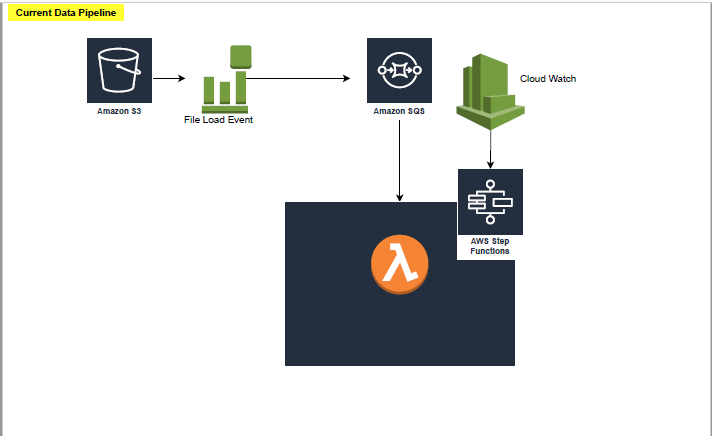
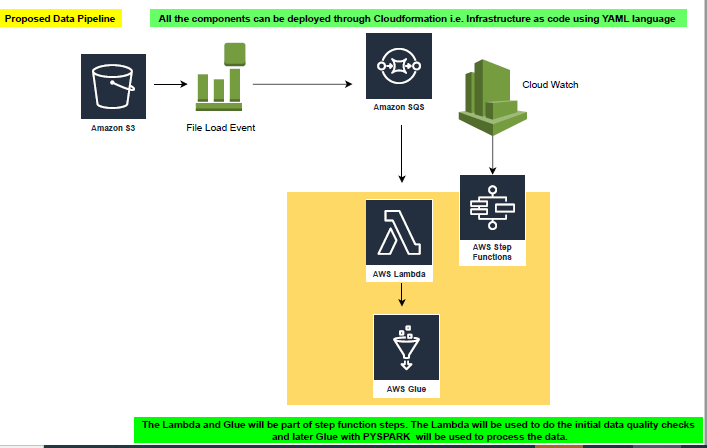
# **EVENT DRIVEN – BATCH DATA PIPELINE USING AWS**

As a Data Engineer, I wanted to create an event driven - batch data pipeline in AWS to process the newly uploaded csv files i.e., to perform all the required transformations.

**Current and Proposed architecture: -** I have created a simple architecture, however, this can be optimised further to achieve better results.







**AWS Data pipeline technical design: -**

I will be using the below AWS components to create the data pipeline: -

* **S3** - Storing files
* **SQS** - Storing the input events from S3
* **Lambda** - Compute Engine / **Lambda** and **Glue** combination to leverage spark functionality
* **Cloud Watch** – To set the rule for invoking the step function for batch processing.
* **Step function** – This is to Orchestrate the entire process.

