TYSON FURY TO ANTHONY JOSHUA - YOU AINT SEEN **** LIKE THIS YOU WEIGHT-LIFTER!- w/ CHISORA & BUFFER
9	UNtoI5RzCek	17.14.11	Kyle Hits Cartman REAL HARD!!!
10	S0A4hBJHND4	17.14.11	Svjetska ekskluziva: Provala u stan Dejana Lovrena

18. Create Final S3 Bucket for the Analytical purpose.

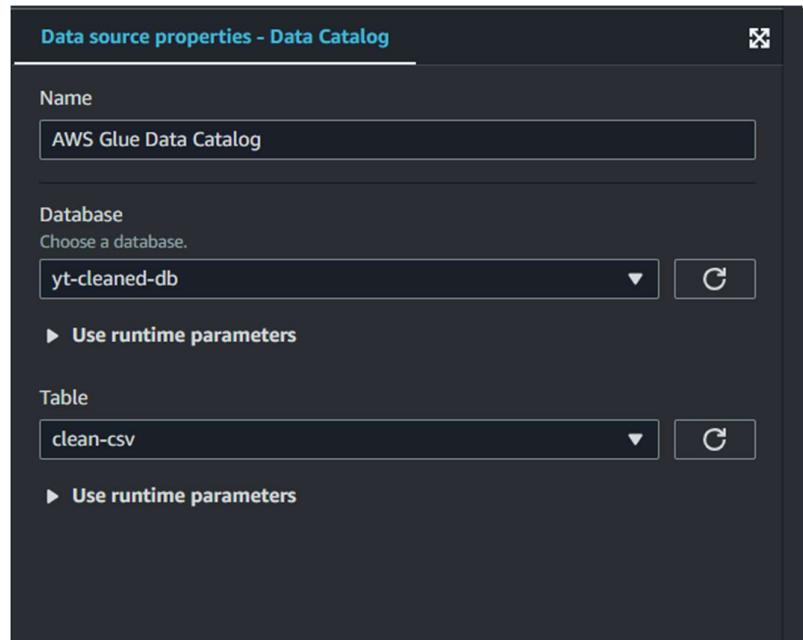


19. Create a ETL job combining both json and csv Data catalogs

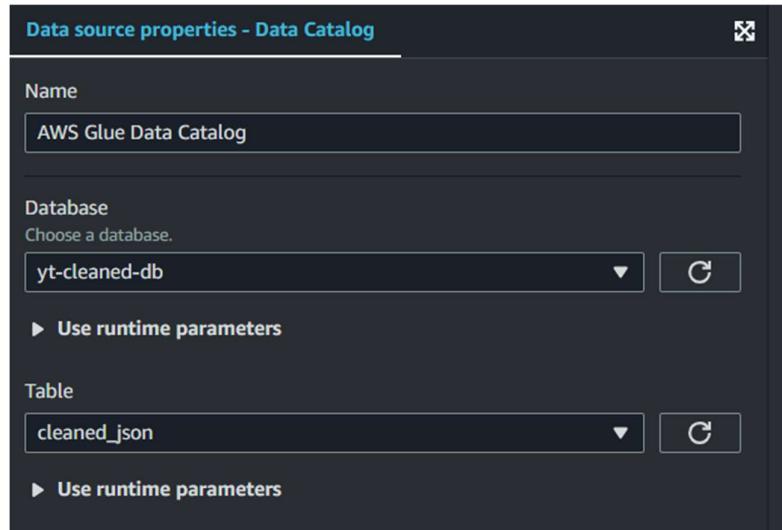
- We create the Glue ETL job for combining the csv and json Data Catalog and store it in a new table, so that we can perform data visualization on the that single table using some visualisation tools.
- We use Visual ETL for designing the job flow.



- Set Data source properties of csv data i.e. csv database and csv data table.



