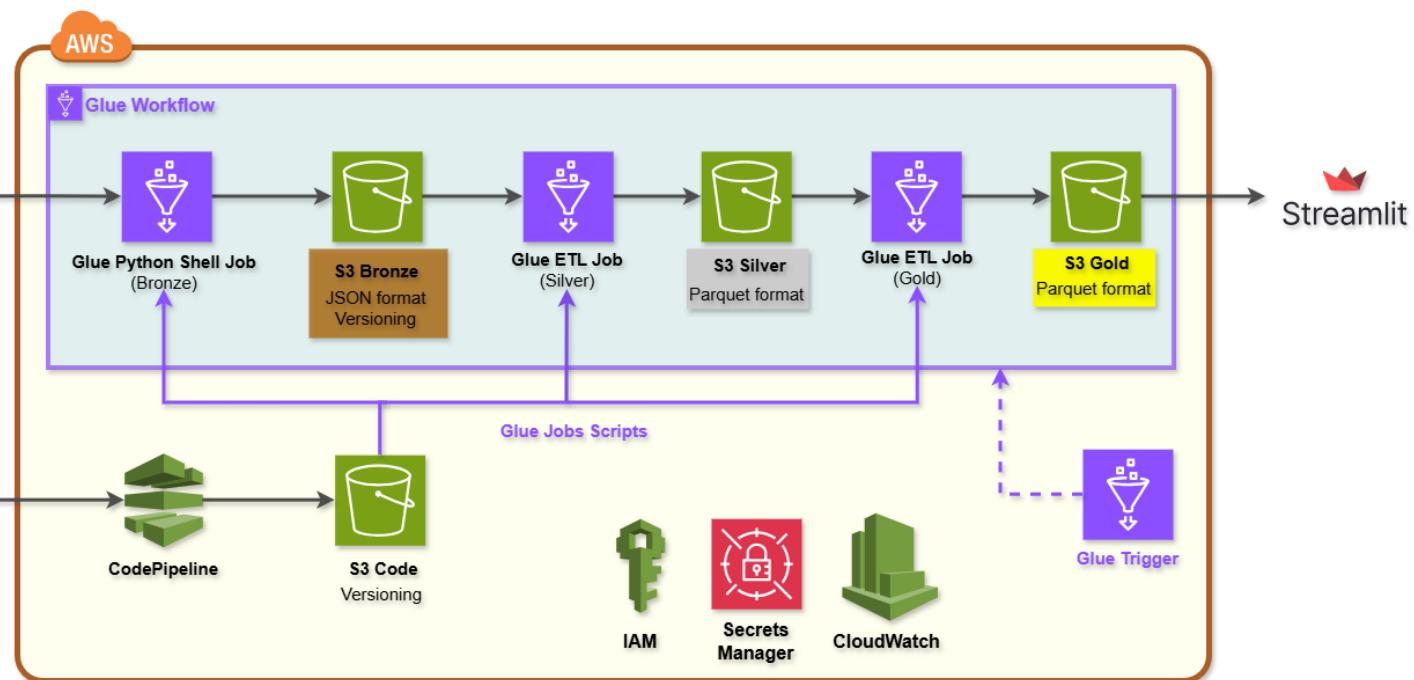


Project 4: Wistia Video Analytics

Overview

In this project, we will work with JSON format data, and the goal is to pull them from the Wistia Portal. Before, Designing 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 analysis of the the Wistia API data files, I have prepared the proposal and got the approval from Jay. Now, here is the data model pipeline, CI/CD operation and orchestration 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.

The screenshot shows the Wistia API documentation for the Stats API. On the left, a sidebar lists endpoints under 'STATS API': Stats:Account, Stats:Projects, Stats:Media (with sub-options: Stats:Media Show, Stats:Media By Date, Stats:Media Engagement), Stats:Visitors (with sub-options: Stats:Visitors List, Stats:Visitors Show). Each endpoint has a 'GET' button next to it. The main content area features a large title 'Getting Started with the Stats API' and a sub-section 'Programmatically access your video metrics, viewer activity, and more.' Below this, there's a text box with a question about video recommendations and another about visitor tracking. At the bottom, a note says 'You'll need an API token to use the Stats API. Head on over to [Making API Requests](#) for instructions on'.

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**.

The screenshot shows the 'Stats:Media Show' endpoint details. It includes the method (GET), URL (<https://api.wistia.com/v1/stats/medias/{mediaId}>), a description ('Retrieve stats for a video. This endpoint provides statistics for a specific video identified by its media-id.'), and a note 'Requires api token with one of the following scopes:'. On the right, there's a 'CREDENTIALS' section with a 'BEARER' input field containing a placeholder token '0323ade64e13f79821bcd0f2a9410d9ec3873aa9df01f8a4'. A red box highlights this field.

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 project description, I have to focus on only two media with mediaId of “**v08dlgr7v**” 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 consider the **Python language** and click on the **Try It!** button, as below image.

The screenshot shows the Wistia API documentation for the `get_stats-medias-mediaid` endpoint. The left sidebar lists various API endpoints under categories like `STATS API`, `UPLOAD API`, and `LIVE STREAM API`. The main content area is titled **Stats:Media Show** and describes retrieving stats for a video. It requires an API token with permissions to read, update, and delete anything, as well as read all data and all folder/media data. Below this, there's a **Recent Requests** table showing three recent successful requests. The **Path Params** section shows a required `mediaId` parameter with the value `v08dlgr7v` highlighted by a red box. The **REQUEST** section contains a code snippet for Python using the `requests` library to make a GET request to the endpoint with the provided mediaId. The **Try It!** button is also highlighted with a red box. The **RESPONSE** section shows the JSON data returned from the API call, which includes metrics like load count, play count, play rate, hours watched, engagement, and visitors.

```
$ python -m pip install requests
import requests
url = "https://api.wistia.com/v1/stats/medias/v08dlgr7v"
headers = {
    "accept": "application/json",
    "authorization": "Bearer 0323ade64e13f79821bdc0f2a9410d9ec3873aa9df01f8a4"
}
response = requests.get(url, headers=headers)
print(response.text)
```

```
{ "load_count": 110734, "play_count": 43810, "play_rate": 0.43503941167769067, "hours_watched": 2651.986969632, "engagement": 0.500445, "visitors": 94642 }
```

For example, the `mediaId` of “**v08dlgr7v**”, 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.

Two screenshots of the API response for different media IDs. The left screenshot shows the response for `mediaId: v08dlgr7v`, which has a red box around its mediaId parameter. The right screenshot shows the response for `mediaId: gskhw4w4lm`, also with a red box around its mediaId parameter. Both responses are 200 OK. The **RESPONSE** section for each shows detailed statistics including load count, play count, play rate, hours watched, engagement, and visitors.

