Screen shot to show the process of triggering lambda which triggers crawler to create Data Catalogue and Table to access data.csv file in S3 bucket (complete process)

Resource created :-

A screenshot of a computer

AI-generated content may be incorrect.

Output details :-

A screenshot of a computer

AI-generated content may be incorrect.

Initially no crawler when Stacke is run as soon file is loaded in S3 bucket ,lambda will be triggered and crawler will be created along with database name and table

| **Timestamp** | | **Logical ID** | **Status** | | **Detailed status** | | **Status reason** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Timestamp** | **Logical ID** | | | **Status** | | **Detailed status** | | **Status reason** |
| 2025-03-02 13:22:28 UTC+0000 | [st-stack-Lambda-crawler](https://eu-west-2.console.aws.amazon.com/cloudformation/home?region=eu-west-2#/stacks/stackinfo?stackId=arn%3Aaws%3Acloudformation%3Aeu-west-2%3A277707117797%3Astack%2Fst-stack-Lambda-crawler%2F48816100-f769-11ef-8dda-06eedc5dd103) | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:28 UTC+0000 | [MyS3Bucket](https://eu-west-2.console.aws.amazon.com/s3/home?region=eu-west-2&bucket=my-example-s3-bucket-2025) | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:20 UTC+0000 | PassRolePolicy | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:14 UTC+0000 | [MyS3Bucket](https://eu-west-2.console.aws.amazon.com/s3/home?region=eu-west-2&bucket=my-example-s3-bucket-2025) | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:22:14 UTC+0000 | LambdaInvokePermission | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:14 UTC+0000 | LambdaInvokePermission | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:22:13 UTC+0000 | MyS3Bucket | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:22:13 UTC+0000 | LambdaInvokePermission | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:22:13 UTC+0000 | [CreateCrawlerFunction](https://eu-west-2.console.aws.amazon.com/lambda/home?region=eu-west-2#functions/CreateCrawlerOnFileUpload) | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:06 UTC+0000 | MyGlueJob | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:22:05 UTC+0000 | PassRolePolicy | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:22:05 UTC+0000 | [CreateCrawlerFunction](https://eu-west-2.console.aws.amazon.com/lambda/home?region=eu-west-2#functions/CreateCrawlerOnFileUpload) | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:22:04 UTC+0000 | MyGlueJob | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:22:04 UTC+0000 | MyGlueJob | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:22:04 UTC+0000 | PassRolePolicy | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:22:04 UTC+0000 | CreateCrawlerFunction | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:22:03 UTC+0000 | [LambdaExecutionRole](https://eu-west-2.console.aws.amazon.com/iam/home?#/roles/st-stack-Lambda-crawler-LambdaExecutionRole-2CDuZ43plULF) | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:21:59 UTC+0000 | [S3OutputBucket](https://eu-west-2.console.aws.amazon.com/s3/home?region=eu-west-2&bucket=my-example-s3-bucket-2025-output) | | | CREATE\_COMPLETE | | - | | - |
| 2025-03-02 13:21:46 UTC+0000 | [S3OutputBucket](https://eu-west-2.console.aws.amazon.com/s3/home?region=eu-west-2&bucket=my-example-s3-bucket-2025-output) | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:21:46 UTC+0000 | [LambdaExecutionRole](https://eu-west-2.console.aws.amazon.com/iam/home?#/roles/st-stack-Lambda-crawler-LambdaExecutionRole-2CDuZ43plULF) | | | CREATE\_IN\_PROGRESS | | - | | Resource creation Initiated |
| 2025-03-02 13:21:45 UTC+0000 | S3OutputBucket | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:21:45 UTC+0000 | LambdaExecutionRole | | | CREATE\_IN\_PROGRESS | | - | | - |
| 2025-03-02 13:21:43 UTC+0000 | [st-stack-Lambda-crawler](https://eu-west-2.console.aws.amazon.com/cloudformation/home?region=eu-west-2#/stacks/stackinfo?stackId=arn%3Aaws%3Acloudformation%3Aeu-west-2%3A277707117797%3Astack%2Fst-stack-Lambda-crawler%2F48816100-f769-11ef-8dda-06eedc5dd103) | | | CREATE\_IN\_PROGRESS | | - | | User Initiated |

A screenshot of a computer

AI-generated content may be incorrect.