**High Level Data flow: -**

S3🡪SQS🡪Lambda (compute engine)🡪S3 Transformed data

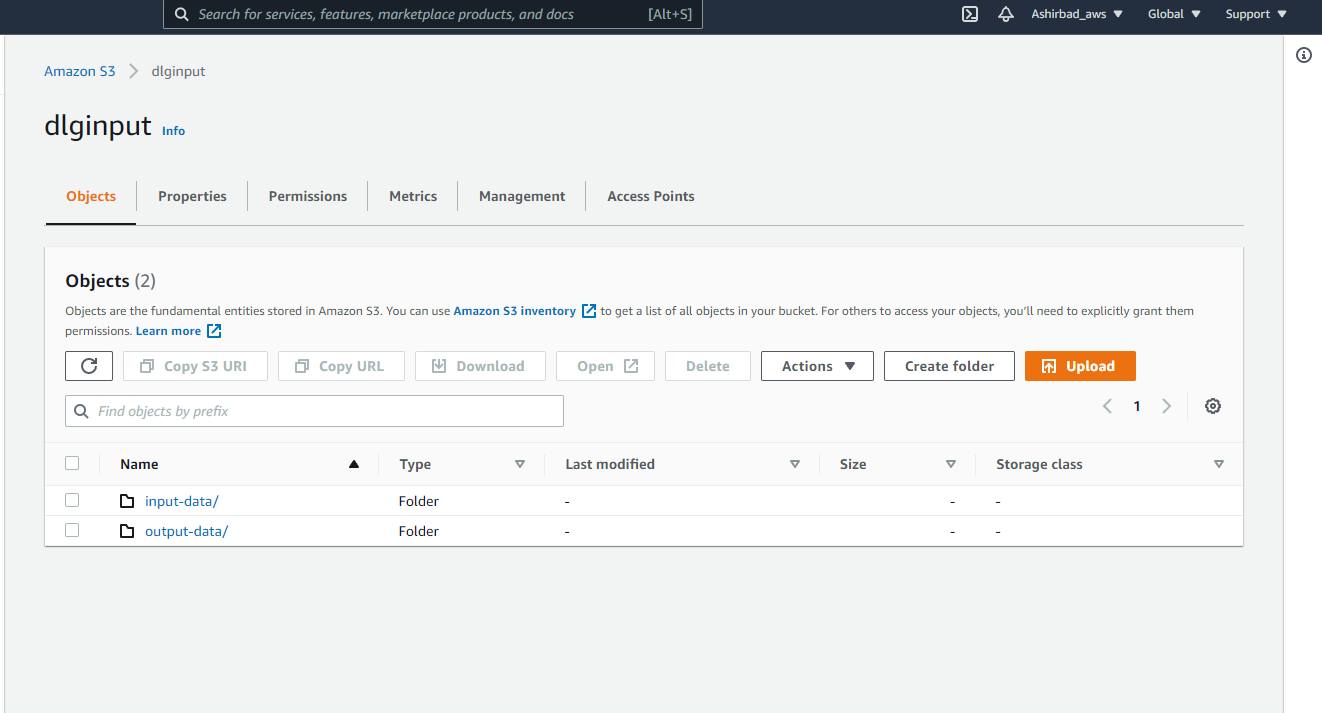
**OR**

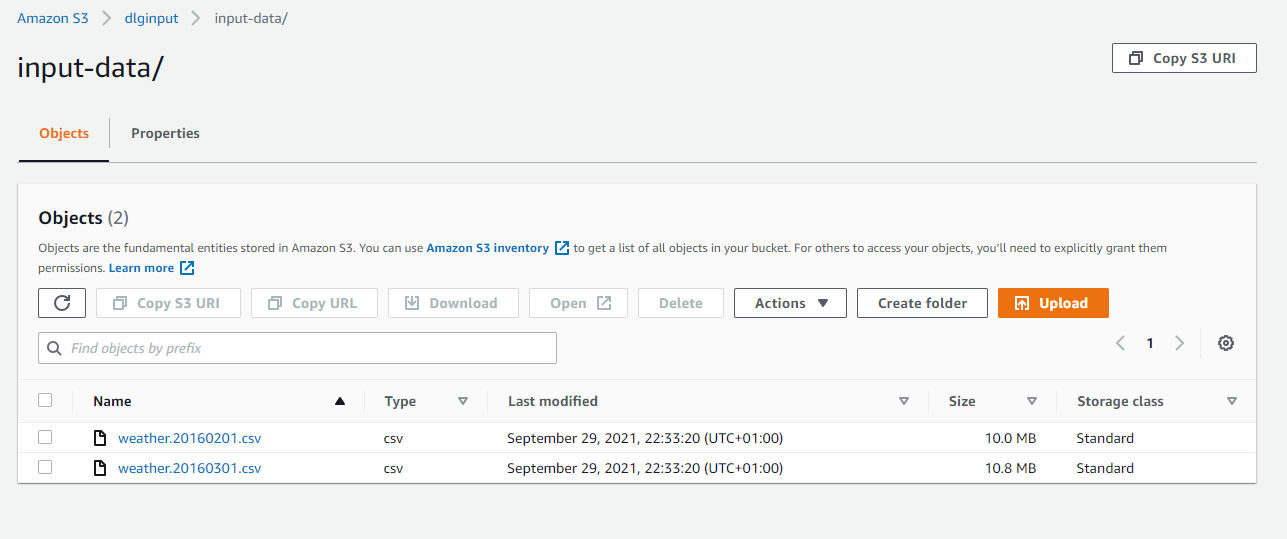
S3🡪SQS🡪Lambda (compute engine)🡪Glue (ETL)🡪S3 Transformed data