Response for mediaId v08dlgr7v:

```
{ "load_count": 110734, "play_count": 43810, "play_rate": 0.43503941167769067, "hours_watched": 2651.986969632, "engagement": 0.500445, "visitors": 94642 }
```

Response for mediaId gskhw4w4lm:

```
{ "load_count": 111165, "play_count": 16681, "play_rate": 0.15495193917077135, "hours_watched": 336.97566205632, "engagement": 0.167007, "visitors": 104555 }
```

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

Stats:Media >
Stats:Visitors >
Stats:Visitors List **GET**
Stats:Visitors Show **GET**
Stats:Events >

UPLOAD API
Getting Started with the Upload API
Update thumbnails via Upload API
Using the Upload API with Dropbox
Examples Using Ruby and Ajax

Query Params

page integer
The page of results based on the per_page parameter.

per_page integer
The maximum number of results to return, capped at 100.

filter string enum
Filtering parameter to narrow down the list of visitors.
Allowed: **has_name** **has_email** **identified_by_email_gate**

search string
Search for visitors based on name or email address.

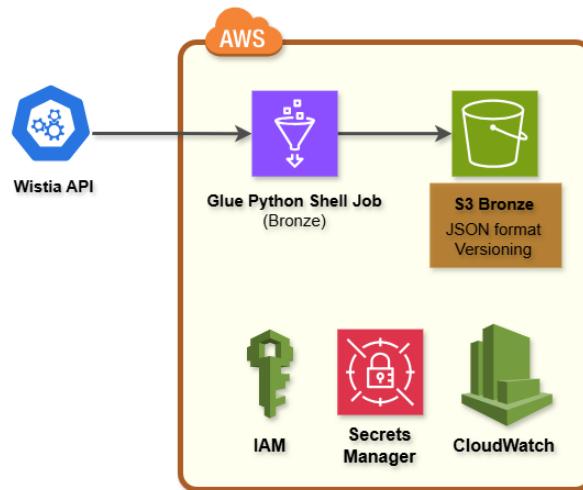
This is information on one of those **110361 visitors**:

```
[  
 {  
   "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)

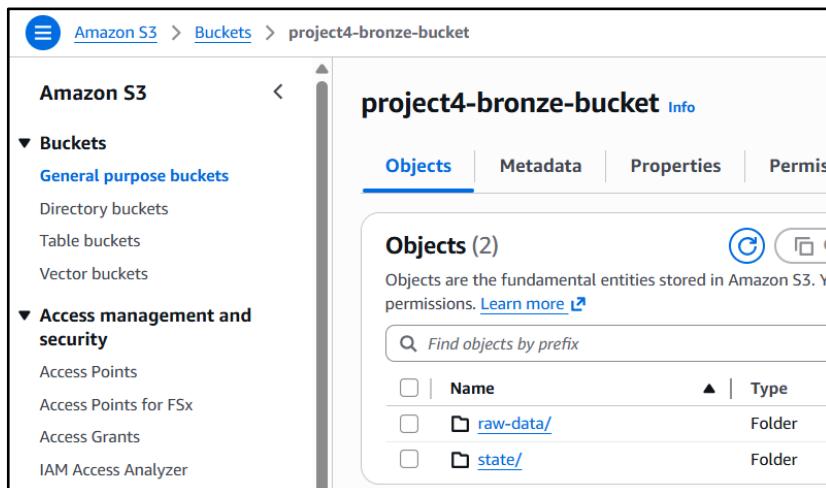
2-1. Objective

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 bucket 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:

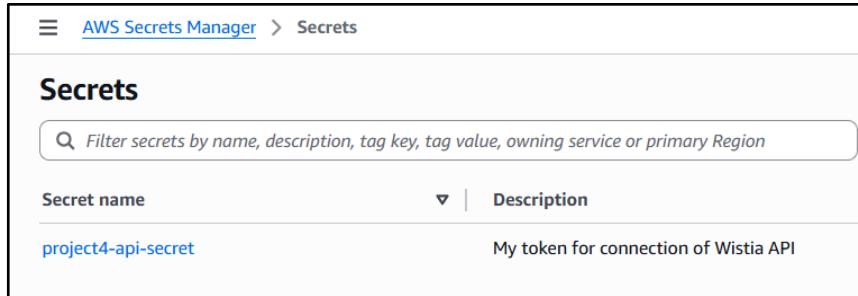
- **AWSGlueConsoleFullAccess:** Attached to my IAM user to allow me to create and manage AWS Glue resources, such as ETL jobs, and workflows, through the AWS Management Console.
- **AWSGlueServiceRole:** Attached to the Glue job runtime role, allowing AWS Glue to assume the role at execution time and 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.

The screenshot shows the AWS IAM Roles page. The left sidebar has a search bar and links for Dashboard, Access Management (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management, Temporary delegation requests), and Access reports (Access Analyzer, Resource analysis, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies, Resource control policies). The main area shows the role 'project4-glue-role' with a summary: Creation date (January 08, 2026, 12:34 (UTC-08:00)), Last activity (10 minutes ago). Below is a table of permissions policies:

Policy name	Type
AmazonS3FullAccess	AWS managed
AWSGlueConsoleFullAccess	AWS managed
AWSGlueServiceRole	AWS managed
CloudWatchLogsFullAccess	AWS managed
SecretsManagerReadWrite	AWS managed

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.

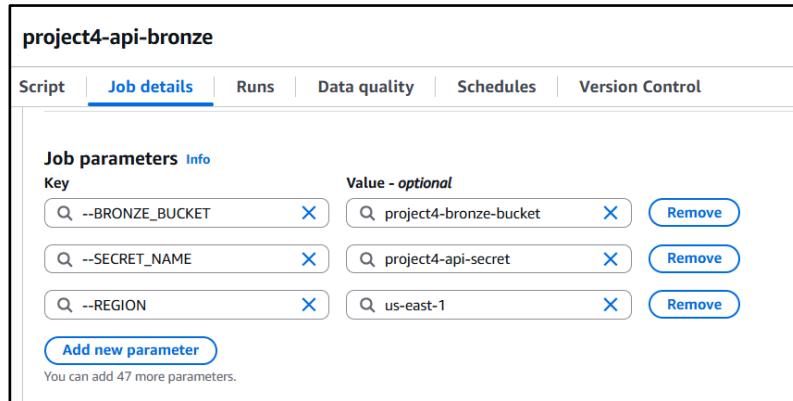


The screenshot shows the AWS Secrets Manager interface. At the top, there's a breadcrumb navigation: AWS Secrets Manager > Secrets. Below that is a search bar with the placeholder "Filter secrets by name, description, tag key, tag value, owning service or primary Region". Underneath is a table with two columns: "Secret name" and "Description". A single row is visible, showing "project4-api-secret" in the first column and "My token for connection of Wistia API" in the second column.

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.



The screenshot shows the "Job details" tab of the AWS Glue job configuration for "project4-api-bronze". Under the "Job parameters" section, there are three parameter pairs listed:

Key	Value - optional
--BRONZE_BUCKET	project4-bronze-bucket
--SECRET_NAME	project4-api-secret
--REGION	us-east-1

Below the table, there are buttons for "Add new parameter" and a note: "You can add 47 more parameters."

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 around 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 folders 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.

The image contains three separate screenshots of the Amazon S3 console, each showing the contents of a specific folder:

- media/**: This folder contains four objects, both of which are JSON files named `media_gskhw4w4lm_20260108_160148.json` and `media_v08dlrgr7v_20260108_160148.json`. Both files are of type json and were last modified on January 8, 2026, at 16:01:49 (UTC-08:00).
- media_engagement/**: This folder contains four objects, both of which are JSON files named `media_engagement_gskhw4w4lm_20260108_160148.json` and `media_engagement_v08dlrgr7v_20260108_160148.json`. Both files are of type json and were last modified on January 8, 2026, at 16:01:49 (UTC-08:00).
- visitors/**: This folder contains one object, a JSON file named `visitors_20260108_170607_batch_110471.json`. The file is of type json and was last modified on January 8, 2026, at 17:06:11 (UTC-08:00).

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 until 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](#)

Filter events - press enter to search

Actions ▾ Start tailing Create metric filter

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: ----
2026-01-08T16:01:48.528-08:00	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_v08dlgrn7v_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_v08dlgrn7v_20260108_160148.json 2026-01-09 00:01:49,027 INFO proj...
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

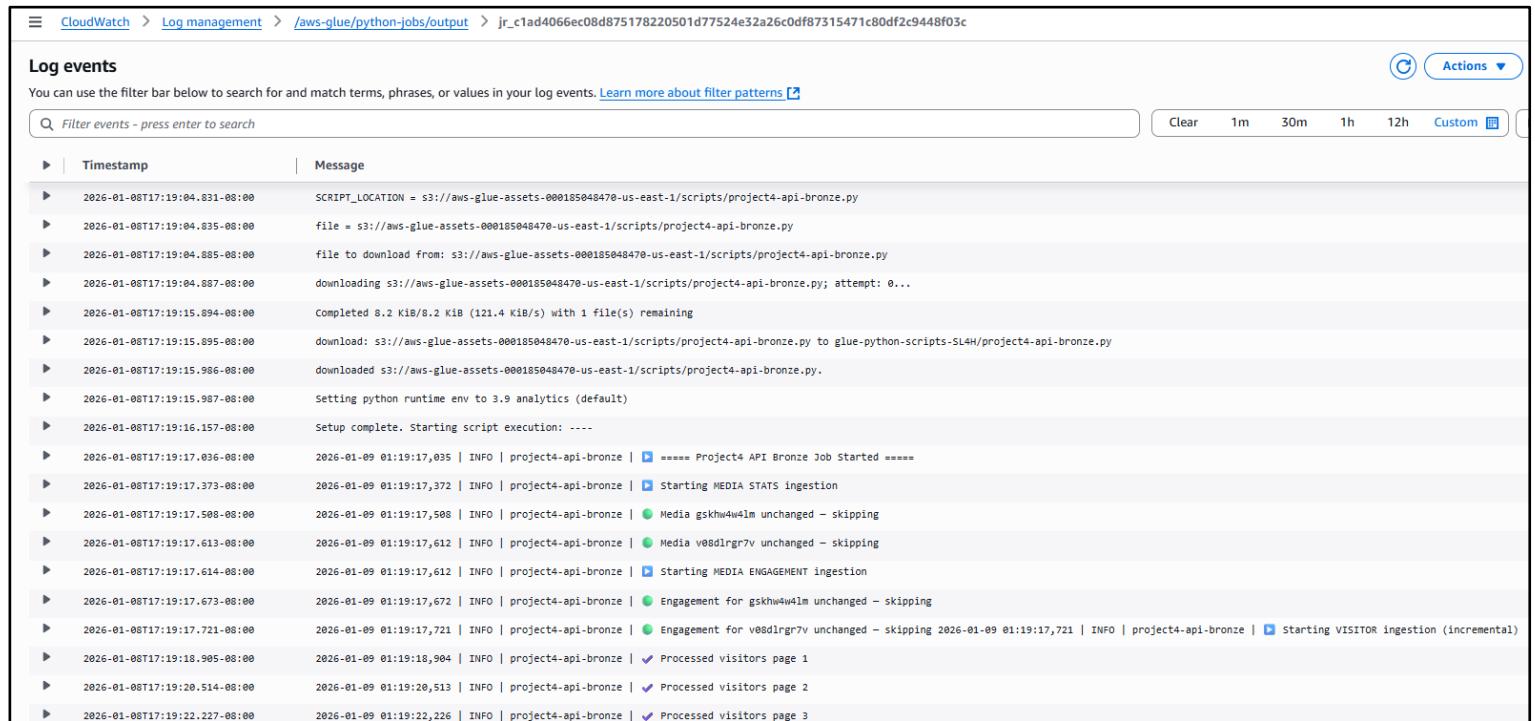
Timestamp	Message
2026-01-08T17:04:44.886-08:00	2026-01-09 01:04:44,885 INFO project4-api-bronze ✓ Processed visitors page 1090
2026-01-08T17:04:50.321-08:00	2026-01-09 01:04:50,321 INFO project4-api-bronze ✓ Processed visitors page 1091
2026-01-08T17:04:55.815-08:00	2026-01-09 01:04:55,815 INFO project4-api-bronze ✓ 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.



