Personal Objective:
In order to gain hands-on experience with AWS and how data flows in the cloud, I created a simple data pipeline using some of the most popular AWS tools: S3, Glue, Lambda, IAM, RedShift, EventBridge, and CloudWatch.

Project Objective:
A common AWS task is creating an ETL pipeline to move data from one service to another. Say I had a large amount of data in S3. Redshift is a data warehouse service that is better suited for holding and querying large amounts of data cost-effectively. Here, I move a dataset from S3 to Redshift using a variety of services, hoping to gain a bit of knowledge on each of them.

Toy Dataset: Fifa World Cup Attendance:
https://www.kaggle.com/datasets/rajkumarpandey02/fifa-world-cup-attendance-19302022

General Pipeline Description:
The data will originally reside in an S3 bucket. We will use two Glue Crawlers to automatically identify the schemas of the dataset and the destination table (an empty destination table with the desired schema created before the movement). We will then use these table schemas (AKA table definitions) in conjunction with a Glue Job to connect and move the dataset from S3 to Redshift.

Additionally, we will create an EventBridge Rule that will monitor for Crawler state changes. Upon a change to a “Successful” state, a lambda function is triggered that invokes the Glue Job. This way, if new data is added to the S3 Bucket and we want the changes to be reflected in RedShift, we can simply run the S3 bucket’s crawler (which will also identify changes in the table such as new columns), and the Glue Job will move new data from the S3 bucket to Redshift.

The following gives an overview of how I completed these tasks.

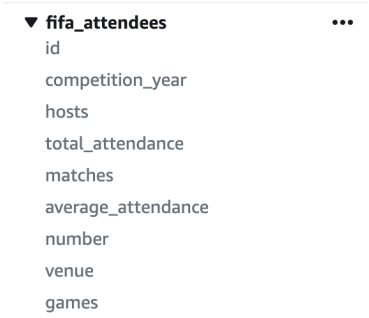
Setup:

- Create an S3 Bucket and drop your data into it.
- Create a Redshift Cluster and use the Query Editor to create a table (that will be empty until data is transferred to it) in your cluster with the correct schema for your data. For my dataset's case, I used the following command:

```
CREATE TABLE fifa_attendees(
    id                  bigint not null,
    competition_year    varchar(100) not null,
    hosts               varchar(100) not null,
    total_attendance    integer not null,
    matches             integer not null,
    average_attendance  integer not null,
    number              integer not null,
    venue               varchar(100) not null,
    games               varchar(100) not null
)
```

(note competition year is not integer because it has some string values)

- You should see a table like this appear under the Resources tab on the left-hand side



- Create an S3 Gateway Endpoint in your Redshift cluster's VPC (which can be found in your cluster's Network and security settings) so that the cluster can be connected to the S3 bucket.
- Create an "All TCP" inbound rule for your Redshift cluster's VPC security group (which can also be found in the cluster's Network and security settings).
- Create an IAM role.
    - Assign three permission policies. It is generally not recommended to add "FullAccess" permissions because it presents potential security vulnerabilities, but I do here for simplicity's sake.
        - AmazonS3FullAccess
        - AWSGlueFullAccess
        - AmazonRedshiftFullAccess
    - Assign two trusted entities.
        - glue.amazonaws.com

            - lambda.amazonaws.com
        - Use this IAM role in all subsequent steps.
- From the Glue Console, create a connection and two crawlers.
    - Create a connection between Glue and your Redshift cluster.
    - S3 Crawler Configuration:
        - Use S3 as your data source.
        - Target the S3 bucket holding your dataset (or just target the dataset itself).
        - Use the IAM role you created above.
        - You will be asked for a database. This is a temporary database used to hold a portion of the data so that the crawler may use it to identify the schema. Create a new one.
    - Redshift Crawler Configuration:
        - Use JDBC as your data source.
        - Use the connection you created above.
        - For “include path,” target the empty table you built in your Redshift cluster. I entered “dev/public/fifa_attendees”.
        - Use the same IAM role.
        - Create a new temporary database for this crawler when prompted.
- In the EventBridge Console, create a rule. In the Event Pattern section, use a Custom Pattern. In the Event Pattern code box, write JSON for what the rule should be listening for: a “Success” state change for your **S3 Crawler**. Such JSON in my case was as follows (code also in GitHub Repo):

```
{
  "detail-type": [
    "Glue Crawler State Change"
  ],
  "source": [
    "aws.glue"
  ],
  "detail": {
    "crawlerName": [
      "DataSourceS3"
    ],
    "state": [
      "Succeeded"
    ]
  }
}
```

Be sure to change “DataSourceS3” to your S3 crawler’s crawler name.

- Create a Lambda Function using the desired Python runtime and your IAM role.
    - After creating the function, go to the “Function Code” section and write a handler that will start a job (make sure to hit deploy after writing the code). For my case, (also in GitHub Repo) I replaced lambda_function.py with

lambda_function × Execution results ×

```python
# Set up logging
import json
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# Import Boto 3 for AWS Glue
import boto3
client = boto3.client('glue')

# Variables for the job:
glueJobName = "fifa_attendees_job"

# Define Lambda function
def lambda_handler(event, context):
    logger.info('## INITIATED BY EVENT: ')
    logger.info(event['detail'])
    response = client.start_job_run(JobName = glueJobName)
    logger.info('## STARTED GLUE JOB: ' + glueJobName)
    logger.info('## GLUE JOB RUN ID: ' + response['JobRunId'])
    return response
```

Be sure to replace “fifa_attendees_job” with the name of your job (which we will create later).

- In the function overview section, add a trigger. In the trigger configuration, select “EventBridge (CloudWatch Events)” and the rule you made above.

Putting it all together:

- In the Glue Console, run the S3 and Redshift crawlers and check the results in the “Tables” section.
- Create a Glue Job.
  - Use S3 as the data source and Redshift as the data target.
  - Inside the job, select the S3 node. Choose the S3 crawler’s database and the corresponding table that was created by the S3 crawler.
  - Select the Redshift node. Choose the Redshift crawler’s database and the corresponding table that was created by the Redshift crawler.
  - In the Transform node, apply the correct mapping from the S3 table’s columns to the Redshift table’s columns.
  - Use your IAM role.
- Run your job.
  - Once the job is complete, you can test its success by querying the database in your redshift cluster. For example,

```sql
SELECT *
FROM fifa_attendees
```

If the query returns the data held in the S3 bucket, then the data has been successfully moved to your Redshift cluster.

  - Additionally, if an error occurs during the job, you can check the CloudWatch logs linked in the run’s description for more information on the error.
- Remember to go back to your Lambda function code and replace the variable

glueJobName with the name of your job (I realize I could've put lambda function creation step last, but I felt it logically made sense to place it above). Test that the lambda function correctly automates the pipeline by running your S3 crawler and ensuring your Glue Job runs after the crawler's successful completion.

    - You can find CloudWatch Metrics in the "Monitor" section for information about invocations of your lambda function, including its success rate. Also, CloudWatch logs are linked there for detailed reports of your lambda function invocations.

Sources:

- https://aws.amazon.com/premiumsupport/knowledge-center/start-glue-job-crawler-completes-lambda/
- https://github.com/RekhuGopal/PythonHacks/tree/main/AWSBoto3Hacks/AWS-ETL-S3-Glue-RedShift
- https://www.youtube.com/watch?v=8tr9kCJTBl4