Crawler created and showing “1 table created”

A screenshot of a computer

AI-generated content may be incorrect.

Table details :-

A screenshot of a computer

AI-generated content may be incorrect.

Database created :-

A screenshot of a computer

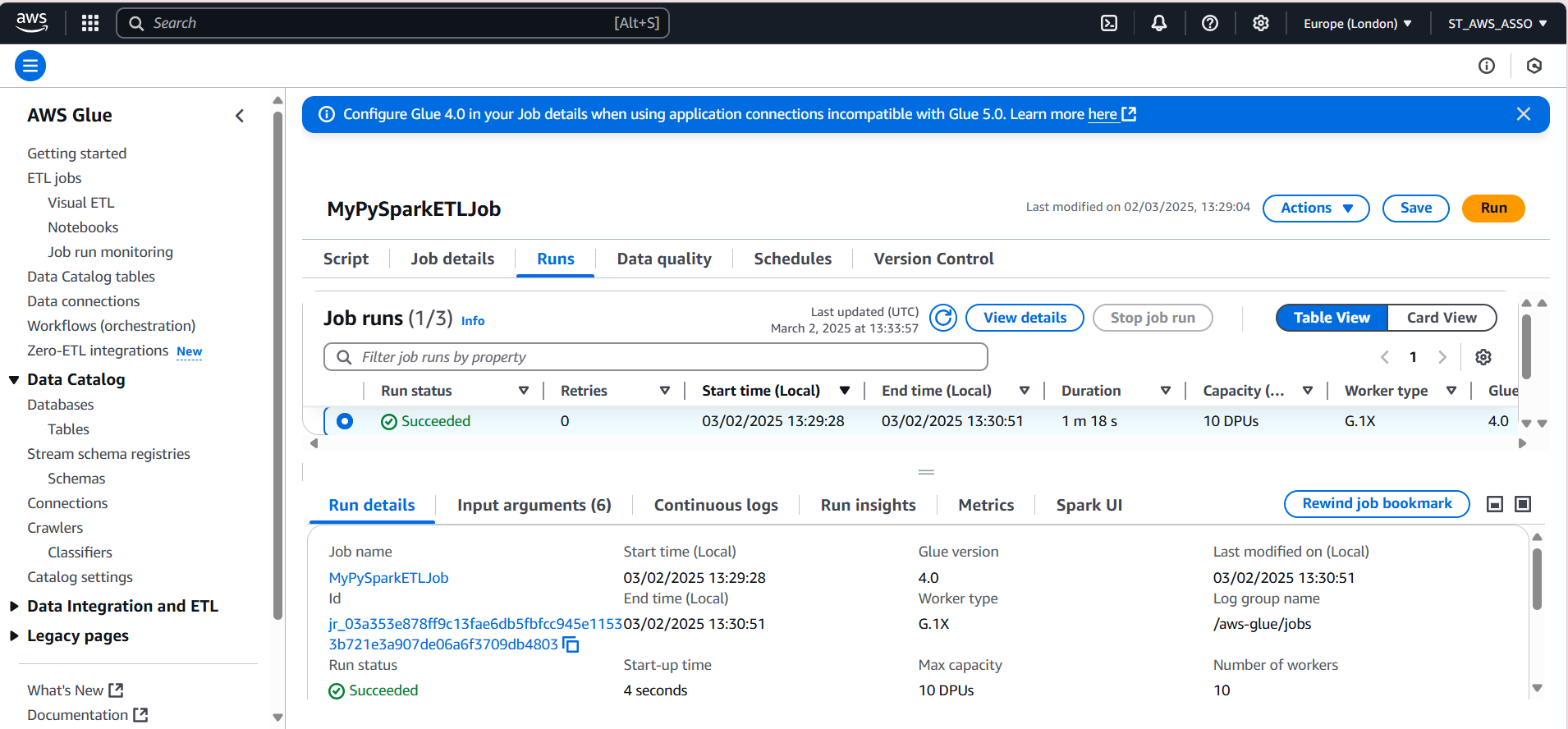
AI-generated content may be incorrect.