The screenshot shows the AWS CloudWatch Log Management interface. The top navigation bar includes 'CloudWatch' > 'Log management' > '/aws-glue/python-jobs/output' > 'jr_c1ad4066ec08d875178220501d77524e32a26c0df87315471c80df2c9448f03c'. Below the navigation is a search bar with placeholder 'Filter events - press enter to search' and a 'Actions' dropdown. A table lists log events with columns for 'Timestamp' and 'Message'. The log entries detail the execution of the script, including file downloads, processing steps like 'Starting MEDIA STATS ingestion', and skipping media items. The final message indicates the start of 'VISITOR ingestion (incremental)'.

Timestamp	Message
2026-01-08T17:19:04.831-08:00	SCRIPT_LOCATION = s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py
2026-01-08T17:19:04.835-08:00	file = s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py
2026-01-08T17:19:04.885-08:00	file to download from: s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py
2026-01-08T17:19:04.887-08:00	downloading s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py; attempt: 0...
2026-01-08T17:19:15.894-08:00	Completed 8.2 KiB/8.2 KiB (121.4 KiB/s) with 1 file(s) remaining
2026-01-08T17:19:15.895-08:00	download: s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py to glue-python-scripts-SL4H/project4-api-bronze.py
2026-01-08T17:19:15.986-08:00	downloaded s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py.
2026-01-08T17:19:15.987-08:00	Setting python runtime env to 3.9 analytics (default)
2026-01-08T17:19:16.157-08:00	Setup complete. Starting script execution: ...
2026-01-08T17:19:17.036-08:00	2026-01-09 01:19:17,035 INFO project4-api-bronze ===== Project4 API Bronze Job Started =====
2026-01-08T17:19:17.373-08:00	2026-01-09 01:19:17,372 INFO project4-api-bronze Starting MEDIA STATS ingestion
2026-01-08T17:19:17.508-08:00	2026-01-09 01:19:17,508 INFO project4-api-bronze Media gskhw4w4lm unchanged - skipping
2026-01-08T17:19:17.613-08:00	2026-01-09 01:19:17,612 INFO project4-api-bronze Media v08dlrgr7v unchanged - skipping
2026-01-08T17:19:17.614-08:00	2026-01-09 01:19:17,612 INFO project4-api-bronze Starting MEDIA ENGAGEMENT ingestion
2026-01-08T17:19:17.673-08:00	2026-01-09 01:19:17,672 INFO project4-api-bronze Engagement for gskhw4w4lm unchanged - skipping
2026-01-08T17:19:17.721-08:00	2026-01-09 01:19:17,721 INFO project4-api-bronze Engagement for v08dlrgr7v unchanged - skipping 2026-01-09 01:19:17,721 INFO project4-api-bronze Starting VISITOR ingestion (incremental)
2026-01-08T17:19:18.905-08:00	2026-01-09 01:19:18,904 INFO project4-api-bronze Processed visitors page 1
2026-01-08T17:19:20.514-08:00	2026-01-09 01:19:20,513 INFO project4-api-bronze Processed visitors page 2
2026-01-08T17:19:22.227-08:00	2026-01-09 01:19:22,226 INFO project4-api-bronze Processed visitors page 3

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

▶	Timestamp	Message
▶	2026-01-08T18:18:18.18732,995-08:00	2026-01-09 02:18:18,18732,995 INFO project4-api-bronze ✓ Processed visitors page 1098
▶	2026-01-08T18:18:38.140-08:00	2026-01-09 02:18:38,140 INFO project4-api-bronze ✓ Processed visitors page 1091
▶	2026-01-08T18:18:44.030-08:00	2026-01-09 02:18:44,029 INFO project4-api-bronze ✓ Processed visitors page 1092
▶	2026-01-08T18:18:50.516-08:00	2026-01-09 02:18:50,516 INFO project4-api-bronze ✓ Processed visitors page 1093
▶	2026-01-08T18:18:56.458-08:00	2026-01-09 02:18:56,457 INFO project4-api-bronze ✓ Processed visitors page 1094
▶	2026-01-08T18:19:02.898-08:00	2026-01-09 02:19:02,897 INFO project4-api-bronze ✓ Processed visitors page 1095
▶	2026-01-08T18:19:07.805-08:00	2026-01-09 02:19:07,805 INFO project4-api-bronze ✓ Processed visitors page 1096
▶	2026-01-08T18:19:13.232-08:00	2026-01-09 02:19:13,231 INFO project4-api-bronze ✓ Processed visitors page 1097
▶	2026-01-08T18:19:18.649-08:00	2026-01-09 02:19:18,649 INFO project4-api-bronze ✓ Processed visitors page 1098
▶	2026-01-08T18:19:24.061-08:00	2026-01-09 02:19:24,061 INFO project4-api-bronze ✓ Processed visitors page 1099
▶	2026-01-08T18:19:28.893-08:00	2026-01-09 02:19:28,893 INFO project4-api-bronze ✓ Processed visitors page 1100
▶	2026-01-08T18:19:34.063-08:00	2026-01-09 02:19:34,062 INFO project4-api-bronze ✓ Processed visitors page 1101
▶	2026-01-08T18:19:39.046-08:00	2026-01-09 02:19:39,045 INFO project4-api-bronze ✓ Processed visitors page 1102
▶	2026-01-08T18:19:44.442-08:00	2026-01-09 02:19:44,441 INFO project4-api-bronze ✓ Processed visitors page 1103
▶	2026-01-08T18:19:49.944-08:00	2026-01-09 02:19:49,944 INFO project4-api-bronze ✓ Processed visitors page 1104
▶	2026-01-08T18:19:54.396-08:00	2026-01-09 02:19:54,395 INFO project4-api-bronze ✅ Saved 4 new visitors to raw-data/visitors/visitors_20260108_181954_batch_4.json
▶	2026-01-08T18:19:54.639-08:00	2026-01-09 02:19:54,638 INFO project4-api-bronze ✅ State successfully updated
▶	2026-01-08T18:19:54.639-08:00	2026-01-09 02:19:54,638 INFO project4-api-bronze ★★★ ===== Project4 API Bronze Job Completed Successfully =====

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!

Log events		Actions	Start tailing
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	1m 30m 1h 12h Custom Local timezone
Timestamp			Message
<pre>> 2026-01-09T10:27:20.636-08:00 file to download from: s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py > 2026-01-09T10:27:20.638-08:00 downloading s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py; attempt: 0... > 2026-01-09T10:27:27.277-08:00 Completed 8.2 kB/8.2 kB (123.0 kB/s) with 1 file(s) remaining > 2026-01-09T10:27:27.278-08:00 download: s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py to glue-python-scripts-z9Dw/project4-api-bronze.py > 2026-01-09T10:27:27.355-08:00 downloaded s3://aws-glue-assets-000185048470-us-east-1/scripts/project4-api-bronze.py > 2026-01-09T10:27:27.356-08:00 Setting python runtime env to 3.9 analytics (default) > 2026-01-09T10:27:27.485-08:00 Setup complete. Starting script execution: ---- > 2026-01-09T10:27:28.098-08:00 2026-01-09 18:27:28,098 INFO project4-api-bronze ===== Project4 API Bronze Job Started ===== > 2026-01-09T10:27:28.515-08:00 