- Data source properties of json data i.e. database and json table data



- Configure the Transformation, here we inner join both Json and csv Data catalogs
- Also provide the joining condition for the tables.

**Transform**

**Name**  
Join

**Node parents**  
Choose which nodes will provide inputs for this one.  
*Choose one or more parent node*

AWS Glue Data Catalog X  
Catalog - DataSource

AWS Glue Data Catalog X  
Catalog - DataSource

**Join type**  
Select the type of join to perform.

Inner join  
Select all rows from both datasets that meet the join condition.

**Join conditions**  
Select a field from each parent node for the join condition.

AWS Glue Data Catalog	AWS Glue Data Catalog
category_id ▾	= id ▾
<input type="button" value="Add condition"/>	

- Provide the Data target properties like, Format (Parquet) of the resultant and the location the final s3 bucket for storing the final result.

**Data target properties - S3**

**Name**  
Amazon S3

**Node parents**  
Choose which nodes will provide inputs for this one.  
*Choose one or more parent node*

Join X  
Join - Transform

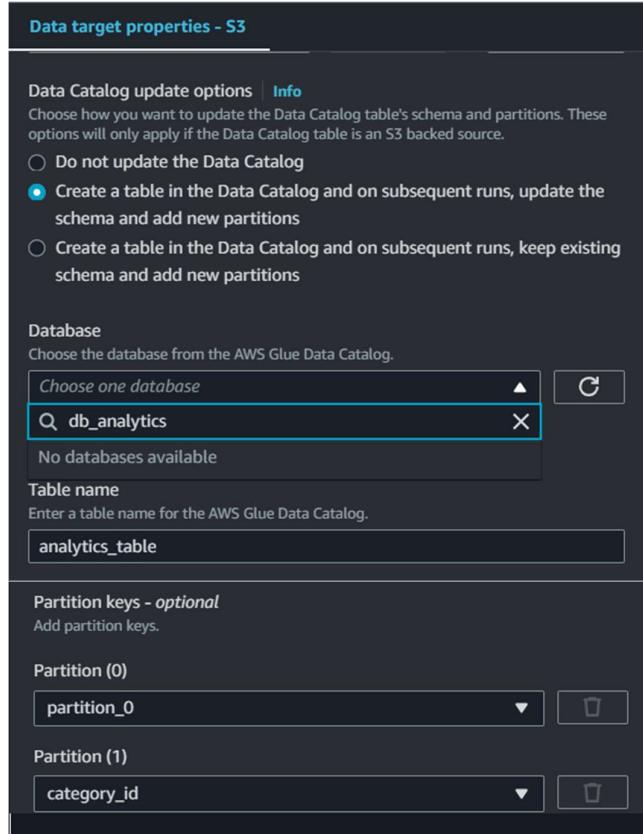
**Format**  
Parquet ▾

ⓘ After you save your job, it will use Glue Studio's optimized Parquet writer. X

**Compression Type**  
Snappy ▾

**S3 Target Location**  
Choose an S3 location in the format s3://bucket/prefix/object/ with a trailing slash (/).

- Also choose the option for creating new table and write the updated schema.
- Also provide Database and table details where new data to be stored.
- Provide the partition key, here region as primary partition key and category\_id as the secondary partition key.



## 20. Run the ETL job.

- Successful run of the ETL job confirms you that new table is generated in the final database.

Job runs (1/1) <a href="#">Info</a>											Last updated (UTC) May 10, 2024 at 12:45:07	<a href="#">Actions</a>	<a href="#">Save</a>	<a href="#">Run</a>
<a href="#">Filter job runs by property</a>											<a href="#">View details</a>	<a href="#">Stop job run</a>	<a href="#">Table View</a>	<a href="#">Card View</a>
Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity...	Worker type	Glue version							
<span style="color: green;">Succeeded</span>	0	05/10/2024 18:13:34	05/10/2024 18:14:55	1 m 12 s	10 DPU	G.1X	4.0							

- Verify the final parquet are generated in the final S3 bucket in accordance with the partition keys specified.

