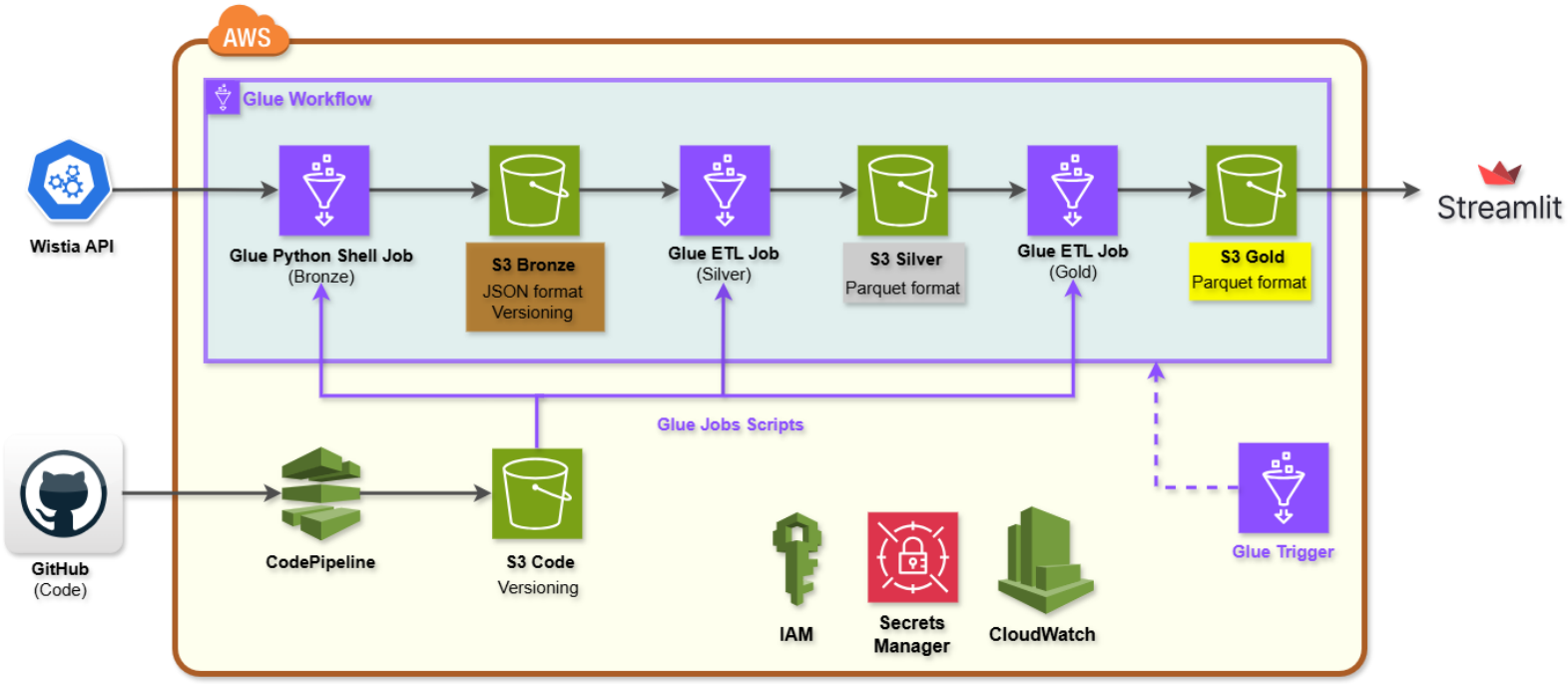
Project 4: Wistia Video Analytics

Overview

In this porject, we will work with JSON format data, and the goal is to pull them from the Wistia Portal. Before, Desiging the data model pipeline, I explored on the Wistia Portal, downloaded the data to my local machine and explored on the data briefly, to understand them well.

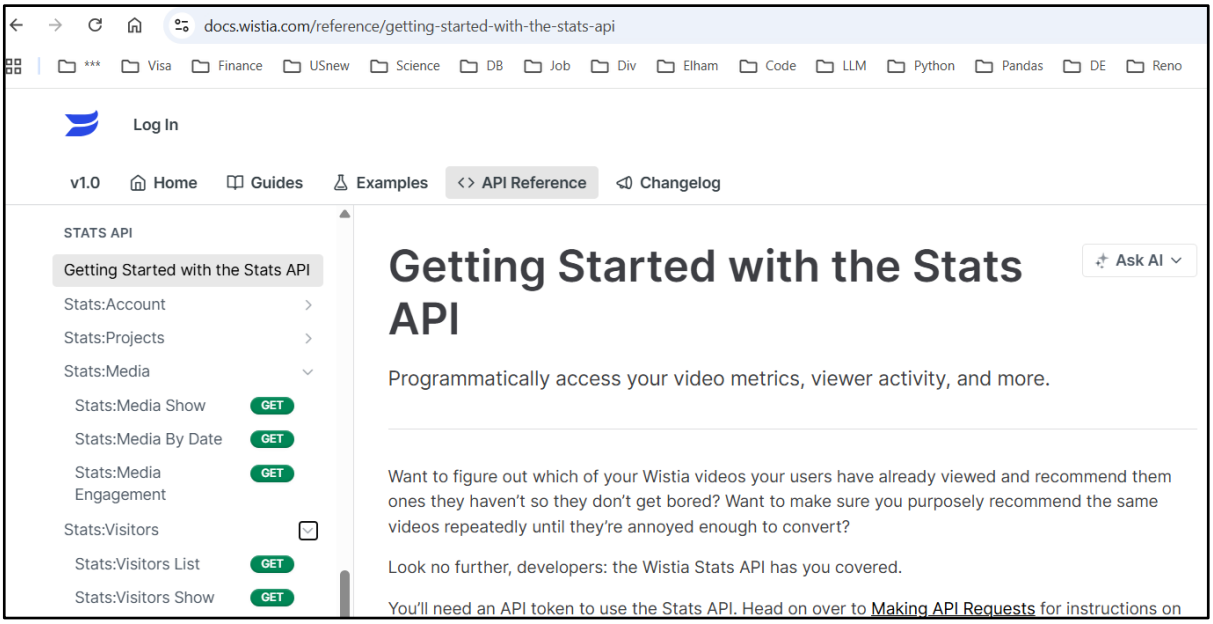
After the exploratory alysis of the the Wistia API data files, I have prepared the proposal and got the approval from Jay. Now, here is the data model pipliene, CI/CD operation and ocrhestration from data ingestion, doing ELT jobs, until visualization to support business insights, customer analytics, and reporting requirements.



Step 1 – Exploratory Analysis (Wistia API)

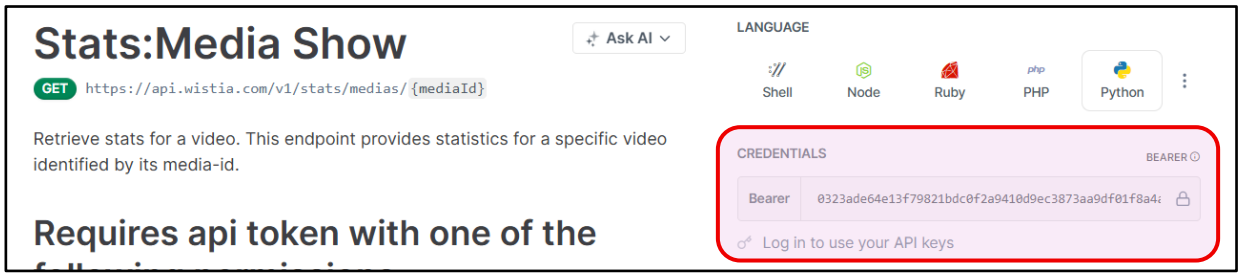
1-1. Wistia API Exploration

In the Wistia API portal, https://docs.wistia.com/reference/getting-started-with-the-stats-api , there is a section, named STATS API, which we will focus on that side.



1-2. Authentication

To access the data of Wistia API, I need a secure API Token. I put it in the **Bearer** section of **Credentials**.



1-3. Endpoints:

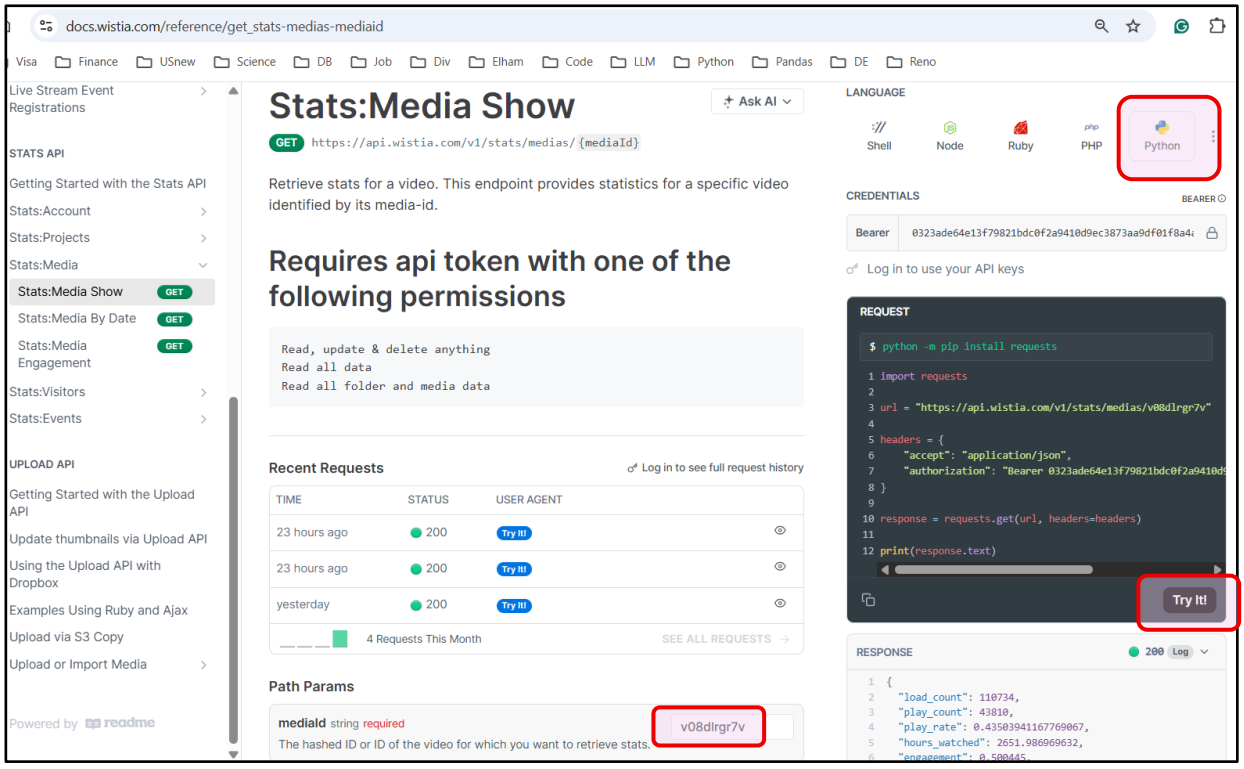
As the project requested, I will focus on two parts of **Stats:Media** and **Stats:Visitors**. In the **Stats:Media**, there are two useful endpoints, and in the **Stats:Visitors**, there are one useful endpoint, which I will use all of them.

1)	**Stats:Media Show**	`Get https://api.wistia.com/v1/stats/medias/{mediaId}`
2)	**Stats:Media Engagement**	`Get https://api.wistia.com/v1/stats/medias/{mediaId}/engagement`
3)	**Stats:Visitors List**	`Get https://api.wistia.com/v1/stats/visitors`

1-4. Get information

Based on projdect description, I have to focus on only two media with `mediaId` of **“v08dlrgr7v”** and **“gskhw4w4lm”**. (I think one of them is related to FaceBook and the other is related to YouTube).

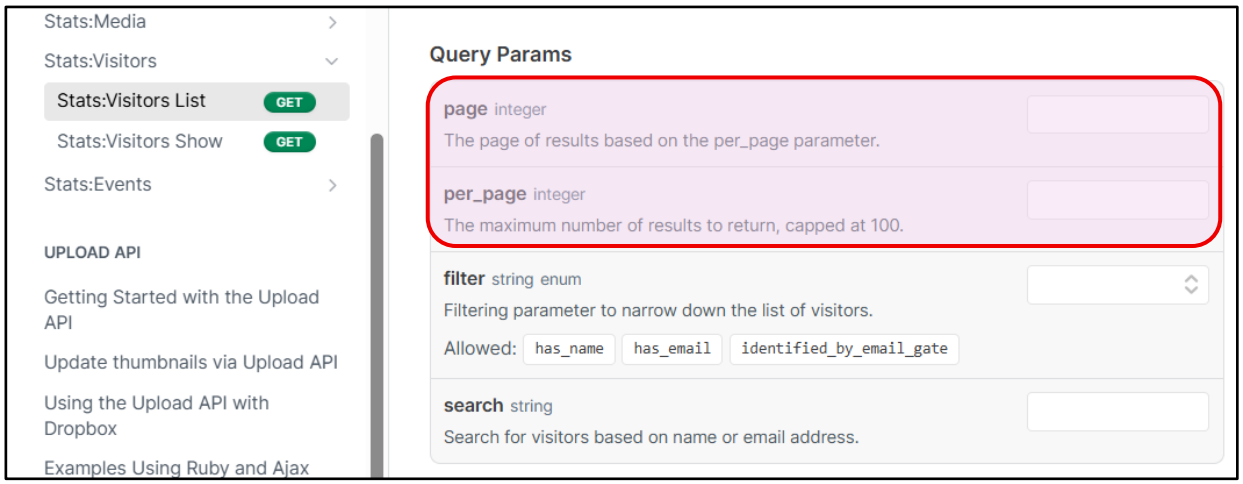
To get the information about these two media, I need to put each of those two `mediaIds` in the parameter section of the first endpoint and then considert the **Python language** and click on the **Try It!** button, as below image.



