

Project 6: Calendly Booking Data Analytics

Overview

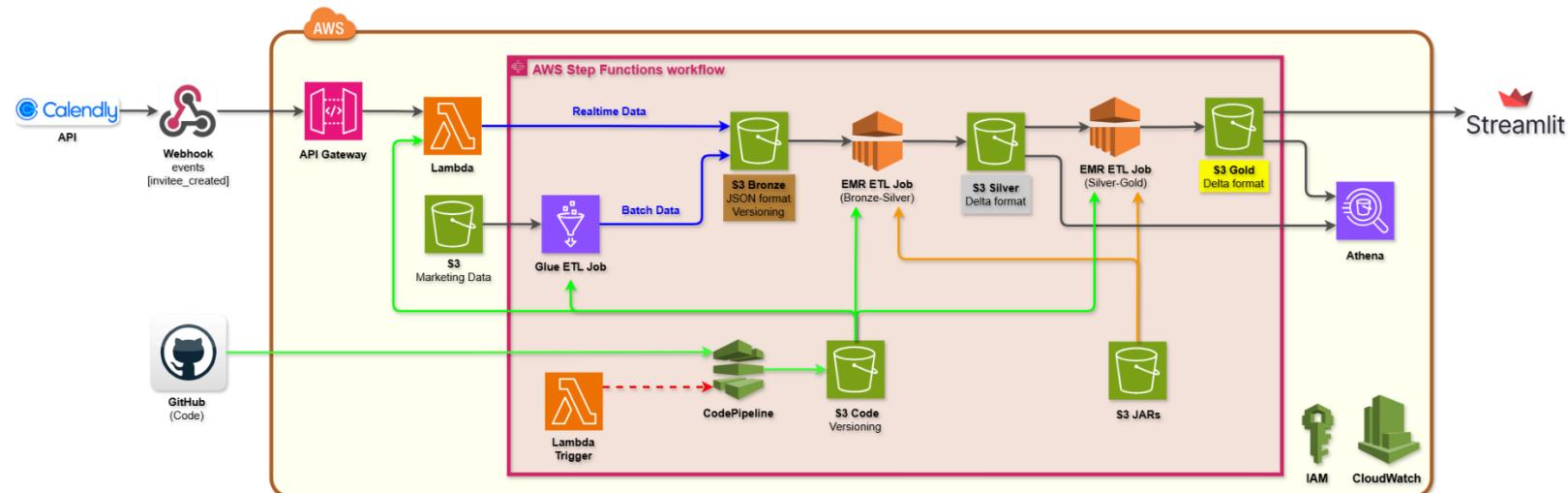
In this project, we work with JSON-formatted data collected from two primary sources: 1) booking (invitee) events from the Calendly platform via Webhooks, and 2) marketing cost data retrieved from a publicly available database stored in the DEA S3 bucket. The goal is to ingest these two data set, integrating them, generate meaningful business metrics, and provide insights that help optimize marketing spend and evaluate campaign conversion rates.

This project implements a hybrid data engineering pipeline that combines real-time, event-driven ingestion (Calendly events) with batch data processing (marketing data). Newly created booking events are captured through Calendly Webhooks, while daily marketing cost data is ingested on a scheduled basis. All data is processed using a fully serverless AWS architecture to ensure scalability, reliability, and cost efficiency.

Real-time booking events are delivered in HTTP format from the Calendly Webhook to an AWS API Gateway endpoint, which triggers a Lambda function. These raw events are stored in the S3 Bronze bucket in their original JSON format. In parallel, an AWS Glue ETL (Spark) job is scheduled to ingest daily marketing cost data from the DEA public S3 bucket (downloading them in the HTTP format) and store it in a dedicated folder within the S3 Bronze layer.

A Bronze–Silver–Gold data modeling approach is applied to ensure clean, structured, and analytics-ready datasets. Two additional Glue ETL jobs perform data transformation and aggregation, progressively refining the data and loading it into the Silver and Gold S3 buckets. The Gold layer contains curated datasets optimized for analytics and reporting. They are storing the data in the format of Delta Tables.

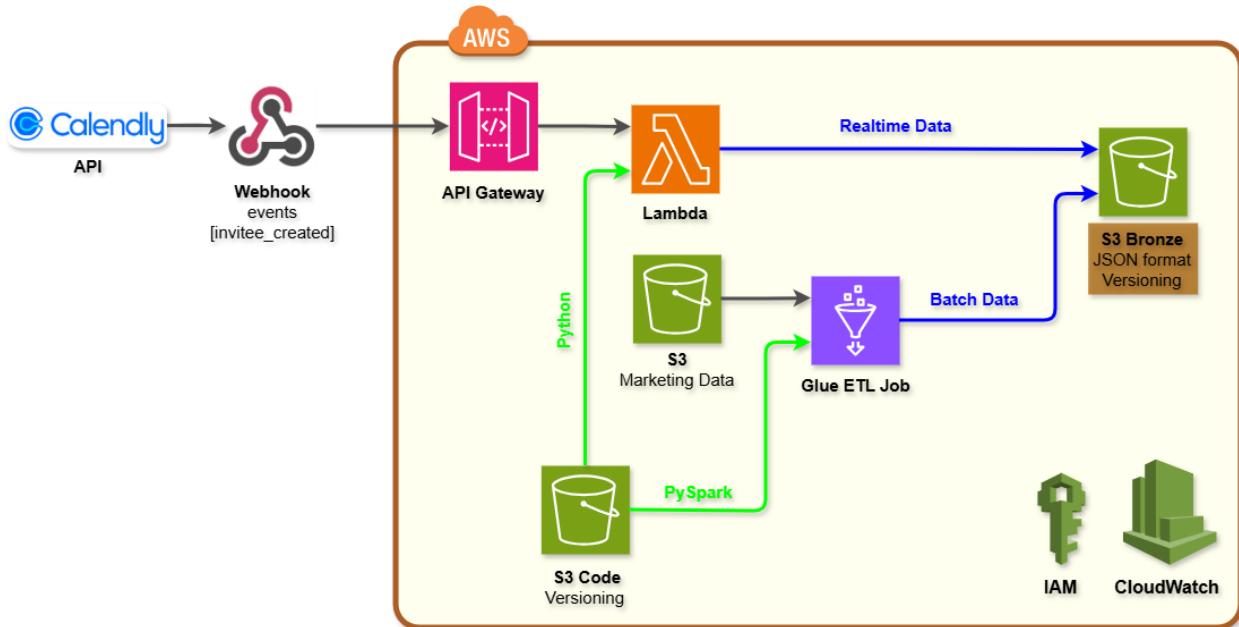
Finally, Streamlit is used to visualize key business KPIs and performance metrics through an interactive dashboard. AWS CodePipeline automates the deployment process by pulling Python and PySpark scripts from a GitHub repository and deploying them to AWS Lambda and Glue jobs. AWS Glue Triggers handle daily scheduling, while AWS Glue Workflows orchestrate the end-to-end pipeline, resulting in a scalable, modular, and production-ready data architecture aligned with modern data engineering best practices.



Step 1 – Data Ingestion

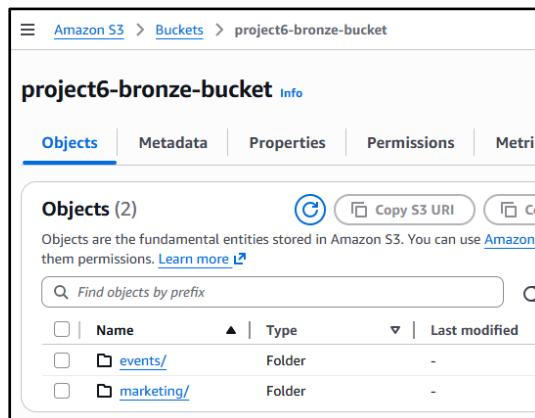
1-1- Objective

The objective of this step is to ingest raw invitee event files from Calendly API Webhook, in **realtime format**, and also daily marketing cost data from the DEA public S3 bucket, in the **batch format**, into an Amazon S3 bucket, named **project6-bronze-bucket**, in the JSON format. To comply with the project requirement that allows me to use Python, I will implement **AWS Lambda** for realtime data ingestion, via **AWS API Gateway**. Moreover, I used **AWS Glue ETL job** for batch data ingestion. This hybrid approach ensures that the ingestion process is both scalable and aligned with other AWS services in orchestration.



1-2- S3 Bronze Layer

I used Amazon S3 service and create a bucket with name of `s3://project6-bronze-bucket/`. After that, I created two folders inside that bucket with the name of `events/` folder (for ingestion of new raw invitee event files in realtime mode), and `marketing/` folder (for ingestion of daily marketing cost for last 30 days).



1-3- IAM Roles

Before starting data ingestion using Lambda, I configured an IAM role to enable Lambda to securely access the required AWS services during job execution. The IAM role was named **project6-lambda-role** and tagged with **project: 6**, and it was attached to the AWS Lambda functions to ensure secure and authorized access during execution.

The following permissions were configured:

- **AmazonS3FullAccess:** Granted to allow Lambda to write to the Bronze S3 bucket.
- **CloudCodePipeline_FullAccess:** Enabled triggering the CodePipeline during CI/CD.
- **AmazonLambda_FullAccess:** Granted to allow creation, execution, and management of Lambda functions involved in webhook ingestion and lead enrichment workflows.
- **CloudWatchLogsFullAccess:** Enabled logging and monitoring of Lambda function execution in Amazon CloudWatch.

Policy name		Type
<input type="checkbox"/>	AmazonS3FullAccess	AWS managed
<input type="checkbox"/>	AWSCodePipeline_FullAccess	AWS managed
<input type="checkbox"/>	AWSLambda_FullAccess	AWS managed
<input type="checkbox"/>	CloudWatchLogsFullAccess	AWS managed

Also, I configured another IAM role to enable AWS Glue jobs to securely access the required AWS services during job execution. The IAM role was named **project6-glue-role** and tagged with **project: 6**, and it was attached to the AWS Glue ETL jobs to ensure secure and authorized access during execution.

The following permissions were configured:

- **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.
- **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.

- **Access_to_dea-data-bucket (inline)**: An inline policy added to the Glue job role to allow reading publicly available data from the DEA bucket. This policy permits `s3:GetObject` and `s3>ListBucket` actions on the dea-data-bucket and all its objects. Here is the policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReadPublicDEABucket",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3>ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::dea-data-bucket",
        "arn:aws:s3:::dea-data-bucket/*"
      ]
    }
  ]
}
```

IAM > Roles > project6-glue-role		
<input type="checkbox"/> Policy name ↴ <input type="checkbox"/> Type		
<input type="checkbox"/>	 CloudWatchLogsFullAccess	AWS managed
<input type="checkbox"/>	 AWSGlueServiceRole	AWS managed
<input type="checkbox"/>	 AWSGlueConsoleFullAccess	AWS managed
<input type="checkbox"/>	 AmazonS3FullAccess	AWS managed
<input type="checkbox"/>	 Access_to_dea-data-bucket	Customer inline

1-4- Lambda Function

I created the Lambda function, named [project6-calendly-webhook-ingestion](#), to perform raw data ingestion from Calendly Webhook API into S3 Bronze bucket. The script file name in the Python language is [project6-calendly-webhook-ingestion.py](#). I will use this function later in CI/CD section. I added one environment variable to this function as Bronze bucket name.

Key: BRONZE_BUCKET

Value: project6-bronze-bucket

project6-calendly-webhook-ingestion

Function overview [Info](#)

[Diagram](#) [Template](#)

project6-calendly-webhook-ingestion

Layers (0)

API Gateway (2) [+ Add destination](#)

[+ Add trigger](#)

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

General configuration

Environment variables (1)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
BRONZE_BUCKET	project6-bronze-bucket

1-5- API Gateway

To get events from Calendly Webhook in the realtime format, I created an API Gateway, named **project6-calendly-webhook-rest-api**, to fetch data from Calendly Webhook. It will post events to my URL using REST API. After that, I created an URL stage in AWS using API Gateway.

After creating the API Gateway successfully, I got my URL address as below. I use this URL in Calendly Webhook configuration.

<https://o7hvem4u66.execute-api.us-east-1.amazonaws.com/prod/webhook>

Name	Description	ID	Protocol	API endpoint type	Created	Security policy	API status
project5-crm-webhook-api		lfrfcwk7j	HTTP	Regional	2026-01-12	-	-
project6-calendly-webhook-rest-api	REST API for ingesting Calendly webhooks into S3 Bronze layer	o7hvem4u66	REST	Regional	2026-01-14	SecurityPolicy_TLS13_1_2_2021_06	Available

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with 'APIs' and 'Stages' sections. Under 'APIs', 'project6-calendly-webhook-rest-api' is selected. In the 'Stages' section, 'prod' is highlighted. The main area is titled 'Stage details' for the 'prod' stage. It shows the stage name 'prod', rate and burst limits (10000 and 5000), and inactive cache and default method-level caching. Below this, there's an 'Invoke URL' field with the value 'https://o7hvem4u66.execute-api.us-east-1.amazonaws.com/prod'. At the bottom, it says 'Active deployment mpby4b on January 13, 2026, 21:33 (UTC-08:00)'. There are 'Stage actions' and 'Create stage' buttons at the top right.

To test my API Gateway, I sent a fake event manually to my URL in VS Code terminal:

```
Invoke-RestMethod -Uri "https://o7hvem4u66.execute-api.us-east-1.amazonaws.com/prod/webhook" ` 
-Method POST ` 
-Headers @{"Content-Type" = "application/json"} ` 
-Body '{"event": {"invitee": "hadi_test_123", "created_at": "2026-01-13", "email": "test@example.com"}}'
```

It works successfully!

The screenshot shows the VS Code interface with a terminal tab open. The terminal window displays a PowerShell session. The user runs the command 'Invoke-RestMethod -Uri "https://o7hvem4u66.execute-api.us-east-1.amazonaws.com/prod/webhook" -Method POST -Headers @{"Content-Type" = "application/json"} -Body '{"event": {"invitee": "hadi_test_123", "created_at": "2026-01-13", "email": "test@example.com"}}''. The output shows the webhook was ingested successfully, creating a file named 'events/2026/01/14/524ce471-9162-49b5-bcd8-332abea81b08.json'.

Amazon S3 > Buckets > project6-bronze-bucket > events/ > 2026/ > 01/ > 14/

14/

Objects Properties

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you must grant them permissions. [Learn more](#)

[Find objects by prefix](#)

Show versions

Name	Type	Last modified	Size	Storage class
524ce471-9162-49b5-bcd8-332abea81b08.json	json	January 13, 2026, 21:27:45 (UTC-08:00)	184.0 B	Standard

Then, I request DEA manager to connect my URL to their available Calendly Webhook. After that I got several JSON files in my S3 bucket.

Amazon S3 > Buckets > project6-bronze-bucket > events/

events/

Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you must grant them permissions. [Learn more](#)

[Find objects by prefix](#)

Show versions

Name	Type	Last modified
event_date=2026-01-14/	Folder	-
event_date=2026-01-15/	Folder	-
event_date=2026-01-16/	Folder	-
event_date=2026-01-17/	Folder	-
event_date=2026-01-18/	Folder	-
event_date=2026-01-19/	Folder	-

Amazon S3 > Buckets > project6-bronze-bucket > events/ > event_date=2026-01-15/

event_date=2026-01-15/

Objects (203)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you must grant them permissions. [Learn more](#)

[Find objects by prefix](#)

Show versions

Name	Type	Last modified	Size	Storage class
00b2f85f-3e04-42f2-a601-3506dbf99a5e.json	json	January 15, 2026, 12:06:55 (UTC-08:00)	4.0 KB	Standard
0110c639-50f3-4a4d-93c9-8a7feb1c01e7.json	json	January 15, 2026, 07:39:17 (UTC-08:00)	2.2 KB	Standard
01b60158-8103-470e-9a84-7ba84e753a47.json	json	January 15, 2026, 12:21:04 (UTC-08:00)	2.2 KB	Standard
038177a0-7564-453e-bd6d-cd300c228601.json	json	January 15, 2026, 12:31:06 (UTC-08:00)	2.4 KB	Standard
053423e0-db8a-4cd8-ae68-df62105ae314.json	json	January 14, 2026, 16:53:01 (UTC-08:00)	2.2 KB	Standard
05f2df56-f4a0-41f1-8457-0548811213e4.json	json	January 15, 2026, 10:36:16 (UTC-08:00)	2.4 KB	Standard
07a8b9f6-bf23-4181-8ab0-451766251b3e.json	json	January 15, 2026, 12:46:53 (UTC-08:00)	3.9 KB	Standard
09ff9943-25b0-4d11-bfbf-add33766263b.json	json	January 15, 2026, 11:54:11 (UTC-08:00)	2.3 KB	Standard
0a068808-805d-4ff8-b9f6-1fa045c24d72.json	json	January 15, 2026, 12:53:19 (UTC-08:00)	2.4 KB	Standard

1-6- Glue ETL Job (S3-Bronze)

After establishing secure access between the DEA S3 bucket and my Bronze S3 bucket, I ingested the raw marketing data from the DEA S3 bucket into the Bronze layer using an **AWS Glue job** named **project6-bronze**. For testing purposes, the ETL logic was implemented as an inline PySpark script within the Glue job. In a production-level CI/CD setup, this script will be stored and managed in a dedicated S3 code repository. The PySpark script is named **project6-bronze.py**.

This Glue job is responsible for ingesting all files related to daily media cost data and storing them under **s3://project6-bronze-bucket/marketing/**. The job was configured with a single input parameter, defined in the **Job Parameters** section. To optimize resource usage during ingestion, minimal compute resources were selected, and the job execution timeout was set to **10 minutes**.

The image consists of three vertically stacked screenshots of the AWS Glue console interface.

- Screenshot 1: Script Tab**

The "Script" tab is selected. It shows the inline PySpark script content:

```
1 # =====
2 # Glue ETL Job: project6-marketing-bronze.py
3 # =====
4
5 # -----
```
- Screenshot 2: Job Details Tab**

The "Job details" tab is selected. Configuration settings include:
 - IAM Role:** project6-glue-role
 - Type:** Spark
 - Glue version:** Glue 5.0 - Supports spark 3.5, Scala 2, Python 3
 - Language:** Python 3
 - Worker type:** G 1X (4vCPU and 16GB RAM)
 - Automatically scale the number of workers:** Unchecked checkbox
- Screenshot 3: Job Parameters Tab**

The "Job parameters" tab is selected. It shows a single parameter entry:
 - Key:** --BRONZE_BUCKET
 - Value - optional:** project6-bronze-bucket
 - Add new parameter:** Button

1-7- Demonestration

After setting up all above services, now I am going start the Bronze layer data ingestion. I asked Azmat to configure the Calendly Webhook setup to send events to my endpoint URL to get the realtime events. Also, in parallel, I ran the Glue function to ingest all marketing data for last 30 days from DEA public S3 bucket into my S3 Bronze bucket in the batch mode. My S3 bucket has structure like this:

The screenshot shows the AWS S3 console interface. At the top, the navigation bar shows 'Amazon S3 > Buckets > project6-bronze-bucket'. Below the navigation, the bucket name 'project6-bronze-bucket' is displayed with a 'Info' link. A horizontal menu bar includes 'Objects' (which is underlined in blue), 'Metadata', 'Properties', 'Permissions', and 'Metrics'. The main content area is titled 'Objects (2)'. It contains a table with two rows, each representing a folder object. The columns are 'Name', 'Type', and 'Last modified'. The first row has a checkbox and the name 'events/'. The second row also has a checkbox and the name 'marketing/'. To the right of the table are three buttons: a blue circular icon with a 'C', a white button with 'Copy S3 URI', and another white button with 'Copy Object'. Below the table is a search bar with the placeholder 'Find objects by prefix' and a magnifying glass icon. A small note below the search bar says 'Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3](#) them permissions. [Learn more](#)'.

Name	Type	Last modified
events/	Folder	-
marketing/	Folder	-

This is structure of **event** files:

```
s3://project6-bronze-bucket/  
└── events/  
    ├── event_date=2026-01-14/      # 212 objects  
    │   └── 008d3007-bec0-4abf-b074-dcb333e6e959.json  
    │   ...  
    ├── event_date=2026-01-15/      # 266 objects  
    │   └── 00b2f85f-3e04-42f2-a601-3506dbf99a5e.json  
    │   ...  
    ├── event_date=2026-01-16/      # 292 objects  
    │   └── 0112157c-3359-4b03-ac3e-1c3d65b81962.json  
    │   ...  
    ├── event_date=2026-01-17/      # 175 objects  
    │   └── 00132870-3a9a-480d-9d98-743a6188686b.json  
    │   ...  
    ├── event_date=2026-01-18/      # 132 objects  
    │   └── 02553eec-5cd8-4332-9b56-24717ca4ea24.json  
    │   ...  
    ├── event_date=2026-01-19/      # 274 objects  
    │   └── 0d6e6bd1-4397-4462-a43b-0a1c3276ecb4.json  
    │   ...  
    └── event_date=2026-01-20/      # 190 objects (so far)  
        └── 02039453-9b50-44da-8a56-8d1016f543d5.json
```

To validate the silver layer data I downloaded them to my local machine and read with VS Code. This is the code for downloading the data using AWS CLI:

```
(base) PS D:\DE\Projects\Projects\Project_6> aws s3 cp s3://project6-bronze-bucket/events/  
./data/Bronze/events/ --recursive --exclude "*" --include "*.json"
```

This is a sample of one of events, which the API Gateway fetch from Calendly Webhook:

0112157c-3359-4b03-ac3e-1c3d65b81962.json

```
{  
  "ingestion_time": "2026-01-16T15:54:43.545190",  
  "source": "calendly_webhook",  
  "data": {  
    "created_at": "2026-01-16T15:54:42.000000Z",  
    "created_by": "https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf",  
    "event": "invitee.created",  
    "payload": {  
      "cancel_url": "https://calendly.com/cancellations/64436c3b-b1a4-4840-b994-eb6eb69d2d3a",  
      "created_at": "2026-01-16T15:54:41.302307Z",  
      "email": "akinsjameson@gmail.com",  
      "event": "https://api.calendly.com/scheduled_events/0a6874c5-b822-4db4-a46f-5cd90f1851d8",  
      "first_name": null,  
      "invitee_scheduled_by": "https://api.calendly.com/users/e43574f3-332f-4da5-9ce1-c10248f49ff5",  
      "last_name": null,  
      "name": "Akin Akinola ",  
      "new_invitee": null,  
      "no_show": null,  
      "old_invitee": "https://api.calendly.com/scheduled_events/d3bfe89e-c98f-498f-92e1-024683233693/invitees/85968cbc-6180-4cf3-b33b-a25023db18f1",  
      "payment": null,  
      "questions_and_answers": [  
        ],  
      "reconfirmation": null,  
      "reschedule_url": "https://calendly.com/reschedulings/64436c3b-b1a4-4840-b994-eb6eb69d2d3a",  
      "rescheduled": false,  
      "routing_form_submission": null,  
      "scheduled_event": {  
        "created_at": "2026-01-16T15:54:41.279214Z",  
        "end_time": "2026-01-20T18:45:00.000000Z",  
        "event_guests": [  
          ],  
        "event_memberships": [  
          {  
            "user": "https://api.calendly.com/users/e43574f3-332f-4da5-9ce1-c10248f49ff5",  
            "user_email": "om@dataengineeracademy.com",  
            "user_name": "Om Gaikhe"  
          }  
        ],  
        "event_type": "https://api.calendly.com/event_types/4392759b-c2cf-49eb-bfba-42da9eec2a49",  
        "invitees_counter": {  
          "total": 1,  
          "active": 1,  
          "limit": 1  
        },  
        "location": {  
          "join_url": "https://calendly.com/events/0a6874c5-b822-4db4-a46f-5cd90f1851d8/google_meet",  
          "status": "processing",  
          "type": "google_conference"  
        },  
        "meeting_notes_html": null,  
        "meeting_notes_plain": null,  
        "name": "VIP Onboarding",  
        "start_time": "2026-01-20T18:15:00.000000Z",  
        "status": "active",  
        "updated_at": "2026-01-16T15:54:41.279214Z",  
        "uri": "https://api.calendly.com/scheduled_events/0a6874c5-b822-4db4-a46f-5cd90f1851d8"  
      },  
      "scheduling_method": "instant_book",  
      "status": "active",  
      "text_reminder_number": null,  
      "timezone": "America/New_York",  
      "tracking": {  
        "utm_campaign": null,  
        "utm_source": null,  
        "utm_medium": null,  
        "utm_content": null,  
        "utm_term": null,  
        "salesforce_uuid": null  
      },  
      "updated_at": "2026-01-16T15:54:41.339370Z",  
      "uri": "https://api.calendly.com/scheduled_events/0a6874c5-b822-4db4-a46f-5cd90f1851d8/invitees/64436c3b-b1a4-4840-b994-eb6eb69d2d3a"  
    }  
  }  
}
```

Also, this is the structure of **marketing** files:

```
s3://project6-bronze-bucket/
└── marketing/
    ├── file_index.json
    ├── spend_date=2025-12-16.json
    ├── spend_date=2025-12-17.json
    ├── ...
    ├── spend_date=2026-01-16.json
    ├── spend_date=2026-01-17.json
    ├── spend_date=2026-01-18.json
    └── spend_date=2026-01-19.json
```

To extract the above information, I used the data which are publicly available at DEA Public Database (only the data of last 30 days):

```
https://dea-data-bucket.s3.us-east-1.amazonaws.com/calendly\_spend\_data/file\_index.json
https://dea-data-bucket.s3.us-east-1.amazonaws.com/calendly\_spend\_data/spend\_data\_2025-12-21.json
https://dea-data-bucket.s3.us-east-1.amazonaws.com/calendly\_spend\_data/spend\_data\_2025-12-22.json
...
https://dea-data-bucket.s3.us-east-1.amazonaws.com/calendly\_spend\_data/spend\_data\_2026-01-18.json
https://dea-data-bucket.s3.us-east-1.amazonaws.com/calendly\_spend\_data/spend\_data\_2026-01-19.json
```

Then, I used the first index file to read the file names and use it as per availability. The rest of files are as a sample for today (2026-01-20). After running the Glue job, I got these data in my S3 bronze bucket. I used this code for downloading the data using AWS CLI:

```
(base) PS D:\DE\Projects\Projects\Project_6> aws s3 cp s3://project6-bronze-bucket/marketing/
./data/Bronze/marketing/ --recursive --exclude "*" --include "*.json"
```

This is the data for the index file at the date of 2026-01-18 (only last month's data will be available daily):

```
{
  "files": [
    "spend_data_2025-12-21.json",
    "spend_data_2025-12-22.json",
    "spend_data_2025-12-23.json",
    "spend_data_2025-12-24.json",
    "spend_data_2025-12-25.json",
    "spend_data_2025-12-26.json",
    "spend_data_2025-12-27.json",
    "spend_data_2025-12-28.json",
    "spend_data_2025-12-29.json",
    "spend_data_2025-12-30.json",
    "spend_data_2025-12-31.json",
    "spend_data_2026-01-01.json",
    "spend_data_2026-01-02.json",
    "spend_data_2026-01-03.json",
    "spend_data_2026-01-04.json",
    "spend_data_2026-01-05.json",
    "spend_data_2026-01-06.json",
    "spend_data_2026-01-07.json",
    "spend_data_2026-01-08.json",
    "spend_data_2026-01-09.json",
    "spend_data_2026-01-10.json",
    "spend_data_2026-01-11.json",
    "spend_data_2026-01-12.json",
    "spend_data_2026-01-13.json",
    "spend_data_2026-01-14.json",
    "spend_data_2026-01-15.json",
    "spend_data_2026-01-16.json",
    "spend_data_2026-01-17.json",
    "spend_data_2026-01-18.json",
    "spend_data_2026-01-19.json"]}
```

Note: I got the information of `spend_date=2026-01-19.json`, yesterday 😊.

Also, this is sample data for daily marketing files, which the Glue ETL job fetches from DEA publick S3 database (the spend amount is in USD currency):