The screenshot shows the AWS S3 console interface for the 'yt-final' bucket. The 'Objects' tab is active, showing three objects: 'partition\_0=CA-csv/' (Folder), 'partition\_0=GB-csv/' (Folder), and 'partition\_0=US-csv/' (Folder). The objects are listed in a table with columns for Name, Type, Last modified, Size, and Storage class. The 'Actions' menu is visible at the top right.

## 21. Visual Analysis using the AWS Quick Sight

- Give, Quick Sight access to AWS Services.

# QuickSight access to AWS services

Make your existing AWS data and users available in QuickSight. [Learn more](#)

## IAM Role

- Use QuickSight-managed role (default)
- Use an existing role

### Allow access and autodiscovery for these resources

- Amazon Redshift
- Amazon RDS
- IAM
- Amazon S3 (7 buckets selected)
  - [Select S3 buckets](#)
- Amazon Athena
  - Make sure you've chosen the right Amazon S3 buckets for QuickSight access
- Amazon S3 Storage Analytics

- Choose dataset from the Aws Athena as data source.

QuickSight

Datasets

SPICE capacity for this region: Auto-purchase enabled

Create a Dataset

FROM NEW DATA SOURCES

The screenshot shows a grid of data source options:

- Upload a file (.csv, .tsv, .clif, .xlsx, .json)
- Salesforce (Connect to Salesforce)
- S3 Analytics
- S3
- Athena
- RDS
- Redshift (Auto-discovered)
- Redshift (Manual connect)
- MySQL

- Choose the database and the Table for the Visualisation of the data and choose Edit/Preview and proceed for the analysis.

Choose your table

youtube\_analytics\_dashboard

Catalog: contain sets of databases.

AwsDataCatalog

Database: contain sets of tables.

db\_analytics

Tables: contain the data you can visualize.

analytics\_table

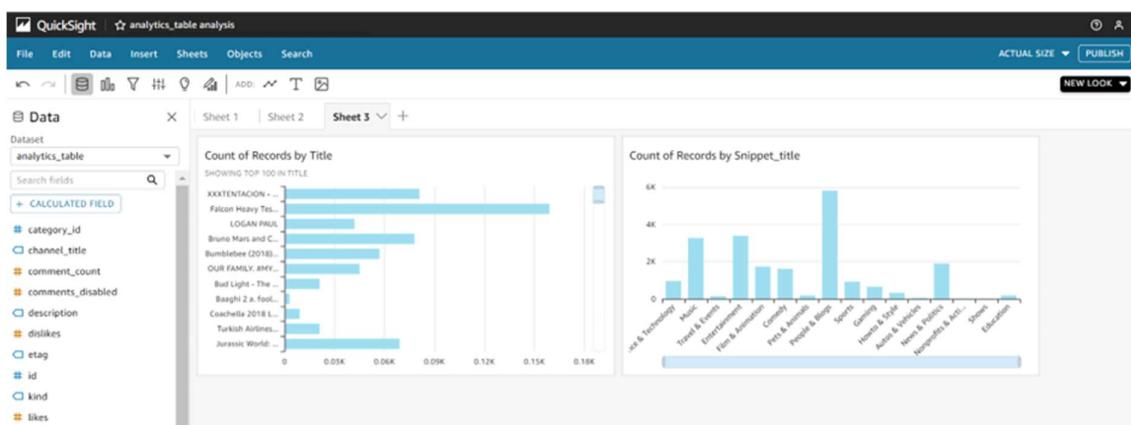
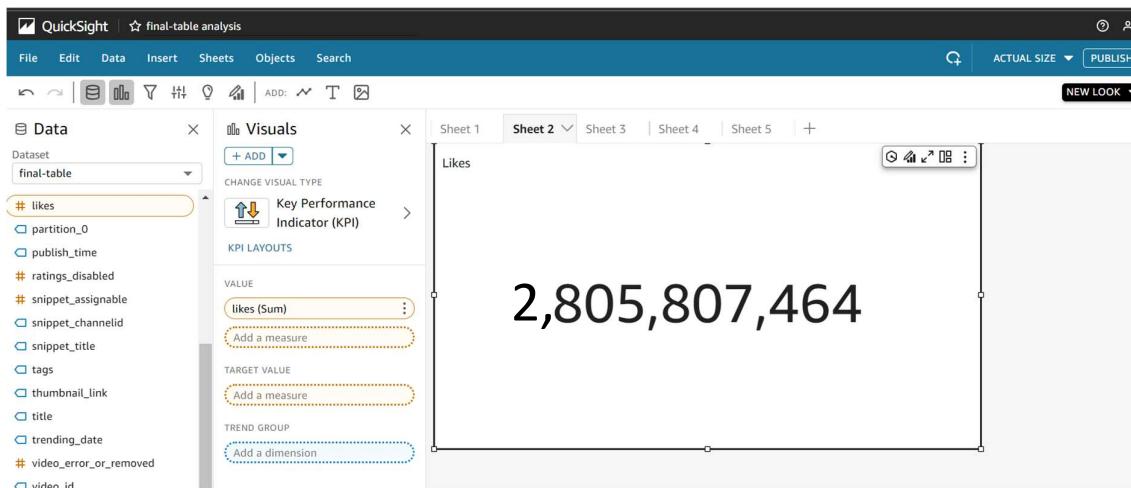
Edit/Preview data    Use custom SQL    Select

