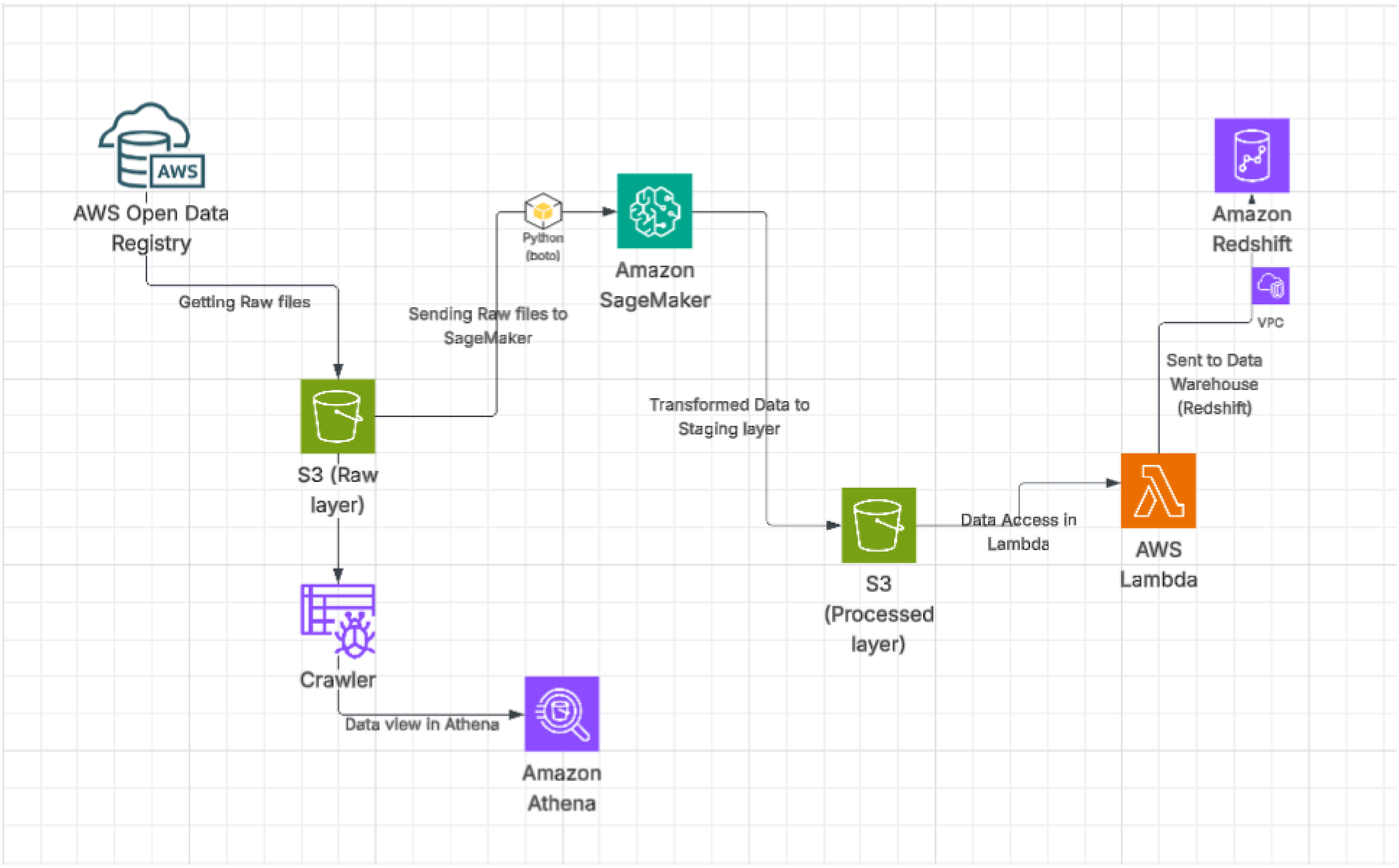
Covid Data Raw to DataWarehouse Ingestion AWS

Git: [Azure-Projects/Covid Data Ingestion from git to Data Warehouse at main · diljotrandhawaa/Azure-Projects]()

Workflow:



The goal of the project:
Get the Raw files about Covid-19 data and transform the data into Dimension and Fact tables to store them in Data Warehouse.