For example, the `mediaId` of **“v08dlrgr7v”**, shows that this media loaded 110734 times by 94642 different visitors. Also, the `mediaId` of **“gskhw4w4lm”**, shows that this media loaded 111165 times by 104555 different visitors.

Also, we can use the second endpoint, **Stats:Visitors List**, to find detailed information aobut users' engagement by selected media; the time of play and pouse of the media by users.

Moreover, at the visitor level, we can use the third endpoint, **Stats:Media Engagement**, to find general information about total 110361 visitors who are engaging with Wistia. This level of information is not connected directly with media-level information. Therefore, in the parameters section, we cannot put the **`mediaId`**. However, as the number records are very huge, 110361, then to get specific information, we can use other filter parameters, like **page**, **per_page**, and **search**. It means that we need to **pagination** during getting this information. The maximum records **per_page** are capped at **100**.



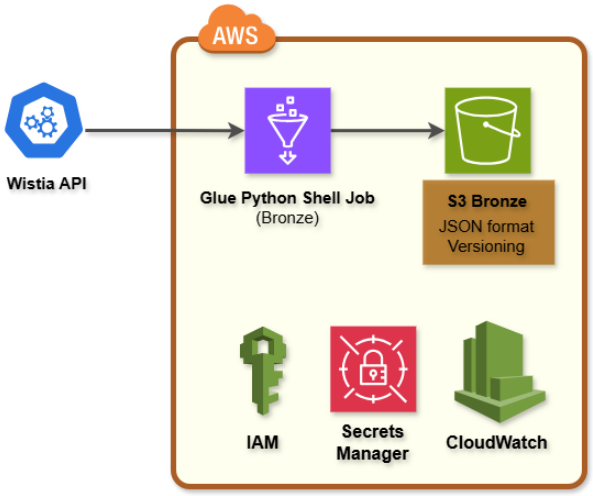
This is information on one of those **110361 visitors**:


```
[
  {
    "visitor_key": "1767733_916148e7-3375-4eed-9e82-0ba235e2155a-c87496ba3-abbd20671cf2-6259",
    "created_at": "2026-01-06T21:03:42.000Z",
    "last_active_at": "2026-01-06T21:03:24.000Z",
    "last_event_key": "1767733_21b83d0a-18fb-4699-81f1-ae70d3abfe37-bb91c8ef9-8079955f2fa3-80ff",
    "load_count": 1,
    "play_count": 1,
    "identifying_event_key": null,
    "visitor_identity": {
      "name": "",
      "email": null,
      "org": {
        "name": null,
        "title": null
      }
    },
    "user_agent_details": {
      "browser": "Instagram",
      "browser_version": "410",
      "platform": "iOS (iPhone)",
      "mobile": true
    }
  }
]
```

Step 2 – Data Ingestion (Wistia API to S3)

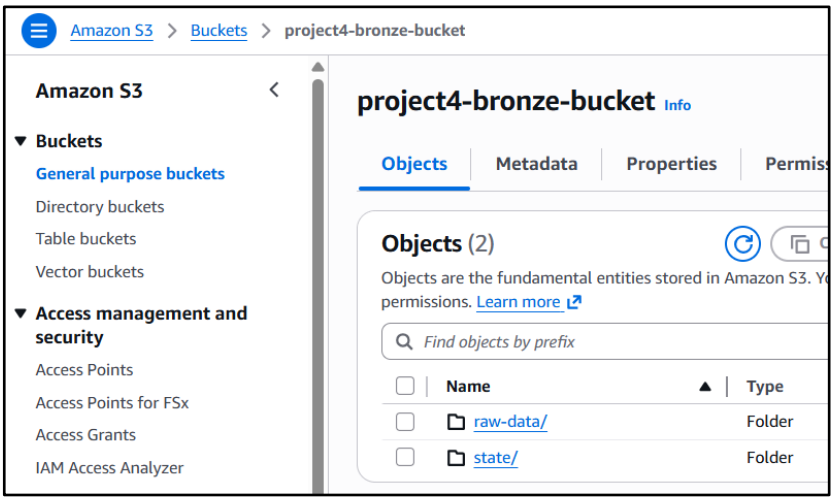
2-1. Objective

The objective of this step is to ingest raw JSON files from Wistia API into an Amazon S3 bucket, named **project4-bronze-bucket**. To comply with the project requirement that allows me to use Python for data ingestion, I will implement **AWS Glue Python Shell Job** for data ingestion, to have orchestration and consistency with Glue Workflow. This approach ensures that the ingestion process is both scalable and aligned with other AWS Glue ETL Jobs in Transformation layers.



2-2. Landing Storage (Bronze Layer)

I used Amazon S3 service and create a bucet with name of **s3://project4-bronze-bucket/**. I have enabled its **versioning** option to store raw and immutable data to support auditability, reprocessing, and downstream transformations. After that, I created two folders inside that bucket with the name of **raw-data** folder for raw JSON files ingestion, and **state** folder for saving the last state of data ingestion.

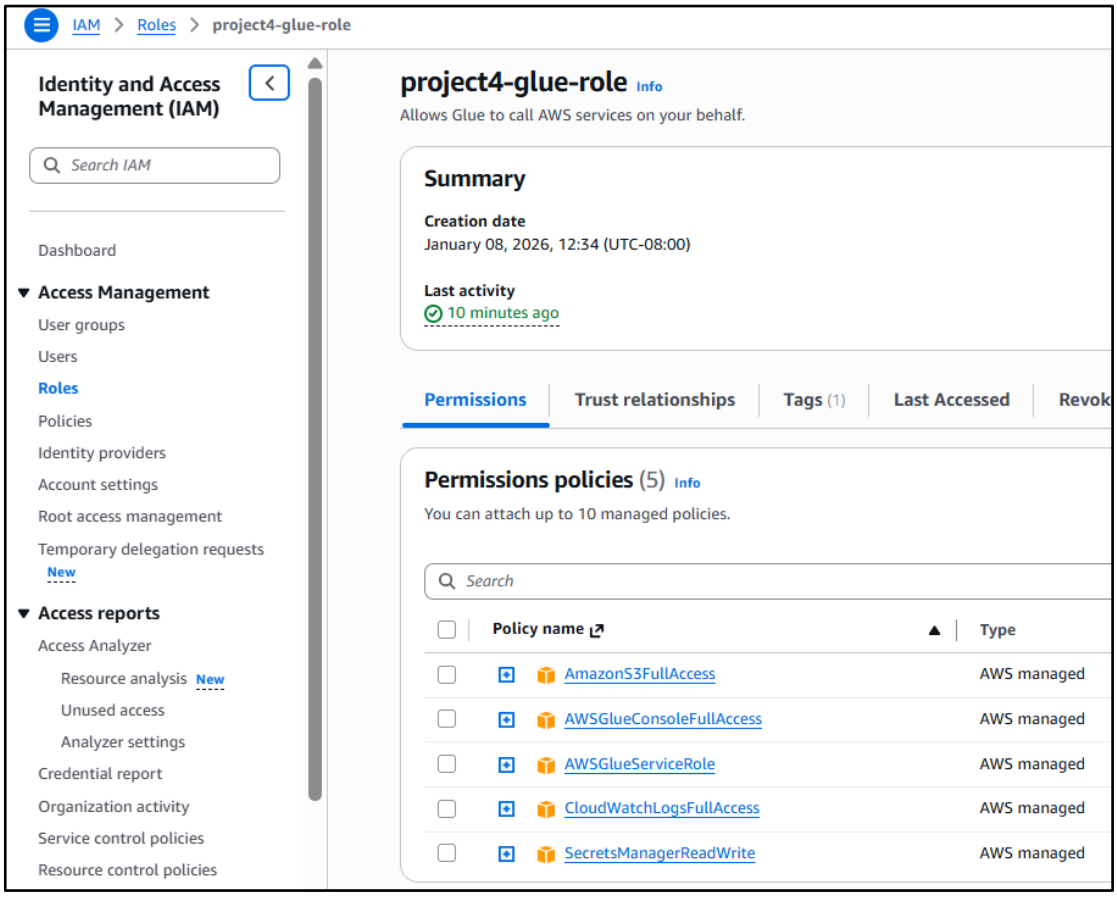


2-3. IAM Role

Before starting data ingestion using AWS Glue ETL jobs, I configured an IAM role to enable AWS Glue to securely access the required AWS services during job execution. The IAM role was named **project4-glue-role** and tagged with **project: 4**, and it was attached to the AWS Glue jobs to ensure secure and authorized access during execution.

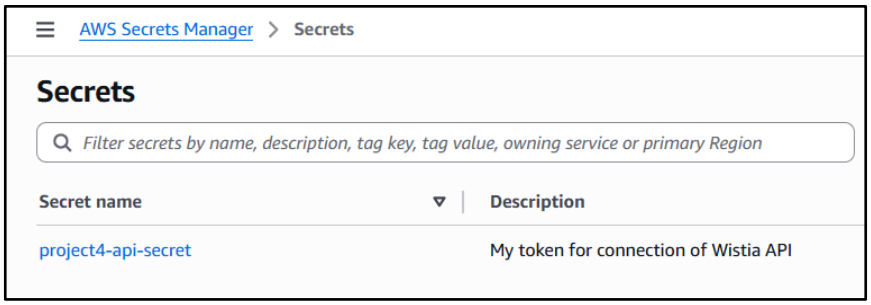
The following permissions were configured:

- **AWSGlueConsoleFullAccess:** Attached to my IAM user to allow me to create and manage AWS Glue resources, such as ETL jobs, and workflows, through the AWS Management Console.
- **AWSGlueServiceRole:** Attached to the Glue job runtime role, allowing AWS Glue to assume the role at execution time and access required services, including Amazon S3, AWS Secrets Manager, and Amazon CloudWatch Logs resources.
- **AmazonS3FullAccess:** Granted to allow AWS Glue to read from and write to the Bronze, Silver, and Gold S3 buckets.
- **CloudWatchLogsFullAccess:** Enabled logging and monitoring of Glue job execution in Amazon CloudWatch.
- **SecretsManagerReadWrite:** Provided access to retrieve and manage secrets, including the RDS credentials.



2-4. Secrets Manager

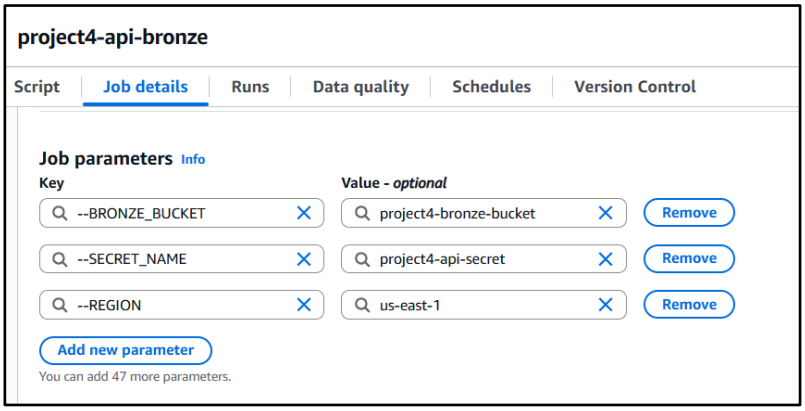
I created a secret, named **project4-api-secret** and tagged with **project: 4**, and put my **Wistia API Token** for API connection in it, in the format of JSON. I do not use the Key/Value format as Glue can only read them in JSON format.



2-5. Glue Python Shell Job (Wistia API - Bronze)

After securing the connection between Wistia API and Glue, now, I am going to create the first AWS Glue job, named **project4-api-bronze**, to perform raw data ingestion from Wistia API into S3 Bronze bucket. For test, the script of this **AWS Glue Python Shell** job is loaded inline script, but later in the CI/CD production level, I will put all Glue Jobs' script in another S3 code bucket. The script file name in the Python language is **project4-api-bronze.py**. If I need the incremental data, I must repeat running this job, like daily! This job is to upload all files related to only two specific media and also all visitors connected to Wistia API into **s3://project4-bronze-bucket/raw-data/**.

In this Glue job, I used 3 arguments and I defined them in the **Job Parameters** section.

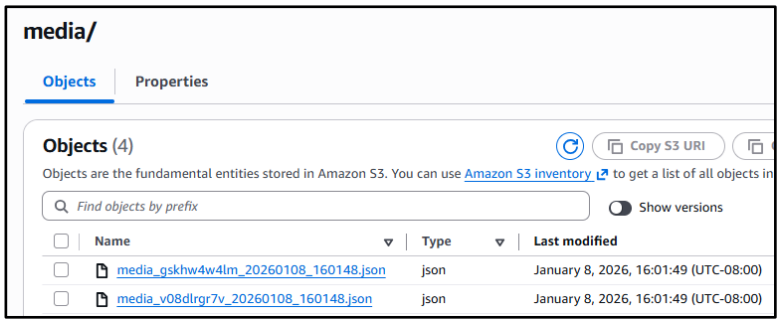


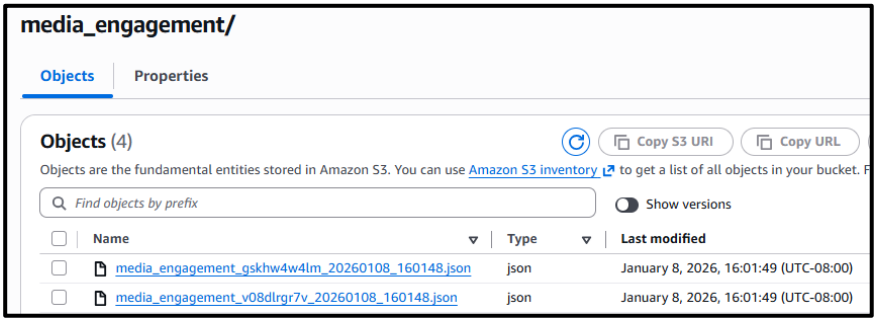
In the configuration, I considered minimal computation system but running time of the job considered for **3 hours**, as my estimation in my local computer was arround 1 hour for the first run.