File created uysing ETL Glue in **my-example-s3-bucket-2025-output** :-

A screenshot of a computer

AI-generated content may be incorrect.

AWS GLUE ETL job (only need to check it should run when Crawler is READY , but manually it runs successfully)



# Define the Glue Crawler properties (code is in Lambda function)

crawler = {

'Name': crawler\_name,

'Role': os.environ['AWS\_LAMBDA\_ROLE\_ARN'], # IAM Role for Glue to access resources

'DatabaseName': glue\_database\_name,

'Targets': {

's3Targets': [

{

'Path': s3\_path,

}

]

},

'Classifiers': [],

'TablePrefix': 'csv\_',

'Schedule': 'cron(0 12 \* \* ? \*)', # Optional: Set up a schedule for the crawler

}

Let's break down the **crawler** dictionary in your Lambda function code to understand each key and its purpose in the Glue Crawler configuration:

**1. 'Name': crawler\_name**

* **Purpose**: Specifies the name of the Glue crawler. In your code, this is dynamically generated from the file name that triggered the Lambda.
* **Example**: If the uploaded file is data/file1.csv, the crawler name could be crawler-file1-csv.

**2. 'Role': os.environ['AWS\_LAMBDA\_ROLE\_ARN']**

* **Purpose**: The IAM role that AWS Glue assumes to perform actions on your behalf. The role needs permissions to read from S3 and write to the Glue Data Catalog (create tables, update schemas, etc.).
* **Example**: If the Lambda function's role ARN is stored in an environment variable, it would be used here.
* **Explanation**: The IAM role must have sufficient permissions for Glue to access the data in S3 and perform operations like creating and updating tables.

You should ensure that this IAM role has the following permissions (at a minimum):

* s3:GetObject (to read data from S3)
* glue:CreateTable, glue:UpdateTable, glue:GetTable, etc. (to interact with the Glue Data Catalog)

**3. 'DatabaseName': glue\_database\_name**

* **Purpose**: Specifies the Glue database where the crawler will store the metadata it discovers from the S3 data.
* **Example**: If your Glue database is called my\_glue\_database, then glue\_database\_name would be "my\_glue\_database".
* **Explanation**: The Glue database is where tables representing your data in S3 will be created. The crawler will scan the data and create or update tables in this Glue database based on the schema it discovers.

**4. 'Targets': { 's3Targets': [ { 'Path': s3\_path, } ] }**

* **Purpose**: Defines the data sources (targets) that the crawler will scan to infer the schema. In this case, it's an S3 location.
* **Example**: s3\_path = "s3://my-upload-bucket/data/file1.csv".
* **Explanation**:
  + The crawler will scan the S3 location specified in 'Path' to infer the structure of the data.
  + You can specify other types of data sources for Glue, such as JDBC or DynamoDB, but in this case, you're using S3 as the target.

**5. 'Classifiers': []**

* **Purpose**: Defines custom classifiers that can be used by the crawler to better understand the format of your data.
* **Explanation**:
  + AWS Glue has built-in classifiers (e.g., CSV, JSON) to recognize common file formats.
  + If your data is in a specific format (e.g., Avro, Parquet) and needs special handling, you can define custom classifiers.
  + In your case, it’s an empty list, meaning no custom classifiers are being used and Glue will rely on built-in ones to process your data.

**6. 'TablePrefix': 'csv\_'**

* **Purpose**: Specifies a prefix that will be added to the names of tables that the crawler creates in the Glue Data Catalog.
* **Example**: If your data has a file named data.csv, Glue will create a table in the catalog named csv\_data.
* **Explanation**: This helps you categorize or distinguish tables created by this specific crawler. If you have multiple crawlers running on different data types, you might want to use different prefixes to organize them.