Tech used:
Amazon S3
IAM
Glue Crawler
Amazon Athena
Amazon SageMaker
Amazon SageMaker Studio

Jupyter Notebook
Python boto3
S3 Event Notifications
AWS Lambda
Psycopg2 (Layer in Lambda) (to connect to Redshift directly)
Amazon VPC
Amazon Redshift

Before starting with Implementation:

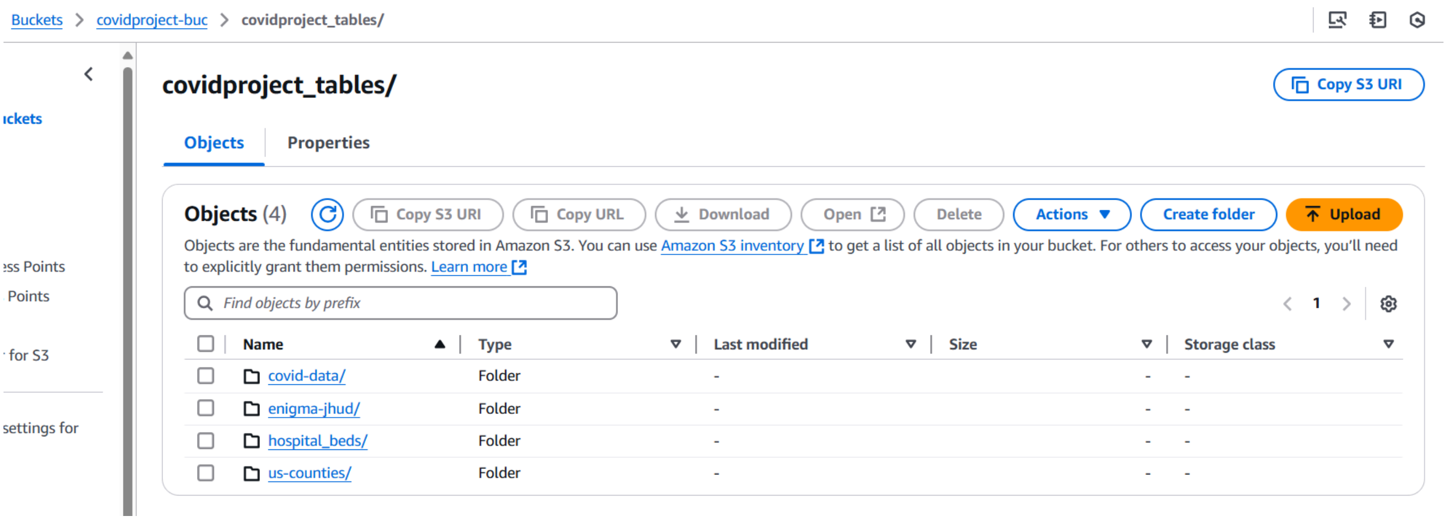
1. Create two S3 buckets, one for Raw layer and one for Transformed/Processed layer.
2. Create a IAM role for the project, give it S3FullAccess, SageMakerFullAccess, LambdaFullAccess and RedshiftFullAccess.
3. Create a SageMaker Studio Domain if not already present.
4. Create a JupyterLab workspace and run it.
5. Create a Lambda function.
6. Download zip file for Pysopg2 layer from the internet. Used zip file is present in the git repository of the project.
7. Create a Redshift namespace and workgroup.

