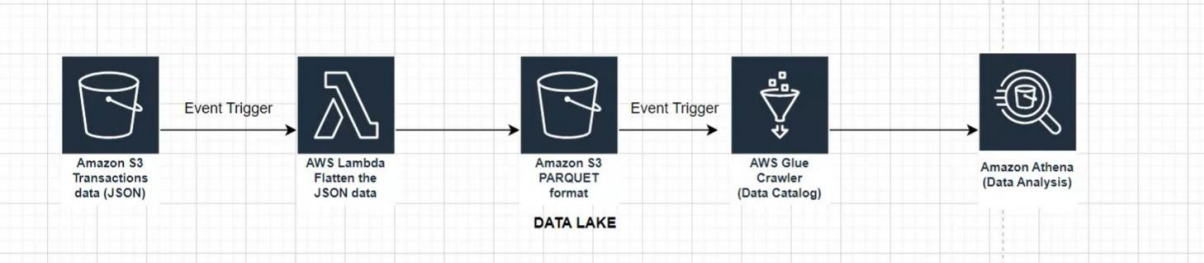
AWS Serverless ETL Pipeline

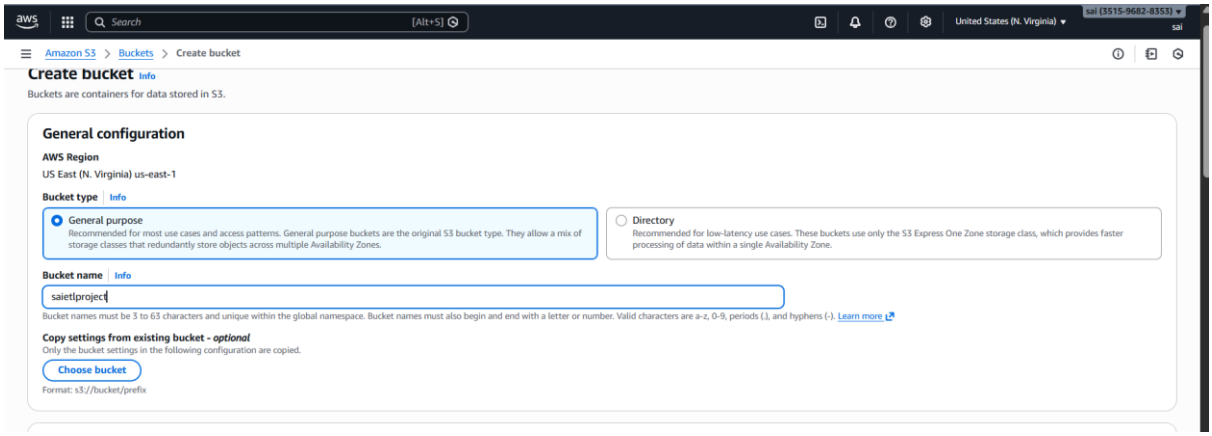
Event Driven Data Pipeline:

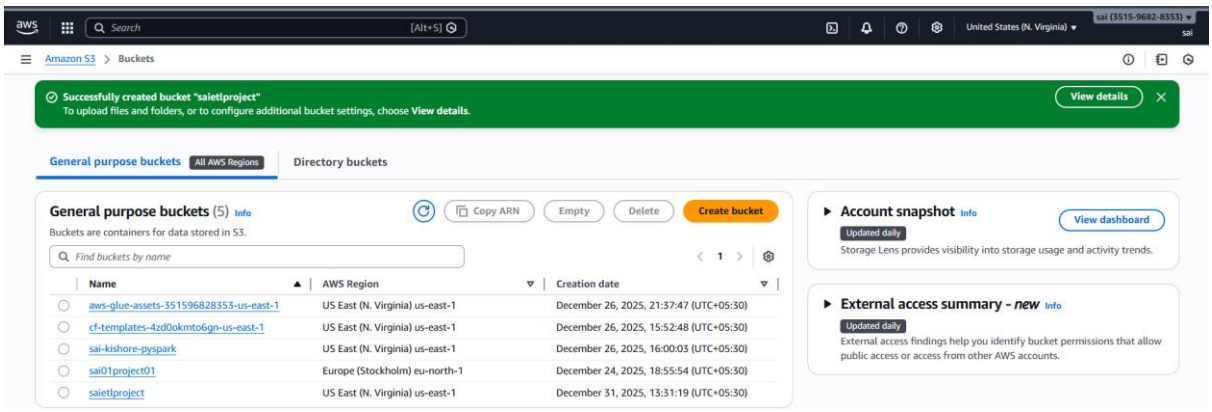
We will have to load a file in S3 and as soon as we load, everything will happen automatically as per the pipeline, we don’t need to schedule