2026-01-09 18:27:28,515 INFO project4-api-bronze ⚡ Starting MEDIA STATS ingestion > 2026-01-09T10:27:28.642-08:00 2026-01-09 18:27:28,642 INFO project4-api-bronze 🟢 Media gsckhw4w4lm unchanged - skipping > 2026-01-09T10:27:28.766-08:00 2026-01-09 18:27:28,765 INFO project4-api-bronze 🟢 Media v08d1ngr7v unchanged - skipping 2026-01-09 18:27:28,765 INFO project4-api-bronze ⚡ Starting MEDIA ENGAGEMENT ingestion > 2026-01-09T10:27:28.829-08:00 2026-01-09 18:27:28,829 INFO project4-api-bronze 🟢 Engagement for gsckhw4w4lm unchanged - skipping > 2026-01-09T10:27:28.883-08:00 2026-01-09 18:27:28,882 INFO project4-api-bronze 🟢 Engagement for v08d1ngr7v unchanged - skipping 2026-01-09 18:27:28,882 INFO project4-api-bronze ⚡ Starting VISITOR ingestion (incremental) > 2026-01-09T10:27:30.982-08:00 2026-01-09 18:27:30,981 INFO project4-api-bronze ✅ Processed visitors page 1 > 2026-01-09T10:27:32.648-08:00 2026-01-09 18:27:32,648 INFO project4-api-bronze ✅ Processed visitors page 2 > 2026-01-09T10:27:34.503-08:00 2026-01-09 18:27:34,502 INFO project4-api-bronze ✅ Processed visitors page 3 > 2026-01-09T10:27:36.767-08:00 2026-01-09 18:27:36,767 INFO project4-api-bronze ✅ Processed visitors page 4 </pre>			

Log events		Actions	Start tailing
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	1m 30m 1h 12h Custom Local timezone
Timestamp			Message
<pre>> 2026-01-09T11:29:34.488-08:00 2026-01-09 19:29:34,487 INFO project4-api-bronze ✅ Processed visitors page 1092 > 2026-01-09T11:29:40.446-08:00 2026-01-09 19:29:40,446 INFO project4-api-bronze ✅ Processed visitors page 1093 > 2026-01-09T11:29:46.702-08:00 2026-01-09 19:29:46,702 INFO project4-api-bronze ✅ Processed visitors page 1094 > 2026-01-09T11:29:52.824-08:00 2026-01-09 19:29:52,823 INFO project4-api-bronze ✅ Processed visitors page 1095 > 2026-01-09T11:29:58.499-08:00 2026-01-09 19:29:58,499 INFO project4-api-bronze ✅ Processed visitors page 1096 > 2026-01-09T11:30:04.111-08:00 2026-01-09 19:30:04,108 INFO project4-api-bronze ✅ Processed visitors page 1097 > 2026-01-09T11:30:09.543-08:00 2026-01-09 19:30:09,542 INFO project4-api-bronze ✅ Processed visitors page 1098 > 2026-01-09T11:30:14.887-08:00 2026-01-09 19:30:14,886 INFO project4-api-bronze ✅ Processed visitors page 1099 > 2026-01-09T11:30:20.358-08:00 2026-01-09 19:30:20,358 INFO project4-api-bronze ✅ Processed visitors page 1100 > 2026-01-09T11:30:25.759-08:00 2026-01-09 19:30:25,758 INFO project4-api-bronze ✅ Processed visitors page 1101 > 2026-01-09T11:30:30.471-08:00 2026-01-09 19:30:30,471 INFO project4-api-bronze ✅ Processed visitors page 1102 > 2026-01-09T11:30:35.526-08:00 2026-01-09 19:30:35,526 INFO project4-api-bronze ✅ Processed visitors page 1103 > 2026-01-09T11:30:40.351-08:00 2026-01-09 19:30:40,351 INFO project4-api-bronze ✅ Processed visitors page 1104 > 2026-01-09T11:30:45.395-08:00 2026-01-09 19:30:45,393 INFO project4-api-bronze ✅ Processed visitors page 1105 > 2026-01-09T11:30:47.690-08:00 2026-01-09 19:30:47,689 INFO project4-api-bronze ✅ Saved 24 new visitors to raw-data/visitors/visitors_20260109_113047_batch_24.json > 2026-01-09T11:30:47.996-08:00 2026-01-09 19:30:47,995 INFO project4-api-bronze ✅ State successfully updated 2026-01-09 19:30:47,995 INFO project4-api-bronze ★★★ ===== Project4 API Bronze Job Completed Successfully ===== </pre>			

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):

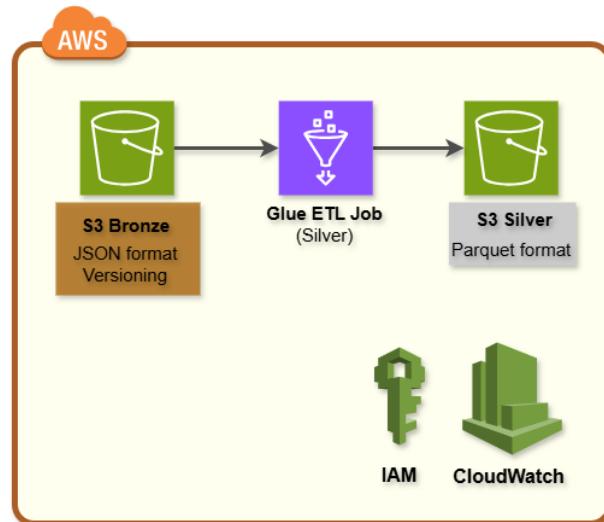
value			
load_count	111166.000000		
play_count	16681.000000		
play_rate	0.154950		
hours_watched	336.975662		
engagement	0.167007		
visitors	104556.000000		
engagement_data			
0	16108	1542	0.167007
1	9796	663	0.167007
2	9394	629	0.167007
3	8996	570	0.167007
4	8649	542	0.167007
...
431	1790	53	0.167007
432	1785	52	0.167007
433	1780	52	0.167007
434	1754	48	0.167007
435	2060	133	0.167007

	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-a191-3d669be414c-8...	2026-01-09 01:03:22+00:00	2026-01-09 01:02:50+00:00	1767920_e2c0b383-bebf-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-f9c9-42f5-af64-e5f280e616b7-9...	5	1	NaN	{"name": "", "email": None, "org": "", "name": No...}	{"browser": "Chrome", "browser_version": "143..."}
2	1767919_27c1c780-d5bf-4fb4-bcf1-34d864e6777c-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-6ad1fc6659d-2...	2026-01-09 00:15:39+00:00	2026-01-09 00:15:27+00:00	1767917_9d603948-2674-45d6-9fae-fb540e19ae57-8...	1	1	NaN	{"name": "", "email": None, "org": "", "name": No...}	{"browser": "Chrome", "browser_version": "143..."}

Step 3 – Transformation (S3 Silver Layer)

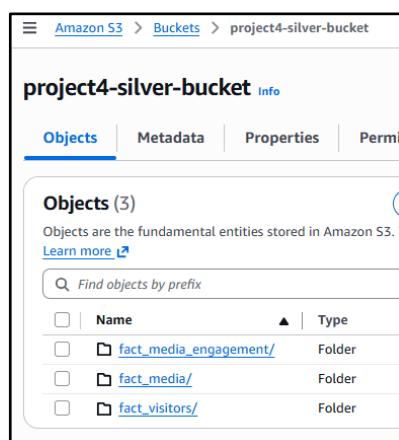
3-1. Objective

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)

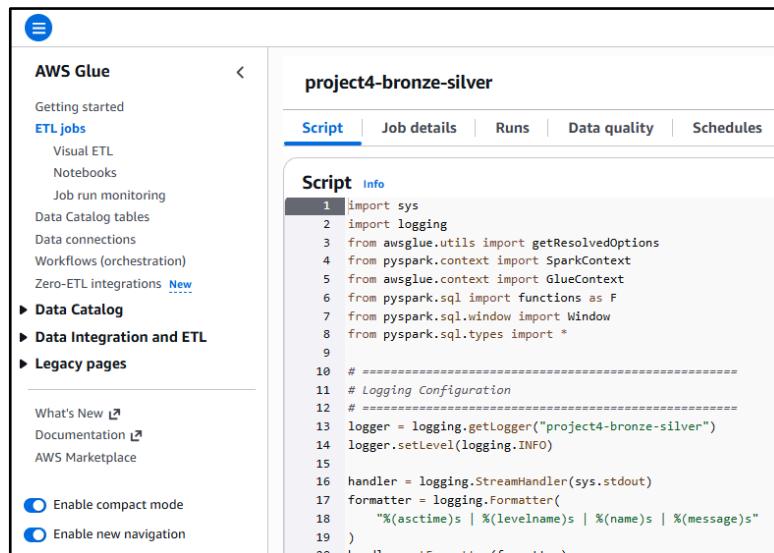
I used Amazon S3 service and create a bucket with name of <s3://project4-silver-bucket/>. All JSON files in the Bronze layer are flattened, then transformed into rational dataset and finally, after cleanup, they are saved in the Parquet format in their own folders in this layer. Data in this layer are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

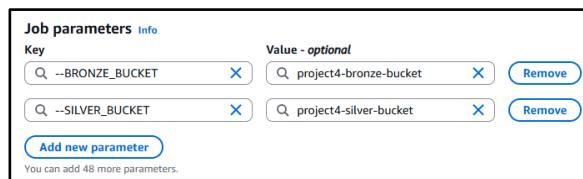
- **Schema Enforcement**
 - Convert JSON files into rational datasets and define schemas for fact tables to prevent schema drifting by flatten nested structures.
 - Ensure consistent column ordering and naming conventions.
- **Data Type Standardization**
 - Cast columns to appropriate data types (e.g., 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.



The screenshot shows the AWS Glue console interface. On the left, there's a sidebar with navigation links like 'Getting started', 'ETL jobs', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main area is titled 'project4-bronze-silver' and has tabs for 'Script', 'Job details', 'Runs', 'Data quality', and 'Schedules'. The 'Script' tab is selected. The code editor contains the following Python script:

```
import sys
import logging
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark.sql.types import *
#
# ===== Logging Configuration =====
#
# ===== Logging Configuration =====
logger = logging.getLogger("project4-bronze-silver")
logger.setLevel(logging.INFO)
handler = logging.StreamHandler(sys.stdout)
formatter = logging.Formatter(
    "%(asctime)s | %(levelname)s | %(name)s | %(message)s"
)
handler.setFormatter(formatter)
logger.addHandler(handler)
```

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.



The screenshot shows the 'Job parameters' section. It has a table with two rows. The first row has a 'Key' column containing '--BRONZE_BUCKET' and a 'Value - optional' column containing 'project4-bronze-bucket'. The second row has a 'Key' column containing '--SILVER_BUCKET' and a 'Value - optional' column containing 'project4-silver-bucket'. There are 'Remove' buttons next to each value. At the bottom, there's a blue 'Add new parameter' button and a note: 'You can add 48 more parameters.'

Key	Value - optional
--BRONZE_BUCKET	project4-bronze-bucket
--SILVER_BUCKET	project4-silver-bucket

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 folders with their file name in **s3://project4-silver-bucket/**, and inside each folder, there are some Parquet files.

The screenshot shows the AWS S3 console interface. The path is `Amazon S3 > Buckets > project4-silver-bucket > fact_visitors/`. The 'Objects' tab is selected. There are four objects listed:

Name	Type
part-00000-439c05aa-194f-43da-ae66-a857183c7adb-c000.snappy.parquet	parquet
part-00001-439c05aa-194f-43da-ae66-a857183c7adb-c000.snappy.parquet	parquet
part-00002-439c05aa-194f-43da-ae66-a857183c7adb-c000.snappy.parquet	parquet
part-00003-439c05aa-194f-43da-ae66-a857183c7adb-c000.snappy.parquet	parquet

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:

fact_media

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

# Read and concatenate them
df_fact_media = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_fact_media.head())
df_fact_media.info()
✓ 0.0s

[ '../data/Silver/fact_media\\part-00000-a7164ec3-7996-404d-a28c-45bacfe49af4-c000.snappy.parquet' ]

   load_count  play_count  play_rate  hours_watched  engagement  visitors  media_id  snapshot_ts
0      111166       16681    0.154950      336.975662     0.167007     104556  gskhw4w4lm  2026-01-08 16:01:48
1      110735       43810    0.435035      2651.986970     0.500445      94643  v08dlgr7v  2026-01-08 16:01:48

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
0   load_count        2 non-null      int64  
1   play_count        2 non-null      int64  
2   play_rate         2 non-null      float64 
3   hours_watched    2 non-null      float64 
4   engagement        2 non-null      float64 
5   visitors          2 non-null      int64  
6   media_id          2 non-null      object  
7   snapshot_ts       2 non-null      datetime64[ns]
dtypes: datetime64[ns](1), float64(3), int64(3), object(1)
memory usage: 260.0+ bytes
```