**Please find below the detailed design description for all the components: -**

**S3**

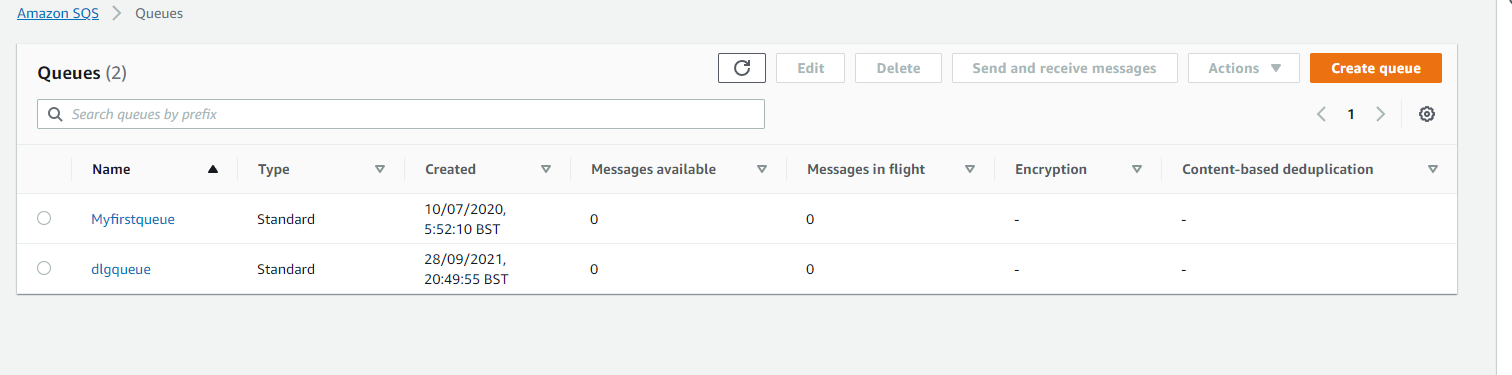
* This will be used for storing the incoming files in AWS cloud environment.
* I have created a bucket and corresponding folder (i.e. gazprom-data-loc/input-data/) to store the incoming files.