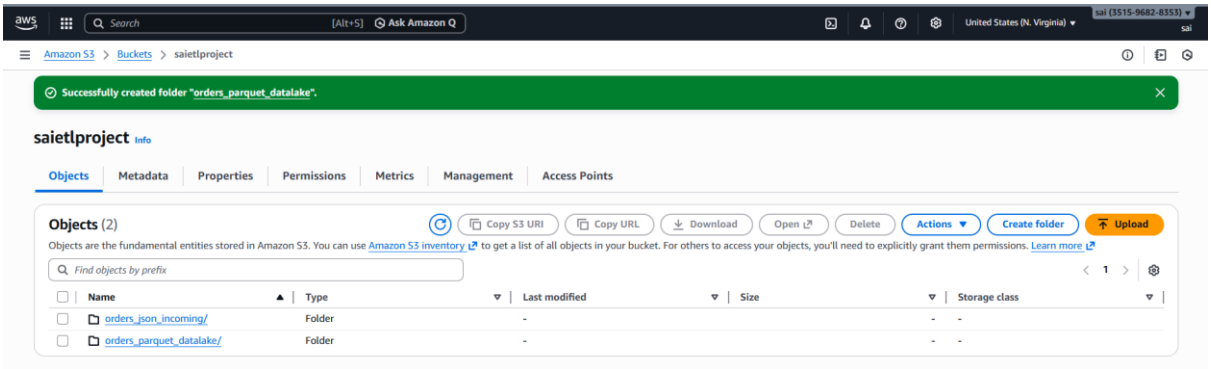
AWS lambda is a serverless compute which can handle any kind of processing without provisioning any EC2 server