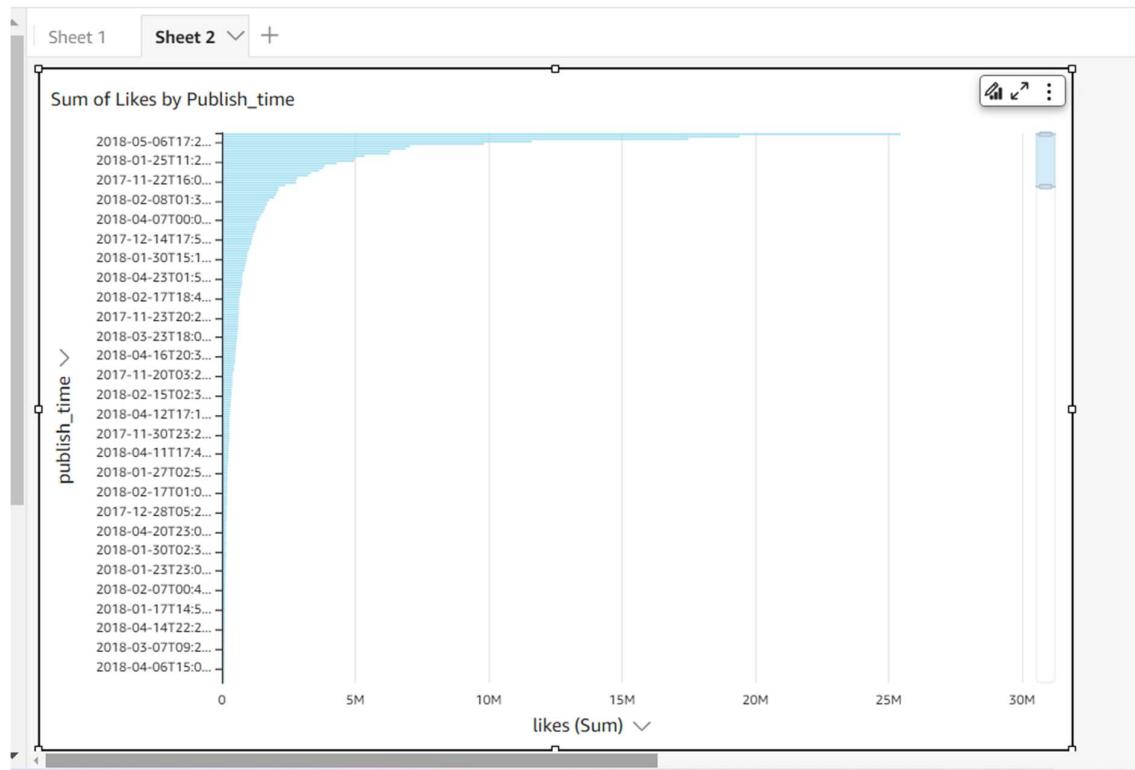
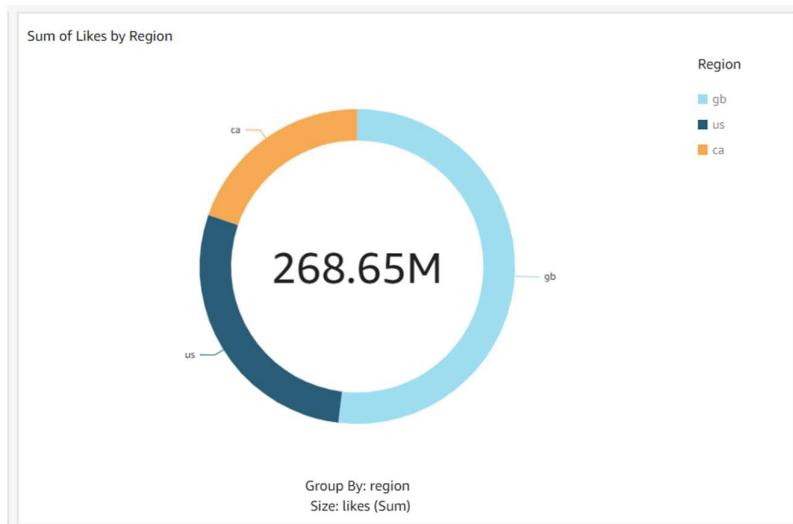
This is a modal dialog box titled "Choose your table". It displays the dataset name "youtube\_analytics\_dashboard". The "Catalog" dropdown is set to "AwsDataCatalog". The "Database" dropdown is set to "db\_analytics". The "Tables" dropdown contains one item, "analytics\_table", which is selected (indicated by a blue circle). At the bottom, there are three buttons: "Edit/Preview data", "Use custom SQL", and a large blue "Select" button.