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_6bfcc196-c93c-487b-a3fe-c24efbc7d4d2-e...	2	2	None	None	None	None	Chrome	126	Windows	False	2026-01-09 18:06:36.625	
1	0023ba9_620b109b-74c4-40f6-8cb1-ebb9bf0f0699-...	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	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	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-a529fc242c8-9...	1	1	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	None	None	None	None	Instagram	325.0.3.34.90	iOS	True	2026-01-09 18:06:36.625	

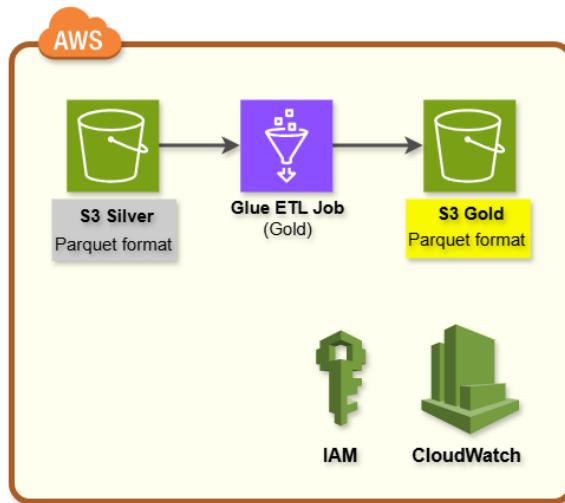
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110499 entries, 0 to 110498
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   visitor_key      110499 non-null   object 
 1   created_at       110499 non-null   datetime64[ns]
 2   last_active_at   110499 non-null   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.