Creating a bucket as shown

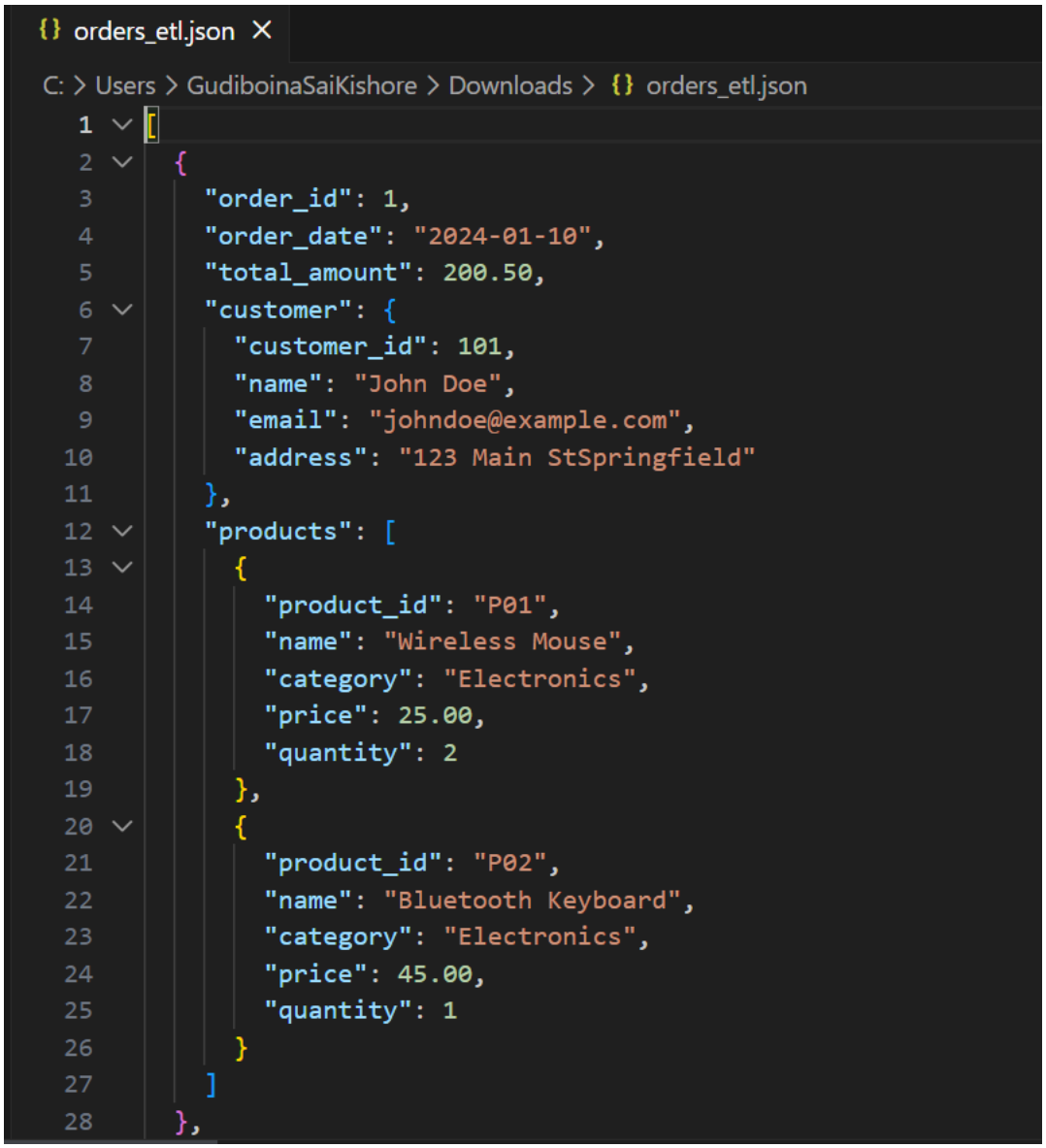




We had create two folders as shown



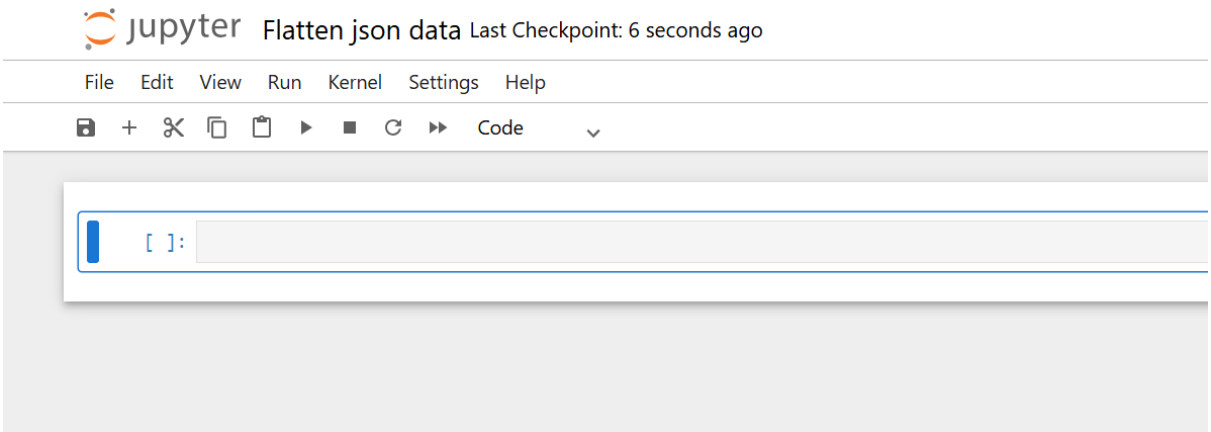
Lets see the orders data file



File is uploaded in GIT

Lets see how to flatten the data

Open ur Jupyter notebook



Load the json data to read

```
[1]: import json
     f = open('orders_etl.json', 'r')
     data = json.load(f)

[2]: data
```

```
[2]: [{'order_id': 1,
       'order_date': '2024-01-10',
       'total_amount': 200.5,
       'customer': {'customer_id': 101,
        'name': 'John Doe',
        'email': 'johndoe@example.com',
        'address': '123 Main StSpringfield'},
       'products': [{'product_id': 'P01',
         'name': 'Wireless Mouse',
         'category': 'Electronics',
         'price': 25.0,
         'quantity': 2},
        {'product_id': 'P02',
         'name': 'Bluetooth Keyboard',
         'category': 'Electronics',
         'price': 45.0,
```

