# Project 2: Healthcare Metrics Project

## Exploratory Data Analysis (EDA)

### Master Data Analysis

First of all, I have tried to read the master file, **'PBJ_Daily_Nurse_Staffing_Q2_2024.csv'**, using Pandas library, but I got a file integrity or structure based problem, which is formal in large files because of windows-based characters, mostly. This is a classic data-engineering problem. The error is:

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x92 in position 173163: invalid start byte
```

To fix this problem I have tried to read the file with the correct encoding, most likely with **'cp1252'**, (otherwise with **'latin1')**:

```
df = pd.read_csv(r'.\data\master_data.csv', encoding='cp1252')
```

After solving this problem, I have tried to analyze deeply this master data, like checking duplication of rows or columns, missing NA values and histogram of all columns.

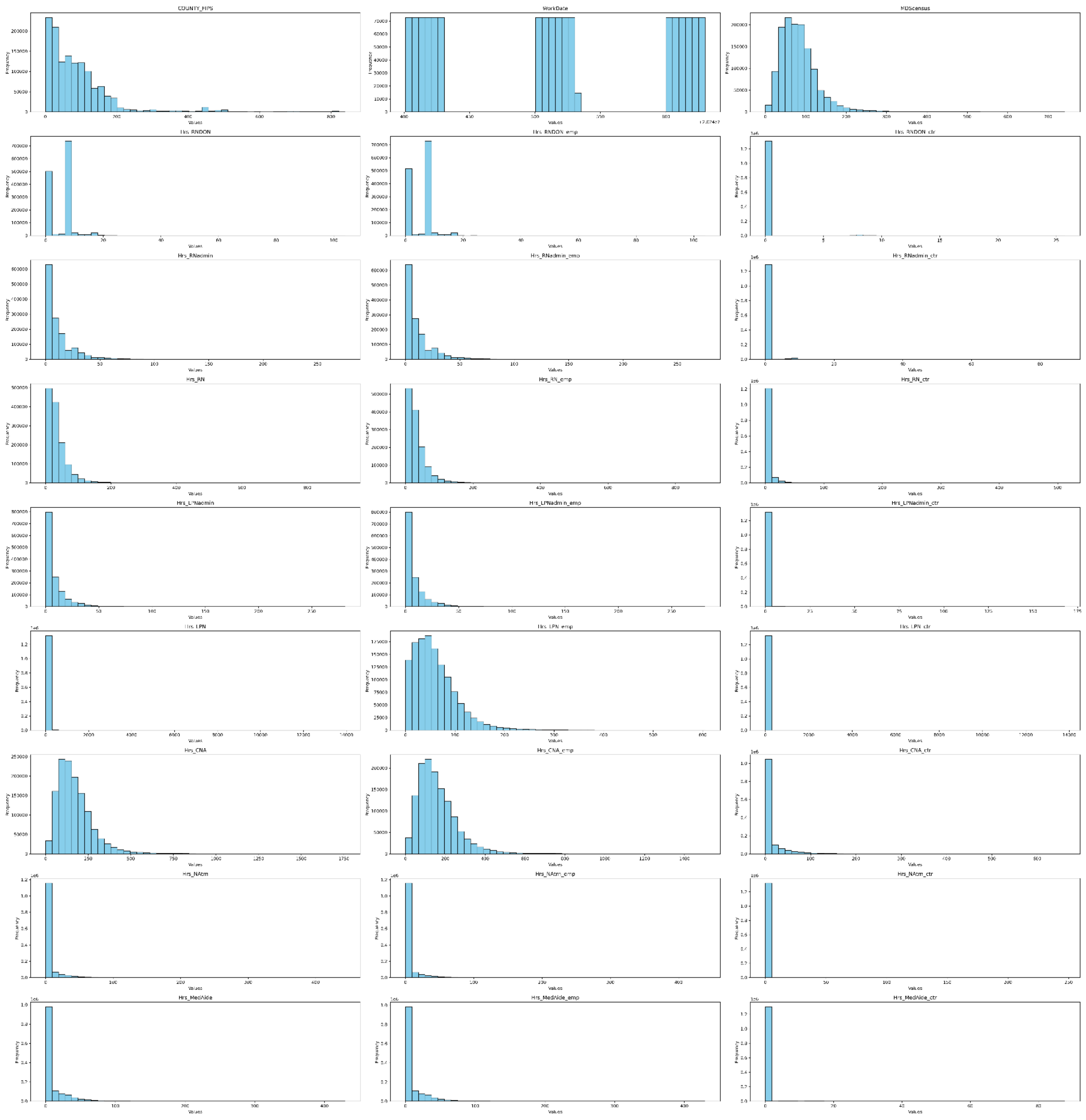
**Data Duplication:**
Check for duplicate rows and columns in master data:

```
No duplicate rows or columns found
```

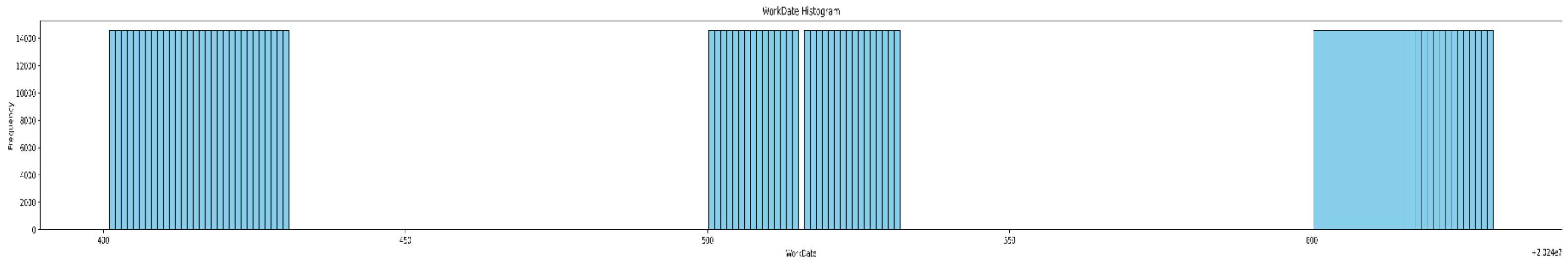
**Missing Values:**
Check for missing values in master data:

```
Is there missing values in the master_dataset?  False
```

**Histogram Analysis:**
Here is the histogram for only numerical columns:


COUNTY_FIPS
WorkDate
MDScensus
Hrs_RNDON
Hrs_RNDON_emp
Hrs_RNDON_ctr
Hrs_RNadmin
Hrs_RNadmin_emp
Hrs_RNadmin_ctr
Hrs_RN
Hrs_RN_emp
Hrs_RN_ctr
Hrs_LPNadmin
Hrs_LPNadmin_emp
Hrs_LPNadmin_ctr
Hrs_LPN
Hrs_LPN_emp
Hrs_LPN_ctr
Hrs_CNA
Hrs_CNA_emp
Hrs_CNA_ctr
Hrs_NAtrn
Hrs_NAtrn_emp
Hrs_NAtrn_ctr
Hrs_MedAide
Hrs_MedAide_emp
Hrs_MedAide_ctr
Values
Frequency

Some columns have a **semi-normal distribution**, but some are **highly skewed**. But for ‘**WorkDate**’ column, we have unified distribution.



Then, I decided to select a primary key for this dataset, which will be unique. I found that the combination of `('PROVNUM', 'WorkDate')` will be a good choice for unique key.

