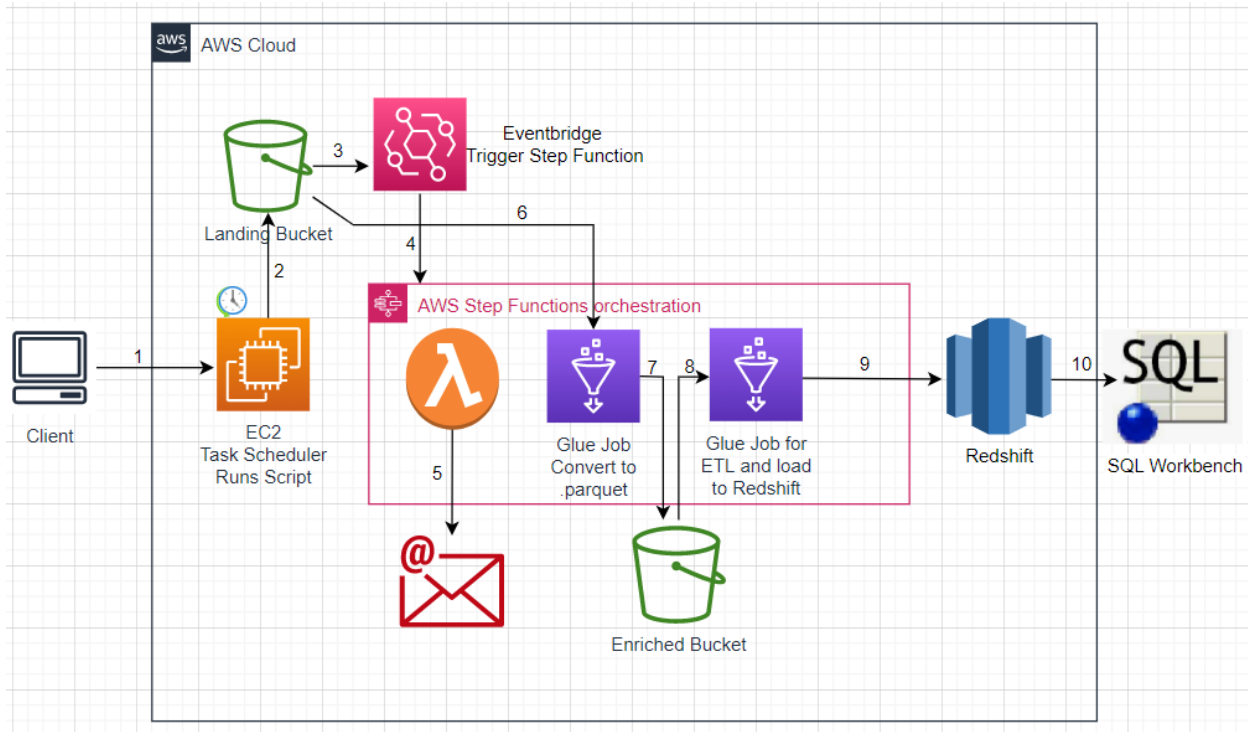


Automated transfer of datasets from local computer to Amazon Redshift with ETL steps

Purpose: Establish an automated pipeline that picks up a dataset from a directory in a local machine, moves it to S3 bucket, conducts ETL and loads it to Redshift Data Store to access it through SQL Workbench for data analytics and machine learning

Take a look at the following Diagram to get a high level glimpse into the expected work awaiting us:



Prerequisites: AWS Account, Installed SQL Workbench, Understanding of: ETL methodology, PySpark, Python, Basic Network Configuration, Downloaded files from folder 'Project 3' in GitHub link provided: <https://github.com/Myself1214/Upwork.git>

Plan of Work (Pseudo Work):