```
spend_data_2026-01-15.json
```

```
[  
  {  
    "date": "2026-01-15",  
    "channel": "youtube_paid_ads",  
    "spend": 381.76  
  },  
  {  
    "date": "2026-01-15",  
    "channel": "facebook_paid_ads",  
    "spend": 473.8  
  },  
  {  
    "date": "2026-01-15",  
    "channel": "tiktok_paid_ads",  
    "spend": 563.78  
  }  
]
```

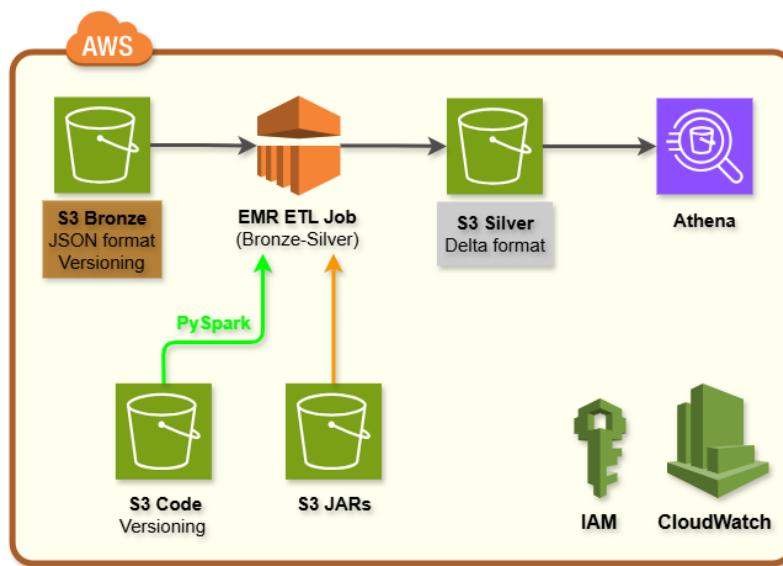
The Bronze layer data was then validated by downloading and manually reviewing the files, and all checks were successful.

Step 2 – Transformation (S3 Silver Layer)

2-1- Objective

The objective of the Silver layer is to transform raw ingested JSON files into clean, standardized, and analytically ready datasets. This layer focuses on enhancing data quality by removing duplicates, enforcing schemas, and preparing fact and dimension tables for downstream analytics and metric computation. Depending on data complexity and project requirements, the data in this layer is stored in Delta table format.

Unlike a previous project in which AWS Glue ETL jobs were used for data transformation, where the output was limited to Parquet format, this project leverages an **EMR Serverless ETL job** for the transformation step. EMR Serverless enables native support for writing **Delta tables**, making it a more suitable choice for implementing the Silver layer in this architecture.



2-2- S3 Silver Layer

I used the Amazon S3 service to create a bucket named <s3://project6-silver-bucket/>. All JSON files from the Bronze layer are first fully and deeply flattened, then transformed into structured datasets, and finally, after data cleansing, stored in Delta format within their respective folders in this layer. The Silver layer data is organized into fact-oriented datasets, enabling efficient joins and aggregations in subsequent processing stages. Two folders, [fact_events/](#) and [fact_marketing/](#), were created in this bucket to store the corresponding fact tables. The structure of this bucket is as follows:

```

s3://project6-silver-bucket/
└── fact_events/
    ├── _delta_log/
    ├── event_date=2026-01-14/
    ├── event_date=2026-01-15/
    ├── event_date=2026-01-16/
    │   └── part-00001-4227ef1f-5cbe-4020-8b97-ad23aa1bc2e1.c000.snappy.parquet
    │   ...
    │   └── part-00033-14ab10a9-262b-46d3-8ecc-434714f6d310.c000.snappy.parquet
    ├── event_date=2026-01-17/
    ├── event_date=2026-01-18/
    ├── event_date=2026-01-19/
    └── event_date=2026-01-20/

└── fact_marketing/
    ├── _delta_log/
    ├── date=2025-12-16/
    │   └── part-00000-cb48c0dd-3a5d-4bd5-8bc9-8de4b8070052.c000.snappy.parquet
    ├── date=2025-12-17/
    │   └── part-00000-e2b9ad1d-9ebe-467c-b2db-95bfce844fbe.c000.snappy.parquet
    ├── date=2026-01-17/
    │   └── part-00002-f120bcef-8eaa-41da-bea8-4146bafcd2dcc.c000.snappy.parquet
    ├── date=2026-01-18/
    │   └── part-00002-4a82d091-7873-4869-9ff2-0f06a351566a.c000.snappy.parquet
    └── date=2026-01-19/
        └── part-00002-9cdabd08-ed39-4182-94c8-1d5f942c5b2c.c000.snappy.parquet

```

Amazon S3 > Buckets > project6-silver-bucket

project6-silver-bucket [Info](#)

- [Objects](#) [Metadata](#) [Properties](#) [Permissions](#)

Objects (2) [Copy](#)

Objects are the fundamental entities stored in Amazon S3. You can upload, download, and manage them permissions. [Learn more](#)

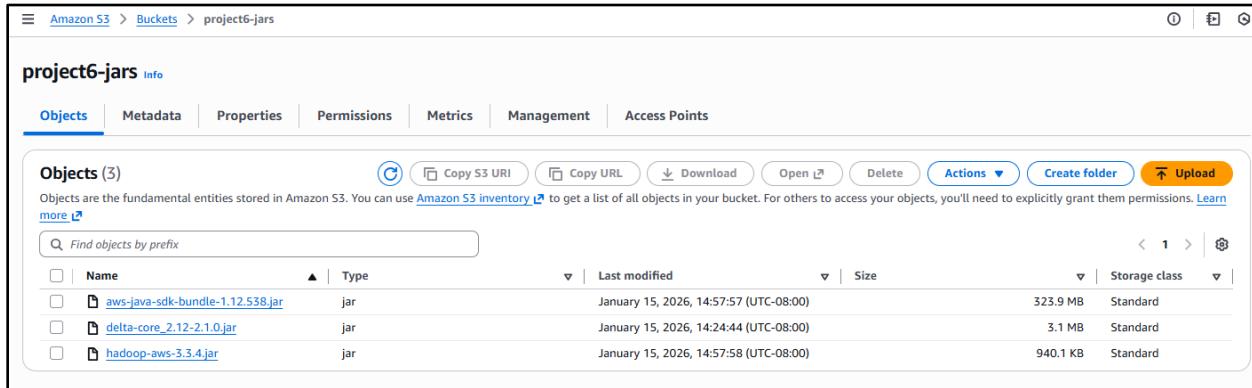
<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	fact_events/	Folder
<input type="checkbox"/>	fact_marketing/	Folder

Additionally, **EMR Serverless** requires external dependencies to be explicitly provided during job configuration. To support this requirement, a dedicated S3 bucket named [s3://project6-jars/](#) was created to store all necessary **JAR dependencies**. These JAR files were downloaded from the **Maven Central repository** (<https://mvnrepository.com/>) and uploaded to this bucket. During EMR Serverless job configuration, the JARs location is explicitly referenced to ensure they are available at runtime.

The following JAR dependencies were attached:

1. **Delta Lake support:** [delta-core_2.12-2.1.0.jar](#)
2. **Amazon S3 filesystem integration:** [hadoop-aws-3.3.4.jar](#)
3. **AWS SDK for Java:** [aws-java-sdk-bundle-1.12.538.jar](#)

This configuration enables EMR Serverless to reliably read from and write to Delta tables stored in Amazon S3 while maintaining transactional consistency and compatibility with AWS services.



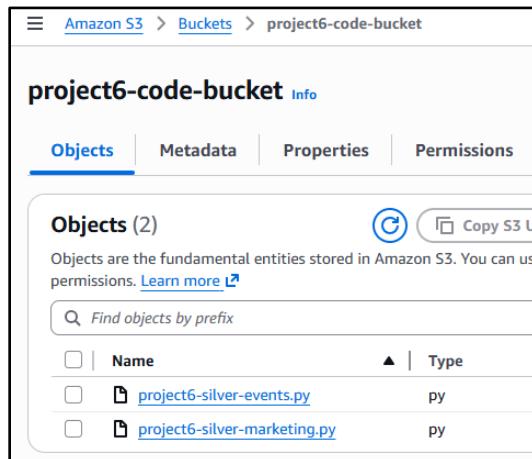
project6-jars [Info](#)

Objects (3)

Name	Type	Last modified	Size	Storage class
aws-java-sdk-bundle-1.12.538.jar	jar	January 15, 2026, 14:57:57 (UTC-08:00)	323.9 MB	Standard
delta-core_2.12-2.1.0.jar	jar	January 15, 2026, 14:24:44 (UTC-08:00)	3.1 MB	Standard
hadoop-aws-3.3.4.jar	jar	January 15, 2026, 14:57:58 (UTC-08:00)	940.1 KB	Standard

Moreover, unlike AWS Lambda or AWS Glue jobs, which provide an inline editor for embedding transformation scripts, **EMR Serverless** does not offer a built-in interface for authoring or storing job code. Instead, EMR Serverless requires transformation scripts to be written externally in **PySpark** and **stored in an Amazon S3 location**, with the script path explicitly provided at job submission time. This approach aligns with best practices for CI/CD-oriented data pipelines.

To support this design, a dedicated S3 bucket was created to store all ETL scripts, named [s3://project6-code-bucket](#). In the production CI/CD workflow, this bucket will be populated automatically from a GitHub repository using AWS CodePipeline. For the Silver layer transformations, two PySpark scripts were developed and stored in this bucket: [project6-silver-events.py](#) and [project6-silver-marketing.py](#). These scripts are responsible for transforming Bronze layer data into structured, Delta-formatted datasets in the Silver layer.



project6-code-bucket [Info](#)

Objects (2)

Name	Type
project6-silver-events.py	py
project6-silver-marketing.py	py

2-3- IAM Role

Before initiating data transformations using **EMR Serverless**, an IAM execution role was configured to allow EMR to securely access the required AWS services during job execution. This role, named **project6-emr-role**, was tagged with **project: 6** and assigned as the execution role for EMR Serverless applications to ensure secure and authorized access at runtime. I considered the IAM trust relationship as below:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "emr-serverless.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

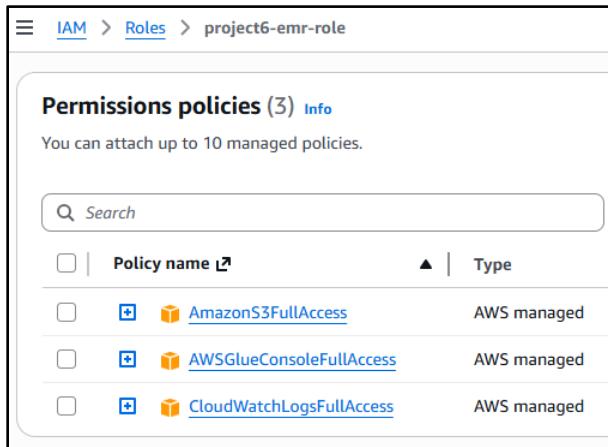
The screenshot shows the AWS IAM 'project6-emr-role' configuration page. The 'Trust relationships' tab is active, displaying the following JSON trust policy:

```
1 {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Principal": {  
7                 "Service": "emr-serverless.amazonaws.com"  
8             },  
9             "Action": "sts:AssumeRole"  
10        }  
11    ]  
12 }
```

Then the following permissions were configured for this role:

- **AmazonS3FullAccess:** Granted to allow EMR Serverless jobs to read from and write to the Bronze, Silver, and Gold S3 buckets.
- **CloudWatchLogsFullAccess:** Enabled centralized logging and monitoring of EMR Serverless job executions in Amazon CloudWatch.
- **AWS Glue permissions:** Provided access to the AWS Glue Data Catalog, enabling EMR Serverless to read table metadata and schemas required for Delta Lake-based transformations.

This IAM configuration ensures secure, least-friction access to all required services while maintaining clear separation between orchestration, execution, and data access responsibilities.



The screenshot shows the 'Permissions policies' section of an IAM role named 'project6-emr-role'. It displays three AWS managed policies attached to the role:

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

2-4- EMR Job (Bronze - Silver)

In this project, the transformation layer was migrated from **AWS Glue ETL jobs** to **EMR Serverless jobs** to better support advanced data processing requirements. While AWS Glue provides a fully managed and serverless solution for data transformations, its native output formats are limited to Parquet and do not provide full support for Delta Lake transactional features. In contrast, EMR Serverless offers greater flexibility and native compatibility with Delta Lake, enabling ACID-compliant writes, efficient handling of incremental data, and improved support for schema evolution and partitioned fact tables. This transition allows the Silver layer to leverage Delta table capabilities more effectively, resulting in improved data reliability, scalability, and suitability for downstream analytics.

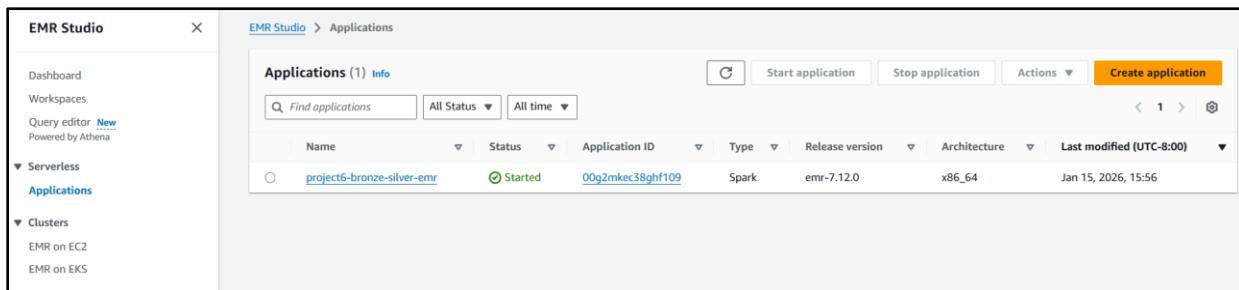
Amazon EMR provides two primary deployment models: **EMR on EC2** and **EMR Serverless**. EMR on EC2 requires provisioning and managing clusters composed of EC2 instances, offering fine-grained control over infrastructure but introducing additional operational overhead related to scaling, maintenance, and cost optimization. In contrast, EMR Serverless abstracts cluster management entirely, automatically provisioning and scaling compute resources based on workload demand. In this project, EMR Serverless was selected to perform data transformations because it eliminates infrastructure management, reduces operational complexity, and enables cost-efficient execution for intermittent batch workloads. This serverless approach aligns well with the project's daily transformation requirements while still providing the flexibility and performance needed to process Delta Lake tables effectively.

Then I started to create an EMR Serverless ETL job for transformation of data from S3 Bronze bucket to S3 Silver bucket. In AWS console, I clicked on the **EMR Serverless**, then clicked on **Create application (or manage the available applications)**.

After that I will pass to another page, named **EMR Studio**! In this page, I am trying to create an EMR Serverless application. I filled in the application form as below:

- **Name:** project6-bronze-silver-emr
- **Type:** Spark
- **Spark version:** 7.12.0 (matches Glue 4.0 and Delta JAR)
- **Execution role:** IAM role (**project6-emr-role**)
- **Tag:** project: 6

I selected the **Use default settings for batch jobs only** icon and left other defaults and clicked **Create**. Then, I got an **Application ID**, which I will use to submit jobs.



The screenshot shows the EMR Studio interface with the 'Applications' section selected. The main table displays one application entry:

Name	Status	Application ID	Type	Release version	Architecture	Last modified (UTC-8:00)
project6-bronze-silver-emr	Started	00g2mkec3Bghf109	Spark	emr-7.12.0	x86_64	Jan 15, 2026, 15:56

To submit the EMR ETL job, I downloaded 4 necessary **JAR dependencies** from the **Maven Central repository** (<https://mvnrepository.com/>) and uploaded to the JAR bucket **s3://project6-jars/**. The following JAR dependencies were attached:

1. **Delta Lake support:** delta-core_2.12-2.1.0.jar
2. **Amazon S3 filesystem integration:** hadoop-aws-3.3.4.jar
3. **AWS SDK for Java:** aws-java-sdk-bundle-1.12.538.jar

Also, I created two PySpark functions and put inside the code bucket **s3://project6-code-bucket** for transformation of Bronze layer data into Silver layer. I named them: **project6-silver-events.py** and **project6-silver-marketing.py**.

2-5- EMR ETL Jobs Submitting

Now, the EMR is ready to job sumitting. This job submitting will be done using VS Code powershell. To sumbit the job I need this information:

Parameters	Values	
Application-id	00g2mkec38ghf109	
Execution-role-arn	arn:aws:iam::000185048470:role/project6-emr-role	
EntryPoint	s3://project6-code-bucket/project6-silver-events.py s3://project6-code-bucket/project6-silver-marketing.py	
EntryPointArguments	["project6-bronze-bucket", "events/", "project6-silver-bucket"] ["project6-bronze-bucket", "marketing/", "project6-silver-bucket"]	
sparkSubmitParameters	--jars	s3://project6-jars/delta-core_2.12-2.1.0.jar, s3://project6-jars/hadoop-aws-3.3.4.jar, s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar
	--conf	spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
	--conf	spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
	--conf	spark.sql.sources.partitionOverwriteMode=dynamic
	--conf	spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore

I used these two formats to submit two different jobs to EMR ETL separately for events and marketing data transformation:

```
aws emr-serverless start-job-run `--application-id 00g2mkec38ghf109` --execution-role-arn arn:aws:iam::000185048470:role/project6-emr-role` --job-driver "{`"sparkSubmit`": {`"entryPoint`": `s3://project6-code-bucket/project6-silver-events.py` ,`"entryPointArguments`": [`"project6-bronze-bucket`", `events/` , `project6-silver-bucket`],`"sparkSubmitParameters`": `"--jars s3://project6-jars/delta-core_2.12-2.1.0.jar,s3://project6-jars/hadoop-aws-3.3.4.jar,s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf spark.sql.sources.partitionOverwriteMode=dynamic --conf spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore`"}`}"`"
```

```
aws emr-serverless start-job-run `--application-id 00g2mkec38ghf109` --execution-role-arn arn:aws:iam::000185048470:role/project6-emr-role` --job-driver "{`"sparkSubmit`": {`"entryPoint`": `s3://project6-code-bucket/project6-silver-marketing.py` ,`"entryPointArguments`": [`"project6-bronze-bucket`", `marketing/` , `project6-silver-bucket`],`"sparkSubmitParameters`": `"--jars s3://project6-jars/delta-core_2.12-2.1.0.jar,s3://project6-jars/hadoop-aws-3.3.4.jar,s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf spark.sql.sources.partitionOverwriteMode=dynamic --conf spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore`"}`}"`"
```

```
(p311) PS D:\DE\Projects\Projects\Project_6> aws emr-serverless start-job-run \
>> --application-id 00g2mkec38ghf109 \
>> --execution-role-arm arn:aws:iam::000185048470:role/project6-emr-role \
>> --job-driver '{
>>     "sparkSubmit": {
>>         "entryPoint": "s3://project6-code-bucket/project6-bronze-silver.py",
>>         "entryPointArguments": ["project6-bronze-bucket", "marketing/", "project6-silver-bucket"],
>>         "sparkSubmitParameters": "-jars s3://project6-jars/delta-core_2.12-2.1.0.jar,s3://project6-jars/hadoop-aws-3.3.4.jar,s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf spark.sql.sources.partitionOverwriteMode=dynamic --conf spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore"
>>     }
>>   }'
>> }'
>> {
>>     "applicationId": "00g2mkec38ghf109",
>>     "jobRunId": "00g2ml8p8bgkb80b",
>>     "arn": "arn:aws:emr-serverless:us-east-1:000185048470:/applications/00g2mkec38ghf109/jobruns/00g2ml8p8bgkb80b"
}
(p311) PS D:\DE\Projects\Projects\Project_6>
```

After submitting the EMR Jobs, and getting the successful running, I found that two fact tables are created in the Silver layer.

The screenshot shows the AWS EMR Studio interface. On the left, there's a sidebar with navigation links: Dashboard, Workspaces, Query editor (New), Powered by Athena, Serverless (Applications selected), Clusters (EMR on EC2, EMR on EKS), Service integrations (SageMaker Data Wrangler New, Low-code data prep and ML), What's New, Submit feedback, and Logout. The main content area is titled 'EMR Studio > Applications'. It shows a table with one row for the application 'project6-bronze-silver-emr'. The columns include Name, Status (Stopped), Application ID (00g2mkec38ghf109), Type (Spark), Release version (emr-7.12.0), Architecture (x86_64), and Last modified (UTC-8:00) (Jan 16, 2026, 00:04). There are buttons for Start application, Stop application, Actions, and Create application.

This screenshot shows the detailed view for the application 'project6-bronze-silver-emr'. The top navigation bar includes links for EMR Studio, Applications, and the specific application name. Below the title 'project6-bronze-silver-emr', there's a 'How it works' section stating 'With Serverless applications, you can submit data processing jobs to it or perform interactive analysis in Jupyter notebooks from EMR Studio workspaces.' The 'Application details' section provides information such as Application ID (00g2mkec38ghf109), ARN (arn:aws:emr-serverless:us-east-1:000185048470:/applications/00g2mkec38ghf109), Status (Stopped), Creation time (Jan 15, 2026, 15:56 (UTC-8:00)), Last updated (Jan 16, 2026, 00:04 (UTC-8:00)), Type (Spark), Release version (emr-7.12.0), and Architecture (x86_64). At the bottom, tabs for Properties, Batch job runs (selected), Streaming job runs - new, and Tags are visible. The 'Batch job runs' tab shows a table with three rows: one successful run ('Success') and one failed run ('Failed'). Each row includes columns for Job run name, Run status, Job run ID, Retry attempts, Driver log files, Start time (UTC-8:00), Run time, and Resource utilization. Buttons for View application UIs, Clone, Cancel job run, and Submit batch job run are also present.

Amazon S3 > Buckets > project6-silver-bucket

project6-silver-bucket Info

- [Objects](#) [Metadata](#) [Properties](#) [Permissions](#)

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon IAM](#) to explicitly grant them permissions. [Learn more ↗](#)

<input type="checkbox"/> Name	Type
fact_events/	Folder
fact_marketing/	Folder

Amazon S3 > Buckets > project6-silver-bucket > fact_events/

fact_events/

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

Objects (7)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon IAM](#) to explicitly grant them permissions. [Learn more ↗](#)

<input type="checkbox"/> Name	Type
_delta_log/	Folder
event_date=2026-01-14/	Folder
event_date=2026-01-15/	Folder
event_date=2026-01-16/	Folder
event_date=2026-01-17/	Folder
event_date=2026-01-18/	Folder
event_date=2026-01-19/	Folder

Amazon S3 > Buckets > project6-silver-bucket > fact_marketing/

fact_marketing/

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

Objects (34)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon IAM](#) to explicitly grant them permissions. [Learn more ↗](#)

<input type="checkbox"/> Name	Type
_delta_log/	Folder
date=2025-12-16/	Folder
date=2025-12-17/	Folder
date=2025-12-18/	Folder
date=2025-12-19/	Folder
date=2025-12-20/	Folder
date=2025-12-21/	Folder
date=2025-12-22/	Folder
date=2025-12-23/	Folder
date=2025-12-24/	Folder
date=2025-12-25/	Folder

There are two folders inside each of them:

- [_delta_log/](#) folder
- [Parquet](#) files for partitions ([event_date=.../](#) or [date=.../](#))

2-6- Silver Layer Data Validation Using AWS CLI

To validate the data in the Silver layer, all [Parquet](#) files (stored as Delta tables) were downloaded from the Silver S3 bucket to the local machine. The following AWS CLI commands were used to recursively copy only the Parquet files for each dataset:

```
fact_events/
aws s3 cp s3://project6-silver-bucket/fact_events/ ./data/Silver/fact_events/ --recursive --
exclude "*" --include "*.parquet"
```

```

fact_marketing/
aws s3 cp s3://project6-silver-bucket/fact_marketing/ ./data/Silver/fact_marketing/ --recursive
--exclude "*" --include "*.parquet"

```

After downloading the Parquet files, the datasets were explored in a Jupyter Notebook using the Pandas library. The Parquet files were loaded into Pandas DataFrames and carefully reviewed to validate schema consistency, data integrity, and transformation correctness. The validation confirmed that the Silver layer data was successfully processed and met the expected quality standards. Screenshots of two representative DataFrames are provided below.

The screenshot shows a Jupyter Notebook cell with the following code:

```

fact_events

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

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

```

Below the code, a preview of the DataFrame is shown:

	ingestion_time	source	data_created_at	data_created_by	data_event	data_payload_cancel_url	data_payload_created_at	data_pa...
3619	2026-01-16T04:31:18.238616	calendly_webhook	2026-01-16T04:31:18.000000Z	39c5-4...	invitee.created	https://calendly.com/cancellations/b9863fb6...	2026-01-16T04:31:17.359022Z	hodgson.douglas...
3620	2026-01-16T00:51:31.376298	calendly_webhook	2026-01-16T00:51:31.000000Z	39c5-4...	invitee.created	https://calendly.com/cancellations/73230f38-25...	2026-01-16T00:51:30.322986Z	
3621	2026-01-16T04:56:41.102266	calendly_webhook	2026-01-16T04:56:40.000000Z	39c5-4...	invitee.created	https://calendly.com/cancellations/ef9c8d67-fa...	2026-01-16T04:56:38.113459Z	ausbai0926...
3622	2026-01-16T00:14:41.686066	calendly_webhook	2026-01-16T00:14:40.000000Z	39c5-4...	invitee.created	https://calendly.com/cancellations/2535600ced...	2026-01-16T00:14:40.005851Z	emao0705...
3623	2026-01-16T00:18:26.390953	calendly_webhook	2026-01-16T00:18:25.000000Z	39c5-4...	invitee.created	https://calendly.com/cancellations/808b395c-5d...	2026-01-16T00:18:24.702123Z	rtoke11...

5 rows x 68 columns

The screenshot shows a Jupyter Notebook cell with the following code:

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9621 entries, 0 to 9620
Data columns (total 68 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ingestion_time  3624 non-null    object 
 1   source          3624 non-null    object 
 2   data_created_at 3624 non-null    object 
 3   data_created_by 3624 non-null    object 
 4   data_event       3624 non-null    object 
 5   data_payload_cancel_url 3624 non-null    object 
 6   data_payload_created_at 3624 non-null    object 
 7   data_payload_email 3624 non-null    object 
 8   data_payload_event 3624 non-null    object 
 9   data_payload_first_name 0 non-null    object 
 10  data_payload_invitee_scheduled_by 2005 non-null    object 
 11  data_payload_last_name 0 non-null    object 
 12  data_payload_name 3624 non-null    object 
 13  data_payload_new_invitee 0 non-null    object 
 14  data_payload_no_show 0 non-null    object 
 15  data_payload_old_invitee 695 non-null   object 
 16  data_payload_payment 0 non-null    object 
 17  data_payload_reschedule_url 3624 non-null    object 
 18  data_payload_rescheduled 3624 non-null    bool   
 19  data_payload_routing_form_submission 23 non-null   object 
 20  data_payload_scheduling_method 701 non-null   object 
 21  data_payload_status 3624 non-null   object 
 22  data_payload_textReminder_number 2443 non-null  object 
 23  data_payload_timezone 3624 non-null   object 
 24  data_payload_updated_at 3624 non-null   object 
 25  data_payload_uri 3624 non-null   object 
 26  data_payload_reconfirmation_confirmed_at 0 non-null   object 
 27  data_payload_reconfirmation_created_at 1 non-null   object 
 28  data_payload_scheduled_event_created_at 3624 non-null  object 
 29  data_payload_scheduled_event_end_time 3624 non-null  object 
 30  data_payload_scheduled_event_event_type 3624 non-null  object 
 31  data_payload_scheduled_event_meeting_notes_html 0 non-null  object 
 32  data_payload_scheduled_event_meeting_notes_plain 0 non-null  object 

```

```

33 data_payload_scheduled_event_name          3624 non-null  object
34 data_payload_scheduled_event_start_time    3624 non-null  object
35 data_payload_scheduled_event_status        3624 non-null  object
36 data_payload_scheduled_event_updated_at    3624 non-null  object
37 data_payload_scheduled_event_uri          3624 non-null  object
38 data_payload_tracking_salesforce_uuid     0 non-null   object
39 data_payload_tracking_utm_campaign        22 non-null   object
40 data_payload_tracking_utm_content        22 non-null   object
41 data_payload_tracking_utm_medium         17 non-null   object
42 data_payload_tracking_utm_source         1 non-null   object
43 data_payload_tracking_utm_term           22 non-null   object
44 data_payload_questions_and_answers_answer 3341 non-null  object
45 data_payload_questions_and_answers_position 3341 non-null  float64
46 data_payload_questions_and_answers_question 3341 non-null  object
47 data_payload_scheduled_event_invitees_counter_active 3624 non-null  int64
48 data_payload_scheduled_event_invitees_counter_limit 3624 non-null  int64
49 data_payload_scheduled_event_invitees_counter_total 3624 non-null  int64
50 data_payload_scheduled_event_location_join_url 3380 non-null  object
51 data_payload_scheduled_event_location_location 50 non-null   object
52 data_payload_scheduled_event_location_status 3380 non-null  object
53 data_payload_scheduled_event_location_type 3624 non-null  object
54 data_payload_scheduled_event_event_guests_created_at 1 non-null   object
55 data_payload_scheduled_event_event_guests_email 1 non-null   object
56 data_payload_scheduled_event_event_guests_updated_at 1 non-null   object
57 data_payload_scheduled_event_event_memberships_user 3624 non-null  object
58 data_payload_scheduled_event_event_memberships_user_email 3624 non-null  object
59 data_payload_scheduled_event_event_memberships_user_name 3624 non-null  object
60 data_payload_scheduled_event_location_data_id 3260 non-null  float64
61 data_payload_scheduled_event_location_data_password 3260 non-null  object
62 data_payload_scheduled_event_location_data_extra_intl_numbers_url 0 non-null   object
63 data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_city 1146 non-null  object
64 data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_country 3247 non-null  object
65 data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_country_name 3247 non-null  object
66 data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_number 3247 non-null  object
67 data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_type 3247 non-null  object
dtypes: bool(1), float64(2), int64(3), object(62)
memory usage: 1.9+ MB

```

As you can see, there are **68 nested values** in the `fact_events` data.