```
unique_key_dups = df[['PROVNUM', 'WorkDate']].duplicated().any()
False
```

I chose all fileds, except three of them, as I need them in downstream analysis.

['PROVNUM', 'PROVNAME', 'CITY', 'STATE', ~~'COUNTY_NAME', 'COUNTY_FIPS', 'CY_Qtr',~~ 'WorkDate', 'MDScensus', 'Hrs_RNDON', 'Hrs_RNDON_emp', 'Hrs_RNDON_ctr', 'Hrs_RNadmin', 'Hrs_RNadmin_emp', 'Hrs_RNadmin_ctr', 'Hrs_RN', 'Hrs_RN_emp', 'Hrs_RN_ctr', 'Hrs_LPNadmin', 'Hrs_LPNadmin_emp', 'Hrs_LPNadmin_ctr', 'Hrs_LPN', 'Hrs_LPN_emp', 'Hrs_LPN_ctr', 'Hrs_CNA', 'Hrs_CNA_emp', 'Hrs_CNA_ctr', 'Hrs_NAtrn', 'Hrs_NAtrn_emp', 'Hrs_NAtrn_ctr', 'Hrs_MedAide', 'Hrs_MedAide_emp', 'Hrs_MedAide_ctr']

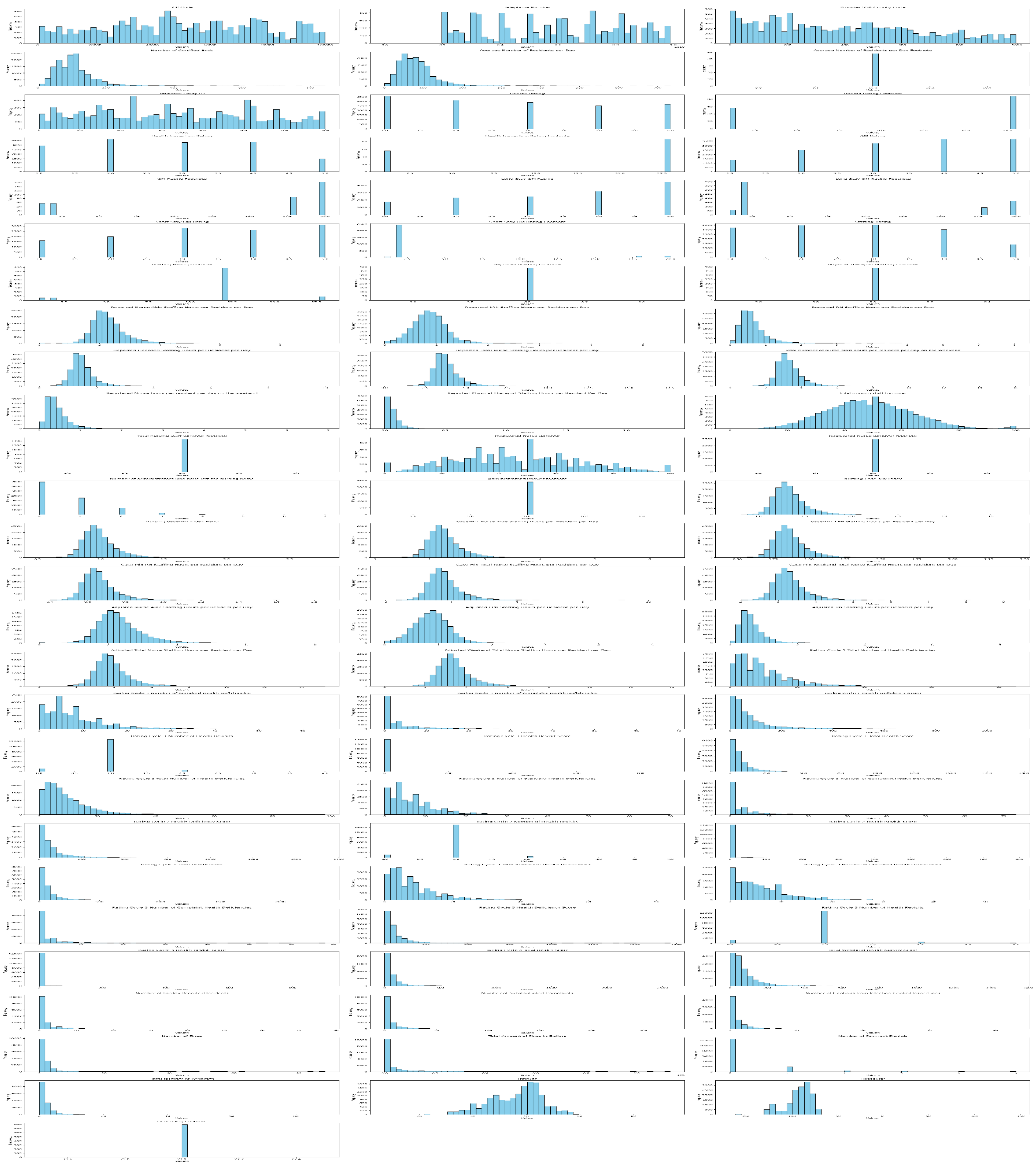
Then, I did some statistical analysis for this dataset:

```
This dataset contains information reagarding quarter of ['2024Q2']
There are 14378 providers in the dataset
There are 5086 cities in the dataset
There are 52 states in the dataset
There are 1672 counties in the dataset
There are 631 MD-Scensus in the dataset
There are 91 different work dates in the dataset starting from 20240401 until 20240630
```

The Juputer Notebook of **master_data_analysis.ipynb** covered this analysis.

## Additional Data Analysis

Next, I tried to analyze 20 additional data. All of them work well. However, based on analysis questions, afrer exploring them, I chose only one additional piece of data as my **dim table**: **‘NH_ProviderInfo_Oct2024.csv’**. This table has a lot of information, which I need for the downstream analysis. For this dataset, I have repeated all above data analysis. There is no duplication in rows or fileds, the data is not corrupted, and there is no missing value. Here is the histogoram of its columns:



Then, I decided to select a primary key for this dataset too, which will be unique. I found that the `'CMS Certification Number (CCN)'`, which is fitted with `'PROVNUM'` in master data, will be a good choice for unique key.