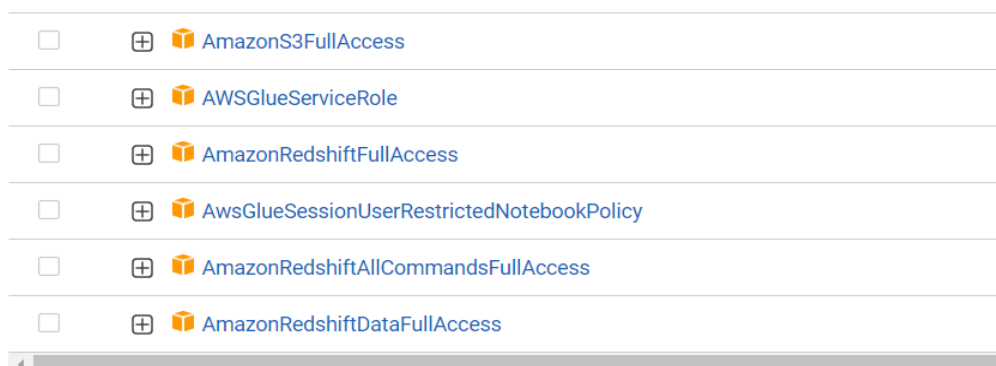
1. Create two S3 buckets: Raw bucket and Enriched bucket
2. Create Security Group to be used by all resources
3. Create an EC2 Windows server and mount a directory from the local machine on EC2 to be able to read local files on EC2 synchronously.
4. Create a script to move the dataset from EC2 server to S3 Raw bucket and put it on a daily execution schedule
5. Create IAM roles to be used by glue jobs, redshift cluster and lambda
6. Create Lambda function to send email notification when Step Function is triggered

7. Create a Glue Job to convert data from Raw bucket to .parquet format and move it to S3 Enriched bucket.
8. Create a Redshift Cluster with loaded sample database
9. Create another Glue Job (Enriched and load job) to read data from Enriched bucket, clean, transform and load it to a database within Redshift Data Store.
10. Create StepFunction to orchestrate lambda and glue jobs executions in order
11. Create EventBridge rule trigger to trigger StepFunction once file is loaded in S3 Ra bucket
12. In SQL Workbench create connection to Redshift Data Store and query loaded dataset to validate it is present in Redshift

Actual Steps:




1. Creating S3 Buckets.
 - In AWS, log in to S3 Console. Click on 'Create Bucket', give the bucket name 'Raw' bucket
 - scroll down and click on 'Create bucket'. Repeat again for the 'Enriched' bucket. Repeat again for the 'tooling' bucket.
2. Create Security Groups
 - navigate to EC2 Console, scroll down and select Security Groups section in the left pane
 - click 'Create security group', give it a name of your choice
 - under Inbound rules select 'Add rule': for type 'RDP', for source 'My IP'
 - add another rule: for type 'HTTPS', for source 'My IP'
 - add another rule: for type 'Redshift', for source 'Anywhere IPv4'
 - under Outbound rules, add rule: for type 'All traffic', for source 'Anywhere IPv4V'.
 - scroll down and click 'Create Security Group'
 - open previously created security group and add another inbound rule: for type 'All TCP', for source select this security group's name from the list. Save changes
3. Launch EC2 Instance.
 - navigate to EC2 Console, Click on 'Launch instance', give it a name of your choice
 - under AMI section choose Windows image
 - under Key Pair section select 'Create new key pair' and follow directions to save you key pair file and select this key pair from the drop down list
 - under network settings section choose 'Select existing security group' and choose previously created security group
 - scroll all the way down and click 'Launch Instance'
4. Mounting the local folder as a disk drive to ec2. To do it, we will need to create a disk drive pointing at shared folder
 - create folder in a directory of your choice in your local machine
 - open file '*to turn folder into disk drive.txt*' from GitHub link provided above and update it with your details as instructed
 - copy and run this code in CLI of your choice. You should see a new drive created in your machine

- connect to your ec2 instance by selecting it in AWS console, click connect on top right, select RDC, click on 'decrypt password', select 'browse', choose 'Key pair' file you created when launching your ec2 instance, click on 'decrypt'
- open Remote Desktop Client app on your local machine and input connection details from ec2 console. Click on 'show options', select 'Local Resources' tab on top, click on 'more', under drives select your newly created drive. Connect to ec2
- 5. Install AWS Command Line Interface on your EC2.
 - Open Command Prompt within your EC2, enter following command, follow instructions to install AWS CLI: `msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi`
- 6. Enable Network Discovery on your EC2.
 - On your EC2 navigate to Network => Properties => Change Advanced Sharing Settings => Turn On Network Discovery and save changes
- 7. Create .bat file to sync files in the mounted folder on EC2 and S3 raw bucket.
 - open file `s3_sync.bat` downloaded from GitHub link, adjust it with your account details, disk drive letter and S3 raw bucket path and save
 - place this `s3_sync.bat` file on your shared folder in your local machine. It should show up in your EC2 instance under `\\tsclient\` path. Move this file to another location of your choice within your EC2 file system
- 8. Create a task on Windows task scheduler to run `s3_sync.bat` file on a daily basis.
 - Instructions on how to create task can be found here: <https://techdirectarchive.com/2020/03/24/how-to-create-a-scheduled-task-on-windows-server-2019/>
- 9. Create IAM roles for glue job to access s3 buckets, redshift, run sql queries and IAM pass role and another IAM role for lambda to access S3 bucket and to report logs to CloudWatch
 - Log in to IAM console. Select 'Roles', then select 'Create Role'. Under 'Trusted Entity type' select 'AWS Service', under "use case" select 'Glue' and click 'Next'. Add following permissions:



Also, add an 'Inline policy', select 'JSON' tab and add json string from `PassRole.json` file provided in GitHub link. Finish creating this role.

- On IAM console create another role. Under 'Trusted Entity type' select 'AWS Service', under "use case" select 'Lambda' and click 'Next'. Add following permissions::

<input type="checkbox"/>	Policy name 	Type
<input type="checkbox"/>	  AmazonS3FullAccess	AWS managed
<input type="checkbox"/>	  CloudWatchFullAccess	AWS managed

10. Create Lambda function to send email notification once file is loaded in S3 raw bucket
 - log in to the Lambda console. Select 'Functions' on the left pane and then click on 'Create function'. Leave 'Author from scratch' selected, provide function name, under 'Runtime' select 'Python 3.9'. Expand 'Change default execution role' and select 'Use an existing role' and from the menu select the IAM role created for Lambda in step 8. Finish creating function.

- once a function is created, scroll down and select the 'Code' tab, delete all default code and replace it with the code from 'lambda-code-email-sender.py' file from github and click on 'Deploy'. After, click on 'Test', provide a name for the test event and save.

Note: To use this code, you must create a third party app password for your email and save it in 'p3_config.yaml' file from provided in GtiHub link, or for Gmail accounts, **allow less secure apps** to access your gmail account in your gmail settings and use your gmail account password in 'p3_config.yaml' file.

- Navigate to 'Configuration' tab, select 'Environment variables', click 'Edit', and click on 'Add environment variables' and add following variables and save:

Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
<input type="text" value="config_bucket"/>	<input type="text" value="<tooling bucket name>"/>	<input type="button" value="Remove"/>
<input type="text" value="config_file"/>	<input type="text" value="p3_config.yaml"/>	<input type="button" value="Remove"/>
<input type="text" value="raw_bucket"/>	<input type="text" value="<raw bucket name>"/>	<input type="button" value="Remove"/>
<input type="text" value="raw_trigger_folder"/>	<input type="text" value="<folder path (if any)>"/>	<input type="button" value="Remove"/>
<input type="button" value="Add environment variable"/>		

► Encryption configuration

- Under 'Configuration' tab switch to 'General configuration', click 'Edit' and set 'Timeout' to 10 seconds and save

11. Now since our code using 'yaml' python module that is not a built-in module in Lambda, we will need to add our own library package to Lambda in .zip file
- follow instructions on this page to build your own python package into a zip file and add it to your Lambda as a layer:

<https://www.linkedin.com/pulse/add-external-python-libraries-aws-lambda-using-layers-gabe-olokun/>

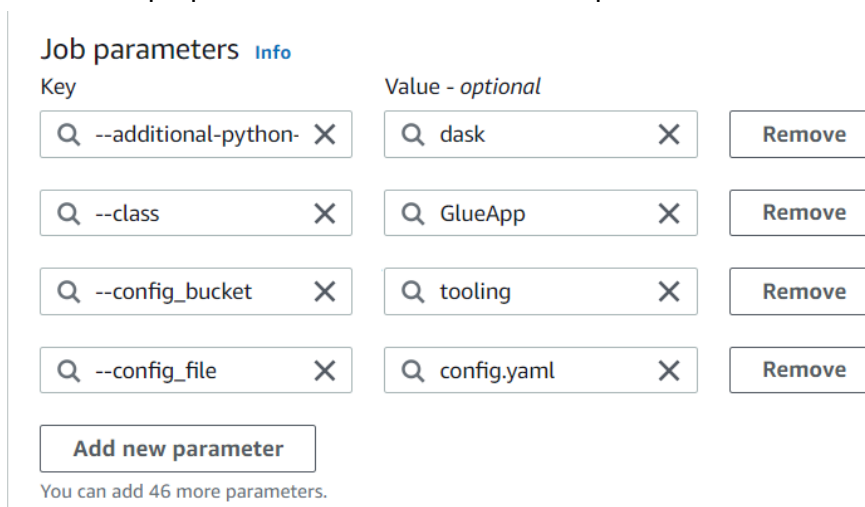
Don't forget to first install the 'yaml' module before you build the python package into a zip file.