Project Implementation:

Steps:

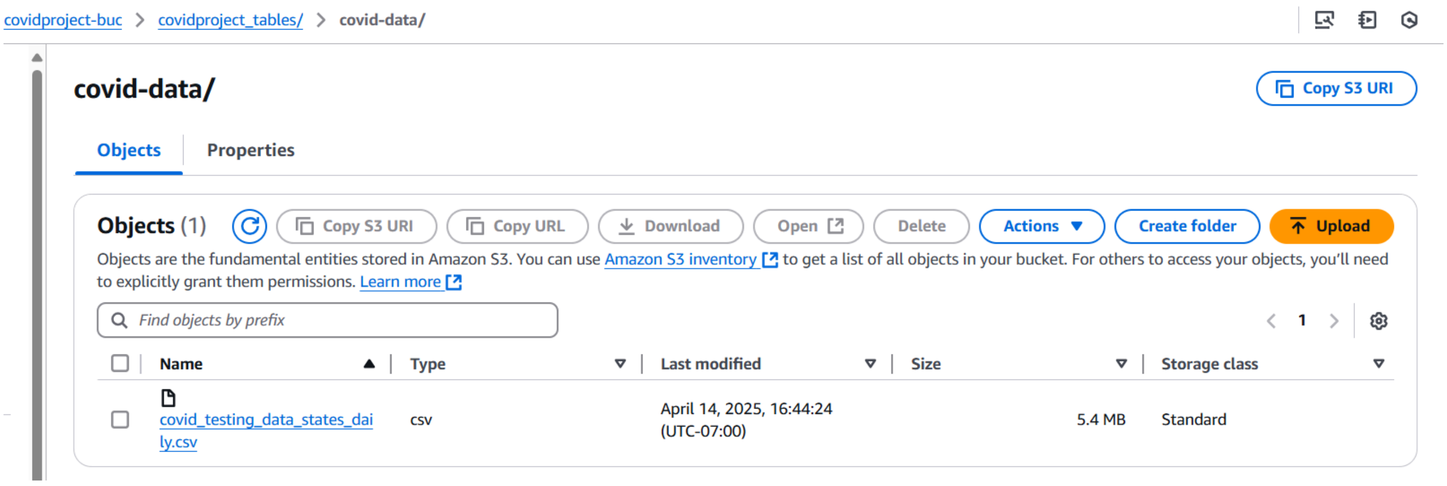
Step 1: Raw Data Collection and Data Ingestion

In the first step, we manually download the raw files from an AWS Open Data Registry that has covid-19 Data. It's called Covid Data Lake.
We upload the files to S3 bucket (Raw layer).
Inside the Raw layer bucket, the four raw files are stored in 4 different folders (directories) so that the crawler creates one table for each raw file.
Below is the structure of the Raw bucket:



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
covid-data/	Folder	-	-	-
enigma-jhud/	Folder	-	-	-
hospital_beds/	Folder	-	-	-
us-counties/	Folder	-	-	-


covidproject-buc > covidproject_tables/ > covid-data/

covid-data/

Objects (1)