We loaded the data into variable ‘data’

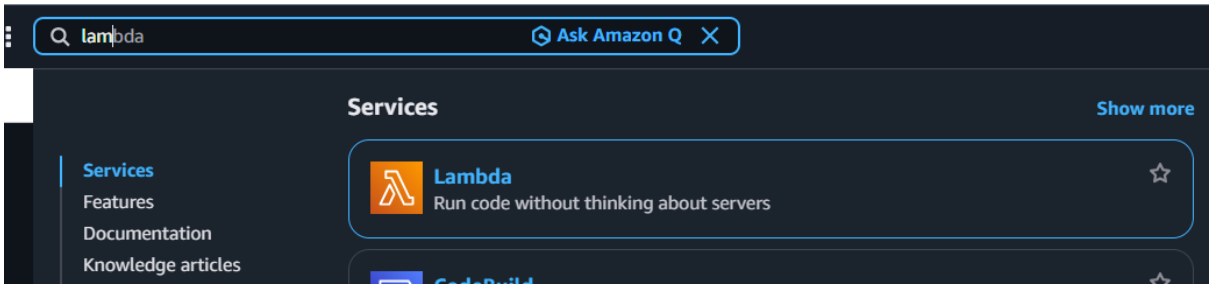
```
[6]: #Lets flatten data and convert to a pandas dataframe
import pandas as pd
def flatten(data):
    orders_data=[]
    for order in data:
        for product in order['products']:
                    row_orders = {
                        "order_id" : order["order_id"],
                        "order_date" : order["order_date"],
                        "total_amount" : order["total_amount"],
                        "customer_id" : order["customer"]["customer_id"],
                        "customer_name" : order["customer"]["name"],
                        "email" : order["customer"]["email"],
                        "address" : order["customer"]["address"],
                        "product_id" : product["product_id"],
                        "product_name" : product["name"],
                        "category" : product["category"],
                        "price" : product["price"],
                        "quantity" : product["quantity"]
                    }
                    orders_data.append(row_orders)
    df_orders = pd.DataFrame(orders_data)
    return df_orders
```

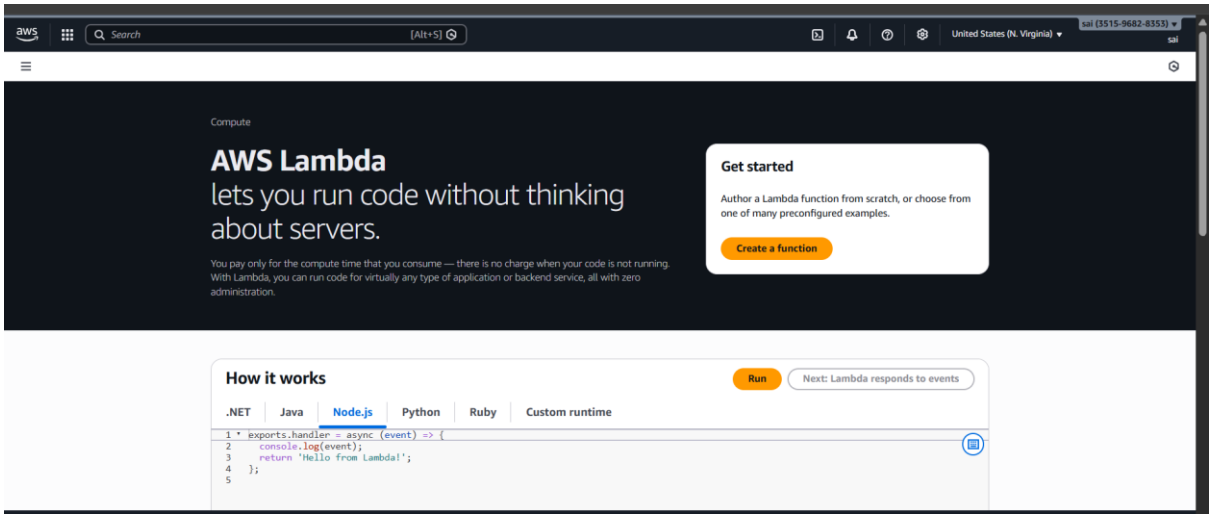
```
[7]: df = flatten(data)
df
```

[7]:

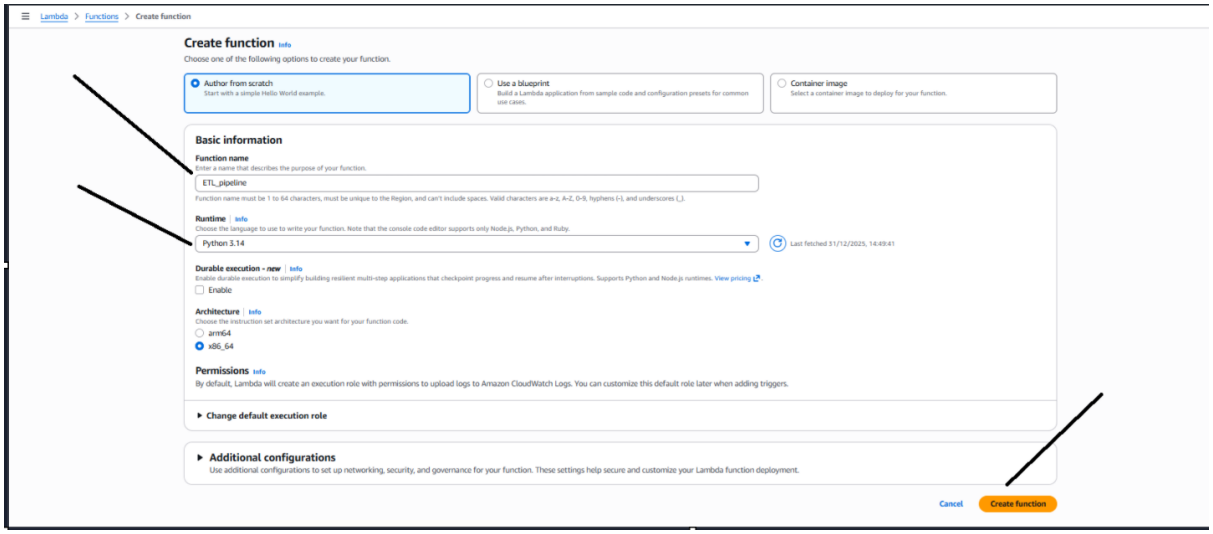
	order_id	order_date	total_amount	customer_id	customer_name	email	address	product_id	product_name	category	price	quantity
0	1	2024-01-10	200.5	101	John Doe	johndoe@example.com	123 Main StSpringfield	P01	Wireless Mouse	Electronics	25.0	2
1	1	2024-01-10	200.5	101	John Doe	johndoe@example.com	123 Main StSpringfield	P02	Bluetooth Keyboard	Electronics	45.0	1
2	2	2024-01-12	150.0	102	Jane Smith	janesmith@example.com	456 Oak StSpringfield	P03	Laptop Stand	Electronics	75.0	1
3	3	2024-01-12	120.0	103	Alice Johnson	alicejohnson@example.com	789 Birch StSpringfield	P04	Gaming Headset	Electronics	60.0	2
4	4	2024-01-13	300.5	101	John Doe	johndoe@example.com	123 Main StSpringfield	P01	Wireless Mouse	Electronics	25.0	3
5	4	2024-01-13	300.5	101	John Doe	johndoe@example.com	123 Main StSpringfield	P02	Bluetooth Keyboard	Electronics	45.0	2
6	5	2024-01-14	180.0	104	Bob Brown	bobbrown@example.com	101 Maple StSpringfield	P03	Laptop Stand	Electronics	75.0	1
7	5	2024-01-	180.0	104	Bob Brown	bobbrown@example.com	101 Maple	P04	Gaming	Electronics	60.0	1