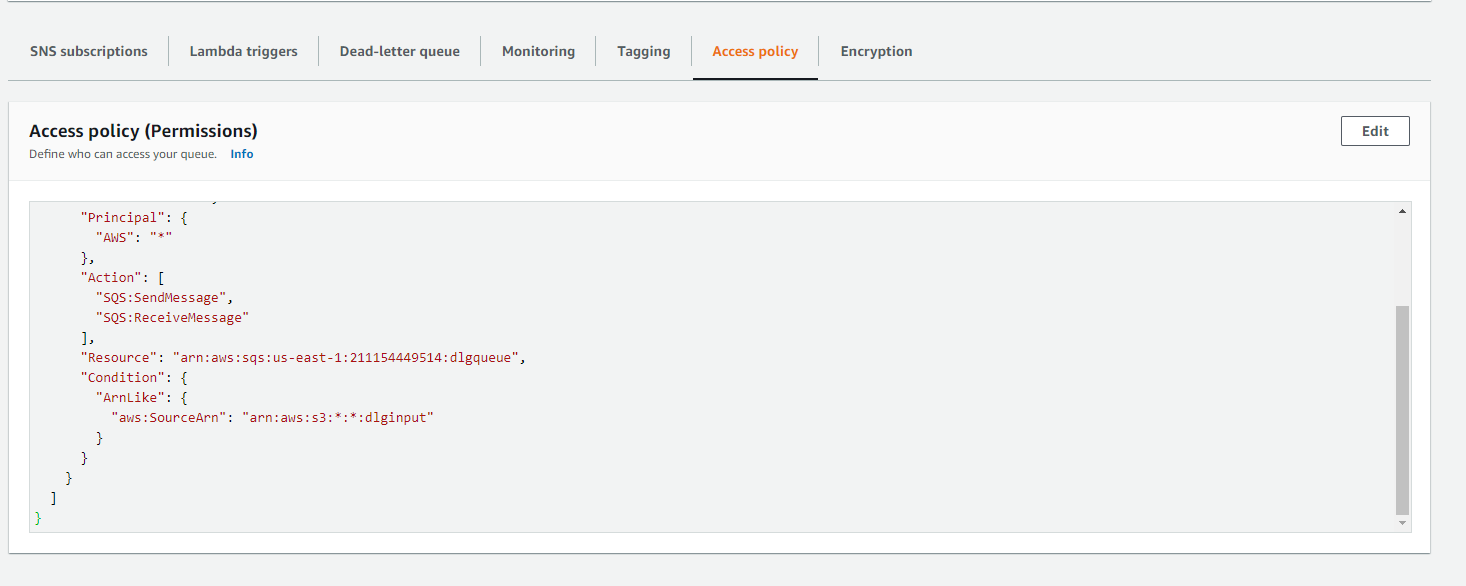




**SQS**

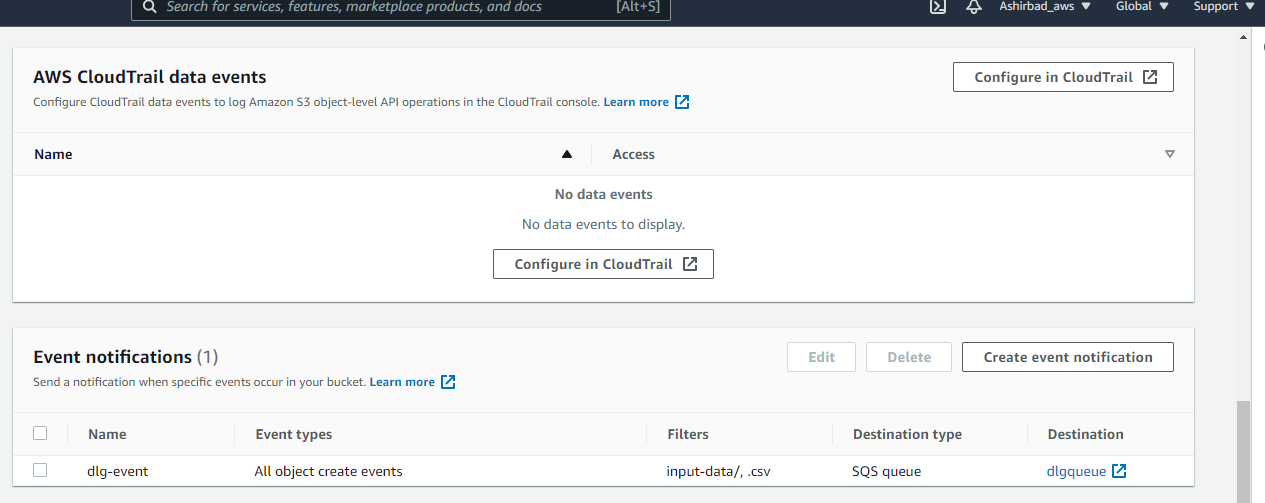
* A SQS has been created to store the input events from the S3. This will basically store the file load events details as messages in the queue.