You may directly upload your .zip file when adding it as a Lambda layer or link it to S3 bucket where you upload it first when building the Lambda layer.

Note: You may skip Lambda creation part since we already created our Lambda function

Note: Alternatively you may use python package zip file provided in GitHub link

12. Create a glue job to partition raw bucket data, convert it to .parquet format and load it to the enriched bucket.
- log in to the AWS Glue console. Select 'Jobs' on the left pane. Under 'Create job' select 'Spark script editor' and click on 'create'.
 - delete the prefilled code then copy code from 'raw-to-enriched-gljob.py' from GitHub and paste it as your script.
 - navigate to 'Job details' tab, change name to 'raw-to-enriched.gljob', under 'IAM-Role' select glue job role created in step 8, for 'Number of retries' change to '0', open 'Advanced properties' section and under 'Job parameters' section enter following details:



Key	Value - optional	
--additional-python-	dask	Remove
--class	GlueApp	Remove
--config_bucket	tooling	Remove
--config_file	config.yaml	Remove

[Add new parameter](#)

You can add 46 more parameters.

then scroll all the way up and click on 'Save'. (Note: it's --additional-python-modules)

13. Create a Redshift cluster with a sample database, set up VPC S3 Endpoint
- log in to the Redshift console, select clusters on the left pane and click on 'Create cluster'. Change cluster name to one of your choice under "Cluster identifier", select 'Free trial', scroll down and provide your username and password and click on 'create cluster'

- once the cluster is created and available, open it and navigate to the 'Properties' tab. Under 'Network and security settings' section click on 'Edit'. Under 'VPC security groups' delete default security group and from drop-down menu choose security group created in step 2. and save changes. This will allow connection on resource level for our cluster.
- once changes are saved and the cluster available (takes a couple of minutes), open cluster again and 'Actions' tab select 'Modify public accessibility settings' and change it to 'Enable' and save changes. This will allow us to connect to the Redshift database from an external source like SQL workbench, having a security group already configured.

- now, we need to allow connections between glue job and our VPC where our redshift cluster resides since glue job will be transferring data from Amazon S3 to a resource in our VPC - redshift cluster. To do this we need to create a *VPC S3 Gateway Endpoint*.

- log in to the VPC console. On the left pane select 'Endpoint', and click on 'Create endpoint'. Give it a name, for 'Services', add the filter 'Type: Gateway' and select 'com.amazonaws.*region*.s3'. For 'VPC' select VPC in which the redshift cluster resides (default VPC). Put a checkmark on route table provided, Scroll down and click on 'Create endpoint'

14. Now we need to create a glue connection to the redshift database to allow glue job to use it as a channel to write data to the redshift database.

- log in to the Glue console, select 'Connectors' from the left pane, under 'Connections' section click on 'Create connection'. Provide it a name, for 'Connection type' select 'Network'. Under 'Network options' section, for 'VPC' select VPC in which redshift cluster resides in, for 'Subnet' select any subnet associated with redshift VPC, and for 'Security group' select security group created in step 2 and click on 'Create connection'

15. Create another glue job to do data transformation and load it to the redshift sample database as a table.

- repeat steps for glue job creation provided in step 11, except, use code from 'to-redshift gljob.py', change glue job name to 'to-redshift-gljob' and under 'Job parameters' add:

Job parameters [Info](#)

Key	Value - optional	
<input type="text" value="--class"/>	<input type="text" value="GlueApp"/>	<button>Remove</button>
<input type="text" value="--config_bucket"/>	<input type="text" value="tooling"/>	<button>Remove</button>
<input type="text" value="--config_file"/>	<input type="text" value="config.yaml"/>	<button>Remove</button>
<button>Add new parameter</button>		

You can add 47 more parameters.