```
unique_key_dups = df ['CMS Certification Number (CCN)'].duplicated().any()
False
```

Finally, I chose only 23 usefull fileds from this dataset, which I need in downstream analysis. The Juputer Notebook of **additional_data_analysis.ipynb** covered this analysis.

Here are my selected fields for **dim** and **fact** tables from Bronze layer!

| **dim_providere** Source: NH_ProviderInfo_Oct2024.csv | |
|---|---|
| **#** | **Fields** |
| **1** | **CMS Certification Number (CCN) [PK]** |
| 2 | Provider Name |
| 3 | Provider Type |
| 4 | State |
| 5 | City/Town |
| 6 | ZIP Code |
| 7 | Number of Certified Beds |
| 8 | Average Number of Residents per Day |
| 9 | Average Number of Residents per Day Footnote |
| 10 | Reported Nurse Aide Staffing Hours per Resident per Day |
| 11 | Reported LPN Staffing Hours per Resident per Day |
| 12 | Reported RN Staffing Hours per Resident per Day |
| 13 | Reported Licensed Staffing Hours per Resident per Day |
| 14 | Reported Total Nurse Staffing Hours per Resident per Day |
| 15 | Total number of nurse staff hours per resident per day on the weekend |
| 16 | Registered Nurse hours per resident per day on the weekend |
| 17 | Reported Physical Therapist Staffing Hours per Resident Per Day |
| 18 | Total nursing staff turnover |
| 19 | Total nursing staff turnover footnote |
| 20 | Registered Nurse turnover |
| 21 | Registered Nurse turnover footnote |
| 22 | Number of administrators who have left the nursing home |
| 23 | Administrator turnover footnote |

| **fact_table** Source: PBJ_Daily_Nurse_Staffing_Q2_2024.csv | |
|---|---|
| **#** | **Fields** |
| **1** | **PROVNUM [PK]** |
| 2 | PROVNAME |
| 3 | CITY |
| 4 | STATE |
| **5** | **WorkDate [PK]** |
| 6 | MDScensus |
| 7 | Hrs_RNDON |
| 8 | Hrs_RNDON_emp |
| 9 | Hrs_RNDON_ctr |
| 10 | Hrs_RNadmin |
| 11 | Hrs_RNadmin_emp |
| 12 | Hrs_RNadmin_ctr |
| 13 | Hrs_RN |
| 14 | Hrs_RN_emp |
| 15 | Hrs_RN_ctr |
| 16 | Hrs_LPNadmin |
| 17 | Hrs_LPNadmin_emp |
| 18 | Hrs_LPNadmin_ctr |
| 19 | Hrs_LPN |
| 20 | Hrs_LPN_emp |
| 21 | Hrs_LPN_ctr |
| 22 | Hrs_CNA |
| 23 | Hrs_CNA_emp |
| 24 | Hrs_CNA_ctr |
| 25 | Hrs_NAtrn |
| 26 | Hrs_NAtrn_emp |
| 27 | Hrs_NAtrn_ctr |
| 28 | Hrs_MedAide |
| 29 | Hrs_MedAide_emp |
| 30 | Hrs_MedAide_ctr |

Now, after exploratory alysis of the master and additional data, and choosing them, here is the project pipliene from data ingestion, doing ELT jobs and visualization.

## Step 1 – Data Ingestion (S3 Bronze Layer)

The first goal is to pull raw master file and additional CSV file from Google Drive into the S3 Bronze bucket. Given the moderate size of the source CSV files (≤ 1 million rows), AWS Lambda provides a simpler and more cost-efficient ingestion mechanism compared to AWS Glue Python Shell, while still fully meeting performance and reliability requirements. Lambda is well suited for this use case because:

- The ingestion workload is batch-based and completes well within Lambda's execution time limits.
- The CSV files can be downloaded from Google Drive using streaming, avoiding memory constraints.
- Lambda has lower operational overhead and cost compared to Glue jobs for lightweight ingestion tasks.

**Implementation:**

1-1- Create an IAM role to grant for Lambda and Glue services to have access to other services, specifically to S3 buckets.
1-2- Create S3 Bronze bucket and activate its versioning to retain all versions of the raw CSV data. The Bronze layer preserves the raw, original CSV files for auditing and traceability.
1-3- Create a Google Drive API key to have access to Google Drive, specifically for large files.
1-4- Store Google Drive credentials securely in AWS Secrets Manager.
1-5- Use Lambda to fetch CSV files from Google Drive. The Lambda can be run manually (for development and testing), or automatically scheduled via Amazon EventBridge (daily or weekly). I chose manual testing. Then based on orcestraion, everything will be run one by one automatically using Glue Workflows orcestration.

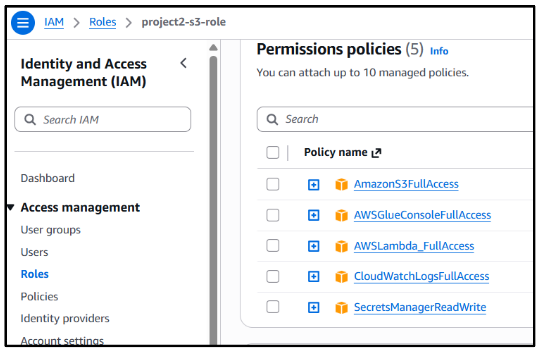
### 1-1- IAM Role

First of all, I have created an IAM role for two AWS services: **Lambda** and **Glue** to have full access to other AWS services:

- AWSGlueConsoleFullAccess (for Glue Workflows and Glue ETL Jobs)
- AWSLambda_FullAccess (for data ingestion)
- AmazonS3FullAccess (for Bronze, Silver, and Gold buckets)
- CloudWatchLogsFullAccess (for logging)
- SecretsManagerReadWrite (for Google Drive API key)

I have named this role **project2-s3-role** and tagged as **project:2**. I put this below code for this role.

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



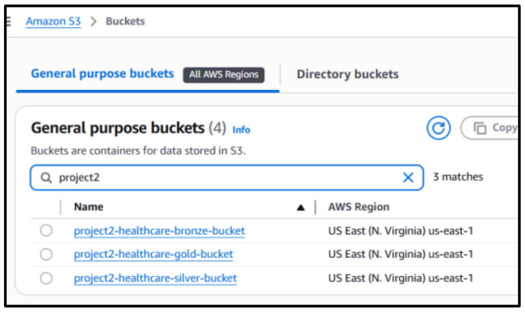
Permissions policies (5) Info

You can attach up to 10 managed policies.

| Policy name |
|---|
| AmazonS3FullAccess |
| AWSGlueConsoleFullAccess |
| AWSLambda_FullAccess |
| CloudWatchLogsFullAccess |
| SecretsManagerReadWrite |

## 1-2- S3 Bronze Bucket

Then, I have tried to create three S3 buckets with name of **project2-healthcare-bronze-bucket**, **project2-healthcare-silver-bucket**, and **project2-healthcare-gold-bucket**, and tagged them as **project:2**.



Amazon S3 > Buckets

General purpose buckets | All AWS Regions | Directory buckets

General purpose buckets (4) Info

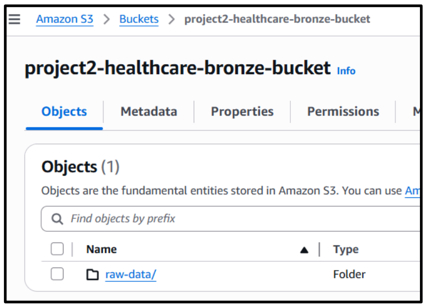
Buckets are containers for data stored in S3.

project2 — 3 matches

| Name | AWS Region |
|---|---|
| project2-healthcare-bronze-bucket | US East (N. Virginia) us-east-1 |
| project2-healthcare-gold-bucket | US East (N. Virginia) us-east-1 |
| project2-healthcare-silver-bucket | US East (N. Virginia) us-east-1 |

For ingestion I used **project2-healthcare-bronze-bucket**, and activated **versioning**. After that, I created a folder inside that bucket with the name of **raw-data**.

`project2-healthcare-bronze-bucket/raw-data`



Amazon S3 > Buckets > project2-healthcare-bronze-bucket

project2-healthcare-bronze-bucket Info

Objects | Metadata | Properties | Permissions

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use

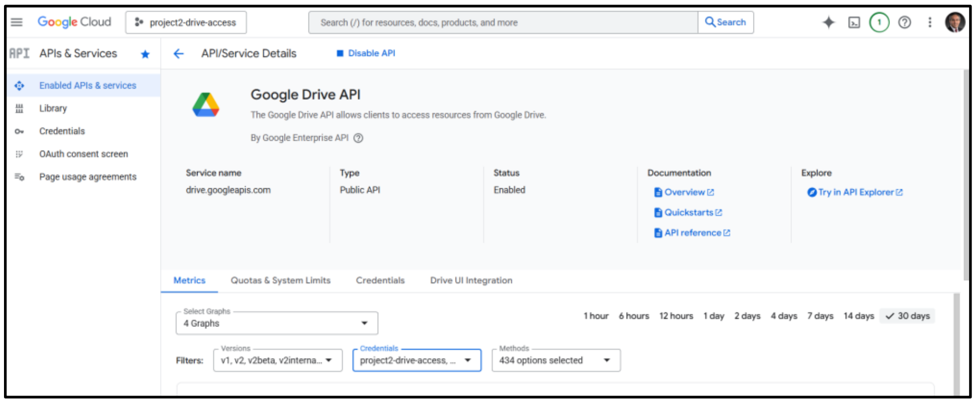
Find objects by prefix

| Name | Type |
|---|---|
| raw-data/ | Folder |

## 1-3- Google Drive API Key

As one of the raw data (master data) is a large file, therefore, I could not download it directly from Google Drive into S3 bucket using Lambda Function. I got virus scan problem from Google side. Therefore, I have tried to create a Google API and for that I need to create a Secret inside AWS Secrets Manager to keep the authenticatno key. This key will be used in Lambda function during connecting to Google Driver.

For the large files, Google Drive did not give access to download them into S3 bucket, and I got virsus scan problem, Then, I decide to create a Google Drive API Key. I named it as **project2-drive-access.**



Google Drive API

The Google Drive API allows clients to access resources from Google Drive.

By Google Enterprise API

| Service name | Type | Status | Documentation | Explore |
|---|---|---|---|---|
| drive.googleapis.com | Public API | Enabled | Overview, Quickstarts, API reference | Try in API Explorer |

Metrics | Quotas & System Limits | Credentials | Drive UI Integration

Select Graphs: 4 Graphs

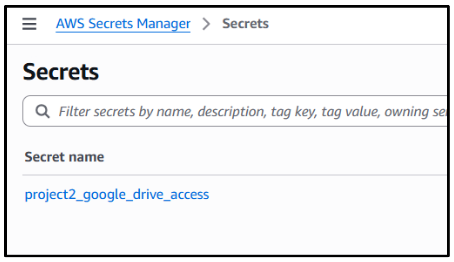
1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days ✓ 30 days

Filters: Versions: v1, v2, v2beta, v2interna... Credentials: project2-drive-access, ... Methods: 434 options selected

## 1-4- Secrets Manager

As one of the raw data (master data) is a large file, therefore, I could not download it directly from Google Drive into S3 bucket using Lambda Function. I got virus scan problem from Google Drive side. Therefore, I have tried to create a Google Drive API and for that reason, I need to create a secret inside AWS Secrets Manager to keep the Google Drive credentials. This key will be used in Lambda function during connecting to Google Driver. The secret name is **project2_google_drive_access** which is tagged as **project:2**. In this secret, I used **Key/Value** option:

```
Key: project2_google_service_account
Value: JSON file { "type": "service_account",   "project_id": "project2-drive-access", *****}
```



AWS Secrets Manager > Secrets

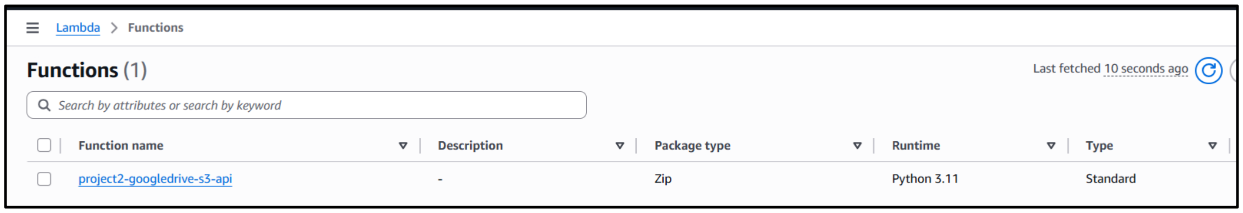
Secrets

Filter secrets by name, description, tag key, tag value, owning se

| Secret name |
|---|
| project2_google_drive_access |

## 1-5- Lambda Function

Then, I created a Lambda function and named it **project2-googledrive-s3-api**, in python 3.11 script format and tagged as **project:2**.



Lambda > Functions

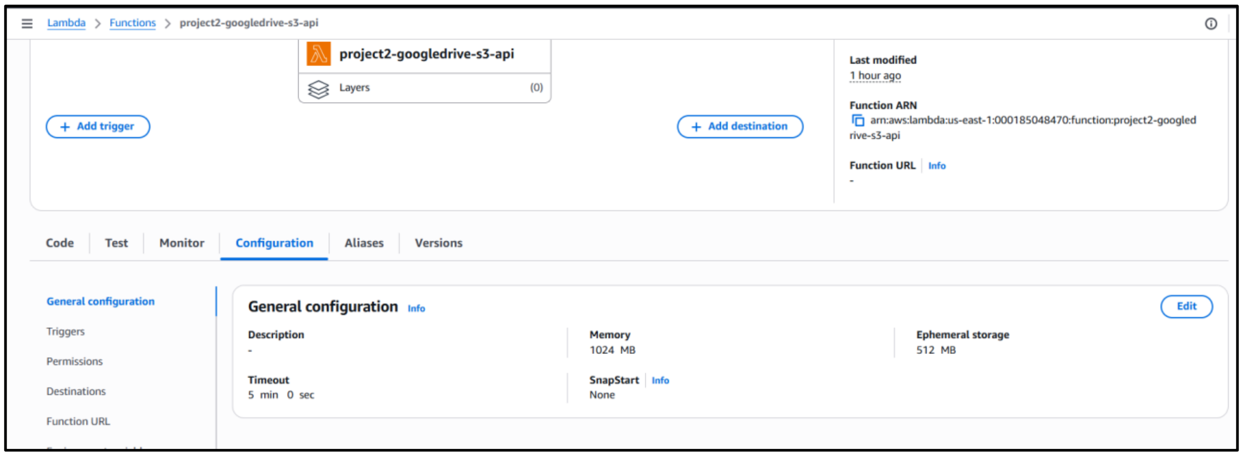
Functions (1)

Last fetched 10 seconds ago

Search by attributes or search by keyword

| Function name | Description | Package type | Runtime | Type |
|---|---|---|---|---|
| project2-googledrive-s3-api | - | Zip | Python 3.11 | Standard |

Our data is about 220 MB, therefore, I have increased the time-out from 3 sec to **5 min** and the memory from 128 MB into **1024 MB** in the configuration page for Lambda.



Lambda > Functions > project2-googledrive-s3-api

project2-googledrive-s3-api

Layers (0)

Last modified
1 hour ago

Function ARN
arn:aws:lambda:us-east-1:000185048470:function:project2-googledrive-s3-api

Function URL Info
-

Code | Test | Monitor | Configuration | Aliases | Versions

General configuration Info

Description
-

Memory
1024 MB

Ephemeral storage
512 MB

Timeout
5 min 0 sec

SnapStart Info
None

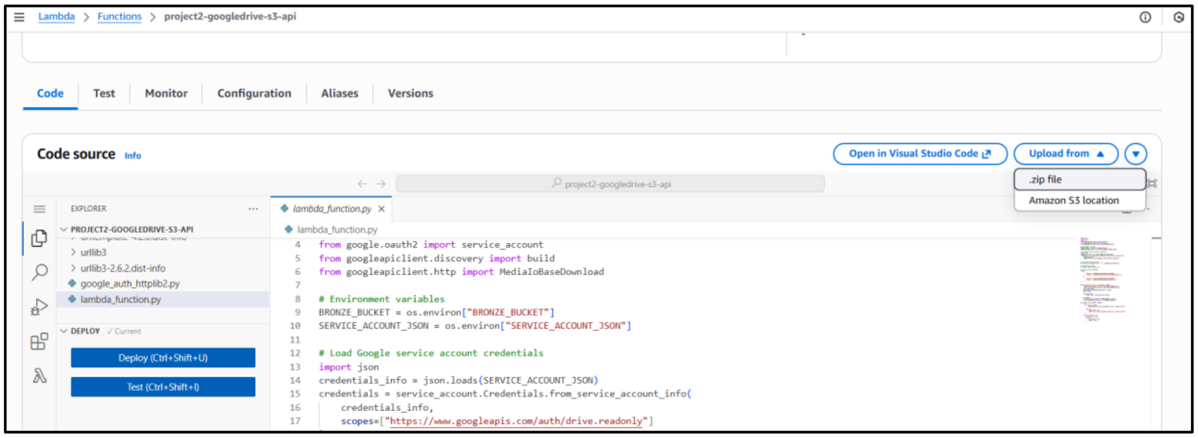
Then, I created a **lambda_function.py** file and put all required import libraries, helper functions, and **lambda_handler function** in this file.
Also, as Lambda doesn't have **google-api-python-client, google-authentication libraries,** and **boto3** preinstalled, then I have run this code in terminal to create a folder named **lambda_package/** in my local machine to create and install dependencies for Google and Lambda in the target folder:

```
mkdir .\Python_codes\lambda_packageee

pip install --target .\Python_codes\lambda_packageee `
boto3 `
google-api-python-client `
google-auth `
google-auth-httplib2 `
google-auth-oauthlib
```

Then, inside `lambda_package` there are a lot of folders and a file with name `google_auth_httplib2.py`. I must add the above `lambda_function.py` file to this folder too, manually. Then, inside this folder, I select all of them with mouse and zip them. The zip file should be like this:

```
lambda_package.zip/
├─ lambda_function.py
├─ google_auth_httplib2.py
├─ google/
├─ ...
├─ ...
```

At the end, I zipped the folder of **lambda_package/** and uploaded **`lambda_package.zip`** file as Lambda deployment package in **Lambda → Code → Upload from → .zip file** section.

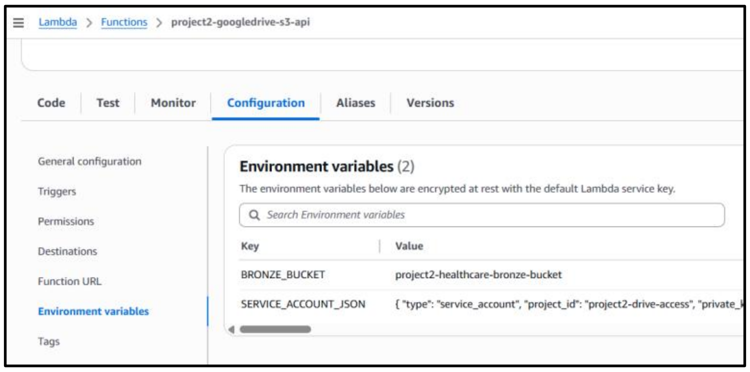


```
from google.oauth2 import service_account
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload

# Environment variables
BRONZE_BUCKET = os.environ["BRONZE_BUCKET"]
SERVICE_ACCOUNT_JSON = os.environ["SERVICE_ACCOUNT_JSON"]

# Load Google service account credentials
import json
credentials_info = json.loads(SERVICE_ACCOUNT_JSON)
credentials = service_account.Credentials.from_service_account_info(
    credentials_info,
    scopes=["https://www.googleapis.com/auth/drive.readonly"]
```

After fixing everythings and giving IAM , I will give the Secrets Manager permission using adding environmnt variables in the configuration page.

**Set environment variables in Lambda:**

- **`Key:`** `BRONZE_BUCKET`
- **`Value:`** `project2-healthcare-bronze-bucket`
- **`Key:`** `SERVICE_ACCOUNT_JSON`
- **`Value:`** `{ "type": "service_account",   "project_id": "project2-drive-access", *****}`



Environment variables (2)

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

| Key | Value |
|---|---|
| BRONZE_BUCKET | project2-healthcare-bronze-bucket |
| SERVICE_ACCOUNT_JSON | { "type": "service_account", "project_id": "project2-drive-access", "private_ |

After running the Lambda funtion, this function will download all csv files from Google Drive and put them into the Bronze S3 bucket, in the raw_data folder. Now, it is ready to run the test. But, I need to create the Glue jobs for the Silver and Gold layers, first.

## Step 2 – Transformation (S3 Silver Layer)

In this step, the goal is to clean and standardize the raw data, organizing it into dimensional and fact tables for downstream analysis.

**Implementation:**

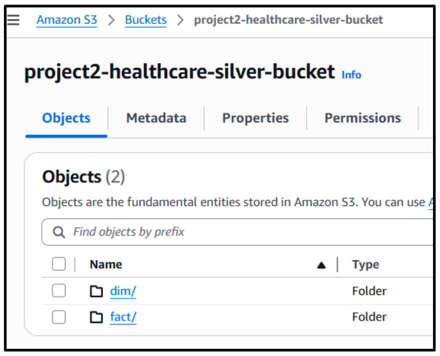
2-1- Use another S3 bucket for silver layer, **project2-healthcare-silver-bucket**.

2-2- Create and run an AWS Glue PySpark Job to perform the following tasks:

- Enforce schemas for dimensional and fact tables
- Normalize column names
- Correct data types (if needed)
- Remove duplicates (if needed)
- Handle missing values and outliers (if needed)
- Convert data to Parquet to improve performance and scalability

### 2-1- S3 Silver Bucket

In the Silver layer, for bucket **project2-healthcare-silver-bucket**, I have created two folders named **dim** and **fact**, without versioning.


Amazon S3 > Buckets > project2-healthcare-silver-bucket
project2-healthcare-silver-bucket Info
Objects
Metadata
Properties
Permissions
Find objects by prefix

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use

| Name | Type |
|---|---|
| dim/ | Folder |
| fact/ | Folder |

### 2-2- Silver Glue Job:

I created the Silver Glue Job with name of **project2-silver-glue**, added available IAM role to this Glue Job, considered Worker Type is **G 1X**, and number of workers, as minimum as **2** in the configuration. Finally, in the Script section, I put the **python** code, whihch saved at **project2-silver-glue.py** file.

Instead of manually clicking on the **Run** button to start the Glue ETL Job, which works well, I have triggered the Glue job using Glue Workflow to orcestrate the ELT jobs and run Silver Glue job immidiately afte finishing the lambda function and after downloading the data into S3 Bronze layer successfully. Then, our needed dim and fact tables will becreated in the Silver bucket automatically after downloading the raw data from Google Drive.

## Step 3 – Analytics & Aggregation (S3 Gold Layer)

In this step, the goal is to create aggregated, dashboard-ready datasets by joining the dimensional and fact tables from the Silver layer.

**Implementation:**

3-1- Store the outputs in a new S3 bucket, **project2-healthcare-gold-bucket**.

3-2- Create second Glue PySpark Job to generate metrics based on the analytics questions and using dim table.

- Schema enforced for BI analyss
- Convert data format from CSV to Parquet to improve performance and scalability
- **Metrics:**
  - Number of beds and patients by state
  - Bed utilization rate by state based on provider type
  - Nurse (different level) to patient ratio by state

3-3- Create third Glue PySpark Job to generate metrics based on the analytics questions and using fact table.

- Schema enforced for BI analyss
- Convert format WORKDATE to date format
- Extract year, month, and day from WORKDATE
- If there is no contract for each staff type, I put the ratio as 10000
- The Gold layer is optimized for Streamlit consumption and easy visualization.
- **Metrics:**
  - Provider staffing at different levels and census per day
  - Searchable provider information based on provider's name or number
  - Average different levels of staff hours by state based on provider types

### 3-1- S3 Gold Bucket

Finally for the bucket **project2-healthcare-gold-bucket**, I did not create any folder at this time, but after running all of the project with Glue Workflow, three tables will be created in this bucket directly for visualization and BI analysis. (No versioning).

### 3-2- Gold Dim Glue Job

For the Gold layer aggregation, I created two Glue Jobs. The first one is **project2-gold-dim-glue**, which handles the data of dim table in the Silver layer. The configuration of this Glue job is like the previous Glue job, but its Spark code are different, which its **python** code, is in **project2-gold-dim-glue.py** file. This Glue job, read data directly from Silver layer's **dim_table**, and create two new BI-based dim tables of **dim_nurse** and **dim_bed**. These two tables are saving in the **project2-healthcare-gold-bucket**.

Again, instead of manually clicking on the **Run** button to start the Glue ETL Job, which works well, I have triggered the Glue job using Glue Workflow to orcestrate the ELT jobs and run **project2-gold-dim-glue** job immidiately afte finishing the **project2-silver-glue** job successfully. Then, our needed dim tables (**dim_bed** and **dim_nurse**) will be created in the Gold bucket automatically after transformation.

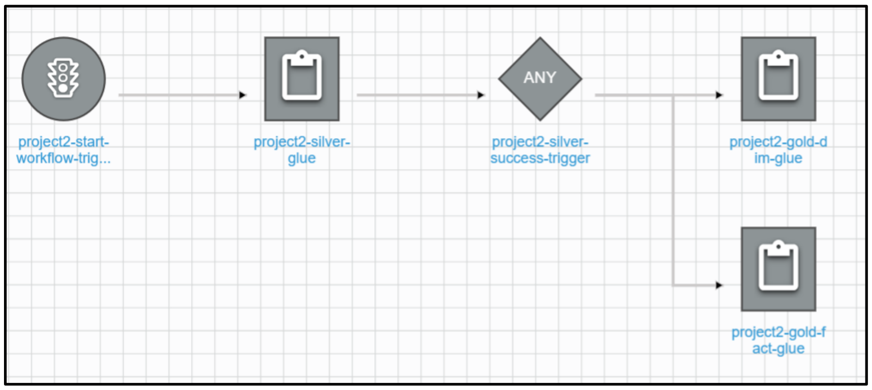
### 3-3- Gold Fact Glue Job

As I mentioned before, for the Gold layer aggregation, I created two Glue Jobs. The first one is **project2-gold-dim-glue**, and the other, **project2-gold-fact-glue**, Glue job handle the fact table. After creating the **project2-gold-fact-glue**, I added available IAM role to this Glue Job, considered 'Worker Type' as '**G 1X'**, and 'number of workers' as minimum as **2** in the configuration, like other glue jobs. Finally, in the Script section, I put the **PySpark** code of **project2-gold-fact-glue.py**, which creates a fact table of **fact_staff** from **fact_table** at tge Silver layer. This table is saving directly in the **project2-healthcare-gold-bucket**.

Again, instead of manually running the **project2-gold-fact-glue** job, I triggered it using Glue Workflow to orcestrate the ELT jobs and run it immidiately afte finishing the **project2-silver-glue** job in parallel with **project2-gold-dim-glue** job. Then, our needed fact table (**fact_staff**) will be created in the Gold bucket automatically after transformation.

## Step 4 – Glue Workflow

As I mentioned, for orchestration, I created a Glue Workflow, named **project2-bronze-silver-gold**, and connect all Glue jobs in this project to this Workflow. Here is the dag of this Workflow:



This Glue Workflow, is triggered On-Demmand. However, in the lambda_function.py, in the section of lambda_handler function, we triggered this Glue Workflow after successfully downloading data from Google Drive into S3 Bronoze layer:

```
### Trigger Glue Workflow to run 3 jobs
response = glue_client.start_workflow_run(
    Name="project2-bronze-silver-gold",
    RunProperties={"input_path": f"s3://{BRONZE_BUCKET}/raw-data/"})
print(f"Glue workflow triggered: {response['RunId']}")
```

Therefore the pipeline orchestration follows a dependency-based, event-driven approach:

### 1. Lambda Ingestion

- Triggered manually by clicking on the Test button (or via EventBridge (daily/weekly)).
- Downloads CSV files from Google Drive and writes them to the Bronze S3 bucket.

### 2. Silver Layer Transformation

- Triggered automatically by Glue Workflow after successful ingestion using lambda function.
- The Silver Glue PySpark job performs data cleaning, standardization, modeling, and converting CSV format data to Parquet.

### 3. Gold Layer Aggregation

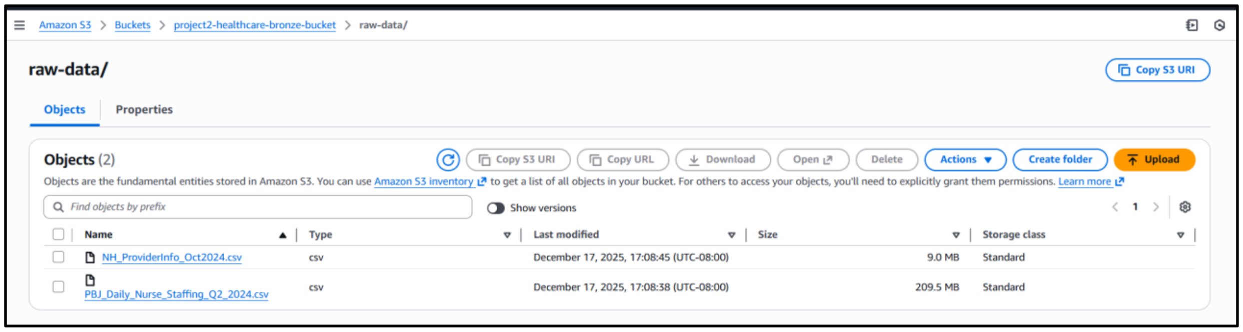
- Triggered automatically by Glue Workflow after successful first AWS Glue job completes successfully.
- The Gold Glue job generates aggregated, analytics-ready datasets and writes them to the Gold S3 bucket in the format of Parquet.

## Step 5 – Demonestration

Now, we want to demonestrate the project and run the Lambda function to download the csv files automatically from Google Drive into Bronze S3 bucket, and after complete of that, immediately trigger the Silver Glue job to do some cleaning stuff on the raw CSV files and convert them into parquet files and save as dim and fact tables inside Silver S3 bucket, by orchestration of Glue Workflow. And finly the second and third Glue jobs will be run after the first Glue job finished, and all final BI-related dim and fact tables will be created in the Gold S3 bucket.

## 5-1- Data Ingestion (S3 Bronze Layer)

To start, I am clicking on the **test** in Lambda function. As you can see in the **lambda_function.py** file, in the lambda handler section, firstly, this function will download all csv files from Google Drive and put them in the S3 Bronze bucket. After that, this function will triger the Glue Workflow of **project2-bronze-silver-gold**. As a result I can see the files downloaded into S3 Bronze layer successfully.



Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

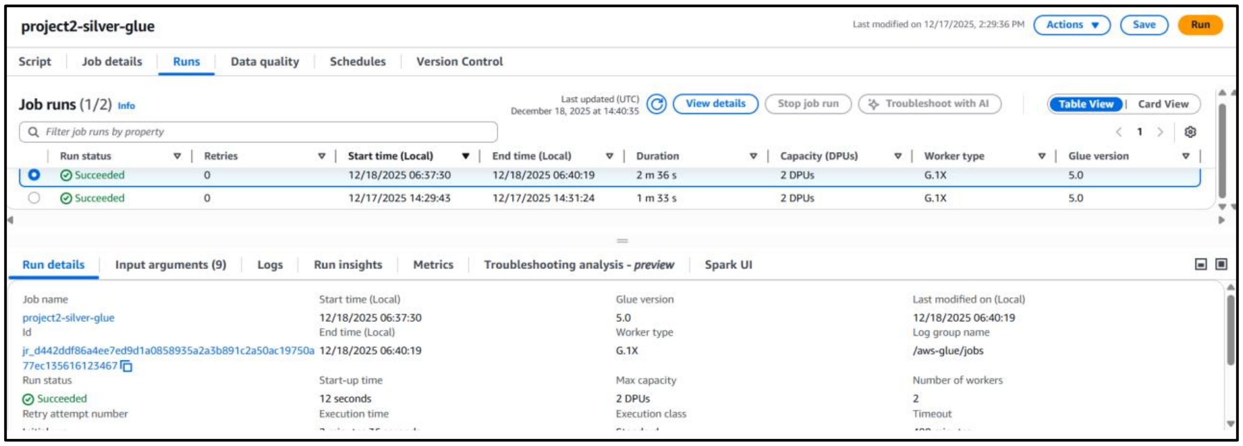
| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| NH_ProviderInfo_Oct2024.csv | csv | December 17, 2025, 17:08:45 (UTC-08:00) | 9.0 MB | Standard |
| PBJ_Daily_Nurse_Staffing_Q2_2024.csv | csv | December 17, 2025, 17:08:38 (UTC-08:00) | 209.5 MB | Standard |

To validate the downloaded data, I have downloaded two CSV files from AWS into my local machine, and checked them carefully. The raw data is successfully downloaded!