* This will then be used by the Lambda or Glue to create a **batch processing** for all the files loaded for a given day.
* Please find below the screenshots for SQS creation: -





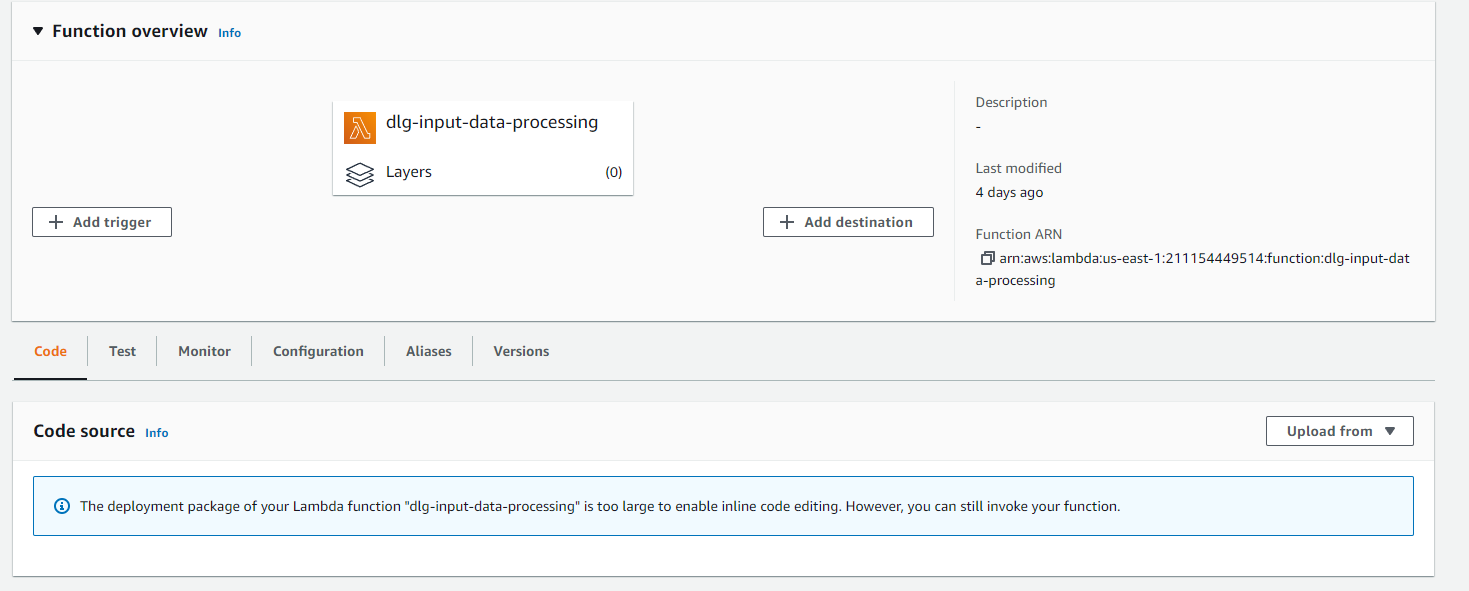
**S3 Trigger on SQS – To Notify file upload:**