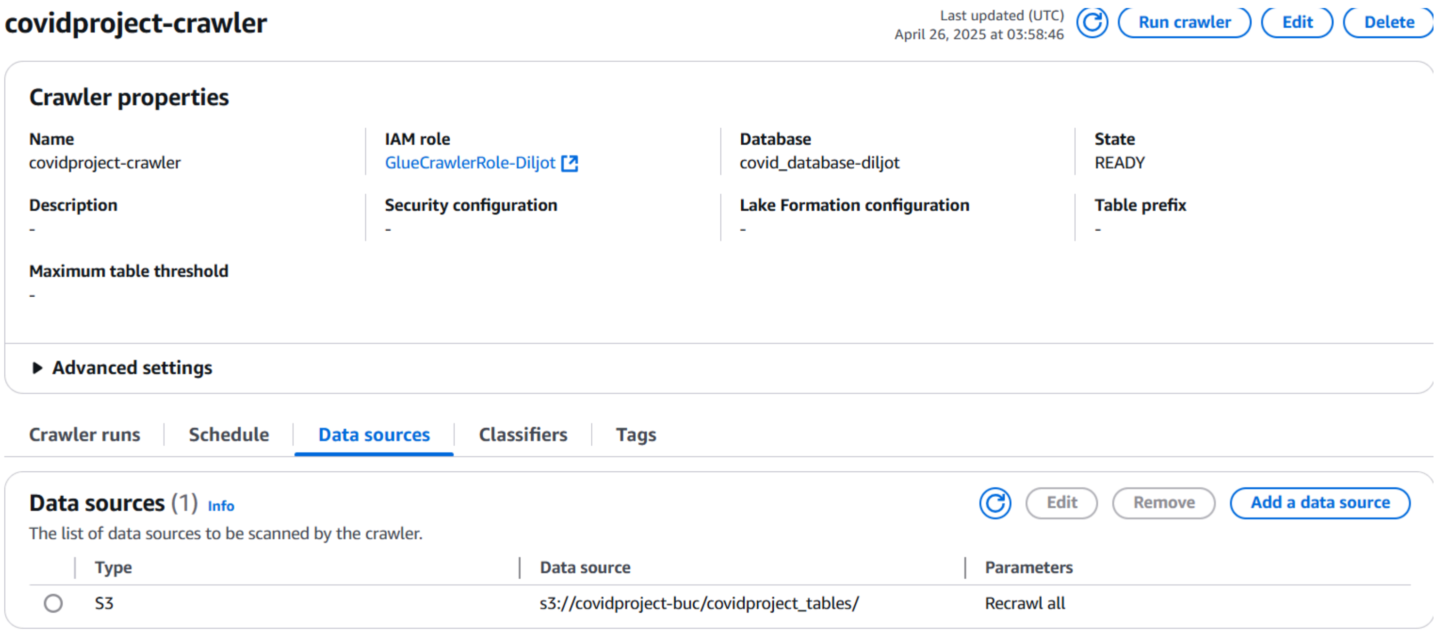
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
covid_testing_data_states_daily.csv	csv	April 14, 2025, 16:44:24 (UTC-07:00)	5.4 MB	Standard

Each folder is named after the Raw file that is stored inside it.

Step 2: Create Crawler for raw bucket and processed bucket (Data Quality Check)

We **create a Crawler for the Raw data**. The crawler points to the Raw S3 bucket. The source is directory containing four directories for raw files.


covidproject-crawler

Last updated (UTC) April 26, 2025 at 03:58:46

Crawler properties

Name: covidproject-crawler	IAM role: GlueCrawlerRole-Diljot	Database: covid_database-diljot	State: READY
Description: -	Security configuration: -	Lake Formation configuration: -	Table prefix: -
Maximum table threshold: -			