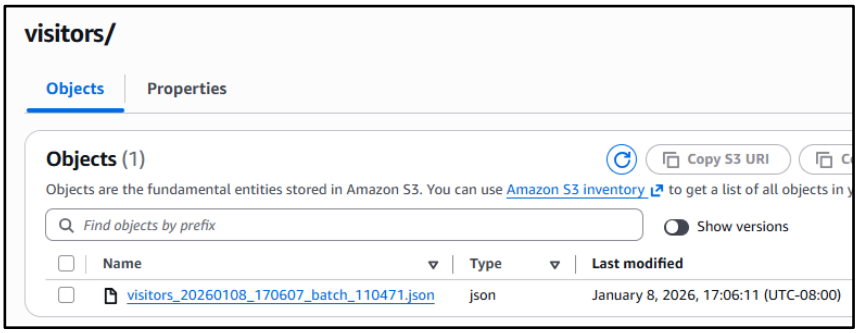
After configuring the Glue job, I am trying to test it and run the job 3 times. I will run again the whole pipeline for 7 days in production mode later.

2-6. Data Ingestion 1st Run

To ingest the data, I ran the **project4-api-bronze** job successfully and got the JSON files in the S3 Bronze bucket. After running the Glue job (for 1h 4m 35 s) all JSON files landed from Wistia API into S3 Bronze bucket successfully. As you can see, there are three foders with file name in **s3://project4-bronze-bucket/raw-data/**, and inside each folder, there are some JSON files. There are two JSON files inside **media** folder, two JSON files inside **media_engagement** folder, and one JSON file inside **visitors** folder.

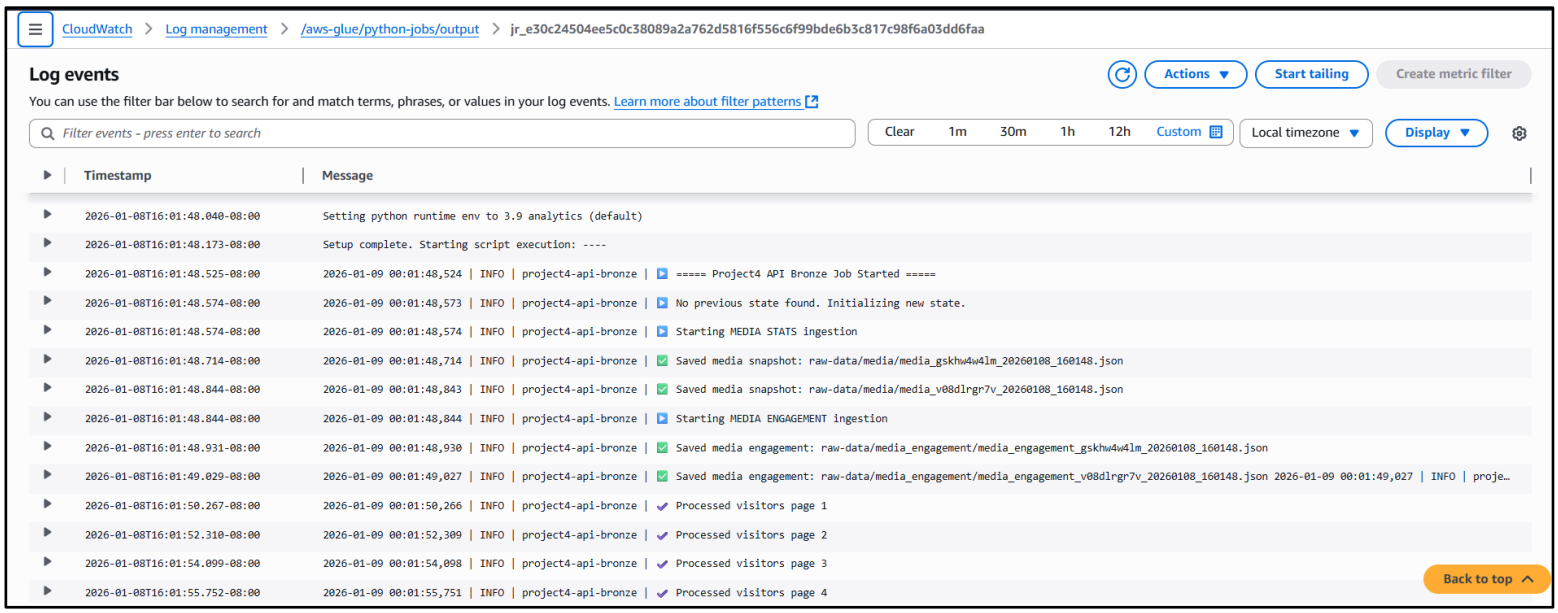


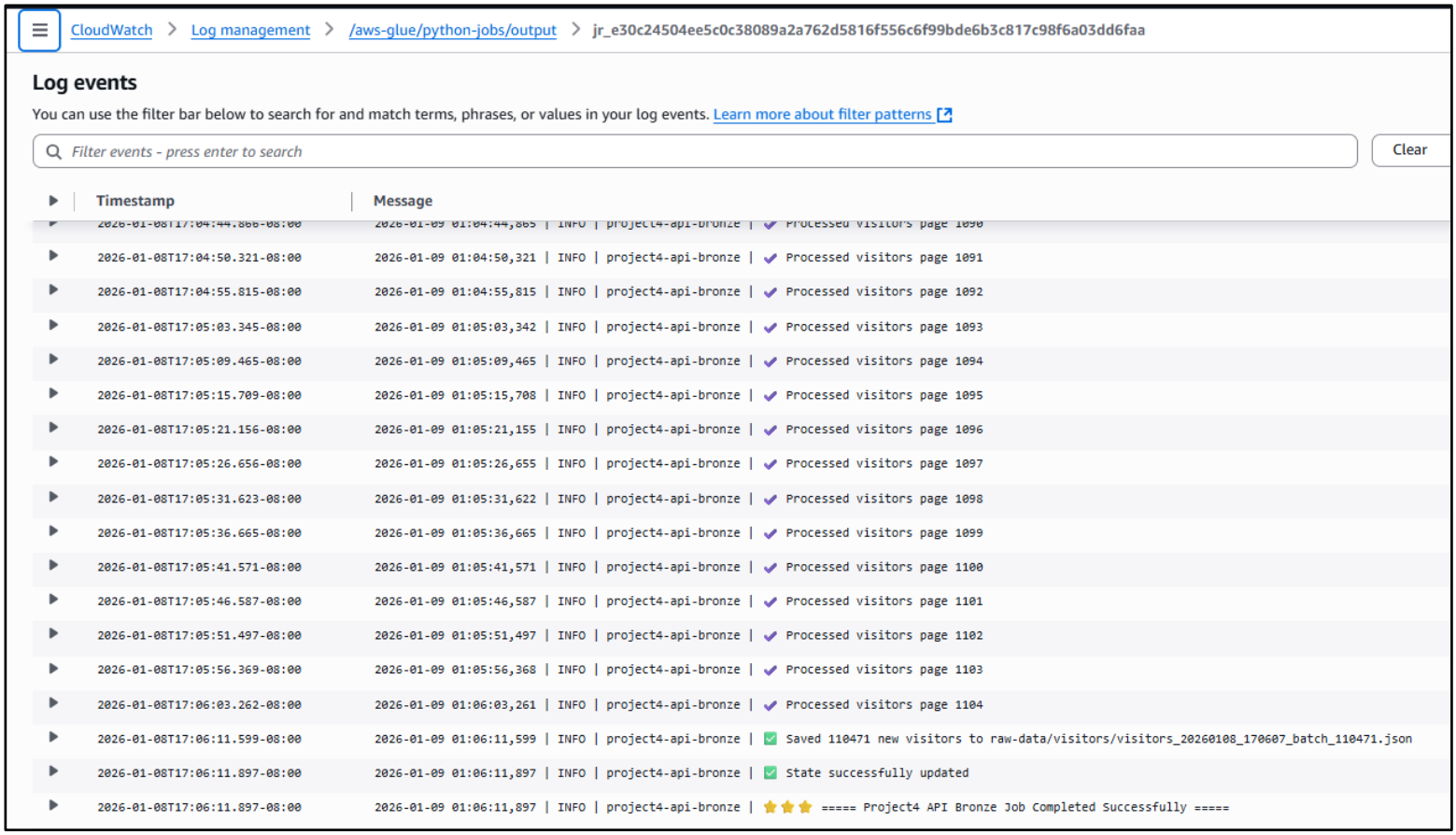




This **visitors_20260108_170607_batch_110471.json** file consists of information of total **110471 visitors**.

The screenshot of output log file from starting the Glue job tuntil finishing the Glue job successfully, is shown below.

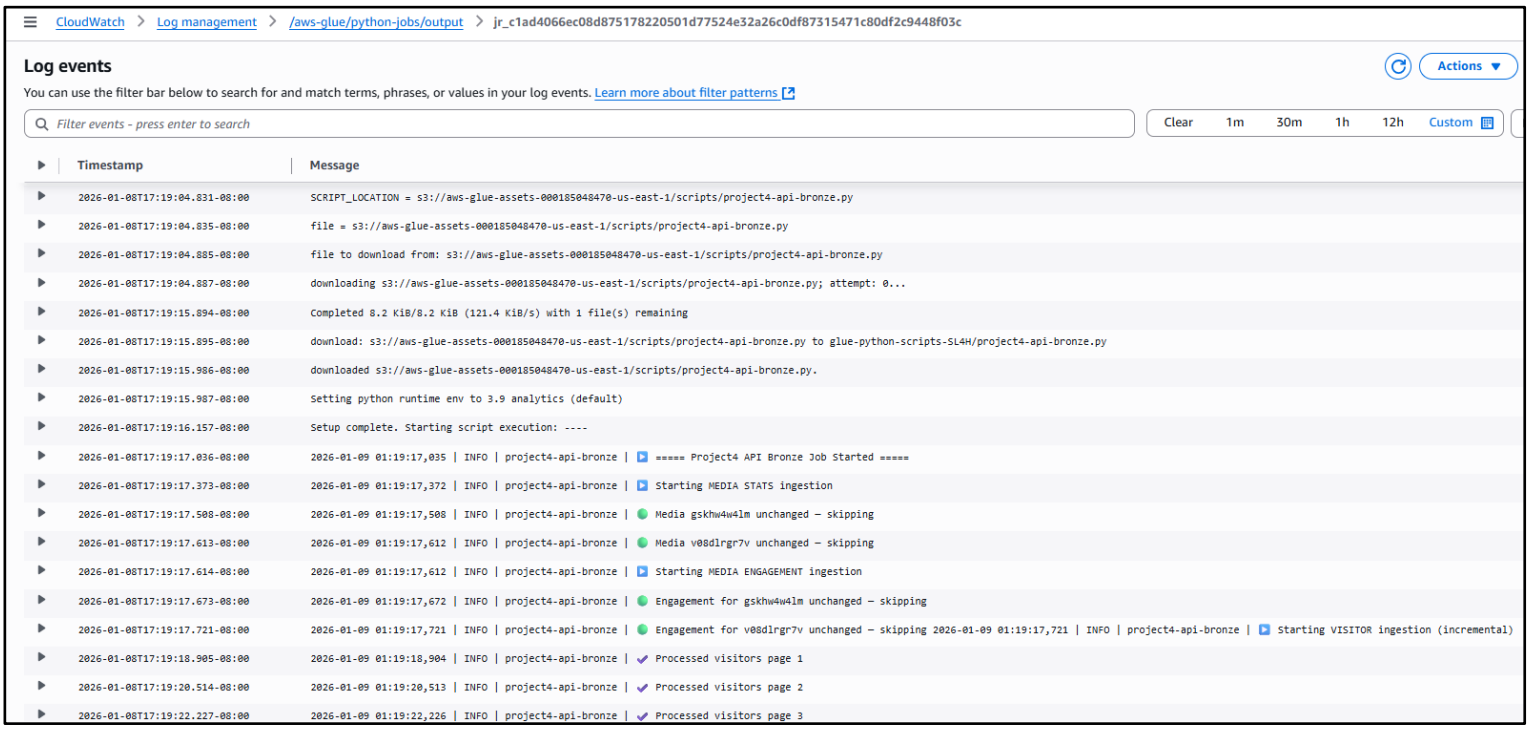

CloudWatch > Log management > /aws-glue/python-jobs/output > jr_e30c24504ee5c0c38089a2a762d5816f556c6f99bde6b3c817c98f6a03dd6faa
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Actions
Start tailing
Create metric filter
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
Local timezone
Display
Timestamp
Message
2026-01-08T16:01:48.040-08:00 Setting python runtime env to 3.9 analytics (default)
2026-01-08T16:01:48.173-08:00 Setup complete. Starting script execution: ----
2026-01-08T16:01:48.525-08:00 2026-01-09 00:01:48,524 | INFO | project4-api-bronze | ===== Project4 API Bronze Job Started =====
2026-01-08T16:01:48.574-08:00 2026-01-09 00:01:48,573 | INFO | project4-api-bronze | No previous state found. Initializing new state.
2026-01-08T16:01:48.574-08:00 2026-01-09 00:01:48,574 | INFO | project4-api-bronze | Starting MEDIA STATS ingestion
2026-01-08T16:01:48.714-08:00 2026-01-09 00:01:48,714 | INFO | project4-api-bronze | Saved media snapshot: raw-data/media/media_gskhw4w4lm_20260108_160148.json
2026-01-08T16:01:48.844-08:00 2026-01-09 00:01:48,843 | INFO | project4-api-bronze | Saved media snapshot: raw-data/media/media_v08dlrgr7v_20260108_160148.json
2026-01-08T16:01:48.844-08:00 2026-01-09 00:01:48,844 | INFO | project4-api-bronze | Starting MEDIA ENGAGEMENT ingestion
2026-01-08T16:01:48.931-08:00 2026-01-09 00:01:48,930 | INFO | project4-api-bronze | Saved media engagement: raw-data/media_engagement/media_engagement_gskhw4w4lm_20260108_160148.json
2026-01-08T16:01:49.029-08:00 2026-01-09 00:01:49,027 | INFO | project4-api-bronze | Saved media engagement: raw-data/media_engagement/media_engagement_v08dlrgr7v_20260108_160148.json 2026-01-09 00:01:49,027 | INFO | proje…
2026-01-08T16:01:50.267-08:00 2026-01-09 00:01:50,266 | INFO | project4-api-bronze | Processed visitors page 1
2026-01-08T16:01:52.310-08:00 2026-01-09 00:01:52,309 | INFO | project4-api-bronze | Processed visitors page 2
2026-01-08T16:01:54.099-08:00 2026-01-09 00:01:54,098 | INFO | project4-api-bronze | Processed visitors page 3
2026-01-08T16:01:55.752-08:00 2026-01-09 00:01:55,751 | INFO | project4-api-bronze | Processed visitors page 4
Back to top