Next, I filter the **Facebook** events and find that there are **210 events** related to this channel.

Facebook								
df_fact_events[df_fact_events['data_payload_scheduled_event_event_type'] == 'https://api.calendly.com/event_types/d639ecd3-8718-4068-955a-436b10d72c78']								
	ingestion_time	source	data_created_at	data_created_by	data_event	data_payload_cancel_url	data_payload_created_at	data_payload_email
✓ 0.0s	1629 2026-01-14T16:40:57.927Z	calendly_webhook	141716:40:57.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/c0daec90-4f...	2026-01-14T16:40:56.536746Z	srameshn@gmail.com https://api.calendly.com/sch
	1684 2026-01-14T17:04:58.000000Z	calendly_webhook	141717:04:58.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/4b210ca1-54...	2026-01-14T17:04:55.35946Z	abhilashreddykonda12@gmail.com https://api.calendly.com/sch
	1685 2026-01-14T23:43:08.481918	calendly_webhook	141723:43:08.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/e4500662-b4...	2026-01-14T23:43:07.14544Z	biswasamalendu@gmail.com https://api.calendly.com/sch
	1686 2026-01-14T16:33:11.155932	calendly_webhook	141716:33:10.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/3891a3de-b9...	2026-01-14T16:33:08.442961Z	christiankaby@hotmail.com https://api.calendly.com/sch
	1687 2026-01-14T19:01:02.363233	calendly_webhook	141719:01:02.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/0f00c474-a6...	2026-01-14T19:01:00.959698Z	bkeymest@gmail.com https://api.calendly.com/sch
	--	--	--	--	--	--	--	--
	9605 2026-01-20T02:24:51.11476	calendly_webhook	2026-01-20T02:24.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/30c34f4-5c...	2026-01-20T02:23.315557Z	boomwicrox@gmail.com https://api.calendly.com/sch
	9606 2026-01-20T00:30:56.415345	calendly_webhook	2026-01-20T00:30:56.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/33af425-b4...	2026-01-20T00:30:55.643373Z	gblahandolini@gmail.com https://api.calendly.com/sch
	9611 2026-01-20T06:47:32.428321	calendly_webhook	2026-01-20T06:47:32.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/b1c4f21-2f...	2026-01-20T06:47:31.364528Z	tonydegreat1@gmail.com https://api.calendly.com/sch
	9614 2026-01-20T02:37:49.235179	calendly_webhook	2026-01-20T02:37:48.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/6b7adcc4-c2...	2026-01-20T02:37:47.483393Z	krinwears@gmail.com https://api.calendly.com/sch
	9618 2026-01-20T05:30:40.200711	calendly_webhook	2026-01-20T05:30:39.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/5189c6b4-f3...	2026-01-20T05:30:38.470285Z	ghost@alphabetas.com https://api.calendly.com/sch

Next, I filter the **YouTube** events and find that there are **6 events** related to this channel.

YouTube

df_fact_events[df_fact_events['data_payload_scheduled_event_event_type'] == 'https://api.calendly.com/event_types/dbb4ec50-38cd-4bcd-bbff-efb7b5a6f098']

0.0s

	ingestion_time	source	data_created_at	data_created_by	data_event	data_payload_cancel_url	data_payload_created_at	data_payload_email	data_j
1693	2026-01-14T21:15:11.11236	calendly_webhook	2026-01-14T21:15:10.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/f100630-34...	2026-01-14T21:15:09.599437Z	csg2243@gmail.com	https://api.calendly.com/scheduled
3320	2026-01-15T00:26:39.585998	calendly_webhook	2026-01-15T00:26:39.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/c38ec406-2c...	2026-01-15T00:26:38.443718Z	eledweozy@gmail.com	https://api.calendly.com/scheduled
5882	2026-01-17T15:36:07.869818	calendly_webhook	2026-01-17T15:36:07.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/539b1ae0-83...	2026-01-17T15:36:06.389923Z	bruce_hofman@yahoo.com	https://api.calendly.com/scheduled
6454	2026-01-18T18:11:06.250220	calendly_webhook	2026-01-18T18:11:06.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/8447f529-c2...	2026-01-18T18:11:04.759031Z	bruce_hofman@yahoo.com	https://api.calendly.com/scheduled
6465	2026-01-18T18:11:25.723491	calendly_webhook	2026-01-18T18:11:25.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/4d3b1ca1-20...	2026-01-18T18:11:24.426405Z	bruce_hofman@yahoo.com	https://api.calendly.com/scheduled
6498	2026-01-18T08:55:43.745930	calendly_webhook	2026-01-18T08:55:43.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/d273404a-69...	2026-01-18T08:55:42.126713Z	d966768@gmail.com	https://api.calendly.com/scheduled
8558	2026-01-19T23:38:37.929269	calendly_webhook	2026-01-19T23:38:37.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/0beba927-dd...	2026-01-19T23:38:36.748628Z	cynthia.okumu1@gmail.com	https://api.calendly.com/scheduled
9582	2026-01-20T00:03:20.608205	calendly_webhook	2026-01-20T00:03:20.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-4...	invitee.created	https://calendly.com/cancellations/d7eaef-0a...	2026-01-20T00:03:19.420357Z	izzy777@duck.com	https://api.calendly.com/scheduled

8 rows × 68 columns

Next, I filter the **TikTok** events and find that there are **0 events** related to this channel.

df_fact_events[df_fact_events['data_payload_scheduled_event_event_type'] == 'https://api.calendly.com/event_types/bb339e98-7a67-4af2-b584-8dbf95564312']

0.0s

	ingestion_time	source	data_created_at	data_created_by	data_event	data_payload_cancel_url	data_payload_created_at	data_payload_email	data_payload_event	data_payload_first_name	data_payload_scheduled_event_event_memberships_us
0 rows × 68 columns											

Then, focus on another database, **fact_marketing**:

fact_marketing

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

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

```

1.3s

	spend_date	channel	spend
0	2025-12-16	youtube	622.83
1	2025-12-16	facebook	510.96
2	2025-12-16	tiktok	260.15
3	2025-12-17	youtube	775.06
4	2025-12-17	facebook	636.82

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105 entries, 0 to 104
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   spend_date  105 non-null    object  
 1   channel     105 non-null    object  
 2   spend       105 non-null    float64
dtypes: float64(1), object(2)
memory usage: 2.6+ KB

```

2-7- Silver Layer Data Validation Using Amazon Athena

In addition to local validation, the Silver layer data was also validated directly within AWS using **AWS Glue Crawlers** and **Amazon Athena**. Two AWS Glue Crawlers were created to catalog the Silver layer datasets stored in the **fact_events** and **fact_marketing** folders within the Silver S3 bucket. These crawlers automatically inferred the schema and registered the corresponding tables in the AWS Glue Data Catalog:

- Database: **event_db** Table Name: **fact_events**
- Database: **marketing_db** Table Name: **fact_marketing**

Crawlers

Name	State	Last run	Last run timestamp	Log	Table changes from last run
project6-silver-event-crawler	Ready	Succeeded	January 16, 2026 at 08:12:...	View log	1 created
project6-silver-marketing-crawler	Ready	Succeeded	January 16, 2026 at 07:11:...	View log	1 updated

Tables

Name	Database	Location	Classification
fact_events	event_db	s3://project6-silver-bucket/fact_events/	Parquet
fact_marketing	marketing_db	s3://project6-silver-bucket/fact_marketing/	Parquet

After the tables were created in the **Glue Data Catalog**, **Amazon Athena** was used to query the **Silver layer data**. Athena queries were executed to verify schema correctness, partition recognition, record counts, and data consistency across different partitions. Sample analytical queries, including row counts and filtered selections by date, were run successfully without errors, confirming that the Silver layer data was properly structured and accessible for downstream analytics.

This validation step ensured that the Silver layer datasets were fully queryable using serverless SQL, accurately reflected the underlying Delta-generated Parquet files, and were ready for further processing and analysis in the Gold layer.

In Athena, first I did several queries on **fact_events** table. I was looking number of fields on this table. As you saw above, the **Pandas Dataframe** shows there are only **68 columns**. It did not consider the Delta partitioning column. But here in **Amazon Athena**, it reads the Delta Table and shows number of fields as **69 columns**.

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Data' sidebar is open, showing the 'Data source' set to 'AwsDataCatalog', 'Catalog' set to 'None', and 'Database' set to 'event_db'. Under 'Tables and views', there is a list of tables, with 'fact_events' selected. The main area displays two queries:

```
Query 8 : X
SELECT COUNT(*) AS num_columns
FROM information_schema.columns
WHERE table_schema = 'event_db'
AND table_name = 'fact_events';

Query 9 : X
```

Below the queries, the status bar shows 'SQL Ln 5, Col 1'. There are buttons for 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. The 'Query results' tab is selected, showing a single row of results:

#	num_columns
1	69

When I describe the database, I can see the last field is **event_date**, which is the partition key for this Delta Table.

Amazon Athena > Query editor

Editor | Recent queries | Saved queries | Query settings

Data

Data source: AwsDataCatalog

Catalog: None

Database: event_db

Tables and views

Tables (1) < 1 >

- + fact_events Partitioned ...

Views (0) < 1 >

Query 8 : X **Query 9 : X**

1 DESCRIBE event_db.fact_events;

2

3

SQL Ln 1, Col 1

Run again Explain Cancel Clear Create

Query results | **Query stats**

Completed

```
ingestion_time      string
source              string
data_created_at     string
data_created_by     string
data_event          string
data_payload_cancel_url string
data_payload_created_at string
data_payload_email   string
data_payload_event   string
data_payload_first_name string
data_payload_invitee_scheduled_by   string
data_payload_last_name  string
data_payload_name    string
data_payload_new_invitee   string
```

```

data_payload_updated_at string
data_payload_uri string
data_payload_reconfirmation_confirmed_at string
data_payload_reconfirmation_created_at string
data_payload_scheduled_event_created_at string
data_payload_scheduled_event_end_time string
data_payload_scheduled_event_event_type string
data_payload_scheduled_event_meeting_notes_html string
data_payload_scheduled_event_meeting_notes_plain string
data_payload_scheduled_event_name string
data_payload_scheduled_event_start_time string
data_payload_scheduled_event_status string
data_payload_scheduled_event_updated_at string
data_payload_scheduled_event_uri string
data_payload_tracking_salesforce_uuid string
data_payload_tracking_utm_campaign string
data_payload_tracking_utm_content string
data_payload_tracking_utm_medium string
data_payload_tracking_utm_source string
data_payload_tracking_utm_term string
data_payload_questions_and_answers_answer string
data_payload_questions_and_answers_position bigint
data_payload_questions_and_answers_question string
data_payload_scheduled_event_invitees_counter_active bigint
data_payload_scheduled_event_invitees_counter_limit bigint
data_payload_scheduled_event_invitees_counter_total bigint
data_payload_scheduled_event_location_join_url string
data_payload_scheduled_event_location_location string
data_payload_scheduled_event_location_status string
data_payload_scheduled_event_location_type string
data_payload_scheduled_event_event_guests_created_at string
data_payload_scheduled_event_event_guests_email string
data_payload_scheduled_event_event_guests_updated_at string
data_payload_scheduled_event_event_memberships_user string
data_payload_scheduled_event_event_memberships_user_email string
data_payload_scheduled_event_event_memberships_user_name string
data_payload_scheduled_event_location_data_id bigint
data_payload_scheduled_event_location_data_password string
data_payload_scheduled_event_location_data_extra_intl_numbers_url string
data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_city string
data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_country string
data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_country_name string
data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_number string
data_payload_scheduled_event_location_data_settings_global_dial_in_numbers_type string
event_date string

# Partition Information
# col_name      data_type      comment
event_date      string

```

Then look at the header of this table:

#	ingestion_time	source	data_created_at	data_created_by	data_event	data_payload_cancel_url
1	2026-01-15T19:28:50.532961	calendly_webhook	2026-01-15T19:28:50.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf	invitee.created	https://calendly.com/cancellations/1f8a6099-d294-4da7-a806-04cc0cc7191d
2	2026-01-15T19:28:50.532961	calendly_webhook	2026-01-15T19:28:50.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf	invitee.created	https://calendly.com/cancellations/1f8a6099-d294-4da7-a806-04cc0cc7191d
3	2026-01-15T19:28:50.532961	calendly_webhook	2026-01-15T19:28:50.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf	invitee.created	https://calendly.com/cancellations/1f8a6099-d294-4da7-a806-04cc0cc7191d
4	2026-01-15T19:28:50.532961	calendly_webhook	2026-01-15T19:28:50.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf	invitee.created	https://calendly.com/cancellations/1f8a6099-d294-4da7-a806-04cc0cc7191d
5	2026-01-15T19:28:50.532961	calendly_webhook	2026-01-15T19:28:50.000000Z	https://api.calendly.com/users/8ec9d4df-39c5-463a-8b40-bcb5d0a70edf	invitee.created	https://calendly.com/cancellations/1f8a6099-d294-4da7-a806-04cc0cc7191d

After that I did a query on **fact_marketing** table.

The screenshot shows the Amazon Athena Query editor interface. The top navigation bar includes tabs for 'Editor', 'Recent queries', 'Saved queries', and 'Query settings'. A 'Workgroup' dropdown is set to 'primary'. The main area displays a list of five completed queries (Query 8, 9, 10, 11) and one pending query (Query 11). The pending query is a SELECT statement from the 'marketing_db.fact_marketing' table with a LIMIT of 5 rows. Below the queries is a SQL editor with a cursor at 'Ln 2, Col 9'. Action buttons include 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. To the right, there's a 'Reuse query results' link. The 'Query results' tab is selected, showing a table with 5 rows of data. The table has columns: #, spend_date, channel, spend, and date. The data is as follows:

#	spend_date	channel	spend	date
1	2025-12-30	youtube	491.35	2025-12-30
2	2025-12-30	facebook	537.74	2025-12-30
3	2025-12-30	tiktok	210.1	2025-12-30
4	2026-01-11	youtube	408.58	2026-01-11
5	2026-01-11	facebook	662.43	2026-01-11

Below the table, performance metrics are listed: Time in queue: 107 ms, Run time: 938 ms, Data scanned: 0.55 KB. Buttons for 'Copy' and 'Download results CSV' are available.