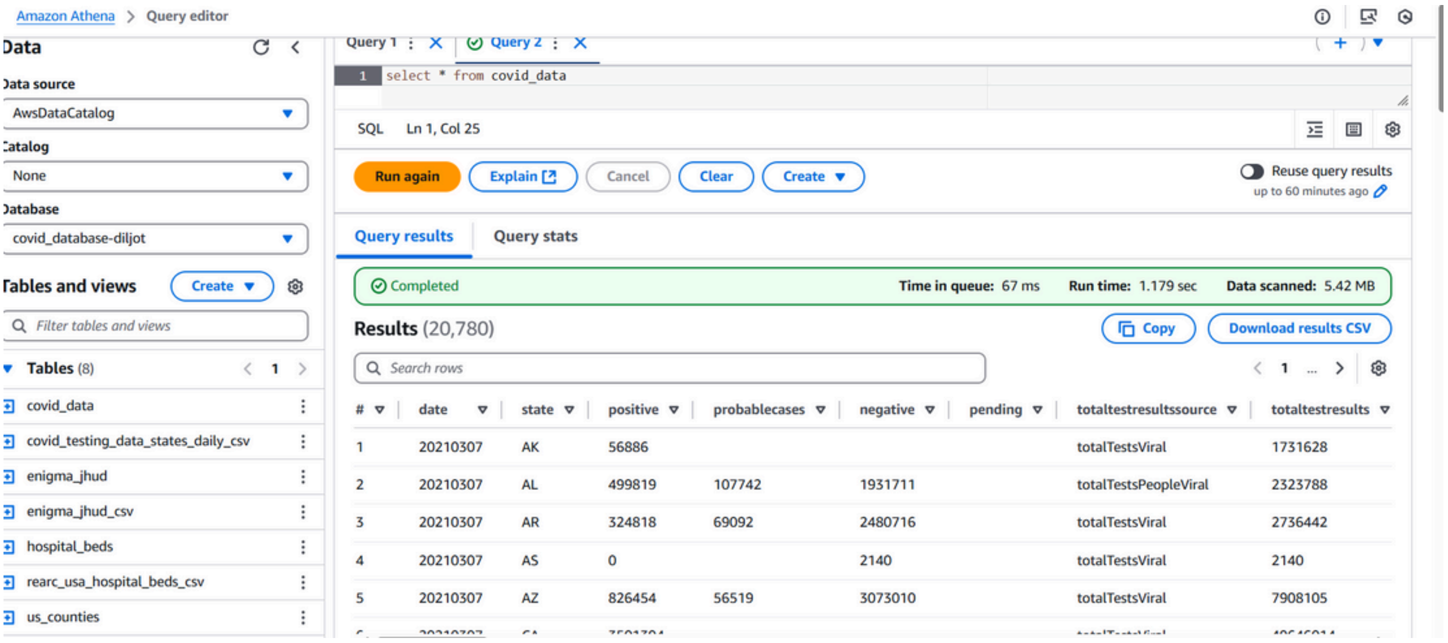
Advanced settings

Data sources (1)

The list of data sources to be scanned by the crawler.

Type	Data source	Parameters
S3	s3://covidproject-buc/covidproject_tables/	Recrawl all

After creating, we run the crawler.
The data can be viewed in Athena for query purposes and to understand the structure of the raw data.
Below is what the "covid_data" raw file looks like


Amazon Athena > Query editor

Data

Data source: AwsDataCatalog

Catalog: None

Database: covid_database-diljot

Tables and views

Tables (8)
- covid_data
- covid_testing_data_states_daily_csv
- enigma_jhud
- enigma_jhud_csv
- hospital_beds
- rearc_usa_hospital_beds_csv
- us_counties