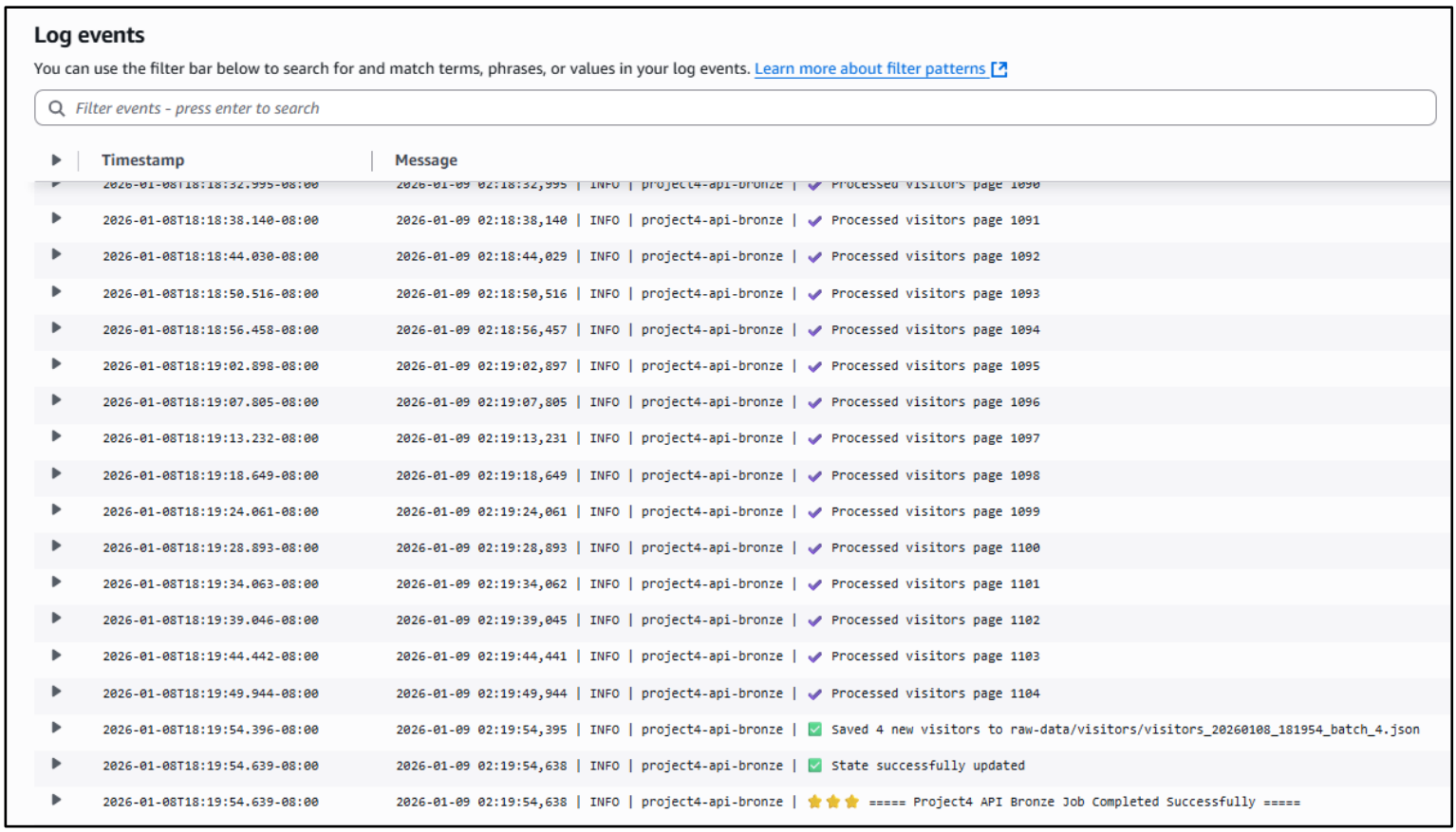

CloudWatch > Log management > /aws-glue/python-jobs/output > jr_e30c24504ee5c0c38089a2a762d5816f556c6f99bde6b3c817c98f6a03dd6faa
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Filter events - press enter to search
Clear
Timestamp
Message
2026-01-08T17:04:50.321-08:00 2026-01-09 01:04:50,321 | INFO | project4-api-bronze | Processed visitors page 1091
2026-01-08T17:04:55.815-08:00 2026-01-09 01:04:55,815 | INFO | project4-api-bronze | Processed visitors page 1092
2026-01-08T17:05:03.345-08:00 2026-01-09 01:05:03,342 | INFO | project4-api-bronze | Processed visitors page 1093
2026-01-08T17:05:09.465-08:00 2026-01-09 01:05:09,465 | INFO | project4-api-bronze | Processed visitors page 1094
2026-01-08T17:05:15.709-08:00 2026-01-09 01:05:15,708 | INFO | project4-api-bronze | Processed visitors page 1095
2026-01-08T17:05:21.156-08:00 2026-01-09 01:05:21,155 | INFO | project4-api-bronze | Processed visitors page 1096
2026-01-08T17:05:26.656-08:00 2026-01-09 01:05:26,655 | INFO | project4-api-bronze | Processed visitors page 1097
2026-01-08T17:05:31.623-08:00 2026-01-09 01:05:31,622 | INFO | project4-api-bronze | Processed visitors page 1098
2026-01-08T17:05:36.665-08:00 2026-01-09 01:05:36,665 | INFO | project4-api-bronze | Processed visitors page 1099
2026-01-08T17:05:41.571-08:00 2026-01-09 01:05:41,571 | INFO | project4-api-bronze | Processed visitors page 1100
2026-01-08T17:05:46.587-08:00 2026-01-09 01:05:46,587 | INFO | project4-api-bronze | Processed visitors page 1101
2026-01-08T17:05:51.497-08:00 2026-01-09 01:05:51,497 | INFO | project4-api-bronze | Processed visitors page 1102
2026-01-08T17:05:56.369-08:00 2026-01-09 01:05:56,368 | INFO | project4-api-bronze | Processed visitors page 1103
2026-01-08T17:06:03.262-08:00 2026-01-09 01:06:03,261 | INFO | project4-api-bronze | Processed visitors page 1104
2026-01-08T17:06:11.599-08:00 2026-01-09 01:06:11,599 | INFO | project4-api-bronze | Saved 110471 new visitors to raw-data/visitors/visitors_20260108_170607_batch_110471.json
2026-01-08T17:06:11.897-08:00 2026-01-09 01:06:11,897 | INFO | project4-api-bronze | State successfully updated
2026-01-08T17:06:11.897-08:00 2026-01-09 01:06:11,897 | INFO | project4-api-bronze | ⭐⭐⭐ ===== Project4 API Bronze Job Completed Successfully =====

2-7. Data Ingestion 2nd Run

To test the incremental data fetching, I ran manually the **project4-api-bronze** job successfully after about **1 hour** for the second time. I found that the **media-level** information is not changed. After running the Glue job (for 1h 1m) all JSON files landed from Wistia API into S3 Bronze bucket successfully.

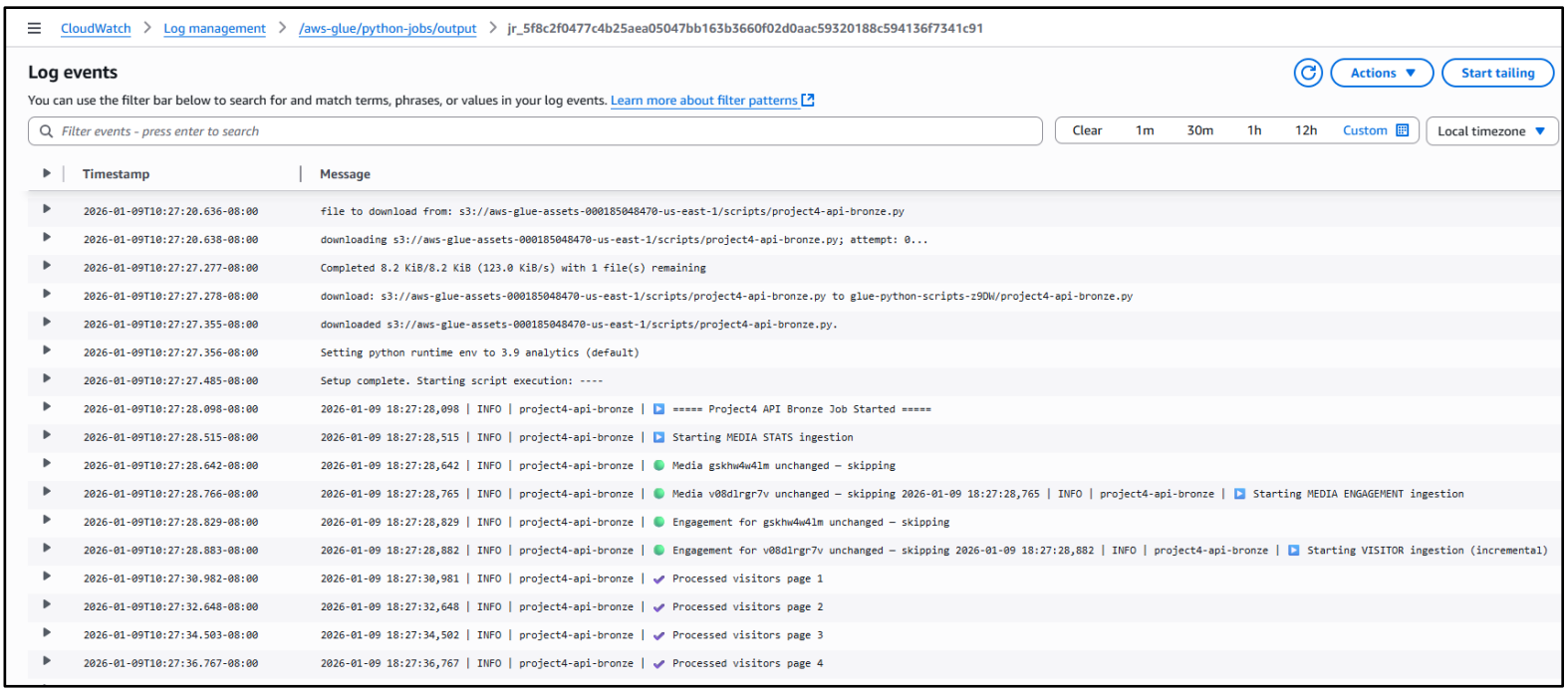
However, after finishig running the **project4-api-bronze** job, I noticed that during about **1 hour**, there are some updates in the **visitor-level**. There were **4 new visitors**, and the new information is saved in new JSON file. As you can see, in the primary visitors file **visitors_20260108_170607_batch_110471.json**, which saved at the time of 17:06:07, there are 110471 visitors, however in the second file, **visitors_20260108_181954_batch_4.json**, which saved at the time of 18:19:54, there are only 4 new visitors.

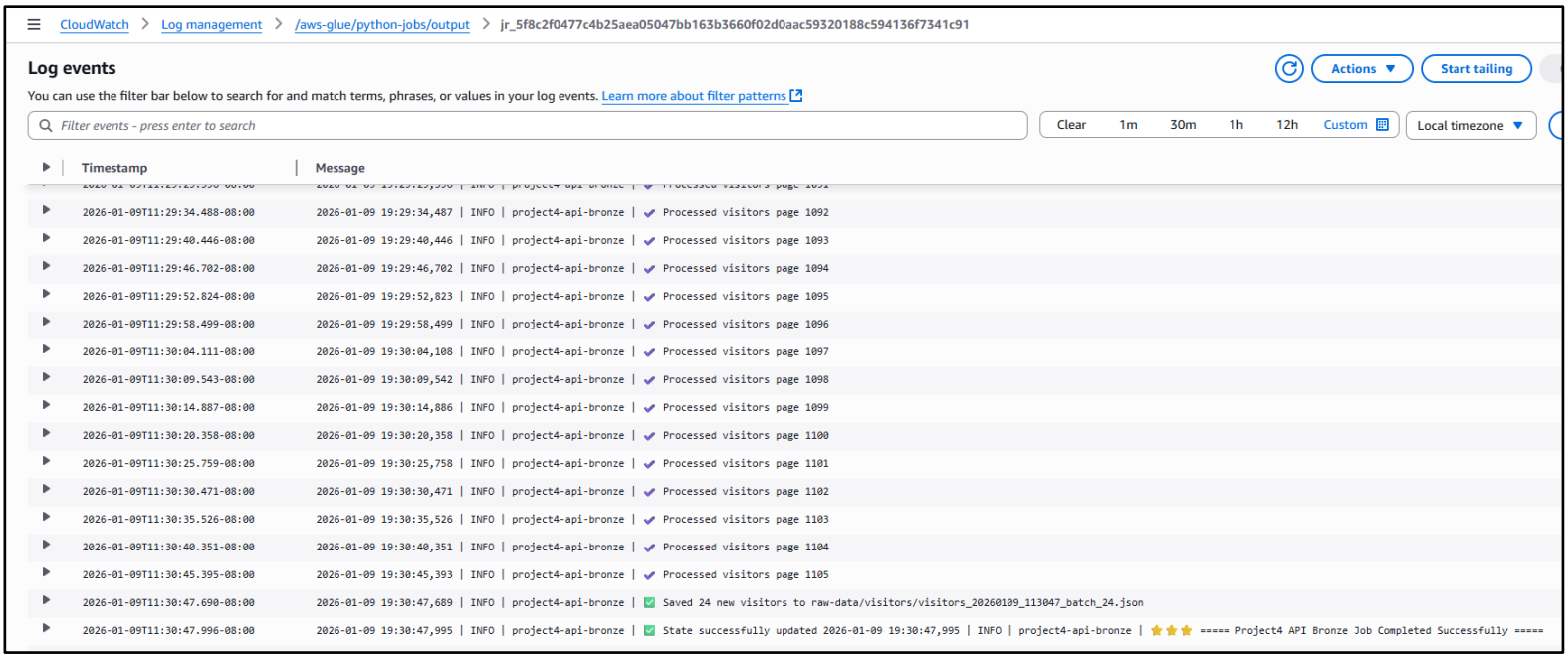




2-8. Data Ingestion 3rd Run

Again, to test the incremental data fetching, I ran manually the **project4-api-bronze** job successfully the **next day** for the third time. I found that the **media-level** information is not changed. However, for **1 day**, there are some updates in the **visitor-level**. There are **24 new visitors**, which is saved in new JSON file **visitors_20260109_113047_batch_24.json**. As you can see, in the primary visitors file **visitors_20260108_170607_batch_110471.json**, which saved at the time of 00:37:48, there are 110471 visitors. Also, in the second visitors file **visitors_20260108_181954_batch_4.json**. Moreover, in the third file, **visitors_20260109_113047_batch_24.json**, which saved at the time of 11:30:47 at the next day, there are only 24 new visitors. It means that the incremental data are fetched successfully!