* We have configured a S3 trigger i.e. S3 event notification to store the messages in SQS as soon as there is an file upload event in the storage layer.
* This trigger will be in action when we upload a new file in the designated bucket and folder.
* Please find below the screenshot for event trigger configuration: -

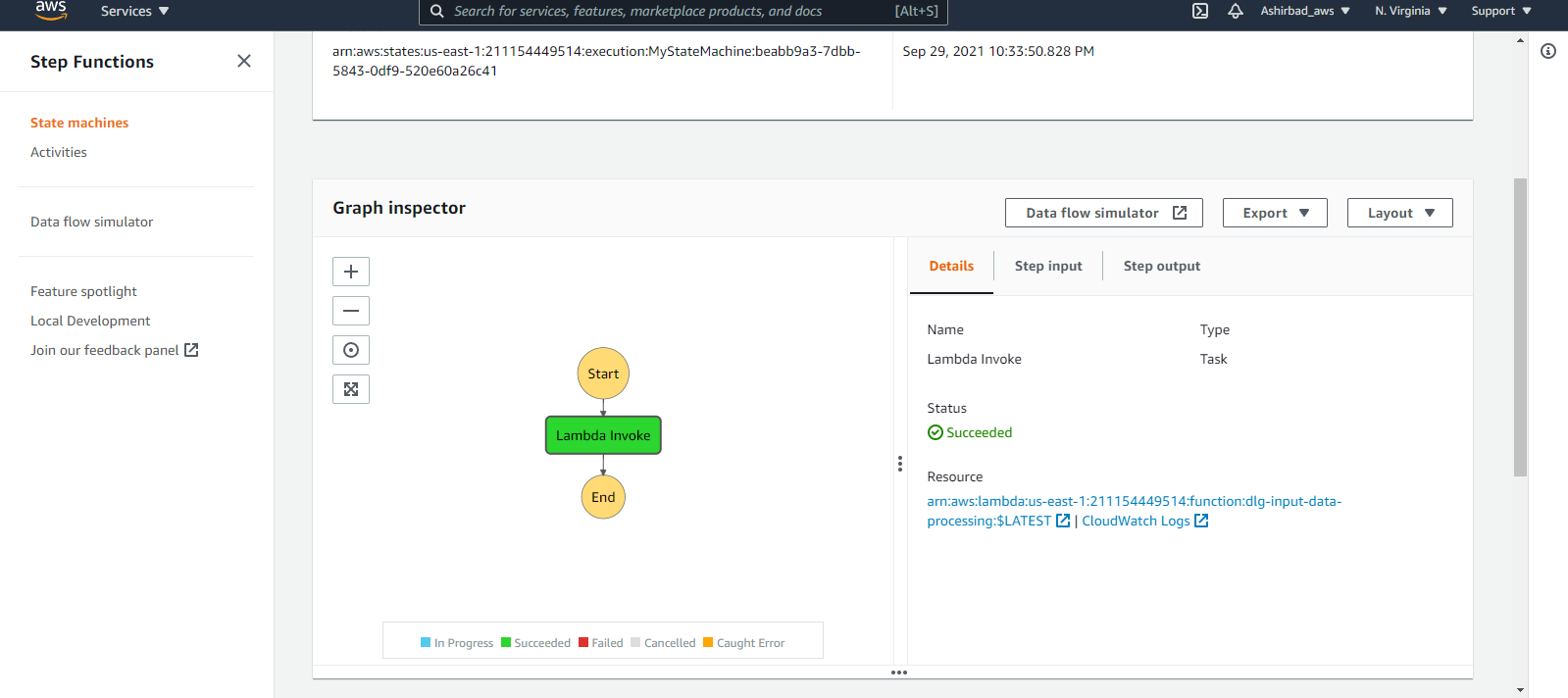


**Lambda:**

* After configuring the S3, we have created a compute engine to receive the incoming file events from S3 storage i.e., captured as messages in the SQS queue.
* Our python code will be deployed to this function and will be executed against the incoming file to perform all the validations and transformations i.e., before loading data into database.



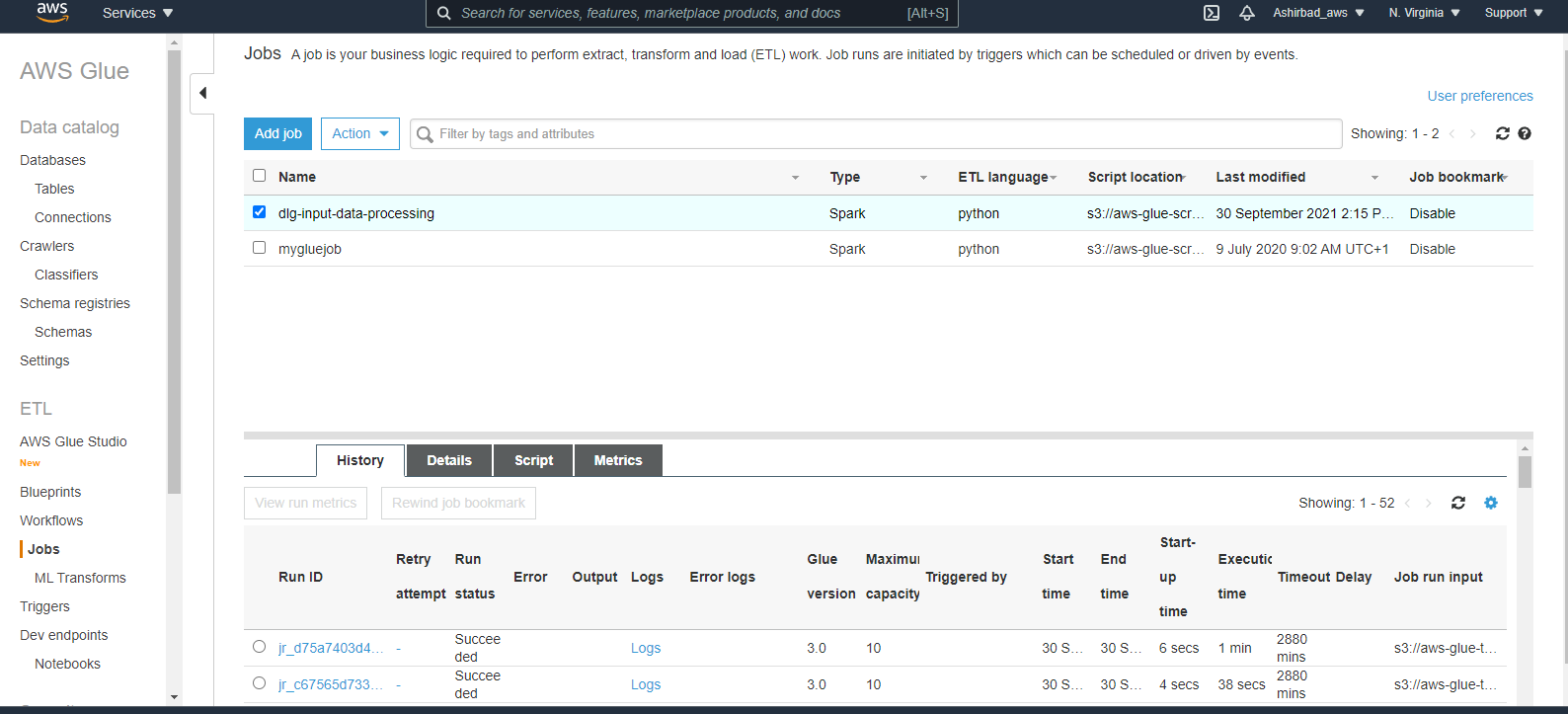
* All the permission from lambda to S3 have been granted through the IAM role attached to the lambda.
* This lambda will be called through the step function i.e., during a batch processing window set in the cloud watch rule.