So, everything is OK with queries, and the Silver layer data are validated successfully.

Step 3 – Analytics & Aggregation (S3 Gold Layer)

3-1- Objective

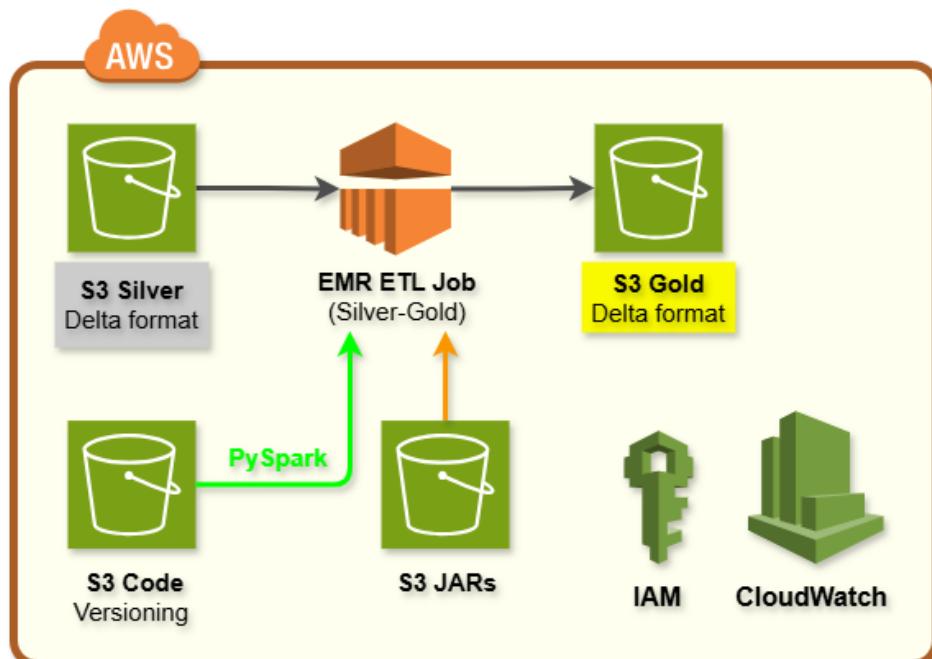
The objective of the **Gold layer** is to transform the curated Silver layer datasets into **business-focused, analytics-ready tables** that directly support reporting, KPI calculation, and decision-making. This layer is designed to aggregate, enrich, and join **events** and **marketing** datasets to produce meaningful metrics that answer core business questions related to marketing performance, lead conversion efficiency, and meeting utilization.

In the Gold layer, data is modeled around business use cases rather than raw events, enabling efficient computation of key performance indicators such as daily bookings by source, cost per booking (CPB) by channel, booking trends over time, channel attribution, time-based booking patterns, and employee meeting load. These transformations involve aggregations, joins between **Calendly events** and **marketing spend data**, and **derivation of time-based dimensions** (date, hour, day of week, and week).

All Gold layer outputs are stored as **Delta tables on top of Amazon S3**, ensuring transactional reliability, schema enforcement, and optimized query performance for downstream analytics. By maintaining a clean separation between Silver (data quality and standardization) and Gold (business logic and metrics), the architecture promotes scalability, reusability, and clear ownership of business definitions.

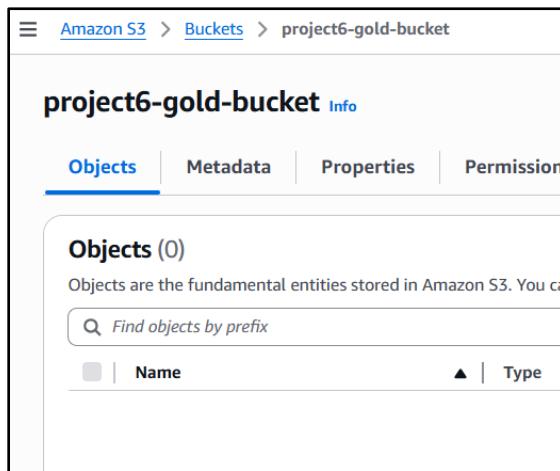
The Gold layer serves as the single source of truth for analytics and visualization, feeding Amazon **Athena queries** and the **Streamlit dashboard**, and enabling stakeholders to gain actionable insights into marketing effectiveness, booking behavior, and operational workload.

This is the Architecture of Gold Layer:

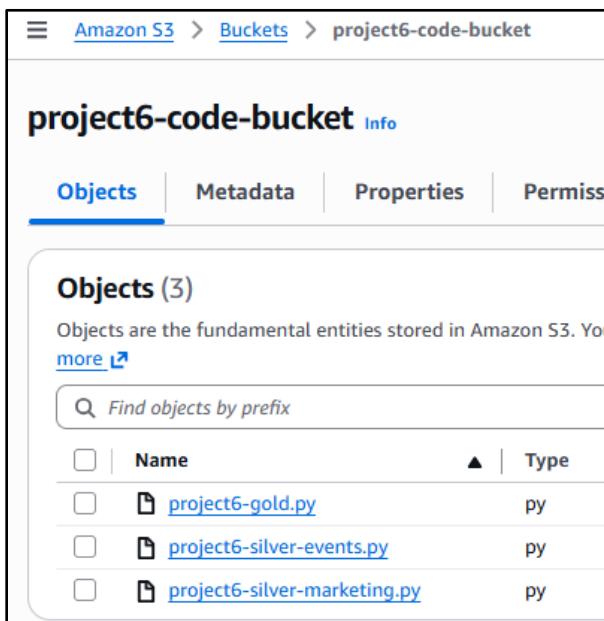


3-2- S3 Gold Layer

I used the Amazon S3 service to create a bucket named <s3://project6-gold-bucket> to store the curated, business-ready datasets produced in the Gold layer. Data from the Silver layer Delta tables is further aggregated, enriched, and optimized to support analytical queries and business intelligence use cases. In this layer, multiple fact datasets are joined and transformed into metric-driven tables that directly support reporting requirements such as cost per booking, daily bookings by channel, and trend analysis. The resulting datasets are stored in Delta Lake format, ensuring data consistency, ACID compliance, and efficient query performance. The Gold layer is structured around analytics-ready tables, enabling seamless consumption by downstream tools such as **AWS Athena** and **Streamlit dashboards**.



Also, as EMR Serverless requires transformation scripts to be written externally in **PySpark** and **stored in an Amazon S3 location**, with the script path explicitly provided at job submission time, I add a new PySpark script to this bucket: [project6-gold.py](#). This script is responsible for creating a few tables in the Gold layer in the Delta-formatted datasets.



3-3- EMR Job (Silver - Gold)

For the Gold layer, **EMR Serverless job** process curated Silver-layer datasets into business-ready, aggregated tables such as daily bookings by channel, cost per booking (CPB), and channel attribution.

Amazon EMR offers two deployment models: **EMR on EC2** and **EMR Serverless**. Again, EMR Serverless was chosen because it eliminates infrastructure management, scales automatically with workload, and provides cost-efficient execution, while delivering the performance and flexibility needed to transform **Delta Lake tables** for downstream analytics and visualization. EMR Serverless provides native Delta Lake support, enabling ACID-compliant writes, incremental updates, schema evolution, and efficient partition handling.

I started to create another EMR Serverless ETL job for aggregation of data from S3 Silver bucket. In AWS console, I clicked on the **EMR Serverless**, then clicked on **Create application (or manage the available applications)**. After that in the **EMR Studio** page, I am trying to create an EMR Serverless application. I filled in the application form as below:

- **Name:** project6-silver-gold-emr
- **Type:** Spark
- **Spark version:** 7.12.0 (matches Glue 4.0 and Delta JAR)
- **Execution role:** IAM role (**project6-emr-role**)
- **Tag:** project: 6

I selected the **Use default settings for batch jobs only** icon and left other defaults and clicked **Create**. Then, I got an **Application ID**, which I will use to submit jobs.

The screenshot shows the EMR Studio interface with the 'Applications' tab selected. On the left sidebar, 'Serverless' is expanded, showing 'Applications' under it. The main area displays a table of applications with the following data:

Name	Status	Application ID	Type	Release version	Architecture	Last modified (UTC-8:00)
project6-silver-gold-emr	Started	00g2o8eatbafn09	Spark	emr-7.12.0	x86_64	Jan 17, 2026, 16:24
project6-bronze-silver-emr	Stopped	00g2mkec58ghf109	Spark	emr-7.12.0	x86_64	Jan 17, 2026, 14:31

3-4- EMR ETL Job Submitting

Now, the EMR is ready to job sumitting. This job submission will be done using VS Code powershell. To submit the job I need this information:

Parameters	Values	
Application-id	00g2o8eatb8fni09	
Execution-role-arn	arn:aws:iam::000185048470:role/project6-emr-role	
EntryPoint	s3://project6-code-bucket/project6-gold.py	
EntryPointArguments	["project6-silver-bucket", "project6-gold-bucket"]	
sparkSubmitParameters	--jars s3://project6-jars/delta-core_2.12-2.1.0.jar, s3://project6-jars/hadoop-aws-3.3.4.jar, s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf spark.sql.sources.partitionOverwriteMode=dynamic --conf spark.delta.logStore.class=org.apache.spark.sql.storage.S3SingleDriverLogStore	

As I do not want to create specific folders in the Gold layer (unlike the Silver layer), then I used this job format to submit the job to EMR ETL in gold layer, which is changed a bit in entryPoint Arguments path:

```
aws emr-serverless start-job-run ` 
--application-id 00g2o8eatb8fni09 ` 
--execution-role-arn arn:aws:iam::000185048470:role/project6-emr-role ` 
--job-driver '{` 
  "sparkSubmit": {` 
    "entryPoint": "s3://project6-code-bucket/project6-gold.py",` 
    "entryPointArguments": ["project6-silver-bucket", "project6-gold-bucket"],` 
    "sparkSubmitParameters": "--jars s3://project6-jars/delta-core_2.12-2.1.0.jar,s3://project6-` 
jars/hadoop-aws-3.3.4.jar,s3://project6-jars/aws-java-sdk-bundle-1.12.538.jar --conf` 
spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf` 
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf` 
spark.sql.sources.partitionOverwriteMode=dynamic --conf` 
spark.delta.logStore.class=org.apache.spark.sql.storage.S3SingleDriverLogStore" ` 
  }` 
}'
```

```
• (p311) PS D:\DE\Projects\Projects\Project_6> aws emr-serverless start-job-run ` 
>>   --application-id 00g2o8eatb8fni09 ` 
>>   --execution-role-arn arn:aws:iam::000185048470:role/project6-emr-role ` 
>>   --job-driver '{` 
>>     "sparkSubmit": {` 
>>       "entryPoint": "s3://project6-code-bucket/project6-gold.py",` 
>>       "entryPointArguments": ["project6-silver-bucket", "project6-gold-bucket"],` 
>>       "sparkSubmitParameters": "--jars s3://project6-jars/delta-core_2.12-2.1.0.jar,s3://project6-jars/hadoop-aws-3.3.4.jar,s3://project6-jars/aws-java-sdk-b` 
undle-1.12.538.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog` 
.DeltaCatalog --conf spark.sql.sources.partitionOverwriteMode=dynamic --conf spark.delta.logstore.class=org.apache.spark.sql.storage.S3SingleDriverLogStor` 
e" ` 
>>     }` 
>>   }` 
>> {` 
  "applicationId": "00g2o8eatb8fni09",` 
  "jobRunId": "00g2o9hrjf500b",` 
  "arn": "arn:aws:emr-serverless:us-east-1:000185048470:/applications/00g2o8eatb8fni09/jobruns/00g2o9hrjf500b" ` 
}` 
○ (p311) PS D:\DE\Projects\Projects\Project_6> []
```

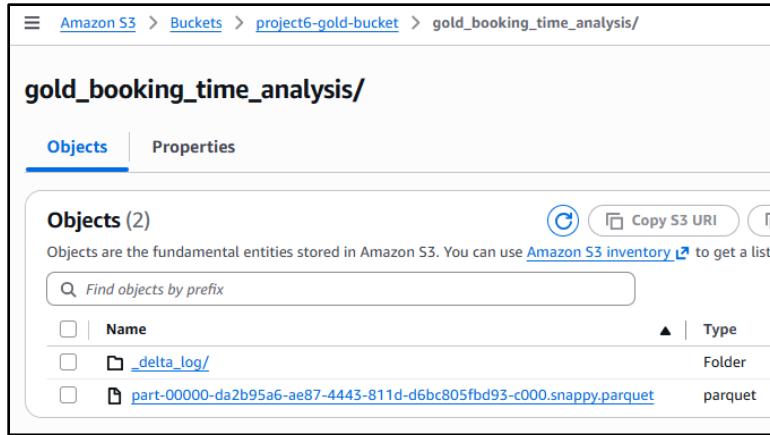
The job is successfully submitted and runned.

The screenshot shows the EMR Studio interface. On the left, there's a sidebar with navigation links like Dashboard, Workspaces, Query editor, Serverless (Applications selected), Clusters, Service integrations, What's New, Submit feedback, and Logout. The main area displays the application 'project6-silver-gold-emr' with its ARN, status (Started), creation time (Jan 17, 2026, 16:24 UTC-8:00), and last updated time (Jan 17, 2026, 16:55 UTC-8:00). Below this, there are tabs for Properties, Batch job runs (selected), Streaming job runs - new, and Tags. Under the Batch job runs tab, it shows one job run: Job run ID 00g2oa9hrjfm5o0b, Status Success, Start time Jan 17, 2026, 18:08, Run time 3 min, 27 secs, and a View logs link.

The screenshot shows the Amazon S3 console. It navigates to the 'Buckets' section and selects 'project6-gold-bucket'. The 'Objects' tab is active, showing 5 objects. A search bar at the top allows finding objects by prefix. The list of objects includes: gold_booking_time_analysis/, gold_channel_attribution/, gold_cpb_by_channel/, gold_daily_bookings_by_channel/, and gold_meeting_load_per_employee/. Each item is a folder icon followed by the folder name.