**7. 'Schedule': 'cron(0 12 \* \* ? \*)'**

* **Purpose**: Defines a schedule on which the crawler should run.
* **Explanation**:
  + This value is in **cron** format, which is used by AWS services to define scheduled tasks.
  + The example cron(0 12 \* \* ? \*) means the crawler will run at 12:00 PM (UTC) every day.
  + You can change the cron expression based on your requirements. For instance:
    - cron(0 0 \* \* ? \*): Runs every day at midnight (UTC).
    - cron(0 18 \* \* ? \*): Runs every day at 6:00 PM (UTC).
  + If you do not want to schedule the crawler (and prefer to run it manually or automatically based on events), you can omit this field.

**Full Explanation:**

In essence, this dictionary is configuring an AWS Glue crawler with the following functionality:

1. **Name**: Dynamically set based on the file name that triggered the Lambda function.
2. **IAM Role**: The Lambda function must specify an IAM role that Glue will assume when it runs to access resources like S3 and Glue Data Catalog.
3. **Database**: The Glue database where the metadata of the data (discovered by the crawler) will be stored.
4. **S3 Target**: The S3 location (or multiple locations) the crawler will scan to infer the schema. This could be a specific file or a directory in S3.
5. **Classifiers**: Custom classifiers for special data formats (in your case, none are provided).
6. **Table Prefix**: A prefix added to the table names in the Glue Data Catalog to help organize them.
7. **Schedule**: Defines when the crawler should run, based on a cron expression.

This configuration allows the Lambda function to create a crawler that is capable of scanning the uploaded data and creating tables in the Glue Data Catalog, all while using a dynamic name based on the file that triggered the Lambda function.

Let me know if you'd like more details on any part of this!

**Crawler Cost:**

* If you use a **Glue Crawler** to discover the file, the crawler will likely process a minimal amount of data as well.
  + Crawlers are billed based on **crawler runtime**, and for a small file, this will probably take a very short time.
  + The Free Tier provides **1 million seconds of crawler runtime per month**, so if you're working within those limits, you won't incur any charges.

**3. Data Catalog:**

* If the data is registered in the **AWS Glue Data Catalog**, you are billed for the number of requests made to the catalog. A small dataset like 10 records will likely incur **one catalog request**.
  + The Free Tier includes **1 million catalog requests per month**, so if you're not exceeding that, you won’t be charged.

**Estimation for a Small Job:**

For a small file with 10 records, assuming minimal ETL or crawler activity, you will **probably not be charged** if you stay within the Free Tier limits. Your charges for such a small job are likely to be negligible unless you’re performing complex transformations or running many jobs.

**Summary:**

* For a small job with 10 records, **you are unlikely to incur any charges** if you stay within the AWS Glue Free Tier limits.
* If your usage exceeds the Free Tier (which is very unlikely for such a small task), then you would be charged based on the actual DPU, crawler time, and catalog requests used.

can we use crawler with Code for Glue Job (PySpark)

**ChatGPT said:**

Yes, you can definitely use a **Glue Crawler** in conjunction with a **PySpark Glue Job**! The Glue Crawler helps automate the process of discovering and cataloging the schema of your data (stored in S3 or other sources), while the Glue PySpark job allows you to process and transform that data.

**How Glue Crawler and PySpark Glue Job Work Together:**

1. **Glue Crawler**:
   * The **Crawler** scans your data (e.g., in S3), infers the schema (columns, data types), and creates a table in the **AWS Glue Data Catalog**.
   * The Glue Data Catalog stores metadata that can be used by both **AWS Glue Jobs** and **Amazon Athena** for querying.
2. **PySpark Glue Job**:
   * The **PySpark Glue Job** can use the Glue Data Catalog to read the metadata of the data cataloged by the crawler.
   * After the crawler runs, the Glue job can access the cataloged table and transform the data (e.g., cleaning, joining, or aggregating).
   * The transformed data can be saved back into S3, or it can be loaded into a data warehouse (e.g., Redshift, RDS).

**Typical Workflow Combining Glue Crawler and PySpark Glue Job:**

1. **Set Up the Glue Crawler**:
   * The Glue Crawler will scan your data source (e.g., CSV, JSON, or Parquet files in S3) and automatically create a table in the Glue Data Catalog.

Here’s how you would create a Crawler:

* + In the AWS Glue Console, go to **Crawlers** and click on **Add crawler**.
  + Select your data source (e.g., an S3 bucket).
  + Set the output target as the Glue Data Catalog database.
  + Once the Crawler runs, it will create a table with metadata in the Glue Data Catalog.