Query 1 | Query 2

```
select * from covid_data
```

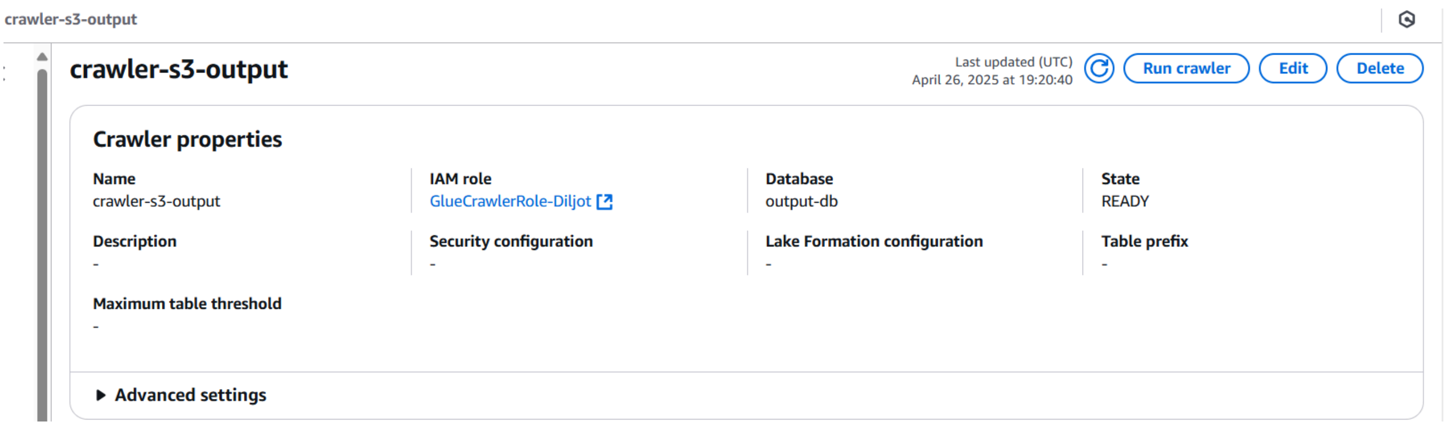
SQL Ln 1, Col 25

Completed — Time in queue: 67 ms — Run time: 1.179 sec — Data scanned: 5.42 MB

Results (20,780)

#	date	state	positive	probablecases	negative	pending	totaltestresultssource	totaltestresults
1	20210307	AK	56886				totalTestsViral	1731628
2	20210307	AL	499819	107742	1931711		totalTestsPeopleViral	2323788
3	20210307	AR	324818	69092	2480716		totalTestsViral	2736442
4	20210307	AS	0		2140		totalTestsViral	2140
5	20210307	AZ	826454	56519	3073010		totalTestsViral	7908105

Next, we **create one Crawler for the Processed S3 bucket** to view the processed data in Athena. We don’t run it yet.


crawler-s3-output

Last updated (UTC) April 26, 2025 at 19:20:40

Crawler properties

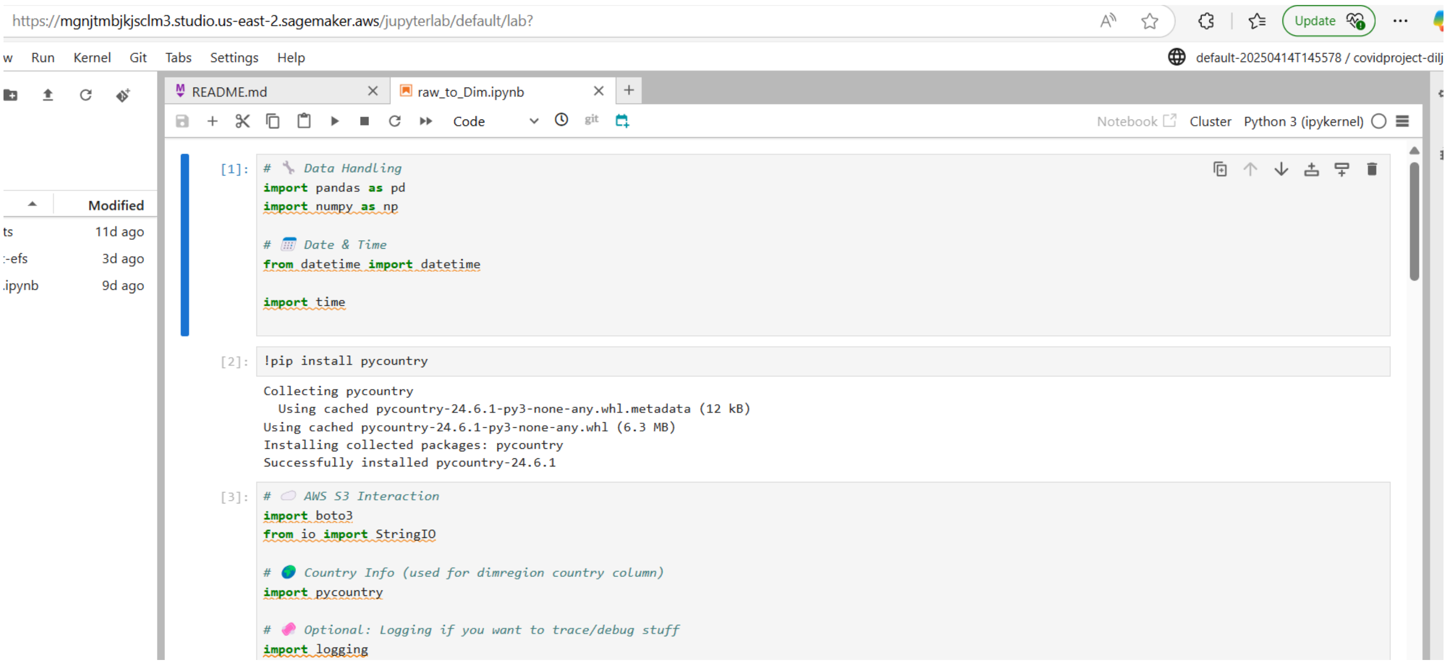
Name: crawler-s3-output	IAM role: GlueCrawlerRole-Diljot	Database: output-db	State: READY
Description: -	Security configuration: -	Lake Formation configuration: -	Table prefix: -
Maximum table threshold: -			