- under the 'Job details' tab navigate to 'Connections' section and select the glue

connection created in step 15 from the drop-down menu. Scroll all the way up and save the glue job.

16. Create Step Function to wrap all executable jobs and arrange them in order
 - Log in to StepFunction Console, click on 'Create State Machine', for 'Type' select 'Standard' and go to the next page. Select Lambda as first step and drag and paste it between 'Start' and 'End' steps, give it a name, select Lambda function created in step 10 under 'API Parameter', under 'Additional configuration' check 'wait for callback'.
 - Select glue job, drag and paste it between 'Lambda' and 'End' steps, give it a name, under 'API parameters' change the name to your glue job's name created in step 12, and under 'Additional configuration' check 'wait for callback'.
 - Add another glue job by repeating previous step, this glue job will be last step in our step function's state machine, then click 'Next' twice, give your state machine a name, for 'Executions role' select 'Create a new role', for 'Logging', select 'Error', scroll down and click on 'Create state machine'
 - Log in to IAM console, choose 'Role' from the left menu, then from list of roles select newly created role for step function (will have step function's name in its name). Inside role select permission that starts with 'GlueStartJobRunFullAccess' then click on 'Edit Policy', select 'JSON' and inside list of values for 'Action' key add 'glue:GetJobRun' and 'glue:GetJobRuns' then save the changes:

Visual editor JSON

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "glue:StartJobRun",
8         "glue:GetJobRun",
9         "glue:GetJobRuns"
10      ],
11      "Resource": [
12        "*"
13      ]
14    }
15  ]
16 }
```

Security: 0 Errors: 0 Warnings: 0 Suggestions: 0

17. Create EventBridge Rule to execute StepFunction based on object creation in raw S3 bucket and allow S3 bucket to sent event to ventBridge rule
 - Log in to EventBridge Console, select 'rules' from left menu, click on 'Create rule'. Give the rule a name then click on 'Next', scroll down and under 'Event pattern' section click on 'Edit pattern' then copy and paste content of 'eventbridge-rule-string.json' file form GitHub link changing bucket name and folder (if any) to your own names and click 'Next'. For 'Target' select 'AWS service' and from drop-down menu select 'Step Function State Machine' and for state machine select step function created in step 16 and click on 'Next' and 'Next' another time, validate all your entries and click on 'Create rule'

- Now log in to S3 console, select raw bucket, under 'Properties' navigate to 'Amazon EventBridge' section and enable 'Send notifications to Amazon EventBridge for all events in this bucket' to allow bucket to send events to EventBridge rule.

Now is the time to test our pipeline with small version of our dataset (about 40 mb), once it runs successful and we validate data in Redshift database, then we will run full dataset size (about 700 MB).

1. Download 'config.yaml' file from GitHub and fill in all the values from your aws resources created:
 - Gather username, password and url - from your redshift cluster details,
 - db_table - set a name of your choice
 - source_file_bucket - name of your raw S3 bucket
 - file path - path inside raw S3 bucket where you'll upload your dataset (path ends with your dataset name and extension'
 - trg_file_bucket - your enriched S3 bucket (in format that is mentioned)
2. Navigate to the S3 console, select the 'tooling' bucket, upload the 'config.yaml' file you adjusted.
3. Download 'p3_config.yaml' file form GitHub and fill in all the values with your own:
 - Email_user - your email to send notification from
 - Email_password - third party application password you created for your email
 - Send_to - to which email you wish to receive your notification of file load
4. Move file '911_Calls_for_Service_(Last_30_Days).csv' to your local shared path that is mounted on your EC2.
5. Wait for task scheduler on your EC2 to run at your scheduled time or execute a manual run

This will sync file in your EC2 shared path with S3 war bucket, which in turn will trigger EventBridge rule that will execute StepFunction. StepFuncion will run Lambda function, raw glue job and enriched glue jobs in step by step manner. Once the file reaches S3 raw bucket you can validate it by checking StepFucntion state, which should say 'Running'. Alternatively you may check raw and then enriched glue jobs' state. Ultimately, once StepFucntion completes execution you should be able to validate data in Redshift by connecting to it with SQL Workbench and running select queries against your chosen table name