```
aws s3 cp s3://project2-healthcare-bronze-bucket/raw-data/ ./data_raw_data/ --recursive --exclude "*"
--include "*.csv"
```

## 5-2- Data Transformation (S3 Silver Layer)

Based on the Glue Workflow orchestration (**project2-bronze-silver-gold**), automatically after downloading the raw data in the Bronze Layer, all of them entered the Silver Glue ETL Job. After finishing, we can see that successfully all parquet dim and fact tables are created in the S3 Silver bucket (**project2-healthcare-silver-bucket**).

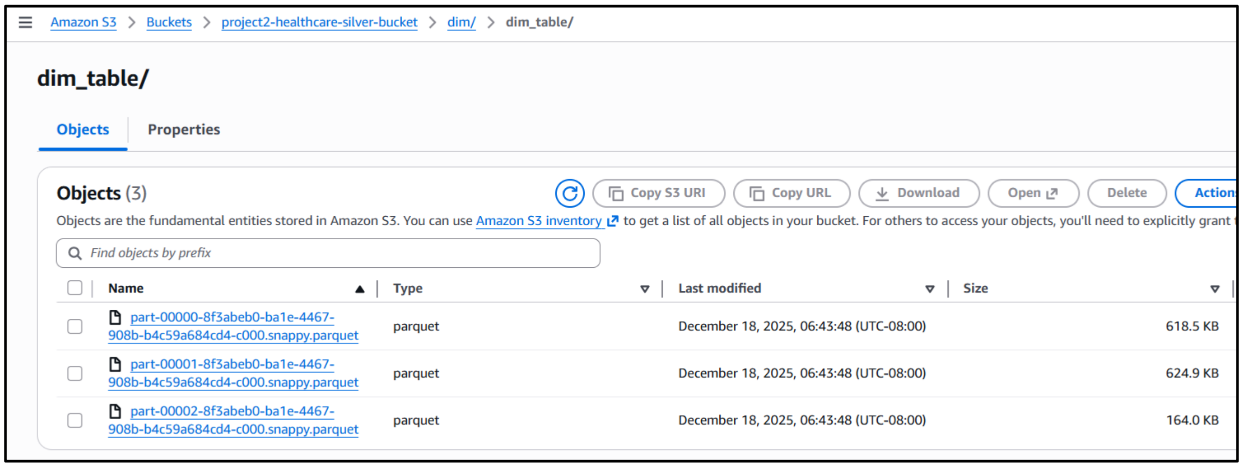


Job runs (1/2) Info

Last updated (UTC) December 18, 2025 at 14:40:35

| Run status | Retries | Start time (Local) | End time (Local) | Duration | Capacity (DPUs) | Worker type | Glue version |
|---|---|---|---|---|---|---|---|
| Succeeded | 0 | 12/18/2025 06:37:30 | 12/18/2025 06:40:19 | 2 m 36 s | 2 DPUs | G.1X | 5.0 |
| Succeeded | 0 | 12/17/2025 14:29:43 | 12/17/2025 14:31:24 | 1 m 33 s | 2 DPUs | G.1X | 5.0 |

| | | | |
|---|---|---|---|
| Job name | Start time (Local) | Glue version | Last modified on (Local) |
| project2-silver-glue | 12/18/2025 06:37:30 | 5.0 | 12/18/2025 06:40:19 |
| Id | End time (Local) | Worker type | Log group name |
| jr_d442ddf86a4ee7ed9d1a0858935a2a3b891c2a50ac19750a77ec135616123467 | 12/18/2025 06:40:19 | G.1X | /aws-glue/jobs |
| Run status | Start-up time | Max capacity | Number of workers |
| Succeeded | 12 seconds | 2 DPUs | 2 |
| Retry attempt number | Execution time | Execution class | Timeout |

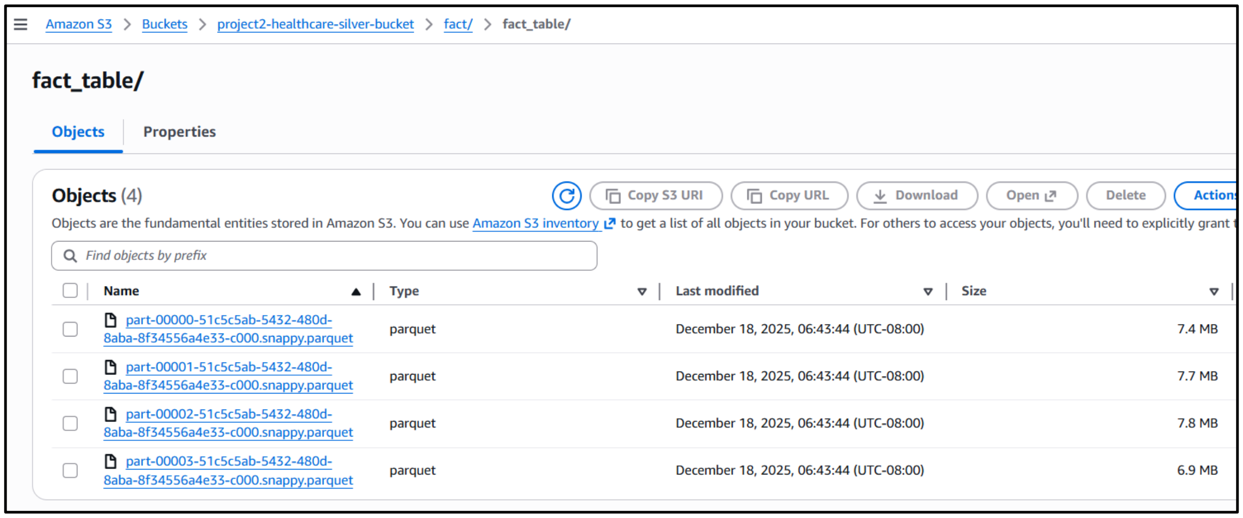

Amazon S3 > Buckets > project2-healthcare-silver-bucket > dim/ > dim_table/

dim_table/

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant

| Name | Type | Last modified | Size |
|---|---|---|---|
| part-00000-8f3abeb0-ba1e-4467-908b-b4c59a684cd4-c000.snappy.parquet | parquet | December 18, 2025, 06:43:48 (UTC-08:00) | 618.5 KB |
| part-00001-8f3abeb0-ba1e-4467-908b-b4c59a684cd4-c000.snappy.parquet | parquet | December 18, 2025, 06:43:48 (UTC-08:00) | 624.9 KB |
| part-00002-8f3abeb0-ba1e-4467-908b-b4c59a684cd4-c000.snappy.parquet | parquet | December 18, 2025, 06:43:48 (UTC-08:00) | 164.0 KB |


Amazon S3 > Buckets > project2-healthcare-silver-bucket > fact/ > fact_table/

fact_table/

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant

| Name | Type | Last modified | Size |
|---|---|---|---|
| part-00000-51c5c5ab-5432-480d-8aba-8f34556a4e33-c000.snappy.parquet | parquet | December 18, 2025, 06:43:44 (UTC-08:00) | 7.4 MB |
| part-00001-51c5c5ab-5432-480d-8aba-8f34556a4e33-c000.snappy.parquet | parquet | December 18, 2025, 06:43:44 (UTC-08:00) | 7.7 MB |
| part-00002-51c5c5ab-5432-480d-8aba-8f34556a4e33-c000.snappy.parquet | parquet | December 18, 2025, 06:43:44 (UTC-08:00) | 7.8 MB |
| part-00003-51c5c5ab-5432-480d-8aba-8f34556a4e33-c000.snappy.parquet | parquet | December 18, 2025, 06:43:44 (UTC-08:00) | 6.9 MB |

Again for validation, I have downloaded the parquet files from AWS into my local machine, and checked them with Pandas library. To download the parquet files into my local machine, in the power shell termianl of Vscode I ran these two scripts:

```
aws s3 cp s3://project2-healthcare-silver-bucket/dim/dim_table/ ./data/dim_table/ --recursive --
exclude "*" --include "*.parquet"