There are five folders (Delta Tables) in the Gold bucket and inside each of them are like this:

- **_delta_log/** folder
- **Parquet** files for partitions



3-5- Gold Layer Data Validation Using AWS CLI

To validate the data in the Gold layer, all **Parquet** files (stored as Delta tables) were downloaded from the Gold S3 bucket to the local machine. The following AWS CLI commands were used to recursively copy only the Parquet files for each dataset:

```
gold_booking_time_analysis/
aws s3 cp s3://project6-gold-bucket/gold_booking_time_analysis/
./data/Gold/gold_booking_time_analysis/ --recursive --exclude "*" --include "*.parquet"

gold_cpb_by_channel/
aws s3 cp s3://project6-gold-bucket/gold_cpb_by_channel/ ./data/Gold/gold_cpb_by_channel/ --
recursive --exclude "*" --include "*.parquet"

gold_channel_attribution/
aws s3 cp s3://project6-gold-bucket/gold_channel_attribution/
./data/Gold/gold_channel_attribution/ --recursive --exclude "*" --include "*.parquet"

gold_daily_bookings_by_channel/
aws s3 cp s3://project6-gold-bucket/gold_daily_bookings_by_channel/
./data/Gold/gold_daily_bookings_by_channel/ --recursive --exclude "*" --include "*.parquet"

gold_meeting_load_per_employee/
aws s3 cp s3://project6-gold-bucket/gold_meeting_load_per_employee/
./data/Gold/gold_meeting_load_per_employee/ --recursive --exclude "*" --include "*.parquet"
```

After downloading the Parquet files, the datasets were explored in a Jupyter Notebook using the Pandas library. The Parquet files were loaded into Pandas DataFrames and carefully reviewed to validate schema consistency, data integrity, and transformation correctness. The validation confirmed that the Gold layer data was successfully processed and met the expected quality standards. Screenshots of five representative DataFrames are provided below.

1) gold_booking_time_analysis

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

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

✓ 0.0s

```
['../data/Gold/gold_booking_time_analysis\\part-00000-e0288799-c164-4e85-b97a-852401db9d45-c000.snappy.parquet',
 '../data/Gold/gold_booking_time_analysis\\part-00002-4a87abae-5921-4880-b4f2-4603c73d585a-c000.snappy.parquet',
 '../data/Gold/gold_booking_time_analysis\\part-00005-18896e56-3920-4c31-aa17-7bcd2c2f7fe5-c000.snappy.parquet',
 '../data/Gold/gold_booking_time_analysis\\part-00008-d2230bae-9638-41c8-b72c-1fcfdfbeac2ec-c000.snappy.parquet',
 '../data/Gold/gold_booking_time_analysis\\part-00011-4d9cc4f8-543b-4133-8a7d-63366c3ab74d-c000.snappy.parquet']
```

	channel	booking_hour	booking_day_of_week	bookings_count
0	facebook	16.0	5.0	7
1	facebook	15.0	2.0	5
2	facebook	0.0	5.0	1
3	facebook	0.0	1.0	2
4	facebook	14.0	2.0	5

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 192 entries, 0 to 191
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   channel          192 non-null    object 
 1   booking_hour     191 non-null    float64
 2   booking_day_of_week  191 non-null    float64
 3   bookings_count   192 non-null    int64  
dtypes: float64(2), int64(1), object(1)
memory usage: 6.1+ KB
```

I classified the channels into four groups: ['facebook', 'youtube', 'tiktok', 'others']. If I filter only the YouTube, then I can see the details.

2) gold_cpb_by_channel

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

# Read and concatenate them
df_gold_cpb_by_channel = pd.concat([pd.read_parquet(f, engine="fastparquet") for f in files], ignore_index=True)
display(df_gold_cpb_by_channel.head())
df_gold_cpb_by_channel.info()
0.0s
[ '../data/Gold/gold_cpb_by_channel\\part-00000-fde0fbbd-affd-494c-9172-75679f25077d-c000.snappy.parquet' ]
```

	channel	total_bookings_count	total_spend	cpb
0	others	9330	0.00	0.00000
1	facebook	283	20260.42	71.59159
2	youtube	8	20207.43	2525.92875
3	tiktok	0	13593.57	0.00000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   channel          4 non-null      object 
 1   total_bookings_count  4 non-null    int64  
 2   total_spend       4 non-null    float64 
 3   cpb              4 non-null    float64 
dtypes: float64(2), int64(1), object(1)
memory usage: 260.0+ bytes
```

3) gold_channel_attribution

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

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

[ '../data/Gold/gold_channel_attribution\\part-00000-6d7ad8b8-e485-4894-84f3-a20472136a1b-c000.snappy.parquet' ]
```

	channel	total_bookings_count	total_spend	cpb	rank_by_volume	rank_by_cpb
0	others	9330	0.00	0.00000	1	1
1	tiktok	0	13593.57	0.00000	4	1
2	facebook	283	20260.42	71.59159	2	2
3	youtube	8	20207.43	2525.92875	3	3

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   channel          4 non-null      object 
 1   total_bookings_count  4 non-null    int64  
 2   total_spend       4 non-null      float64
 3   cpb              4 non-null      float64
 4   rank_by_volume    4 non-null      int32  
 5   rank_by_cpb       4 non-null      int32  
dtypes: float64(2), int32(2), int64(1), object(1)
memory usage: 292.0+ bytes
```

4) gold_daily_bookings_by_channel

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

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

[ '../data/Gold/gold_daily_bookings_by_channel\\part-00000-9a76f5d6-e8ac-4cbd-8ec0-9c7c0d2fa8ab-c000.snappy.parquet',
  '../data/Gold/gold_daily_bookings_by_channel\\part-00002-79185500-1410-432c-9b97-f6340a54b552-c000.snappy.parquet',
  '../data/Gold/gold_daily_bookings_by_channel\\part-00005-f96788ce-6eae-4933-92a4-528841af9d71-c000.snappy.parquet',
  '../data/Gold/gold_daily_bookings_by_channel\\part-00008-db7d6af0-22c6-4fbe-9c1f-e3574cf6eee1-c000.snappy.parquet',
  '../data/Gold/gold_daily_bookings_by_channel\\part-00011-f9671050-1a8b-47c0-8a22-230b7bd0adfe-c000.snappy.parquet']



|   | booking_date | channel  | bookings_count | booking_week |
|---|--------------|----------|----------------|--------------|
| 0 | 2026-01-18   | facebook | 27             | 3            |
| 1 | 2026-01-25   | facebook | 0              | 4            |
| 2 | 2026-02-13   | facebook | 0              | 7            |
| 3 | 2026-01-14   | facebook | 13             | 3            |
| 4 | 2026-01-31   | facebook | 0              | 5            |



<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   booking_date    116 non-null    datetime64[ns]
 1   channel          116 non-null    object  
 2   bookings_count  116 non-null    int64  
 3   booking_week    116 non-null    int32  
dtypes: datetime64[ns](1), int32(1), int64(1), object(1)
memory usage: 3.3+ KB
```

5) gold_meeting_load_per_employee

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

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

[ '../data/Gold/gold_meeting_load_per_employee\\part-00000-7f216ed2-6ec5-4262-a231-f2e14e6159c3-c000.snappy.parquet' ]
```

	user_id	total_meetings_count
0	030a66c4-ad31-42a1-9568-fda1cf399cef	24
1	da226a81-f340-42b1-87b6-adf44d649dd3	19
2	b7a24d00-d0eb-495f-bb86-241ede6e81f2	20
3	98c10b79-1c4d-4135-8bd6-48bf7ec14925	391
4	e03e8f12-1a32-48e5-b01f-1a5996a50f10	19

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61 entries, 0 to 60
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   user_id          61 non-null    object 
 1   total_meetings_count  61 non-null   int64  
 dtypes: int64(1), object(1)
memory usage: 1.1+ KB
```

So, everything is OK with Delta Tables in the Gold layer and are validated successfully.

Step 4 – 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, well-defined job dependencies, and maintainable deployment practices. Both **real-time event data from the Calendly Webhook API** and **batch marketing data from the DEA public S3 bucket** are ingested into AWS, transformed across the **Bronze, Silver, and Gold layers**, and made available for analytics and visualization **without manual intervention**. Real-time Calendly events are ingested through **AWS Lambda functions exposed via Amazon API Gateway**, which automatically receive and process webhook events in near real time. Batch marketing data is ingested through a scheduled **AWS Glue job**.

To support continuous development and deployment, all **AWS Lambda, AWS Glue, and EMR Serverless scripts** are version-controlled in a GitHub repository and automatically deployed to AWS through a CI/CD workflow whenever changes are introduced.

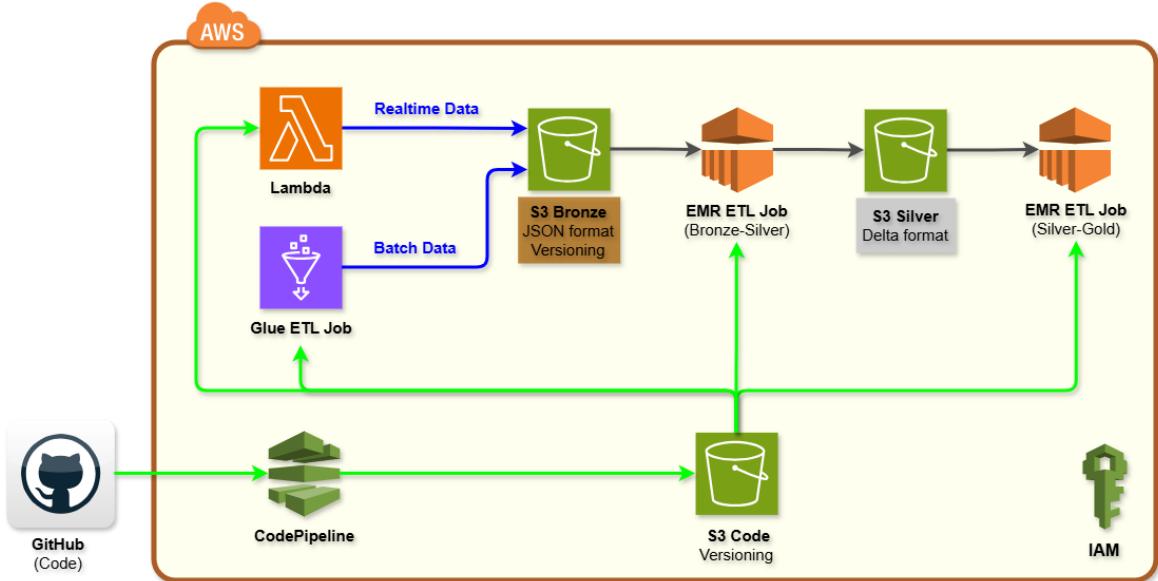
The pipeline is orchestrated using **AWS Step Functions**, which acts as the central control layer. On a daily schedule, a Glue job ingests marketing data into the Bronze layer, after which AWS Step Functions coordinates the execution of the Bronze-to-Silver and Silver-to-Gold transformation jobs in a dependency-driven sequence, ensuring data consistency, fault isolation, and reliable end-to-end processing.

6-2. CI/CD Strategy

This approach enables automated deployment, traceability, and reproducibility of data pipelines. All Python and PySpark scripts for Lambda function, Glue ETL job, and EMR Serverless ETL jobs are stored in a **GitHub** repository: https://github.com/Hadi2468/ELT_Project6/tree/main.

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



6-3. IAM Role

During setting up the AWS CodePipeline, it asked for an IAM role, and I configured a **new role**, and I did not change its' name, named **AWSCodePipelineServiceRole-us-east-1-project6-codepipeline** and tagged with **project: 6**. 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. After that, I added a new “**AWSLambda_FullAccess**” policy to this role, as I want to trigger the CodePipeline with **Lambda Functions** and using the **Step Functions** later.

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.
- **AWSLambda_FullAccess**: Provides permission for CodePipeline to trigger Lambda functions.

Permissions policies (4) Info		
You can attach up to 10 managed policies.		
<input type="text"/> Search		
Policy name ↗	Type	
<input type="checkbox"/> AWSLambda_FullAccess	AWS managed	
<input type="checkbox"/> AWSCodePipelineServiceRole-us-east-1-project6-co...	Customer managed	
<input type="checkbox"/> CodePipeline-CodeConnections-us-east-1-project6-...	Customer managed	
<input type="checkbox"/> CodePipeline-S3Deploy-us-east-1-project6-codepip...	Customer managed	

Also I need a comprehensive access permission for Step Functions, then I created a new IAM role, named **project6-stepfunction** and tagged with **project: 6**. This role grants Step Functions the required permissions to securely connect to all used AWS services in this project.