* This way the entire process will be **orchestrated.**

**Important notes and improvement ideas: -**

* The first part of the pipeline is event driven and second part is batched.
* We can achieve full event driven approach; however, this will lead to invocation for compute engine for each file. E.g. If we have 100 files the there will be 100 invocations.
* By batching it, we are reading all the files together in a single invocation.
* **I couldn’t zip the supporting libraries (pyarrow or fastparquet) to convert the files in to parquet format as the total zip size was going beyond 100MB. I have tried to achieve the same using a Glue code.**
* **Glue: -**
  + I have leveraged the spark functionality in AWS i.e. using Glue to process files and convert it into parquet format.
  + This is one of the best ways to deal with the large files and anyways spark provides better performance compared to python when you are dealing with Big-Data.
  + As it’s my personal account and glue is one of the costliest services, so I couldn’t test the spark code fully(Partially tested). However, I am sure the spark code that I have written will do the job for us.



**Proposed Architecture: -**

* **S3 to store the input files.**
* **S3 event trigger to pass the input events to SQS.**
* **Cloud watch rule and step function to be set to orchestrate the remaining process: -**
  + Lambda to be called at the batch window time to perform the data quality checks first i.e., to validate if it is a 0 kb file/ file with just the headers/ any other validations.
  + Once the validation is successful, lambda will then call the Glue job to process the file and to do the required transformation.
  + **This way we will avoid the calling of Glue in sae of any issues with the file. This will definitely save cost.**
  + **We also need to put different mechanism to handle the replay in elegant ways.**

# **CODE – TECHINCAL DETAILS**

* I have come up with three different versions of the code to do the file read and processing jobs: -
  + **Pyspark: -** To be used in AWS Glue using the spark and sparksql features.
  + **Python for lambda: -** Python code supporting AAWS lambda. This doesn’t have the parquet conversion part i.e., due to the file size part mentioned in the previous sections.
  + **Native python code can be used in PyCharm to access files in local system: -** This code can be executed in the local system using PyCharm. This can read all the files from a local folder and will then try to do the required transformations
* Before getting into the code details, let’s discuss about the **Assumptions**, **Data processing method and improvement ideas**: -
  + **Assumption**: -
    - Fore now, I am just going with one assumption that the ‘-99’ value is being used to notify the ‘Unknown’ records.
  + **Data processing: -**
    - All the different version of the code adheres to the below standards: -
      * Reading multiple files at a time from a single folder.
      * Creating a data frame on top of it to do the further processing.
      * Doing minor data quality checks before processing data.i.e. Dataframe has no records except the headers
      * **Filtering out unnecessary columns and rows.**
      * **One level up aggregation i.e., at Observation date level for a country, region and site.**
      * Converting to parquet (Exception Lambda function)
  + **Improvement ideas: -**
    - More data quality checks: -
      * File size shouldn’t be 0 KB.
      * File with proper headers
      * File with just headers and no data rows
      * And more functional validations if required
* **Code Logic**: -
  + The code will first look for all the files that needs to be processed.
  + It will create dataframe using PYSPARK or PYTHON PANDAS.
  + The consolidated dataframe (i.e., created for all the files in a given particular day), will then be used to do a minor validation i.e., the file contains some data / it shouldn’t contain just the headers.
  + Post the validation, the irrelevant columns are removed.
  + Rows (where screen temp is -99 i.e., assumed to be used for unknown values) are also being filtered out.
  + **Aggregation**: - Post filtering the irrelevant data, I have not copied the entire data into the final dataframe, rather, I have done one level aggregation to have max values per country, region, date. This will get rid of all the irrelevant rows.
  + Then with a simple select query on the data set will help us to extract the hottest temp and corresponding date and region. (I have also included this part in the code).
  + Finally, the data has been converted to parquet wherever possible.
  + **The data processing part is done using by either spark sql/pandas sql/ pandas dataframe native syntax.**

**Output: -**

**Lambda output: -**



**PyCharm output: -**