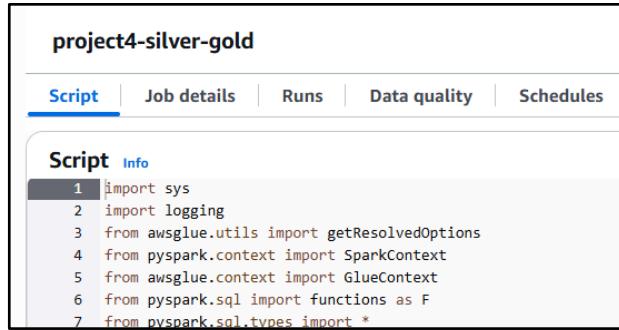
4-2. Gold Layer Storage

I used Amazon S3 service and create a bucket 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 prepared in the Parquet format for machine learning models and BI insights. I named the folders as [fact_media_performance](#) and [fact_audience_insights](#).

A screenshot of the Amazon S3 console showing the 'project4-gold-bucket' bucket. The 'Objects' tab is selected, displaying two objects: 'fact_audience_insights/' and 'fact_media_performance/'. Both objects are listed as 'Folder'. The 'Metadata' tab is also visible at the top.

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.

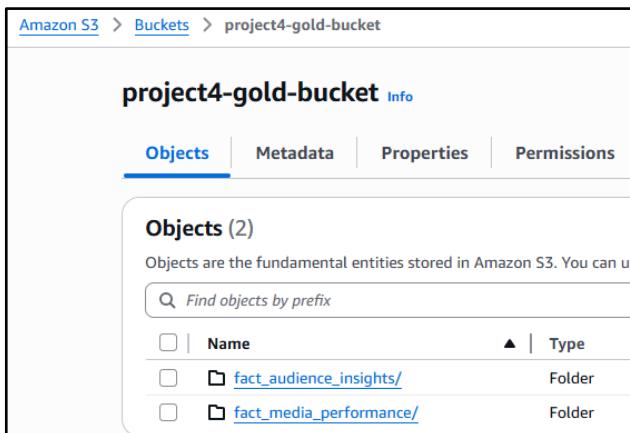


```
import sys
import logging
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql import functions as F
from pyspark.sql.types import *
```

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-bd35-9966e46e7ba-c000.snappy.parquet']

   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-00000-9f25ada7-51dd-4fa9-8a15-675626c5234a-c000.snappy.parquet',
..../data/Gold/Fact_audience_insights\\part-00001-0f25ada7-51dd-4fa9-8a15-675626c5234a-c000.snappy.parquet,
..../data/Gold/Fact_audience_insights\\part-00002-0f25ada7-51dd-4fa9-8a15-675626c5234a-c000.snappy.parquet,
..../data/Gold/Fact_audience_insights\\part-00003-0f25ada7-51dd-4fa9-8a15-675626c5234a-c000.snappy.parquet]

   class 'pandas.core.frame.DataFrame'
RangeIndex: 118499 entries: 0 to 118498
Data columns (total 19 columns):
 # Column           Non-Null Count  Dtype  
--- 
0  visitor_key      118499 non-null  object 
1  created_at       118499 non-null  datetime64[ns]
2  last_active_at  118499 non-null  datetime64[ns]
3  last_event_key   118499 non-null  object 
4  name             118499 non-null  object 
5  play_count       118499 non-null  int64  
6  identifying_event_key  0 non-null   object 
7  name             118499 non-null  object 
8  email            118499 non-null  object 
9  org_name          0 non-null   object 
10 org_title         0 non-null   object 
11 browser          118499 non-null  object 
12 browser_version  118499 non-null  object 
13 platform          118499 non-null  object 
14 mobile            118499 non-null  bool  
15 ingested_at      118499 non-null  datetime64[ns]
16 recency_days     118499 non-null  int32 
17 frequency        118499 non-null  int64  
18 visitor_segment  118499 non-null  object 

dtypes: bool(1), datetime64[ns](1), int32(1), int64(1), 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 (manually), 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 immediately.

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

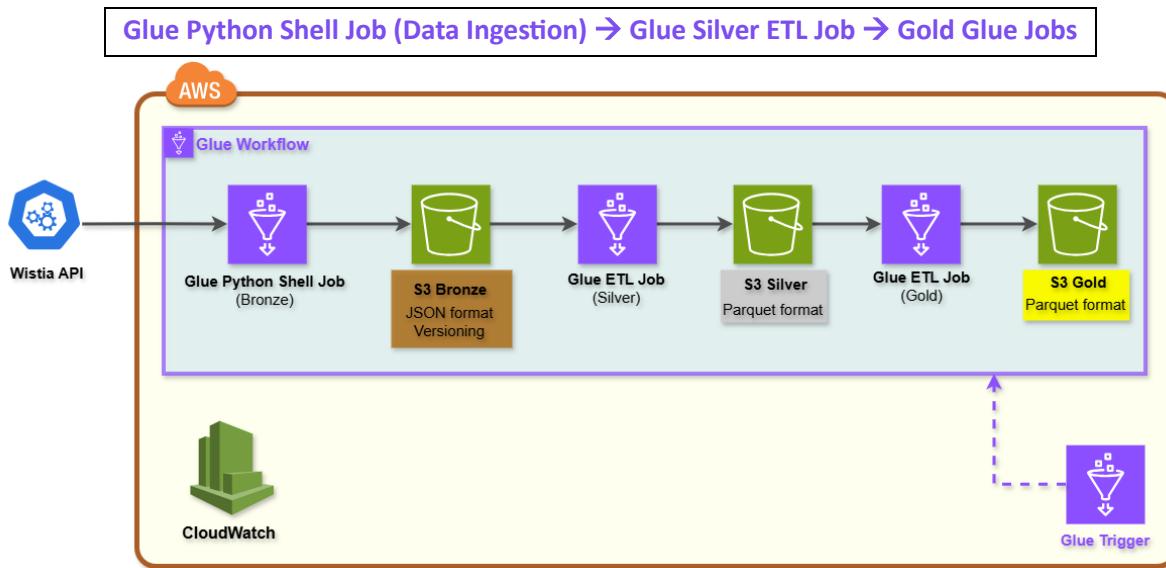
The objective of the CI/CD approach is to automate the end-to-end batch data pipeline on a daily schedule while ensuring reliable orchestration, clear job dependencies, and maintainable CI/CD practices. All data is ingested from Wistia API into AWS, transformed through Bronze, Silver, and Gold layers, 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 orchestration 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 Glue 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:



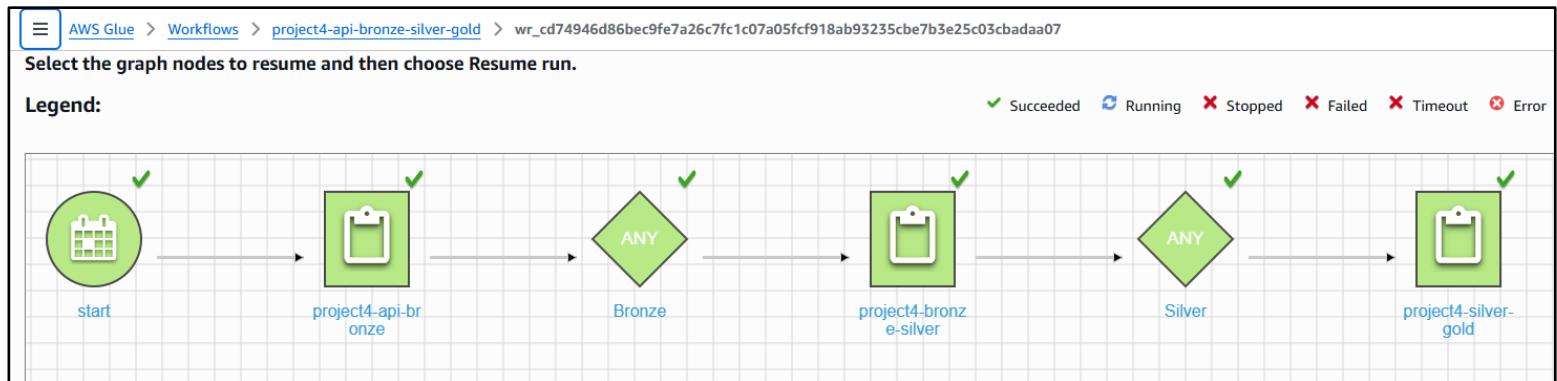
As for triggering the Glue jobs, I created a Glue Trigger, named **start**, which triggers the first Glue Python Shell job at 7:54 am (UTC), daily. As the Glue Python Shell is my first Glue job inside the Glue Workflow, then Glue Workflow will orchestrate all the rest of the jobs after succeeding in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

The screenshot shows the AWS Glue Triggers page. The left sidebar navigation includes: AWS Glue (selected), Getting started, ETL jobs, Visual ETL, Notebooks, Job run monitoring, Data Catalog tables, Data connections, Workflows (orchestration), Zero-ETL integrations, Data Catalog, Data Integration and ETL, Zero-ETL integrations, ETL jobs, Visual ETL, Notebooks, Job run monitoring, Interactive Sessions, Data classification tools, Sensitive data detection, Record Matching, Triggers (selected), Workflows (orchestration), Blueprints, and Security configurations.

The main content area displays the **Triggers** section with the following details:

Name	Status	Type	Parameters	Targets
Bronze	Activated	Conditional	1.condition	1 job: project4-bronze-silver
Silver	Activated	Conditional	1.condition	1 job: project4-silver-gold
start	Activated	Scheduled	At 07:54 AM	1 job: project4-api-bronze

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

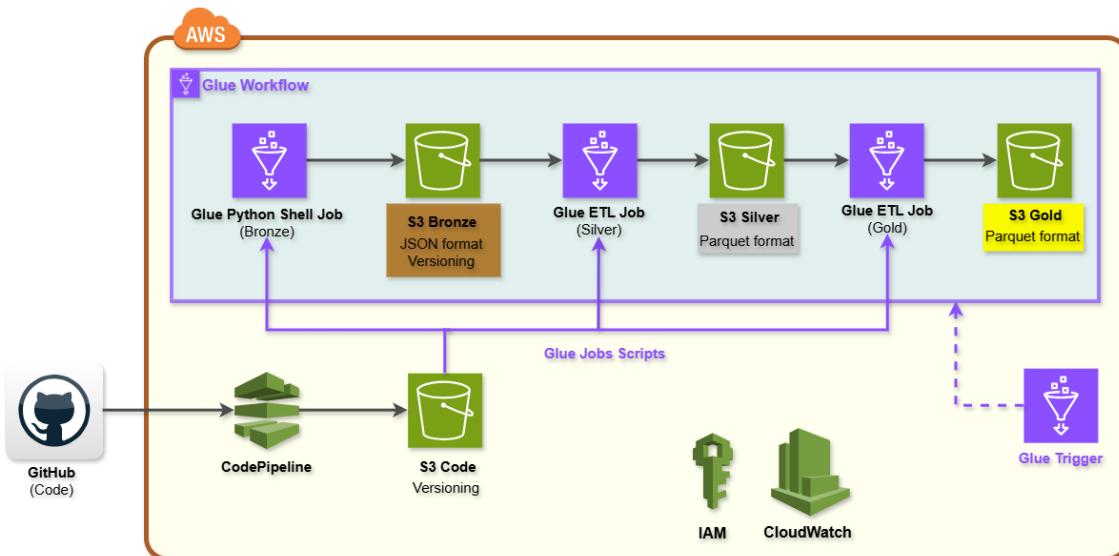
This approach enables automated deployment, traceability, and reproducibility of data pipelines. All Python and PySpark scripts for Glue jobs are stored in a **GitHub** repository:

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 Python 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.

The screenshot shows the 'Permissions' tab of the IAM role configuration. It lists three managed policies attached to the role:

Policy name	Type
AWSCodePipelineServiceRole-us-east-1-project4-github-codepipeline	Customer managed
CodePipeline-CodeConnections-us-east-1-project4-github-codepipeline	Customer managed
CodePipeline-S3Deploy-us-east-1-project4-github-codepipeline	Customer managed

6-5. S3 Code Bucket

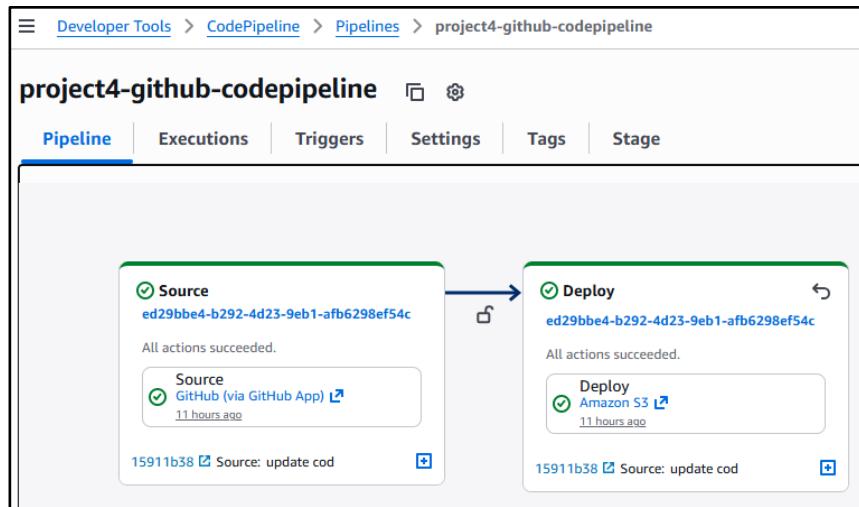
The CI/CD pipeline automatically detects changes or additions to Glue job scripts in the GitHub repo and immediately deploys updated scripts to another S3 (code) bucket, named **s3://project4-code-bucket/** using AWS CodePipeline service. All Glue jobs reference their scripts directly from the S3 code 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.

The screenshot shows the 'Objects' tab of the S3 bucket content view. It displays three objects in the 'Glue_codes/' folder:

Name	Type
project4-api-bronze.py	py
project4-bronze-silver.py	py
project4-silver-gold.py	py

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.