Advanced settings

Step 3: **Jupyter code implementation (Data Transformation)**

Now that we understand the structure of the data better, we export the data to Jupyter notebook in SageMaker Studio’s Jupyterlab to transform it further.

Below is the small snippet of the Jupyter notebook, the code file is present in the git repository.


https://mgnjtmbjkjsclm3.studio.us-east-2.sagemaker.aws/jupyterlab/default/lab?
README.md
raw_to_Dim.ipynb
Code
Python 3 (ipykernel)

```
[1]: # Data Handling
import pandas as pd
import numpy as np

# Date & Time
from datetime import datetime

import time
```

```
[2]: !pip install pycountry
```

```
Collecting pycountry
  Using cached pycountry-24.6.1-py3-none-any.whl.metadata (12 kB)
Using cached pycountry-24.6.1-py3-none-any.whl (6.3 MB)
Installing collected packages: pycountry
Successfully installed pycountry-24.6.1
```

```
[3]: # AWS S3 Interaction
import boto3
from io import StringIO

# Country Info (used for dimregion country column)
import pycountry

# Optional: Logging if you want to trace/debug stuff
import logging
```

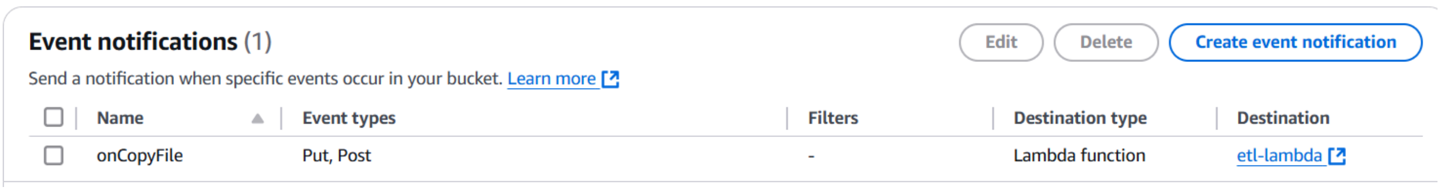
In the notebook, we first import the raw data from S3 using boto3. We then create our Dimension tables and Fact tables using merge and other operations.
The dimension tables for Hospitals, Region and Date are created while a Fact table is created for Covid data.
(dimHospital, dimRegion, dimDate, factsCovid) are created.

Using the code in Jupyter notebook, the dim and fact tables will be exported to our Processed Layer (S3) using boto3 and StringIO buffer in the form of CSV files.

Step 4: Test the Processed data and Create Event notification for automation (Automation)

We can test run the code to see if we receive the correct data in our Processed S3 bucket. Run the Crawler to view data in Athena. After the data is verified and validated, it is then ready to move to Data Warehouse in Redshift.

To automate the ingestion, we create a Event notification on the Processed S3 bucket and connect it to the Lambda function created earlier. The Lambda function does not have any code yet, we'll move on to the Lambda function now.
The event notification created can be seen below.