To validate the data in the Bronze layer, I have downloaded all JSON files from AWS Bronze bucket into my local machine, and checked them carefully.

```
(p311) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-bronze-bucket/raw-data/media/
./data/Bronze/media --recursive --exclude "*" --include "*.json"
(p311) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-bronze-bucket/raw-
data/media_engagement/ ./data/Bronze/media_engagement --recursive --exclude "*" --include "*.json"
(p311) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-bronze-bucket/raw-data/visitors/
./data/Bronze/visitors --recursive --exclude "*" --include "*.json"
```

After downloading JSON files, in Jupyter Notebook and using Pandas library, I explored those JSON files and converted them into dataframes for validation. In the Bronze layer, everything is OK. I will flatten them into rational data format and clean them in the AWS S3 Silver layer and prepare.

Here are three screenshots for three different files (the second-round results):

../data/Bronze/media\media_gskhw4w4lm_20260108_160148.json

	value
load_count	111166.000000
play_count	16681.000000
play_rate	0.154950
hours_watched	336.975662
engagement	0.167007
visitors	104556.000000

../data/Bronze/media_engagement\media_engagement_gskhw4w4lm_20260108_160148.json

	engagement_data	rewatch_data	engagement
0	16108	1542	0.167007
1	9796	663	0.167007
2	9394	629	0.167007
3	8996	570	0.167007
4	8649	542	0.167007
...	...	...	...
431	1790	53	0.167007
432	1785	52	0.167007
433	1780	52	0.167007
434	1754	48	0.167007
435	2060	133	0.167007

436 rows × 3 columns

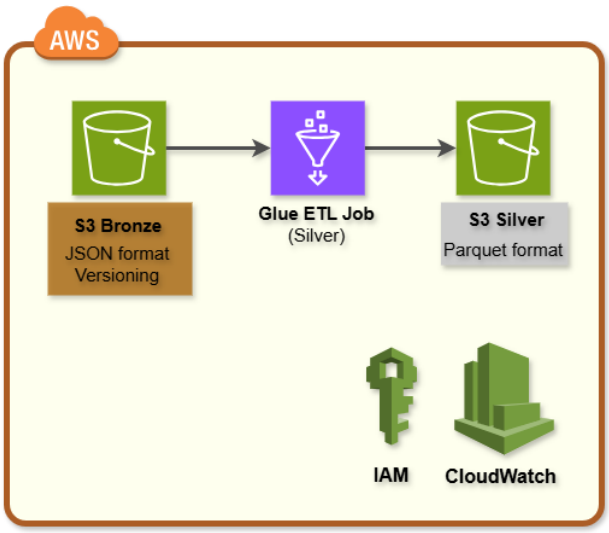
../data/Bronze/visitors\visitors_20260108_181954_batch_4.json

	visitor_key	created_at	last_active_at	last_event_key	load_count	play_count	identifying_event_key	visitor_identity	user_agent_details
0	1767920_38987ced-0b72-4516-a191-3d6696be414c-8...	2026-01-09 01:03:22+00:00	2026-01-09 01:02:50+00:00	1767920_e2c0b383-befb-4383-acd5-b970c3fb1c2d-6...	1	1	NaN	{'name': '', 'email': None, 'org': {'name': No...	{'browser': 'Safari', 'browser_version': '26',...
1	1767918_e9d18517-6ac7-4ff6-bacf-1fe5148edd0d-a...	2026-01-09 00:48:42+00:00	2026-01-09 00:48:16+00:00	1767919_77655319-fc9c-425f-af64-e5f280e616b7-9...	5	1	NaN	{'name': '', 'email': None, 'org': {'name': No...	{'browser': 'Chrome', 'browser_version': '143'...
2	1767919_27c1c780-d5bf-4bfa-bcff-34d864e6777c-8...	2026-01-09 00:41:15+00:00	2026-01-09 00:41:08+00:00	1767919_4e35bc60-4a03-402d-b85e-f0499f220524-c...	1	1	NaN	{'name': '', 'email': None, 'org': {'name': No...	{'browser': 'Chrome', 'browser_version': '143'...
3	1767917_d32b5597-f5d7-4711-906b-6ad1f5c6659d-2...	2026-01-09 00:15:39+00:00	2026-01-09 00:15:27+00:00	1767917_9d603948-2674-456d-9fae-fb540e19ae57-8...	1	1	NaN	{'name': '', 'email': None, 'org': {'name': No...	{'browser': 'Chrome', 'browser_version': '143'...

Step 3 – Transformation (S3 Silver Layer)

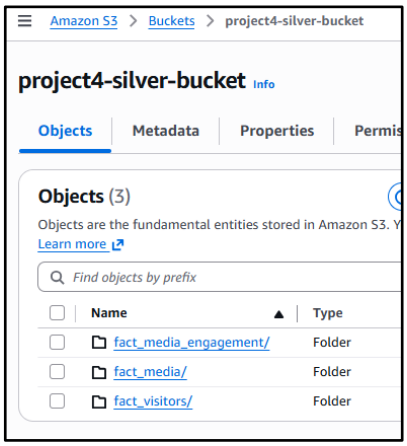
3-1. Objective

The objective of the Silver layer is to transform raw ingested JSON files into clean, standardized, and analytically usable datasets. This step focuses on improving data quality by removing duplicates, enforcing schemas, and preparing fact and dimension tables for downstream analytics and metric computation.



3-2. Transformation Storage (Silver Layer)

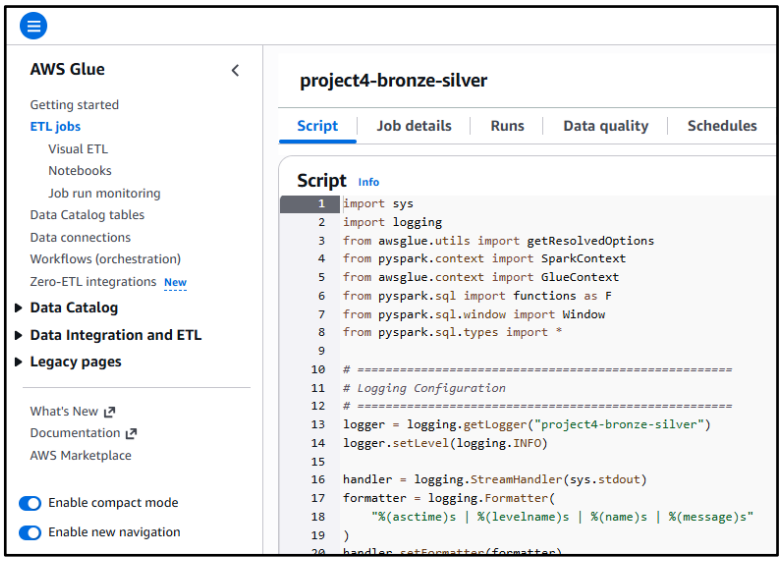
I used Amazon S3 service and create a bucet with name of **s3://project4-silver-bucket/**. All JSON files in the Bronze layer are flattened, then transformed into rational dataset and finally, after cleanup, they are saved in the Parquet format in their own folders in this layer. Data in this layer are structured into fact–oriented datasets, enabling efficient joins and aggregations in subsequent processing stages. I named the forlders as **fact_media**, **fact_media_engagement**, and **fact_visitors**.



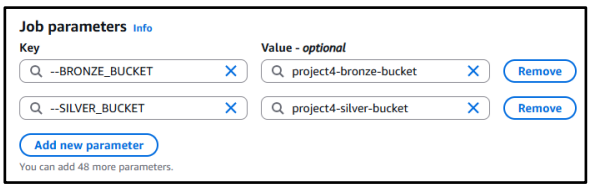
3-3. Glue ETL Job (Bronze - Silver)

An AWS Glue PySpark ETL job, named **project4-bronze-silver**, was executed to perform the following transformations:

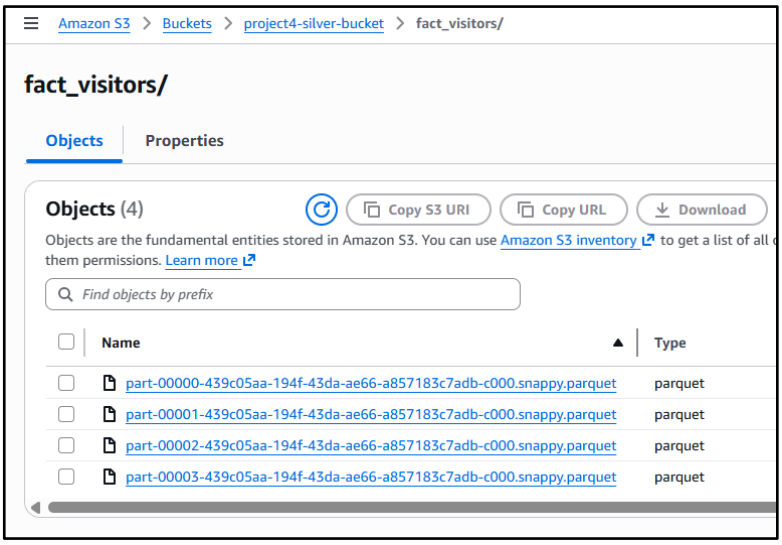
- **Schema Enforcement**
  - Convert JSON files into rational datasets and define schemas for fact tables to prevent schema drifting by flatten nested structures.
  - Ensure consistent column ordering and naming conventions.
- **Data Type Standardization**
  - Cast columns to appropriate data types (e.g., timestamps, numeric fields, boolean flags).
  - Ensure consistency across datasets for join keys and metrics.
- **Duplicate Handling**
  - Identify and remove duplicate records based on defined primary or composite keys.
  - Preserve only the most recent or valid records where applicable.
- **Optimized Storage Format**
  - Persist transformed datasets in **Parquet format** to improve query performance and reduce storage costs.



In this Glue job, In the configuration, I considered minimal computation system. Also, I used 2 arguments and I defined them in the **Job Parameters** section.



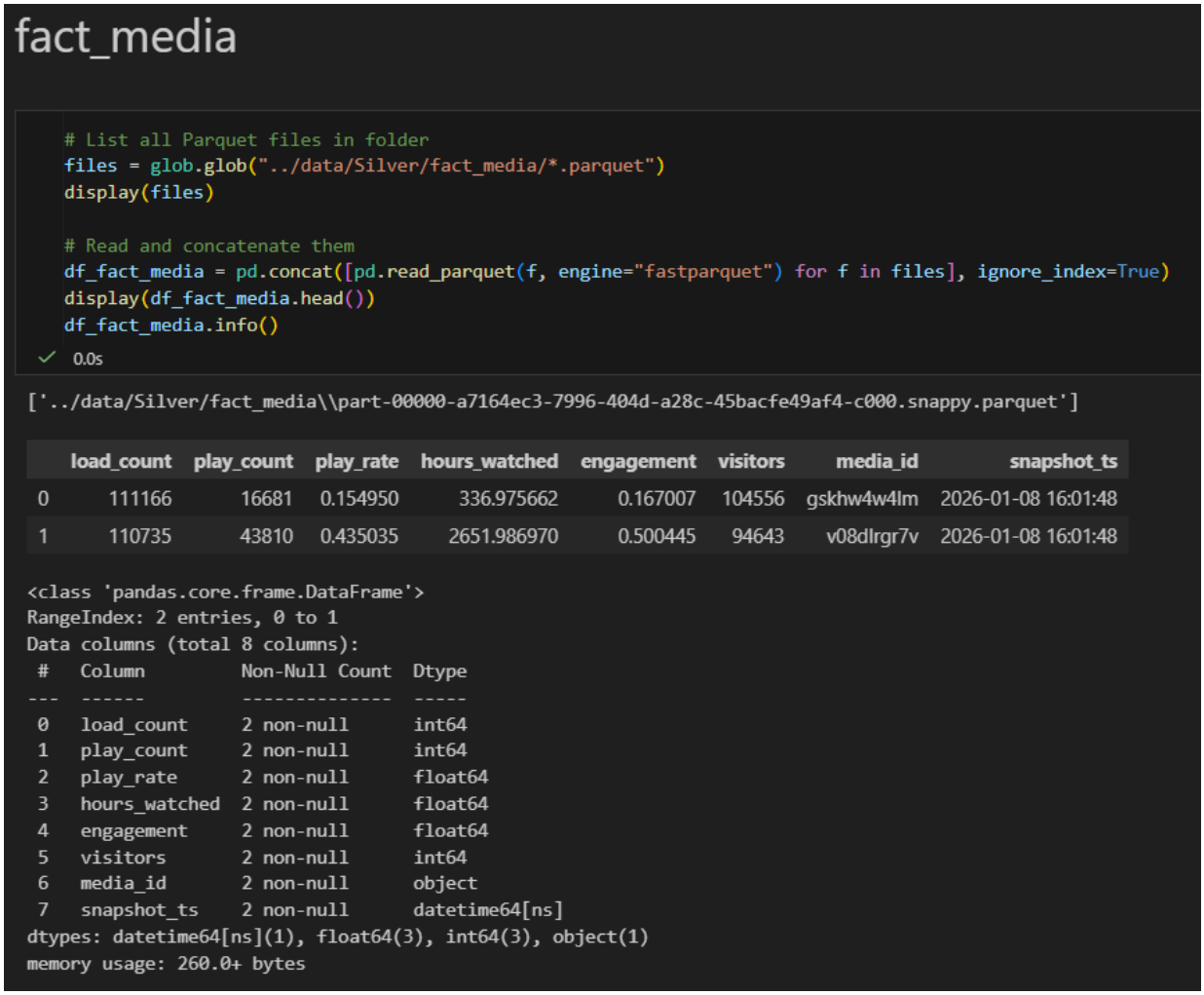
To transform the data, I ran the **project4-bronze-silver** job successfully and got the Parquet files in the S3 Silver bucket. All fact tables are created in three foders with their file name in **s3://project4-silver-bucket/**, and inside each folder, there are some Parquet files.