1. **Write a PySpark Glue Job**:
   * After the data is cataloged, you can write a Glue Job to process this data using PySpark (Python-based ETL scripting).
   * The PySpark script can read data directly from the Glue Data Catalog, apply transformations, and then save the processed data back into S3 (or any other data target).

**Example PySpark Glue Job Code:**

Here’s an example of how you can write a PySpark Glue Job that leverages the **Glue Crawler**’s cataloged table to extract data and transform it:

**1. Crawl Your Data First:**

* Let's assume the **Crawler** created a table in the Glue Data Catalog called sales\_data in a database called sales\_db.

**2. PySpark Script for Glue Job:**

python

Copy

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from pyspark.sql import SparkSession

# Initialize GlueContext and SparkSession

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

# Reading data from Glue Data Catalog (after being crawled)

# Use the catalog database and table name that the crawler created

datasource = glueContext.create\_dynamic\_frame.from\_catalog(

database = "sales\_db", # Glue database name

table\_name = "sales\_data" # Glue table name created by the crawler

)

# Convert the DynamicFrame to a DataFrame for easier manipulation (if needed)

df = datasource.toDF()

# Example Transformation: Clean data (e.g., removing rows with null 'price' column)

df\_clean = df.filter(df['price'].isNotNull())

# Convert back to DynamicFrame (required for saving with Glue)

clean\_dynamic\_frame = glueContext.create\_dynamic\_frame.from\_df(df\_clean, glueContext)

# Save the cleaned data to S3 in Parquet format (optimized for Athena)

output\_dir = "s3://your-bucket/processed\_sales\_data/"

glueContext.write\_dynamic\_frame.from\_options(

clean\_dynamic\_frame,

connection\_type="s3",

connection\_options={"path": output\_dir},

format="parquet"

)

**Explanation of the PySpark Code:**

1. **glueContext.create\_dynamic\_frame.from\_catalog()**:
   * This is where the Glue Job accesses the **Glue Data Catalog** (which was populated by the Glue Crawler) and reads the data from the cataloged table sales\_data in the sales\_db database.
2. **Transform the Data**:
   * After the data is read, you can perform various transformations using Spark, such as filtering out null values in a column (df['price'].isNotNull()).
3. **Save the Data**:
   * Finally, you can save the transformed data back to **S3** in an optimized format (e.g., **Parquet**) for more efficient querying, especially when querying with **Amazon Athena**.

**Key Points:**

* The **Crawler** automatically creates the metadata for your data, making it easy for your Glue Job to access and work with the data.
* **PySpark Glue Jobs** allow you to transform and clean the data before saving it in an optimized format like **Parquet** or **ORC**, which is ideal for querying with Athena.
* **Glue Data Catalog** serves as a central metadata store, which can be used by both **AWS Glue** and **Athena** for accessing schema and data.

**Use Case Example:**

* **Scenario**: You have raw sales data in **CSV files** in an S3 bucket. You use a Glue **Crawler** to discover the structure of the CSV files and create a table in the Glue Data Catalog.
* **Glue Job**: You then write a PySpark Glue Job to:
  + Read the cataloged data (sales data).
  + Perform transformations such as removing incomplete records, cleaning data, and converting it to Parquet.
  + Write the transformed data back to S3 in Parquet format.
* **Athena**: You can query the Parquet files in S3 directly with Athena, leveraging the Glue Data Catalog for schema management.

**Benefits of Using Crawler with Glue Jobs:**

* **Automation**: The Crawler automates the discovery and cataloging of your data, saving you from manually defining schemas.
* **Scalability**: AWS Glue and Spark can scale automatically to handle large data volumes, ensuring your ETL jobs can handle both small and large datasets.
* **Data Quality**: You can use Glue Jobs to clean, filter, and enrich your data before querying it with Athena.
* **Performance**: Using optimized formats like Parquet or ORC in S3 and querying them with Athena leads to significant cost and performance benefits.

Let me know if you need further clarification or examples!

Top of Form

Bottom of Form