Send a notification when specific events occur in your bucket. Learn more

Name	Event types	Filters	Destination type	Destination
onCopyFile	Put, Post	-	Lambda function	etl-lambda

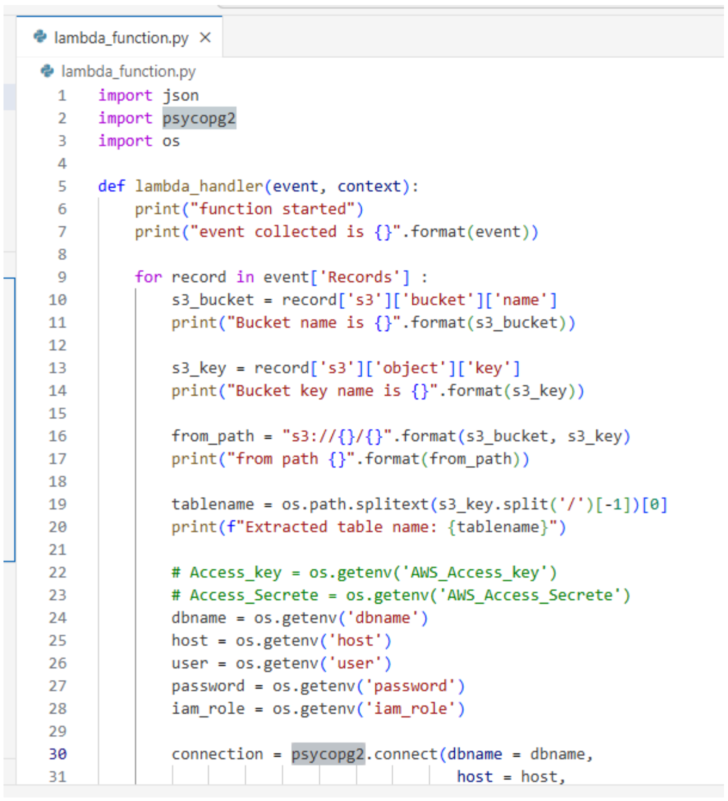
Step 5: Configure the Lambda function (Data Storage)

We go to the Lambda function and write the code. But before that, we need to add a custom layer for Psycopg2 to connect directly to Redshift. To do that, we need a zip file for the layer, the used zip file can be found in the git repository.

After installing and attaching the layer to the Lambda function, we create some Enviornment variables for cleaner code and for safety. The variables are created for Access keys and credentials to connect to Redshift.

The code is written in the Lambda function. Code simply imports the dimension and fact tables from Processed layer. It directly connects to the Redshift and creates four tables in Redshift. It then copies the data from S3 to Redshift.

The lambda code file can be found in the git repository of the project.



```python
import json
import psycopg2
import os

def lambda_handler(event, context):
    print("function started")
    print("event collected is {}".format(event))

    for record in event['Records'] :
        s3_bucket = record['s3']['bucket']['name']
        print("Bucket name is {}".format(s3_bucket))

        s3_key = record['s3']['object']['key']
        print("Bucket key name is {}".format(s3_key))

        from_path = "s3://{}/{}".format(s3_bucket, s3_key)
        print("from path {}".format(from_path))

        tablename = os.path.splitext(s3_key.split('/')[-1])[0]
        print(f"Extracted table name: {tablename}")

        # Access_key = os.getenv('AWS_Access_key')
        # Access_Secrete = os.getenv('AWS_Access_Secrete')
        dbname = os.getenv('dbname')
        host = os.getenv('host')
        user = os.getenv('user')
        password = os.getenv('password')
        iam_role = os.getenv('iam_role')

        connection = psycopg2.connect(dbname = dbname,
                                      host = host,
```

Now deploy the function.

Step 6: Add VPC Security group to the Lambda function

To successfully connect to Redshift, make sure your Lambda function is configured to connect to same VPC Security group as the Redshift workgroup.

Now run the code and the data will be sent to the Data warehouse Redshift.

Code run example:

So all we have to do is run the Jupyter notebook. As soon as we run the notebook and data gets sent to the Processed bucket, it will automatically trigger the Lambda function and copies the data in the Redshift warehouse.