To validate the data in the Silver layer, I downloaded all Parquet files from AWS Silver bucket into my local machine and checked them carefully.

```
(base) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-silver-bucket/fact_media/
./data/Silver/fact_media --recursive --exclude "*" --include "*.parquet"
(base) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-silver-bucket/fact_media_engagement/
./data/Silver/fact_media_engagement --recursive --exclude "*" --include "*.parquet"
(base) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-silver-bucket/fact_visitors/
./data/Silver/fact_visitors --recursive --exclude "*" --include "*.parquet"
```

After downloading Parquet files, in Jupyter Notebook and using Pandas library, I explored on those Parquet files and converted them into dataframes for validation. In the Silver layer, everything is OK. Here are three screenshots for three different dataframe:



As you can see in the screenshot above, in the **fact_media** table, there are only two records for our two **media_Id**. It can be enlarged as we add more **media_Id** to our project.

fact_media_engagement

```
# List all Parquet files in folder
files = glob.glob("../data/Silver/fact_media_engagement/*.parquet")
display(files)

# Read and concatenate them
df_fact_media_engagement = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_fact_media_engagement.tail())
df_fact_media_engagement.info()
```

✓ 0.0s

```
['../data/Silver/fact_media_engagement\\media_gskhw4w4lm.parquet',
 '../data/Silver/fact_media_engagement\\media_v08dlrgr7v.parquet']
```

	media_id	snapshot_ts	engagement_idx	engagement_count	rewatch_count	engagement
867	v08dlrgr7v	2026-01-08 16:01:48	431	15529	440	0.500445
868	v08dlrgr7v	2026-01-08 16:01:48	432	15386	436	0.500445
869	v08dlrgr7v	2026-01-08 16:01:48	433	15225	414	0.500445
870	v08dlrgr7v	2026-01-08 16:01:48	434	14965	400	0.500445
871	v08dlrgr7v	2026-01-08 16:01:48	435	15719	738	0.500445

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 872 entries, 0 to 871
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   media_id          872 non-null    object
 1   snapshot_ts       872 non-null    datetime64[ns]
 2   engagement_idx    872 non-null    int32
 3   engagement_count  872 non-null    int64
 4   rewatch_count     872 non-null    int64
 5   engagement        872 non-null    float64
dtypes: datetime64[ns](1), float64(1), int32(1), int64(2), object(1)
memory usage: 37.6+ KB
```

As you can see in the screenshot above, there are 435 engagements for each **media_Id,** and they are indexed separately with column **'engagement_idx'**. Then, they merge together and we can detect them based on theor **'media_Id'** and **'engagement_idx'** columns. Also, there will be more engagement JSON files with different time snap in the Bronze layer during production. However, the Glue job only considers the **latest engagement** snapshot for each **media_id**.

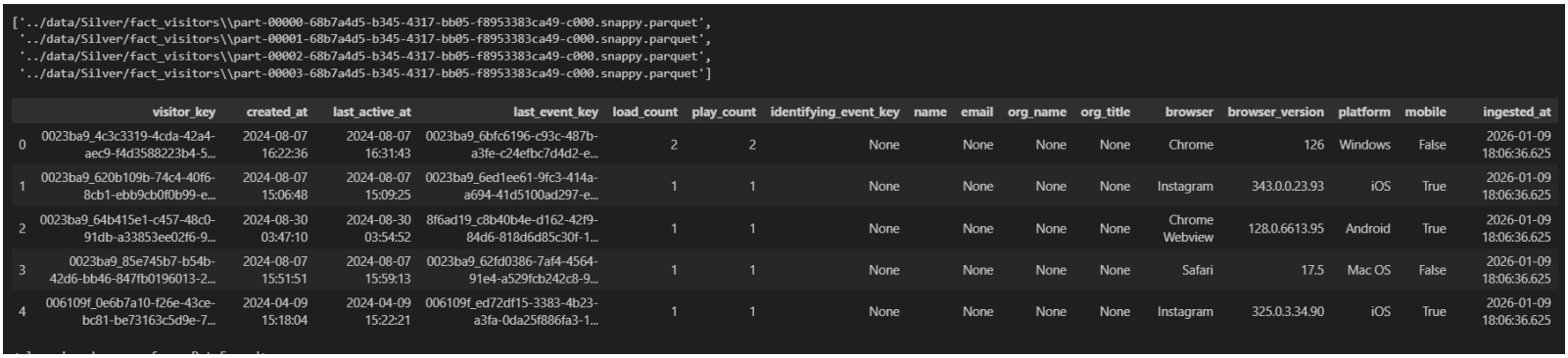
fact_visitors

```
# List all Parquet files in folder
files = glob.glob("../data/Silver/fact_visitors/*.parquet")
display(files)

# Read and concatenate them
df_fact_visitors = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_fact_visitors.head())
df_fact_visitors.info()
```

✓ 0.1s

```
['../data/Silver/fact_visitors\\part-00000-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
 '../data/Silver/fact_visitors\\part-00001-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
 '../data/Silver/fact_visitors\\part-00002-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
 '../data/Silver/fact_visitors\\part-00003-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet']
```



['../data/Silver/fact_visitors\\part-00000-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
'../data/Silver/fact_visitors\\part-00001-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
'../data/Silver/fact_visitors\\part-00002-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet',
'../data/Silver/fact_visitors\\part-00003-68b7a4d5-b345-4317-bb05-f8953383ca49-c000.snappy.parquet']

	visitor_key	created_at	last_active_at	last_event_key	load_count	play_count	identifying_event_key	name	email	org_name	org_title	browser	browser_version	platform	mobile	ingested_at
0	0023ba9_4c3c3319-4cda-42a4-aec9-f4d3588223b4-5...	2024-08-07 16:22:36	2024-08-07 16:31:43	0023ba9_6bfc6196-c93c-487b-a3fe-c24efbc7d4d2-e...	2	2	None	None	None	None	None	Chrome	126	Windows	False	2026-01-09 18:06:36.625
1	0023ba9_620b109b-74c4-40f6-8cb1-ebb9cb0f0b99-e...	2024-08-07 15:06:48	2024-08-07 15:09:25	0023ba9_6ed1ee61-9fc3-414a-a694-41d5100ad297-e...	1	1	None	None	None	None	None	Instagram	343.0.0.23.93	iOS	True	2026-01-09 18:06:36.625
2	0023ba9_64b415e1-c457-48c0-91db-a33853ee02f6-9...	2024-08-30 03:47:10	2024-08-30 03:54:52	8f6ad19_c8b40b4e-d162-42f9-84d6-818d6d85c30f-1...	1	1	None	None	None	None	None	Chrome Webview	128.0.6613.95	Android	True	2026-01-09 18:06:36.625
3	0023ba9_85e745b7-b54b-42d6-bb46-847fb0196013-2...	2024-08-07 15:51:51	2024-08-07 15:59:13	0023ba9_62fd0386-7af4-4564-91e4-a529fcb242c8-9...	1	1	None	None	None	None	None	Safari	17.5	Mac OS	False	2026-01-09 18:06:36.625
4	006109f_0e6b7a10-f26e-43ce-bc81-be73163c5d9e-7...	2024-04-09 15:18:04	2024-04-09 15:22:21	006109f_ed72df15-3383-4b23-a3fa-0da25f886fa3-1...	1	1	None	None	None	None	None	Instagram	325.0.3.34.90	iOS	True	2026-01-09 18:06:36.625

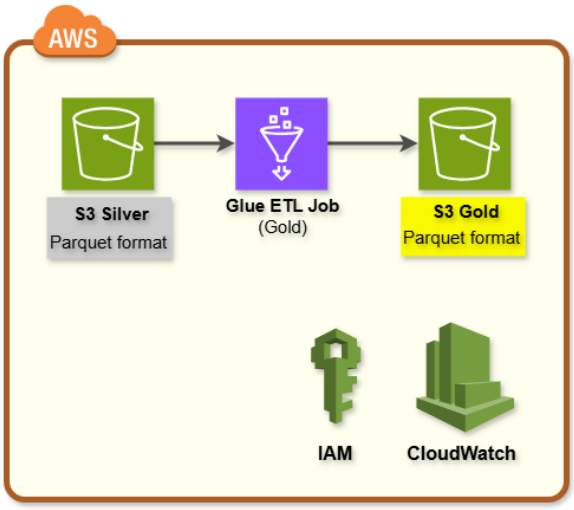
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110499 entries, 0 to 110498
Data columns (total 16 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   visitor_key           110499 non-null  object
 1   created_at            110499 non-null  datetime64[ns]
 2   last_active_at        110499 non-null  datetime64[ns]
 3   last_event_key        110499 non-null  object
 4   load_count            110499 non-null  int64
 5   play_count            110499 non-null  int64
 6   identifying_event_key  0 non-null      object
 7   name                  110499 non-null  object
 8   email                 0 non-null       object
 9   org_name              0 non-null       object
 10  org_title             0 non-null       object
 11  browser               110499 non-null  object
 12  browser_version       110499 non-null  object
 13  platform              110499 non-null  object
 14  mobile                110499 non-null  bool
 15  ingested_at           110499 non-null  datetime64[ns]
dtypes: bool(1), datetime64[ns](3), int64(2), object(10)
memory usage: 12.8+ MB
```

As you can see from the above screen shot for **fact_visitors** table, there are **110499 records**, 110471 visitors at first run, 4 visitors at the second run, and 24 visitors at the third run, which means that the Glue job concatenates all files of visitors from Bronze layer successfully.

Step 4 – Analytics & Aggregation (S3 Gold Layer)

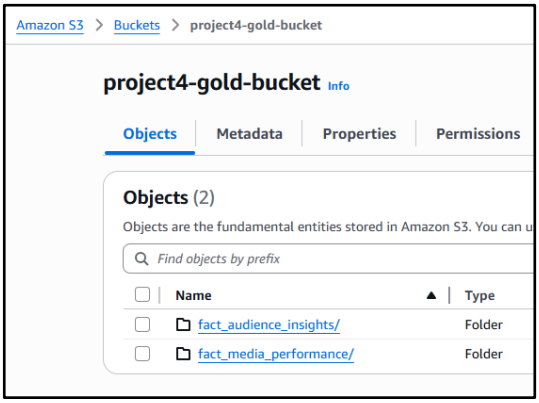
4-1. Objective

In this step, the objective is to produce analytics-ready, aggregated datasets by joining and summarizing the dimensional and fact tables created in the Silver layer. The Gold layer serves as the single source of truth for business metrics and is optimized for direct consumption by visualization and reporting tools.



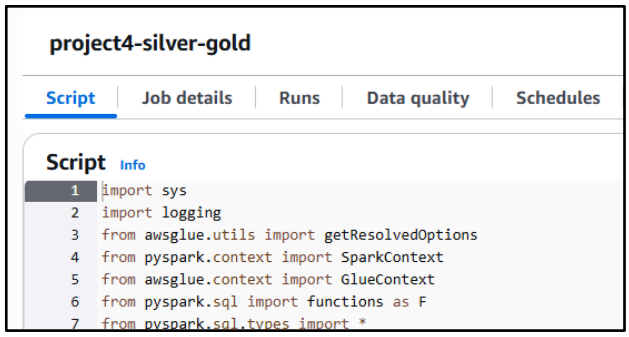
4-2. Gold Layer Storage

I used Amazon S3 service and create a bucet with name of **s3://project4-gold-bucket/**. In this layer, all fact tables and Parquet files of the Silver layer are merged, organized, and after cleanup and normalization, be prapared in the Parquet format for machine learning models and BI insights. I named the forlders as **fact_media_performance** and **fact_audience_insights**.



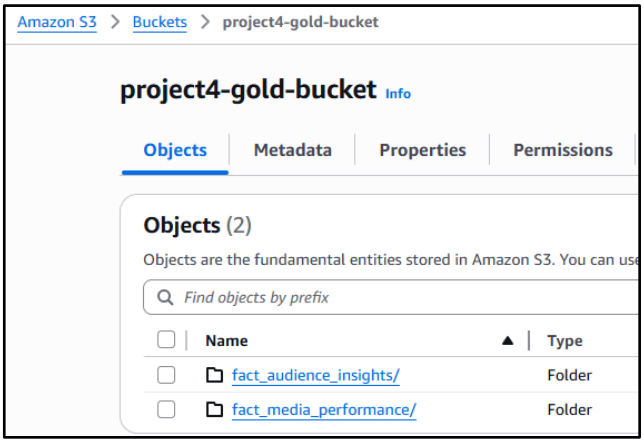
4-3. Glue ETL Job (Silver - Gold)

An AWS Glue PySpark ETL job, named **project4-silver-gold** (for fact-level aggregations and KPI calculations), was executed to produce two aggregated tables of **fact_media_performance**, for media-level metrics aggregation, and **fact_audience_insights**, for visitor-level insights and engagement behaviors.