aws s3 cp s3://project2-healthcare-silver-bucket/fact/fact_table/ ./data/fact_table/ --recursive --
exclude "*" --include "*.parquet"
```

After downloading Parquet files for both dim and fact tables, now in Jupyter Notebook and using Pandas library, explored on those two tables for validation.

As we can see, all the cleaning actions are done successfully.

- Selecting 30 of 33 columns for fact_table
- Selecting 23 columns for dim_table
- Upercase all column names
- Replaceing spaces (` `) with underline character (`_`) in column names
- Replaceing slash character (`/`) with underline character (`_`) in column names

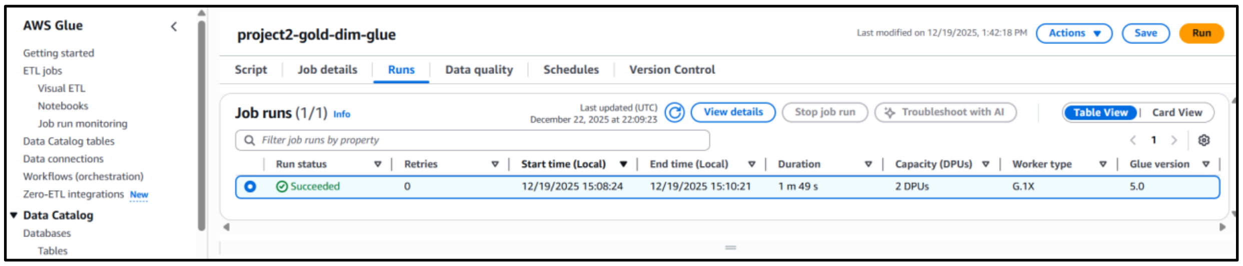
This orchestration strategy ensures clear job dependencies, controlled execution order, failure isolation, minimal operational complexity. Here are the results of transforamtion in the S3 Silver layer:

| **dim_table** (Source: NH_ProviderInfo_Oct2024.csv) | |
|---|---|
| **#** | **Fields** |
| 1 | **PROVIDER_NUM [PK]** |
| 2 | PROVIDER_NAME |
| 3 | PROVIDER_TYPE |
| 4 | STATE |
| 5 | CITY_TOWN |
| 6 | ZIP_CODE |
| 7 | NUMBER_OF_CERTIFIED_BEDS |
| 8 | AVERAGE_NUMBER_OF_RESIDENTS_PER_DAY |
| 9 | AVERAGE_NUMBER_OF_RESIDENTS_PER_DAY_FOOTNOTE |
| 10 | REPORTED_NURSE_AIDE_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 11 | REPORTED_LPN_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 12 | REPORTED_RN_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 13 | REPORTED_LICENSED_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 14 | REPORTED_TOTAL_NURSE_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 15 | TOTAL_NUMBER_OF_NURSE_STAFF_HOURS_PER_RESIDENT_PER_DAY_ON_THE_WEEKEND |
| 16 | REGISTERED_NURSE_HOURS_PER_RESIDENT_PER_DAY_ON_THE_WEEKEND |
| 17 | REPORTED_PHYSICAL_THERAPIST_STAFFING_HOURS_PER_RESIDENT_PER_DAY |
| 18 | TOTAL_NURSING_STAFF_TURNOVER |
| 19 | TOTAL_NURSING_STAFF_TURNOVER_FOOTNOTE |
| 20 | REGISTERED_NURSE_TURNOVER |
| 21 | REGISTERED_NURSE_TURNOVER_FOOTNOTE |
| 22 | NUMBER_OF_ADMINISTRATORS_WHO_HAVE_LEFT_THE_NURSING_HOME |
| 23 | ADMINISTRATOR_TURNOVER_FOOTNOTE |

| **fact_table** (Source: PBJ_Daily_Nurse_Staffing_Q2_2024.csv) | |
|---|---|
| **#** | **Fields** |
| 1 | **PROVNUM [PK]** |
| 2 | PROVNAME |
| 3 | CITY |
| 4 | STATE |
| 5 | **WORKDATE [PK]** |
| 6 | MDSCENSUS |
| 7 | HRS_RNDON |
| 8 | HRS_RNDON_EMP |
| 9 | HRS_RNDON_CTR |
| 10 | HRS_RNADMIN |
| 11 | HRS_RNADMIN_EMP |
| 12 | HRS_RNADMIN_CTR |
| 13 | HRS_RN |
| 14 | HRS_RN_EMP |
| 15 | HRS_RN_CTR |
| 16 | HRS_LPNADMIN |
| 17 | HRS_LPNADMIN_EMP |
| 18 | HRS_LPNADMIN_CTR |
| 19 | HRS_LPN |
| 20 | HRS_LPN_EMP |
| 21 | HRS_LPN_CTR |
| 22 | HRS_CNA |
| 23 | HRS_CNA_EMP |
| 24 | HRS_CNA_CTR |
| 25 | HRS_NATRN |
| 26 | HRS_NATRN_EMP |
| 27 | HRS_NATRN_CTR |
| 28 | HRS_MEDAIDE |
| 29 | HRS_MEDAIDE_EMP |
| 30 | HRS_MEDAIDE_CTR |

## 5-3- Data Aggregation (S3 Gold Layer)

Based on the Glue Workflow orchestration (**project2-bronze-silver-gold**), after the Silver Glue ELT Job finishing, automatically the Gold Glue ELT Jobs (**project2-gold-dim-glue** and **project2-gold-fact-glue**) are running. We can see that successfully all parquet aggregated dim and fact tables are created in the S3 Gold bucket (**project2-healthcare-gold-bucket**).



AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
▼ Data Catalog
Databases
Tables

project2-gold-dim-glue

Last modified on 12/19/2025, 1:42:18 PM Actions ▼ Save Run

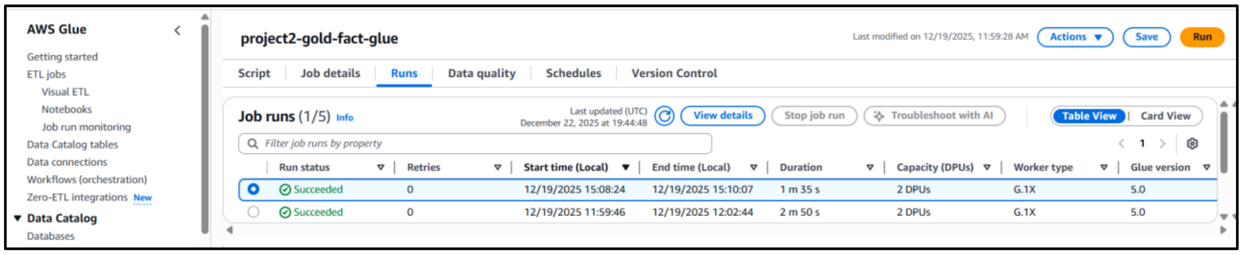
Script | Job details | Runs | Data quality | Schedules | Version Control

Job runs (1/1) Info

Last updated (UTC) December 22, 2025 at 22:09:23 View details | Stop job run | Troubleshoot with AI | Table View | Card View

Filter job runs by property

| | Run status | Retries | Start time (Local) | End time (Local) | Duration | Capacity (DPUs) | Worker type | Glue version |
|---|---|---|---|---|---|---|---|---|
| ● | ✓ Succeeded | 0 | 12/19/2025 15:08:24 | 12/19/2025 15:10:21 | 1 m 49 s | 2 DPUs | G.1X | 5.0 |



AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
▼ Data Catalog
Databases

project2-gold-fact-glue

Last modified on 12/19/2025, 11:59:28 AM Actions ▼ Save Run

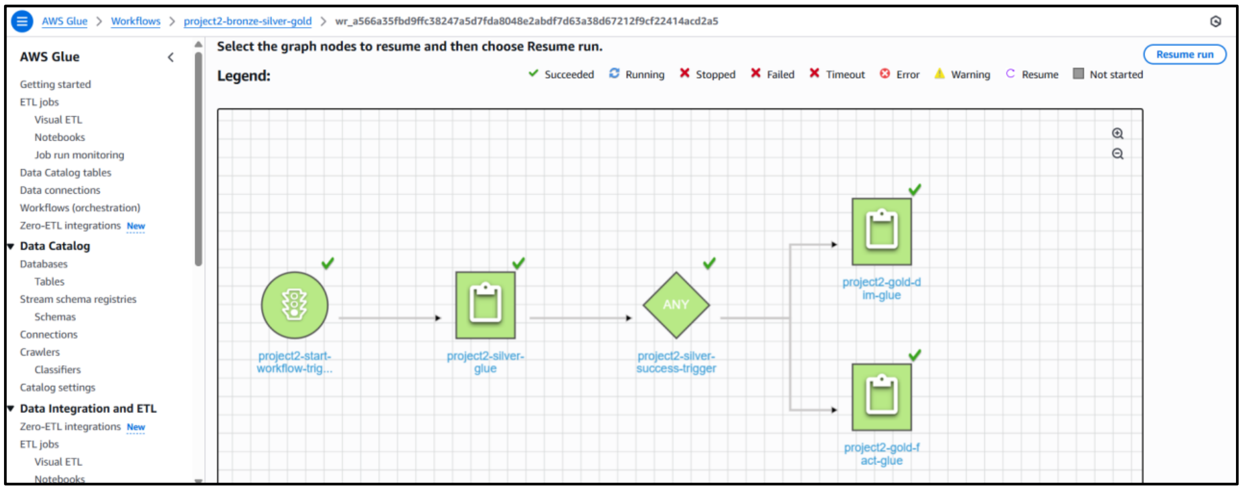
Script | Job details | Runs | Data quality | Schedules | Version Control

Job runs (1/5) Info

Last updated (UTC) December 22, 2025 at 19:44:48 View details | Stop job run | Troubleshoot with AI | Table View | Card View

Filter job runs by property

| | Run status | Retries | Start time (Local) | End time (Local) | Duration | Capacity (DPUs) | Worker type | Glue version |
|---|---|---|---|---|---|---|---|---|
| ● | ✓ Succeeded | 0 | 12/19/2025 15:08:24 | 12/19/2025 15:10:07 | 1 m 35 s | 2 DPUs | G.1X | 5.0 |
| ○ | ✓ Succeeded | 0 | 12/19/2025 11:59:46 | 12/19/2025 12:02:44 | 2 m 50 s | 2 DPUs | G.1X | 5.0 |



AWS Glue > Workflows > project2-bronze-silver-gold > wr_a566a35fbd9ffc38247a5d7fda8048e2abdf7d63a38d67212f9cf22414acd2a5

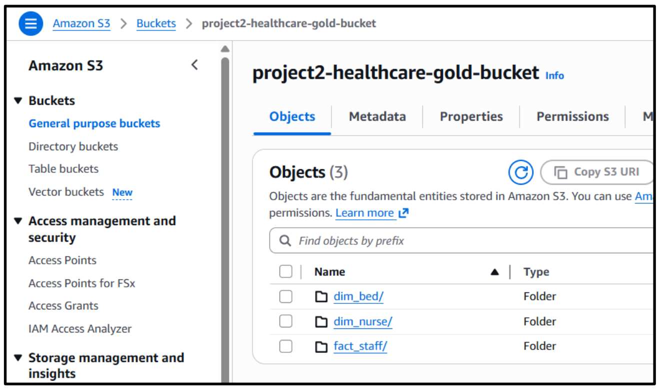
AWS Glue

Getting started
ETL jobs
Visual ETL
Notebooks
Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)
Zero-ETL integrations New
▼ Data Catalog
Databases
Tables
Stream schema registries
Schemas
Connections
Crawlers
Classifiers
Catalog settings
▼ Data Integration and ETL
Zero-ETL integrations New
ETL jobs
Visual ETL
Notebooks

Select the graph nodes to resume and then choose Resume run.

Resume run

Legend: ✓ Succeeded ⟳ Running ✖ Stopped ✖ Failed ✖ Timeout ⊗ Error ⚠ Warning ↻ Resume ■ Not started



Amazon S3 > Buckets > project2-healthcare-gold-bucket

Amazon S3

▼ Buckets
General purpose buckets
Directory buckets
Table buckets
Vector buckets New
▼ Access management and security
Access Points
Access Points for FSx
Access Grants
IAM Access Analyzer
▼ Storage management and insights

project2-healthcare-gold-bucket Info

Objects | Metadata | Properties | Permissions | M

Objects (3) Copy S3 URI

Objects are the fundamental entities stored in Amazon S3. You can use Am
permissions. Learn more

Find objects by prefix

| | Name ▲ | Type |
|---|---|---|
| ☐ | dim_bed/ | Folder |
| ☐ | dim_nurse/ | Folder |
| ☐ | fact_staff/ | Folder |

For validation, I have downloaded the new parquet files from AWS into my local machine and checked them with Pandas library. To download the parquet files into my local machine, in the power shell termianl of Vscode I ran these two scripts:

```
aws s3 cp s3://project2-healthcare-gold-bucket/dim_nurse/ ./data/dim_nurse_table/ --recursive --exclude "*" --
include "*.parquet"

aws s3 cp s3://project2-healthcare-gold-bucket/dim_bed/ ./data/dim_bed_table/ --recursive --exclude "*" --include
"*.parquet"

aws s3 cp s3://project2-healthcare-gold-bucket/fact_staff/ ./data/fact_staff_table/ --recursive --exclude "*" --
include "*.parquet"
```

Here are the results of transforamtion in the S3 Gold layer:

| **dim_bed** | |
|---|---|
| **Columns** | **Type** |
| **PROVIDER_NUM [PK]** | object |
| PROVIDER_NAME | object |
| PROVIDER_TYPE | object |
| STATE | object |
| NUMBER_OF_CERTIFIED_BEDS | int32 |
| AVERAGE_NUMBER_OF_RESIDENTS_PER_DAY | float64 |
| AVERAGE_NUMBER_OF_RESIDENTS_PER_DAY_FOOTNOTE | int32 |
| AVERAGE_NUMBER_OF_RESIDENTS_PER_DAY_TOTAL | float64 |
| BED_UTILIZATION_RATE | float64 |

| **dim_nurse** | |
|---|---|
| **Columns** | **Type** |
| **PROVIDER_NUM [PK]** | object |
| PROVIDER_NAME | object |
| PROVIDER_TYPE | object |
| STATE | object |
| REPORTED_NURSE_AIDE_STAFFING_HOURS_PER_RESIDENT_PER_DAY | float64 |
| REPORTED_LPN_STAFFING_HOURS_PER_RESIDENT_PER_DAY | float64 |
| REPORTED_RN_STAFFING_HOURS_PER_RESIDENT_PER_DAY | float64 |
| REPORTED_TOTAL_NURSE_STAFFING_HOURS_PER_RESIDENT_PER_DAY | float64 |
| TOTAL_NUMBER_OF_NURSE_STAFF_HOURS_PER_RESIDENT_PER_DAY_ON_THE_WEEKEND | float64 |
| REGISTERED_NURSE_HOURS_PER_RESIDENT_PER_DAY_ON_THE_WEEKEND | float64 |
| NURSE_AIDE_TO_PATIENT_RATE | float64 |
| LPN_TO_PATIENT_RATE | float64 |
| RN_TO_PATIENT_RATE | float64 |
| NURSE_TO_PATIENT_RATE | float64 |

| **fact_staff** | |
|---|---|
| **Columns** | **Type** |
| **PROVNUM [PK]** | object |
| PROVNAME | object |
| CITY | object |
| STATE | object |
| **WORKDATE [PK]** | int32 |
| MDSCENSUS | int32 |
| HRS_RNDON_EMP | float64 |
| HRS_RNDON_CTR | float64 |
| HRS_RNADMIN_EMP | float64 |
| HRS_RNADMIN_CTR | float64 |
| HRS_RN_EMP | float64 |
| HRS_RN_CTR | float64 |
| HRS_LPNADMIN_EMP | float64 |
| HRS_LPNADMIN_CTR | float64 |
| HRS_LPN_EMP | float64 |
| HRS_LPN_CTR | float64 |
| HRS_CNA_EMP | float64 |
| HRS_CNA_CTR | float64 |
| HRS_NATRN_EMP | float64 |
| HRS_NATRN_CTR | float64 |
| HRS_MEDAIDE_EMP | float64 |
| HRS_MEDAIDE_CTR | float64 |
| STAFF_RATIO_RNDON | float64 |
| STAFF_RATIO_RNADMIN | float64 |
| STAFF_RATIO_RN | float64 |
| STAFF_RATIO_LPNADMIN | float64 |
| STAFF_RATIO_LPN | float64 |
| STAFF_RATIO_CNA | float64 |
| STAFF_RATIO_NATRN | float64 |
| STAFF_RATIO_MEDAIDE | float64 |

## Step 6 – Visualization (Streamlit)

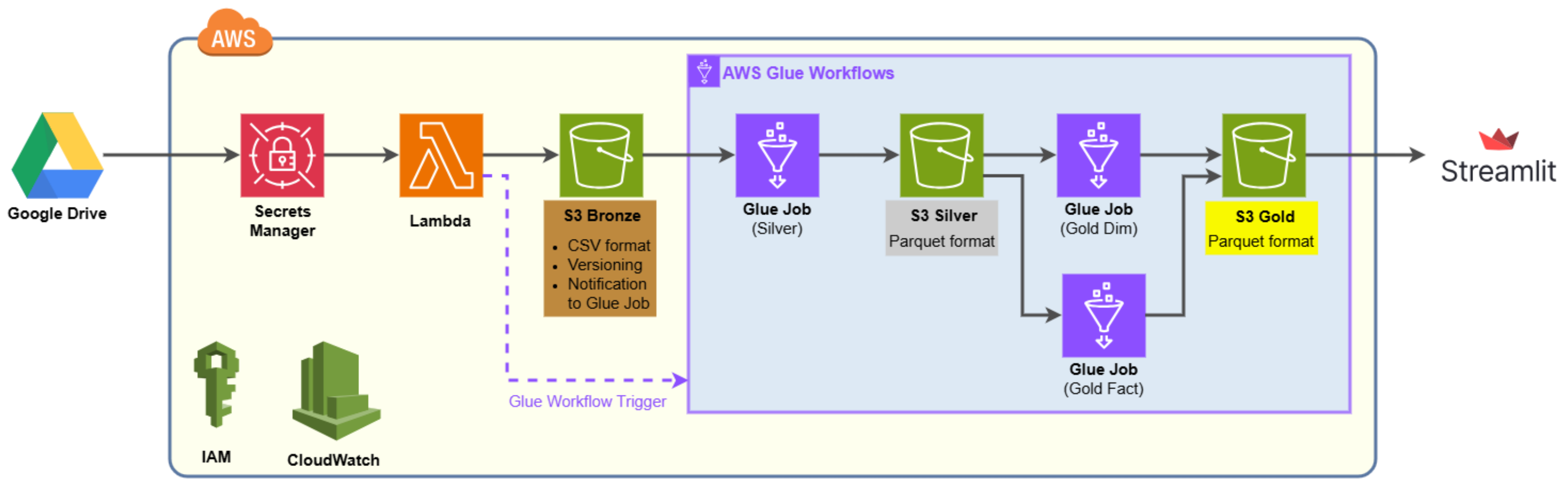
In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket.

**Implementation:**

- I build interactive dashboards for stakeholders with filters, charts, different metrics.
- I triggered lambda function directly from Streamlit.
- Streamlit reads the Gold-layer Parquete files directly from S3.
- I used pandas for data handling.
- No data warehouse or query engine is required. This approach is fully serverless and cost-efficient.
- Dashboards reflect the latest successful Gold Glue job output.

## Conclusion

This solution implements a cost-efficient, serverless batch data pipeline leveraging AWS Lambda, AWS Glue, and Amazon S3, designed around a Bronze–Silver–Gold lakehouse architecture. Raw data is ingested and preserved in the Bronze layer for auditability and traceability, transformed and standardized in the Silver layer, and stored in the Gold layer as analytics-ready Parquet datasets optimized for performance and scalability. Streamlit consumes these curated Parquet files directly from S3, removing the need for additional query engines such as Athena or Redshift while maintaining simple, flexible, and easily extensible architecture.


AWS
AWS Glue Workflows
Google Drive
Secrets Manager
Lambda
S3 Bronze
CSV format
Versioning
Notification to Glue Job
Glue Job (Silver)
S3 Silver Parquet format
Glue Job (Gold Dim)
Glue Job (Gold Fact)
S3 Gold Parquet format
Streamlit
Glue Workflow Trigger
IAM
CloudWatch