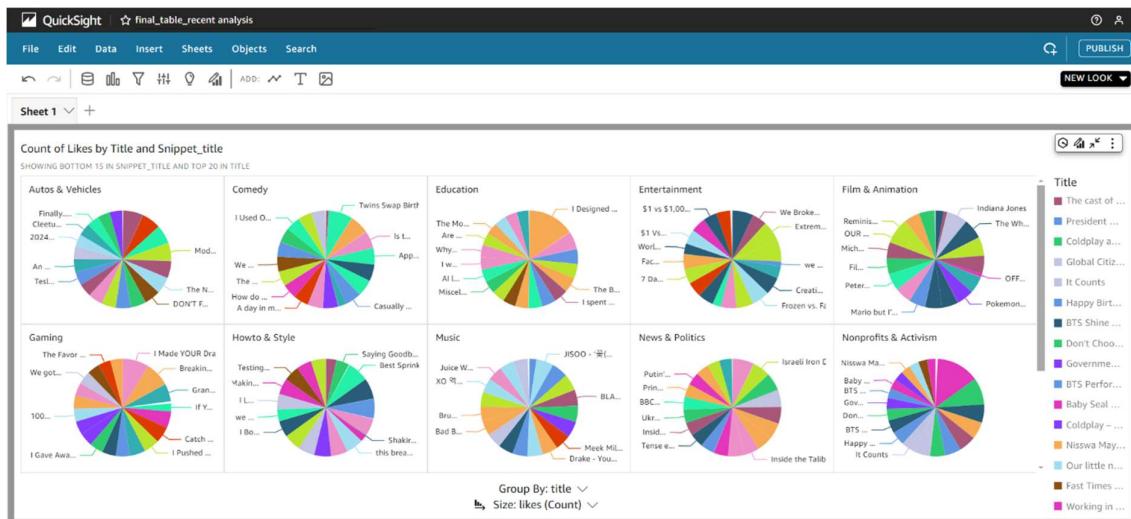
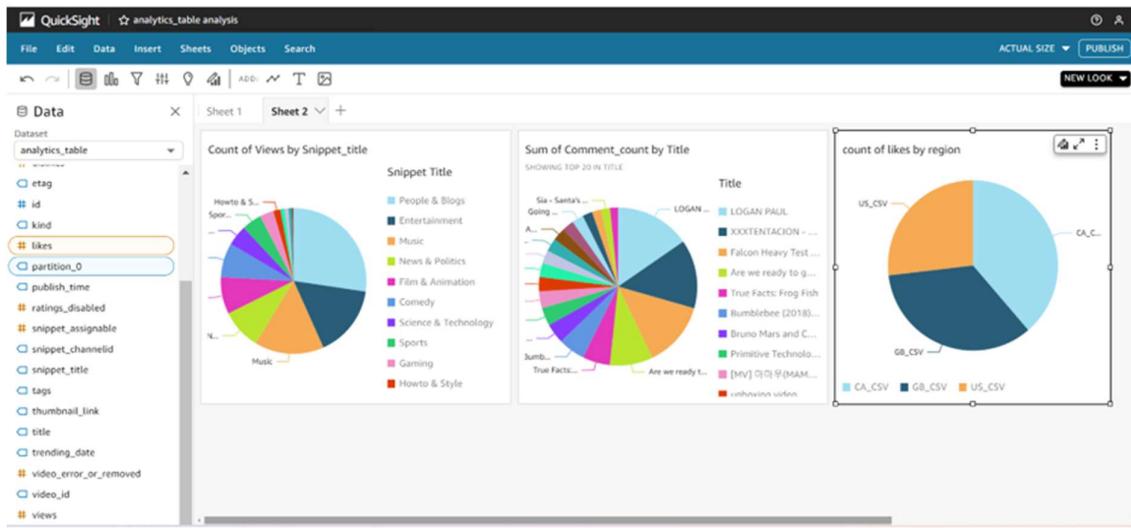
The screenshot shows the QuickSight Data view interface. On the left, there's a sidebar with sections for Fields, Filters, Parameters, Community, and Query mode. The main area is titled "Data" and shows a preview of the "analytics\_table". The table has columns: ratings\_disabled, comments\_disabled, snippet\_title, trending\_date, etag, video\_id, thumbnail\_link, snippet\_assignable, kind, comment\_count, likes, and views. The preview shows four rows of data.

ratings_disabled	comments_disabled	snippet_title	trending_date	etag	video_id	thumbnail_link	snippet_assignable	kind	comment_count	likes	views
0	0	Comedy	17.16.12	"m2yskBQFy... ms9jZ5XkYoU	https://ytli...	1	youtube#vid...	223	2091	139347	
0	0	Comedy	18.23.04	"m2yskBQFy... Tp_JNsBjXQA	https://ytli...	1	youtube#vid...	1108	12806	726455	
0	0	Comedy	18.24.02	"m2yskBQFy... vHwOMWG...	https://ytli...	1	youtube#vid...	5156	16513	6765325	
0	0	Comedy	18.26.04	"m2yskBQFy... Tp_JNsBjXQA	https://ytli...	1	youtube#vid...	1150	13652	762665	

- Choose Different Visual styles and choose appropriate attributes for output to generate.







**Conclusion:** Data Analysis and Visualisation is successfully done on the YouTube Trending data using different AWS Services.