Permissions policies (8) Info		
You can attach up to 10 managed policies.		
<input type="text"/> Search		
Policy name ↗	Type	
<input type="checkbox"/> AmazonEMRFullAccessPolicy_v2	AWS managed	
<input type="checkbox"/> AmazonEMRServicePolicy_v2	AWS managed	
<input type="checkbox"/> AmazonS3FullAccess	AWS managed	
<input type="checkbox"/> AWSCodePipeline_FullAccess	AWS managed	
<input type="checkbox"/> AWSGlueConsoleFullAccess	AWS managed	
<input type="checkbox"/> AWSLambdaRole	AWS managed	
<input type="checkbox"/> CloudWatchLogsFullAccess	AWS managed	
<input type="checkbox"/> EMRServerlessStartJobRunManagementScopedAccessPolicy-a06ad47f-be65-487f-8a0f-2aad4672a111	Customer managed	

6-4. S3 Code Bucket

The CI/CD pipeline automatically detects changes or additions to Lambda function, Glue job, or EMR Serverless scripts in the GitHub repo and immediately deploys updated scripts to another S3 (code)

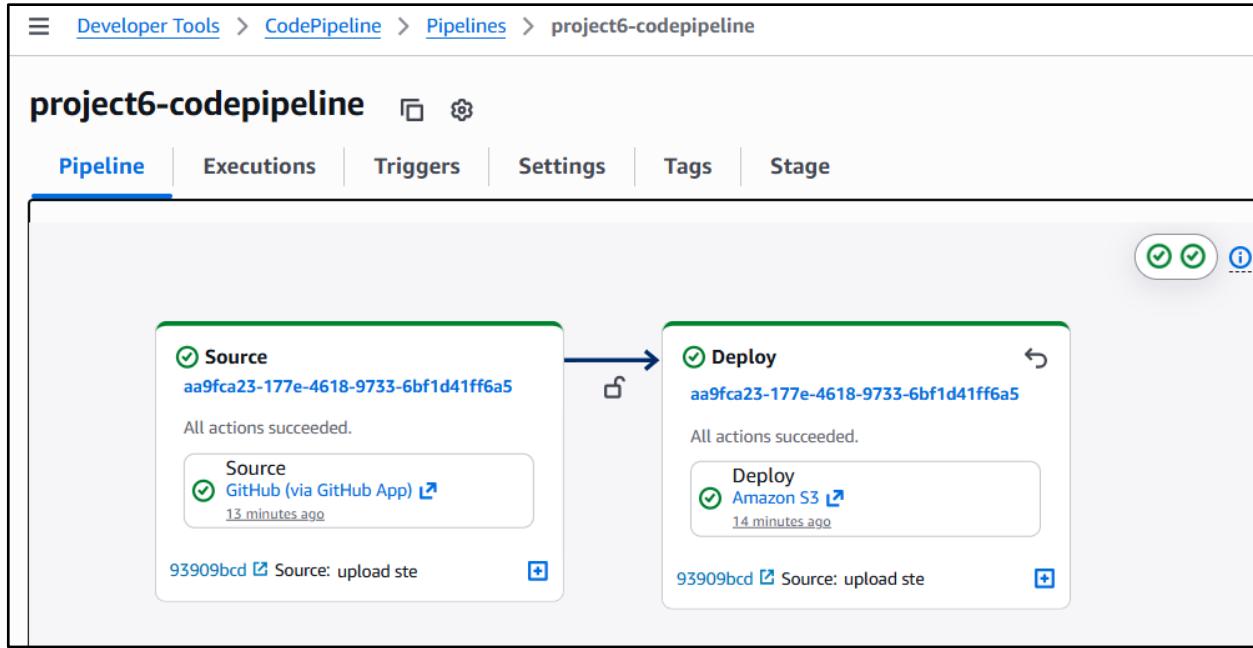
bucket, named <s3://project6-code-bucket> using AWS CodePipeline service. All Python or PySpark jobs reference their scripts directly from the S3 code bucket, ensuring version consistency, easy rollback, and no manual code uploads. I have enabled its **versioning** option to store all versions of codes to support auditability, reprocessing, and downstream transformations.

The image shows two side-by-side screenshots of the Amazon S3 console. The left screenshot shows the contents of the 'Python_codes/' folder in the 'project6-code-bucket'. It lists three objects: 'project6-calendly.py', 'project6-calendly-webhook-ingestion.py', and 'project6-silver-marketing.py'. The right screenshot shows the contents of the 'Spark_codes/' folder in the same bucket. It lists four objects: 'project6-gold.py', 'project6-marketing-bronze.py', 'project6-silver-events.py', and 'project6-silver-marketing.py'. Both screenshots include search and filter functions.

6-5. CodePipeline

Then, I finished setting up the AWS CodePipeline and named it [project6-codepipeline](#) and tagged it with [project: 6](#).

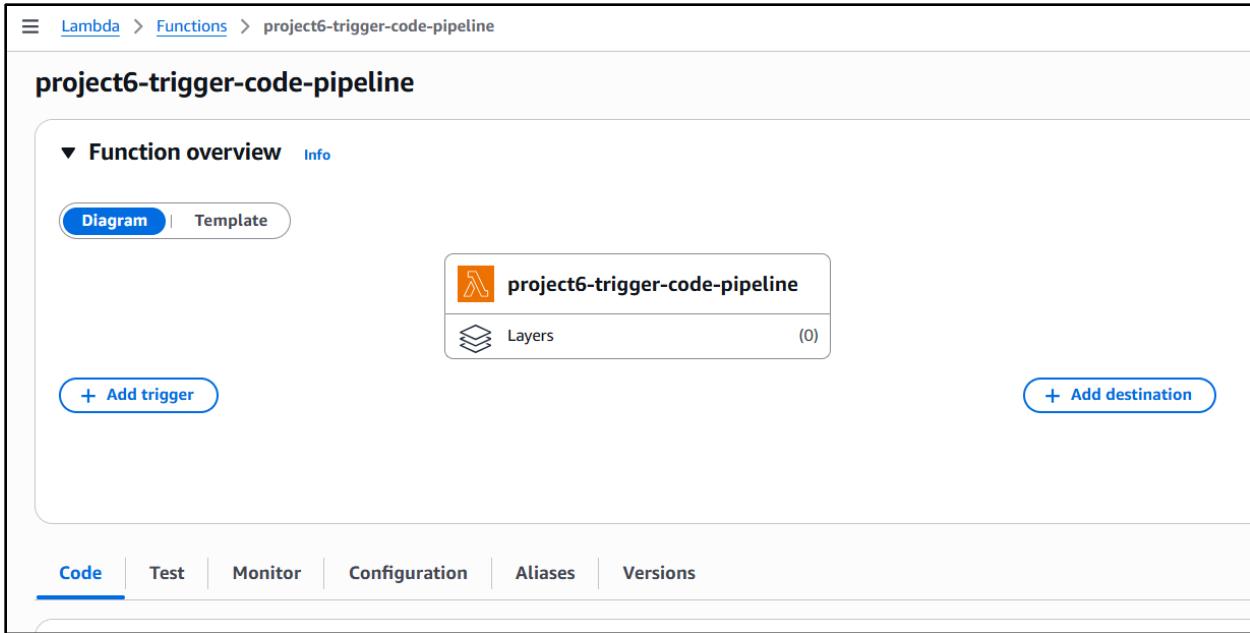
The image shows the AWS CodePipeline console. The left sidebar navigation bar includes 'Developer Tools' (selected), 'CodePipeline' (selected), 'Pipelines' (selected), 'Getting started', and 'Account metrics'. The main area displays a table titled 'Pipelines' with two entries: 'project6-codepipeline' and 'project4-github-codepipeline'. Both pipelines show a 'Succeeded' status, the latest execution started 15 minutes ago, and four recent executions. There are 'View details' buttons for each pipeline entry.



6-6. Lambda

I decided to put CodePipeline as a first step inside the Step Functions, to make sure everything was orchestrated well. Then, I need a simple Lambda function for triggering the CodePipeline. I named this function as [project6-trigger-code-pipeline](#).

The screenshot shows the AWS Lambda console under the 'Functions' section. It displays two functions: 'project6-calendly-webhook-ingestion' and 'project6-trigger-code-pipeline'. Both functions have a status of 'Active'. A search bar at the top is set to 'Search by attributes or search by keyword'.



I used this script for Lambda function for triggering the CodePipeline:

```
import boto3

def lambda_handler(event, context):
    client = boto3.client("codepipeline")
    response = client.start_pipeline_execution(
        name="project6-codepipeline"
    )
    return response
```

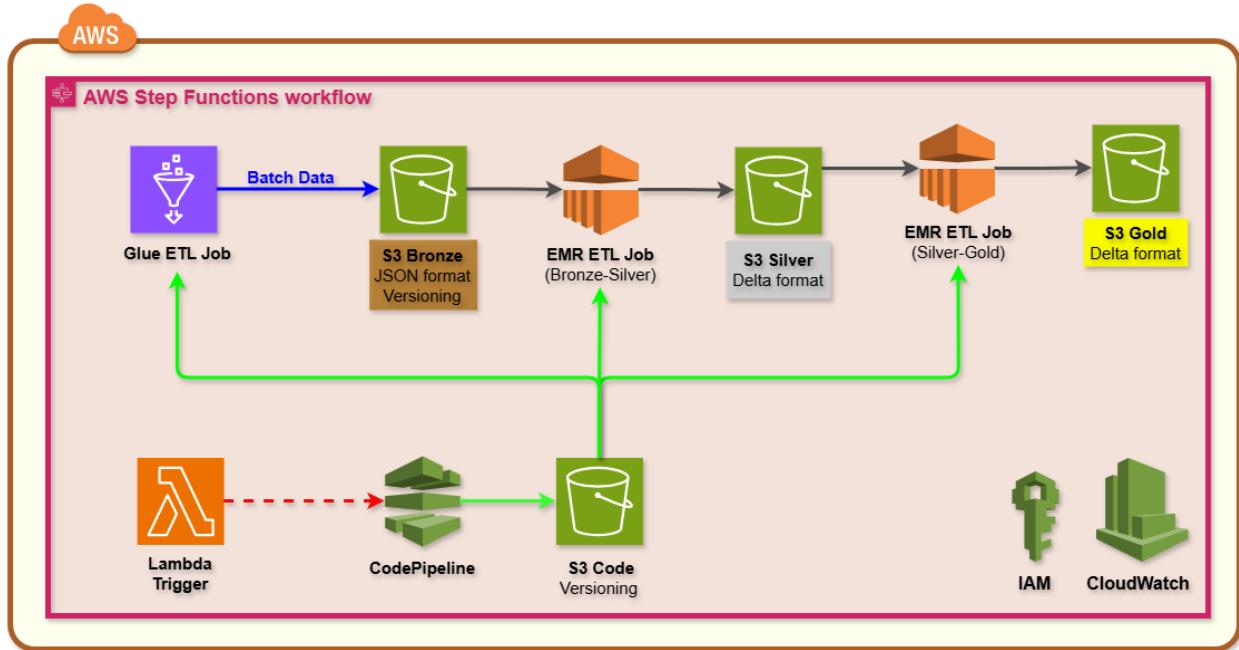
6-7. Orchestration (Step Functions)

For orchestration of the AWS Glue and EMR Serverless ETL jobs, an **AWS Step Functions state machine**, named **project6-state-machine**, is used as the central orchestration engine of the pipeline. This state machine triggered on a **daily schedule** using a **Step Functions time-based trigger** and is responsible for coordinating the execution of all batch processing jobs.

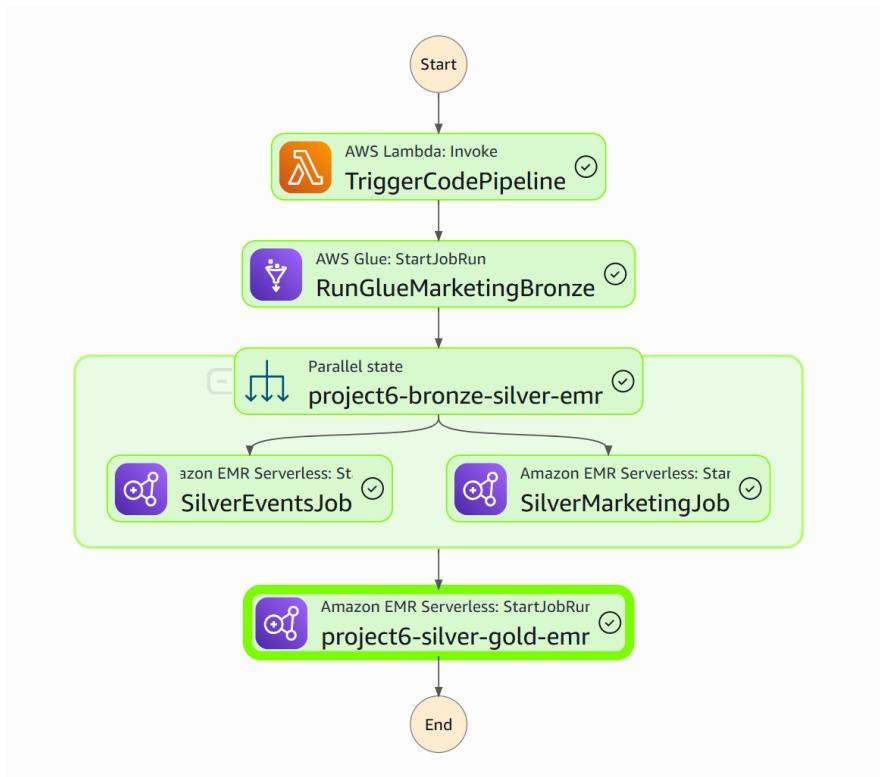
The state machine executes jobs in a **dependency-driven sequence**, ensuring that each processing stage completes successfully before the next stage begins. The execution flow is defined as follows:

```
Glue ETL Job (Data Ingestion to Bronze Layer) →
EMR Serverless ETL Job (Silver Layer Transformation) →
EMR Serverless ETL Job (Gold Layer Aggregation)
```

By centralizing orchestration within **AWS Step Functions**, the pipeline benefits from improved reliability, clear job dependencies, built-in error handling, and easier monitoring of end-to-end execution, making the overall architecture more robust and production-ready.



After creating Step Functions, I executed successfully and orchestrated well. Each step went to be green after finishing, as you can see in this graph:



6-8. 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 Lambda function, Glue ETL job, EMR Serverless ETL jobs 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.

Step 6 – Visualization (Streamlit)

In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket. I have created a **key button (Run ETL Pipeline)** inside the dashboard for manual triggering the project. 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 **Run ETL Pipeline** 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** button in the Streamlit dashboard (manually), the Step Functions is triggered, then all the steps run sequentially:

Lambda will trigger the CodePipeline

CodePipeline started to transfer codes from my GitHub repository into the S3 code bucket

Glue ETL Job gets its script from the code bucket and started to ingest the marketing data from DEA S3 bucket

In parallel two **Silver layer EMR Serveless jobs** run to create the **fact_event** and **fact_marketing** Delta tables in Silver layer

Gold layer EMR Serverless job creates five BI based tables in Delta format

Conclusion

In this project, a fully automated and scalable data pipeline was designed and implemented to process Calendly webhook events in real time. Leveraging AWS Glue for ETL operations and EMR for distributed data processing, the pipeline efficiently transformed raw event data from the Bronze layer into clean and structured Silver-layer Delta tables, ultimately curating high-quality Gold-layer datasets ready for analytics and business insights.

The orchestration of jobs using AWS Step Functions ensured a robust, dependency-based execution flow, while CI/CD integration with Lambda and API Gateway enabled real-time ingestion of events, reducing manual intervention and improving operational efficiency. Logging and monitoring mechanisms further enhanced transparency and maintainability, allowing for seamless tracking of data transformations and job execution status.

This project demonstrates the effectiveness of a well-architected, cloud-based data pipeline in ensuring data reliability, consistency, and readiness for downstream analytics. By implementing a modular, multi-layered structure, the pipeline not only supports current data processing needs but also provides a foundation for future scalability. Potential enhancements include expanding to additional data sources, incorporating advanced monitoring and alerting systems, and optimizing performance for higher event volumes, ensuring long-term resilience and adaptability of the data infrastructure.

Overall, the successful deployment of this pipeline highlights the importance of combining automation, cloud services, and best practices in data engineering to deliver a robust solution that can support real-time decision-making and actionable insights.