In this Glue job, In the configuration, I considered minimal computation system. Also, I used 2 arguments, which defined them in the **Job Parameters** section.

To prepare the aggregated fact tables, I ran the **project4-silver-gold** job successfully and got the Parquet files in the S3 Gold bucket.



These tables are aligned with the business analytics questions, including but not limited to:

- Interactive tables display all media assets with key metrics using **play_count**, **hours_watched**, and **play_rate**
- Aggregated metrics per media using **play_count**, **hours_watched**, and **play_rate**
- Media content will be categorized based on engagement thresholds using **play_count** and **play_rate**
- Audience insights will be segmented by **platform** and **browser**
- Recency, Frequency (RF) analysis using **recency_days** and **frequency**

To validate the data in the Gold layer, I have downloaded all Parquet files from AWS Gold bucket into my local machine and checked them carefully.

```
(base) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-gold-bucket/fact_media_performance/ ./data/Gold/fact_media_performance --recursive --exclude "*" --include "*.parquet"
(base) PS D:\DE\Projects\Projects\Project_4> aws s3 cp s3://project4-gold-bucket/fact_audience_insights/ ./data/Gold/fact_audience_insights --recursive --exclude "*" --include "*.parquet"
```

After downloading Parquet files, in Jupyter Notebook and using Pandas library, I explored on those Parquet files and converted them into dataframes for validation. In the Gold layer, everything is OK. Here are two screenshots for two different dataframe:

fact_media_performance

```
# List all Parquet files in folder
files = glob.glob("../data/Gold/fact_media_performance/*.parquet")
display(files)

# Read and concatenate them
df_fact_media_performance = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_fact_media_performance.head(5))
df_fact_media_performance.info()
```

✓ 0.0s

```
['../data/Gold/fact_media_performance\\part-00000-8be60aaa-ff51-43e6-bd35-9966e46e7ba7-c000.snappy.parquet']
```

	media_id	load_count	play_count	play_rate	hours_watched	engagement	visitors	snapshot_ts	engagement_points	total_engagement_count	total_rewatch_count	avg_engagement_per_second	play_rate_class	hours_class
0	gskhw4w4lm	111166	16681	0.154950	336.975662	0.167007	104556	2026-01-08 16:01:48	436	1297210	57850	2975.252294	Medium	Medium
1	v08dlrgr7v	110735	43810	0.435035	2651.986970	0.500445	94643	2026-01-08 16:01:48	436	10103978	435865	23174.261468	High	High

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   media_id                  2 non-null      object
 1   load_count                2 non-null      int64
 2   play_count                2 non-null      int64
 3   play_rate                 2 non-null      float64
 4   hours_watched             2 non-null      float64
 5   engagement                2 non-null      float64
 6   visitors                  2 non-null      int64
 7   snapshot_ts               2 non-null      datetime64[ns]
 8   engagement_points         2 non-null      int64
 9   total_engagement_count    2 non-null      int64
 10  total_rewatch_count       2 non-null      int64
 11  avg_engagement_per_second 2 non-null      float64
 12  play_rate_class           2 non-null      object
 13  hours_class               2 non-null      object
dtypes: datetime64[ns](1), float64(4), int64(6), object(3)
memory usage: 356.0+ bytes
```

fact_orders_location_performance_daily

```
# List all Parquet files in folder
files = glob.glob("../data/Gold/fact_audience_insights/*.parquet")
display(files)

# Read and concatenate them
df_fact_audience_insights = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_fact_audience_insights.head())
df_fact_audience_insights.info()
```

✓ 0.1s

```
['../data/Gold/fact_audience_insights\\part-00000-0f25ada7-51dd-4fa9-8a15-675626c5214a-c000.snappy.parquet',
 '../data/Gold/fact_audience_insights\\part-00001-0f25ada7-51dd-4fa9-8a15-675626c5214a-c000.snappy.parquet',
 '../data/Gold/fact_audience_insights\\part-00002-0f25ada7-51dd-4fa9-8a15-675626c5214a-c000.snappy.parquet',
 '../data/Gold/fact_audience_insights\\part-00003-0f25ada7-51dd-4fa9-8a15-675626c5214a-c000.snappy.parquet']
```

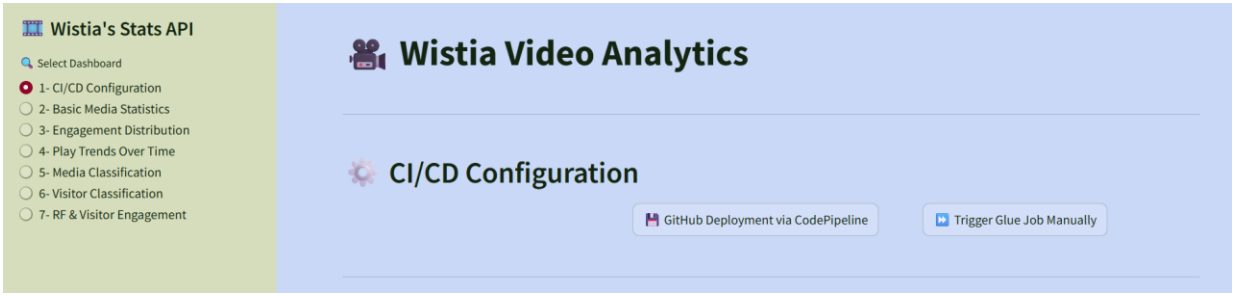
	visitor_key	created_at	last_active_at	last_event_key	load_count	play_count	identifying_event_key	name	email	org_name	org_title	browser	browser_version	platform	mobile	ingested_at	recency_days	frequency	visitor_segment
0	0023ba9_0a0fb71d-2bb1-4bca-bd22-1502509a660c-5...	2024-08-07 15:17:45	2024-08-07 15:25:10	0023ba9_c203e4a2-9102-43b5-b06f-38054eee7c18-1...	1	1	None	None	None	None	None	UIWebView	605.1.15	iOS	True	2026-01-09 22:15:53.188	520	2	Occasional
1	0023ba9_186e597d-0d35-4114-8326-c0737ab24dfc-7...	2024-08-07 15:11:44	2024-08-07 15:13:29	0023ba9_2ebc5c72-d620-4b01-a5a8-a713ca4bd5e0-7...	1	1	None	None	None	None	None	Instagram	343.0.0.23.93	iOS	True	2026-01-09 22:15:53.188	520	2	Occasional
2	0023ba9_498fb836-3d83-4c93-a5f5-bccaf5b2404e-0...	2024-08-07 15:30:41	2024-08-07 15:37:56	0023ba9_7eeb6963-7d50-497e-af5a-689b4634c43b-e...	1	1	None	None	None	None	None	Instagram	343.0.0.23.93	iOS	True	2026-01-09 22:15:53.188	520	2	Occasional
3	0023ba9_9b515b84-0465-48c3-91b7-643677ae989d-3...	2024-08-07 16:01:07	2024-08-07 16:19:50	0023ba9_3a385ed5-2210-41ae-b73b-7ebcdf4acfae-e...	1	1	None	None	None	None	None	Safari	17.5	iOS	True	2026-01-09 22:15:53.188	520	2	Occasional
4	0023ba9_c703b8fd-d24b-4d17-a633-511778646bdf-c...	2024-08-07 16:23:35	2024-08-07 16:29:00	0023ba9_5c71d85b-ca47-471c-8f6e-29a415f11f3e-d...	2	2	None	None	None	None	None	Edge	126	Windows	False	2026-01-09 22:15:53.188	520	4	Occasional

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110499 entries, 0 to 110498
Data columns (total 19 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   visitor_key           110499 non-null  object
 1   created_at            110499 non-null  datetime64[ns]
 2   last_active_at        110499 non-null  datetime64[ns]
 3   last_event_key        110499 non-null  object
 4   load_count            110499 non-null  int64
 5   play_count            110499 non-null  int64
 6   identifying_event_key 0 non-null       object
 7   name                  110499 non-null  object
 8   email                 0 non-null       object
 9   org_name              0 non-null       object
 10  org_title             0 non-null       object
 11  browser               110499 non-null  object
 12  browser_version       110499 non-null  object
 13  platform              110499 non-null  object
 14  mobile                110499 non-null  bool
 15  ingested_at           110499 non-null  datetime64[ns]
 16  recency_days          110499 non-null  int32
 17  frequency             110499 non-null  int64
 18  visitor_segment       110499 non-null  object
dtypes: bool(1), datetime64[ns](3), int32(1), int64(3), object(11)
memory usage: 14.9+ MB
```

Step 5 – Visualization (Streamlit)

5-1. Objective

In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket. I have created two **key buttons** (GitHub Deployment via CodePipeline and Trigger Glue Job Manually) inside the dashboard for manual triggering the orchestration. I used Pandas for data handling (sufficient for moderate-sized parquet files), and built interactive dashboards for stakeholders with filters, charts, and KPI cards.



As soon as we click on **GitHub Deployment via CodePipeline** button in the Streamlit dashboard (maually), or any small change in the GitHub repository, the CodePipeline automatically gets the update from GitHub repo and shares it inside the S3 code bucket. Then, Python and PySpark scripts are available for Glue ETL jobs immedietely.

Also, as soon as we click on **Trigger Glue Job Manually** button in the Streamlit dashboard (manually), the first Glue Job will be triggered, and all Glue jobs get their scripts from the S3 code bucket and start one after one sequentially and successfully.

Step 6 – Jobs Orchestration and CI/CD

6-1. Objective

The objective of the CI/CD approach is to automate the end-to-end batch data pipeline on a daily schedule while ensuring reliable orchestration, clear job dependencies, and maintainable CI/CD practices. All data is ingested from Wistia API into AWS, transformed through Bronze, Silver, and Gold layers, and made available for analytics and visualization without manual intervention. To support continuous development and deployment, all AWS Glue scripts are version-controlled in a GitHub repository and automatically deployed to AWS when changes occur.

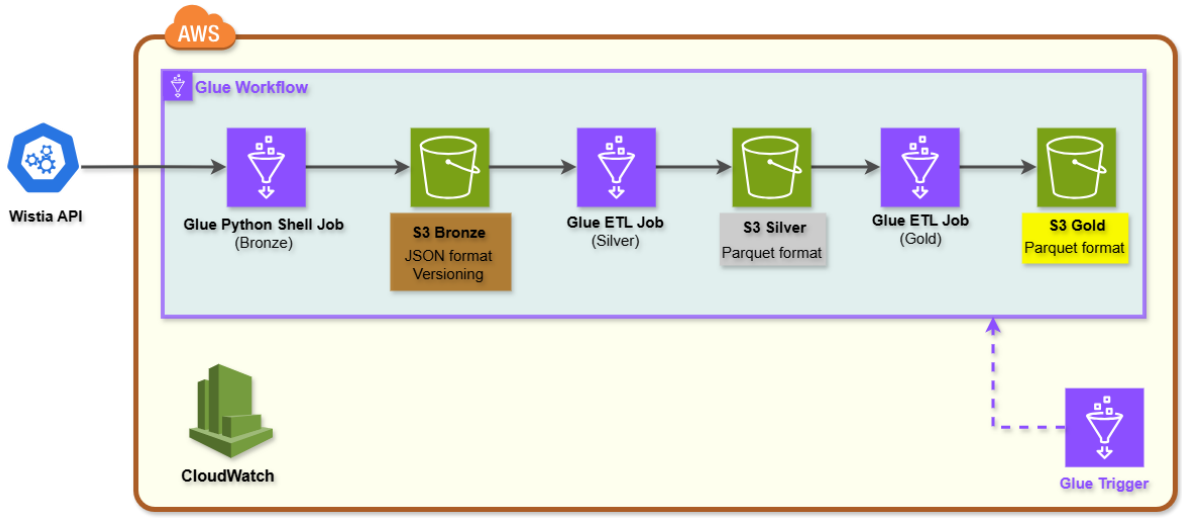
The scheduling and orchestration pipelines are:

- A daily schedule glue jobs using **AWS Glue Trigger**.
- The **Glue Workflow** acts as the central orchestration engine for the pipeline.
- The Glue Workflow executes jobs in a dependency-based sequence:

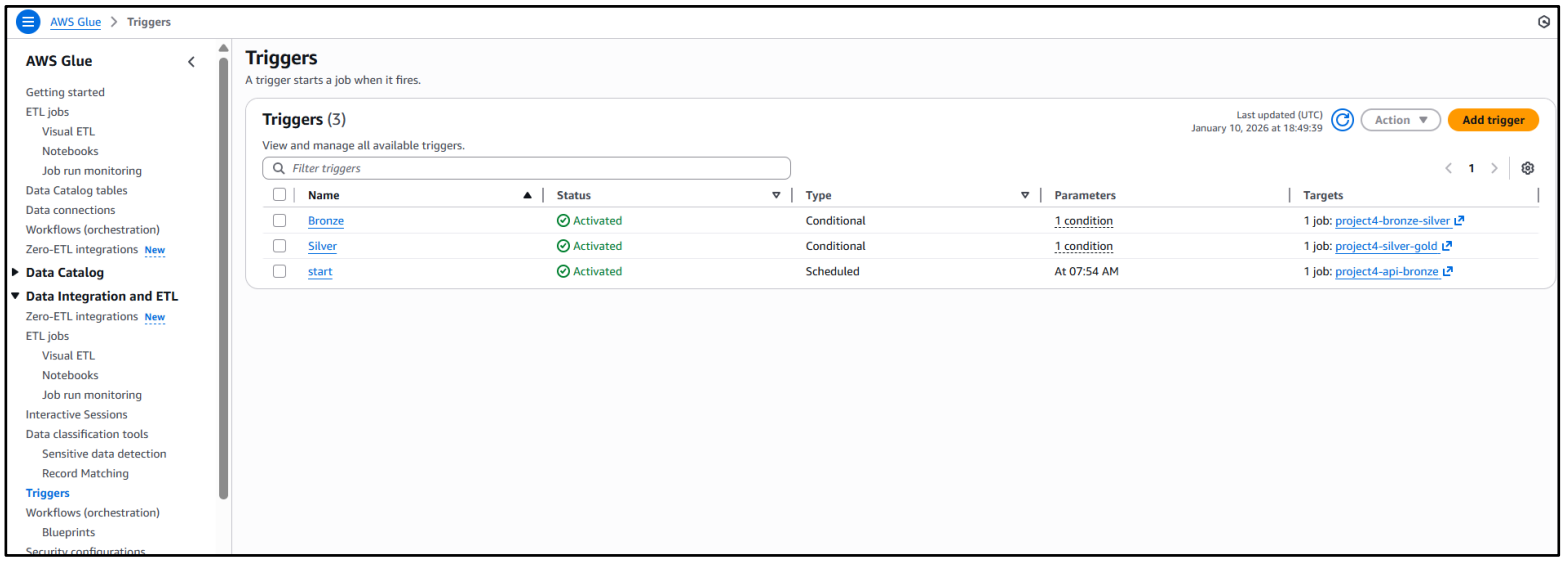
6-2. Orchestration

For occhestraion of Glue ETL jobs, I used Glue Workflow, named **project4-api-bronze-silver-gold**, which acts as the central orchestration engine for the pipeline. This Glow Workflow is triggered using **Glue Trigger**. Glue Trigger is configured to trigger the Glue Workflow daily. The Glue Workflow executes jobs in a dependency-based sequence:

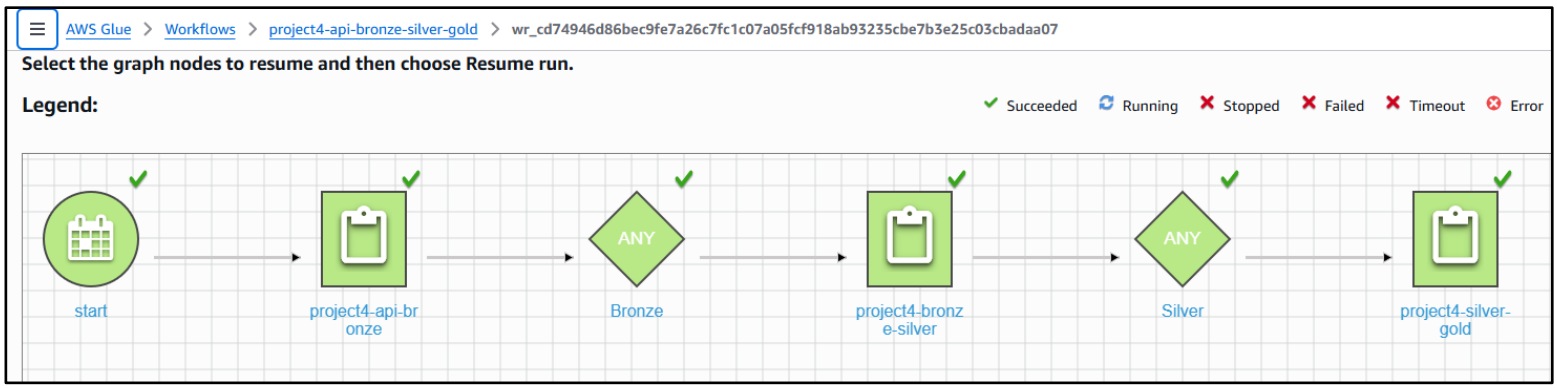
Glue Python Shell Job (Data Ingestion) → Glue Silver ETL Job → Gold Glue Jobs



As for triggering the Glue jobs, I created a Glue Trigger, named **start**, which triggers the first Glue Python Shell job at 7:54 am (UTC), dayily. As the Glue Python Shell is my first Glue job inside the Glow Workflow, then Glue Workflow will orchestrate all the rest of the jobs after succeeding in the first Glue job.



As you can see in picture above, there are three triggers, one of them is scheduled based trigger for the first Glue job, and the others are in the condition-based for the rest of Glue jobs. This is the orchestration steps of Glue Workflow:



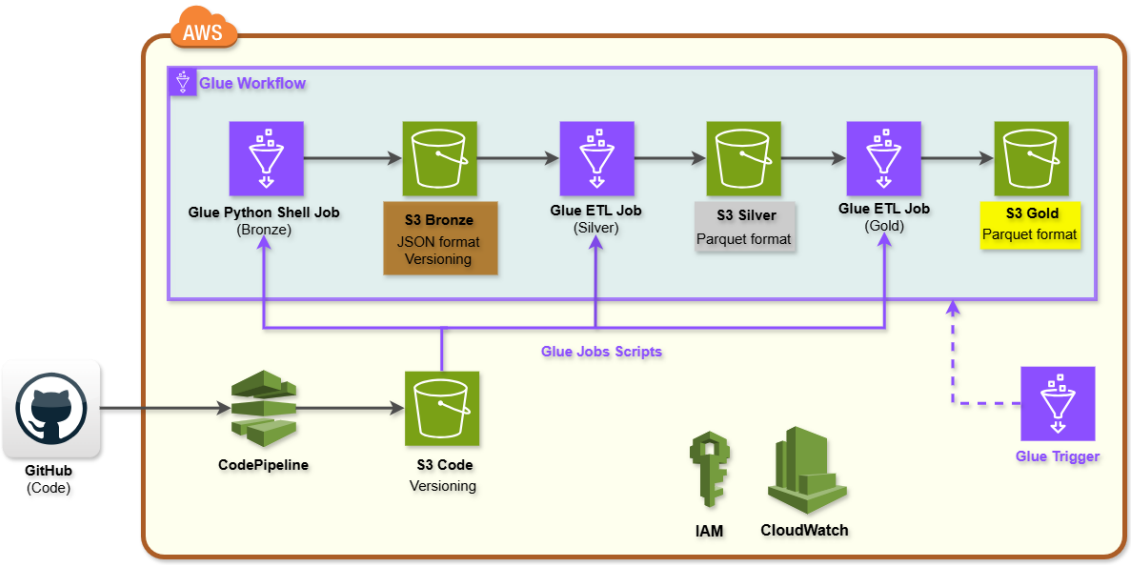
6-3. CI/CD Strategy

This approach enables automated deployment, traceability, and reproducibility of data pipelines. All Python and PySpark scripts for Glue jobs are stored in a **GitHub** repository: https://github.com/Hadi2468/ELT_Project4/tree/main/Glue_codes .

Benefits of this orchestration strategy

This design ensures:

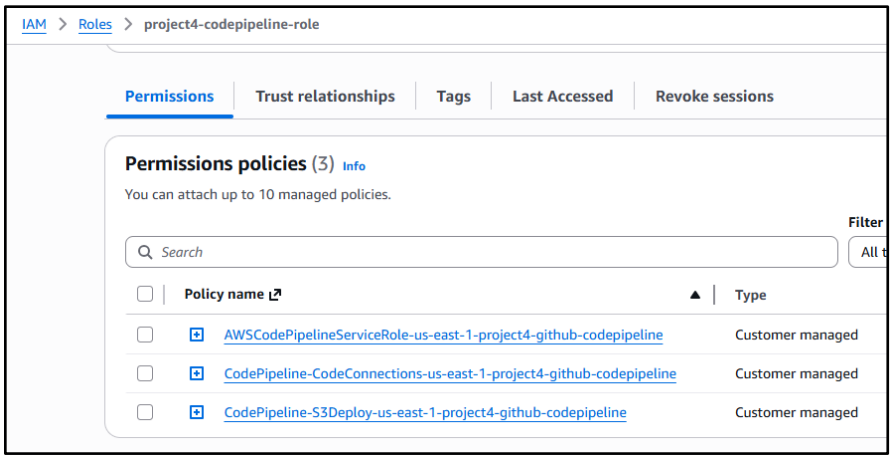
- Clear and deterministic job dependencies
- Controlled execution order
- Parallel processing where appropriate
- Failure isolation and observability
- Minimal operational complexity
- Scalable and production-ready CI/CD practices



6-4. IAM Role

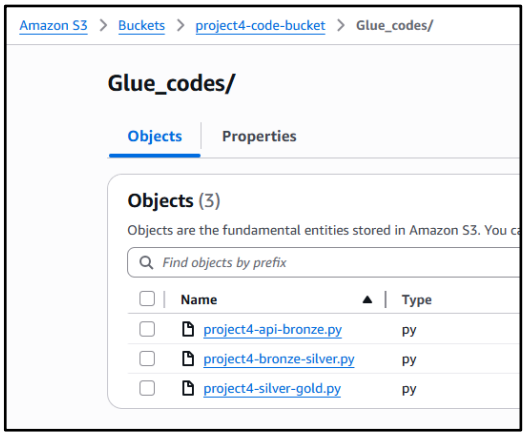
During setting up the AWS CodePipeline, it asked for an IAM role, and I configured a new role, named **project4-codepipeline-role** and tagged with **project: 4**. This role grants CodePipeline the required permissions to securely connect to the GitHub repository, retrieve the Pyton and PySpark source codes, and deploy the artifacts to Amazon S3. The following permissions were configured:

- **AWSCodePipelineServiceRole**: Grants CodePipeline core permissions to orchestrate pipeline stages and interact with AWS services on behalf of the pipeline.
- **CodePipeline-CodeConnections**: Allows CodePipeline to establish and manage secure connections to external source providers such as GitHub using AWS CodeStar Connections.
- **CodePipeline-S3Deploy**: Provides permission for CodePipeline to upload, read, and manage build artifacts in Amazon S3 buckets used during the deployment process.



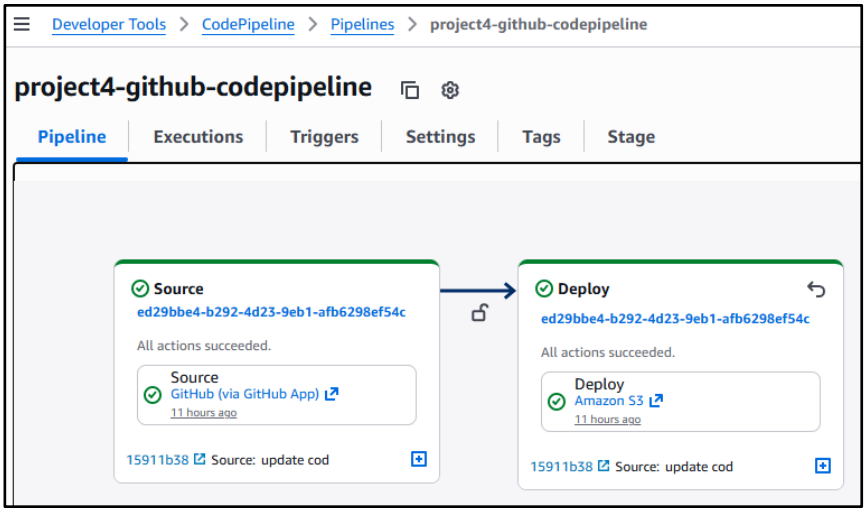
6-5. S3 Code Bucket

The CI/CD pipeline automatically detects changes or additions to Glue job scripts in the GitHub repo and immediately deploys updated scripts to another S3 (code) bucket, named **s3://project4-code-bucket/** using AWS CodePipeline service. All Glue jobs reference their scripts directly from the S3 code bucket, ensuring version consistency, easy rollback, and no manual code uploads. I have enabled its **versioning** option to store all versions of codes to support auditability, reprocessing, and downstream transformations.



6-6. CodePipeline

Then, I finished setting up the AWS CodePipeline and named it **project4-github-codepipeline** and tagged with **project: 4**.



6-7. Monitoring and Failure Handling

To monitoring and failure handling, each Glue job writes logs (Output or Error) to **Amazon CloudWatch**. Then, in case of any job failure:

- The Glue Workflow stops downstream execution
- Error details are available in CloudWatch Logs for troubleshooting

Also, we can follow all logging messages that I created in each block of the code, like Start states, Success, or Exceptions.

Conclusion

This project successfully demonstrates the end-to-end design of a scalable, production-ready data analytics pipeline for Wistia video engagement data, through the Wistia Stats API: media-level, media engagement–level, and visitor-level datasets.

The proposed architecture follows modern data lake best practices by implementing a multi-layered Bronze–Silver–Gold design on Amazon S3. The Bronze layer preserves raw, immutable JSON data for auditability and reprocessing, the Silver layer standardizes and normalizes the data into analytics-ready Parquet formats, and the Gold layer delivers curated fact tables and KPIs optimized for business intelligence, reporting, and advanced analytics. This layered approach ensures data quality, scalability, and long-term maintainability.

AWS Glue was intentionally selected as the core orchestration and transformation engine to provide consistency across ingestion and transformation stages, native scheduling, dependency management, and seamless integration with AWS IAM, Secrets Manager, and CloudWatch. The use of Glue Workflows enables deterministic execution order, clear failure handling, and operational observability, while avoiding unnecessary architectural complexity. Incremental ingestion logic and state management further enhance efficiency by minimizing redundant data processing.

The pipeline is fully automated and extensible, supporting manual execution, scheduled daily runs, and CI/CD-driven deployments through GitHub and AWS CodePipeline. This ensures that changes to ingestion or transformation logic can be deployed safely, reproducibly, and with minimal operational overhead. The final Streamlit-based visualization layer provides a lightweight, serverless interface for stakeholders to explore engagement metrics and audience insights directly from the Gold layer.

Overall, this proposal delivers a robust, cost-efficient, and enterprise-aligned analytics solution that transforms raw Wistia API data into actionable insights. The design is intentionally modular, secure, and scalable, making it well-suited for future extensions such as additional media sources, longer retention periods, advanced audience segmentation, and machine learning–based engagement analysis.