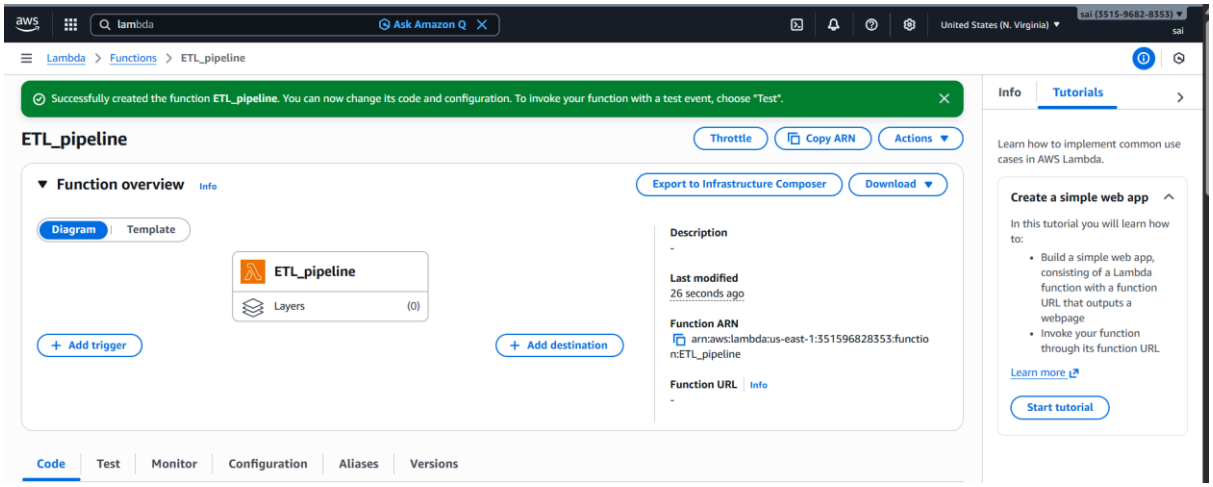
Now lets go to lambda to process this data



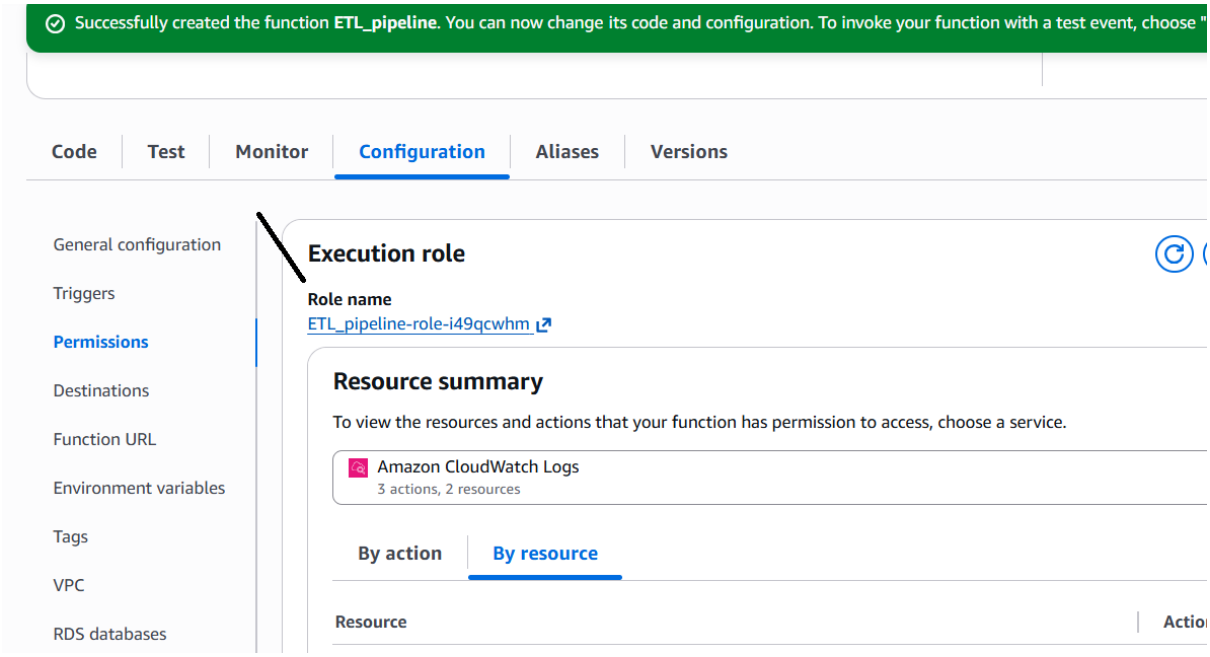


Lets create a function



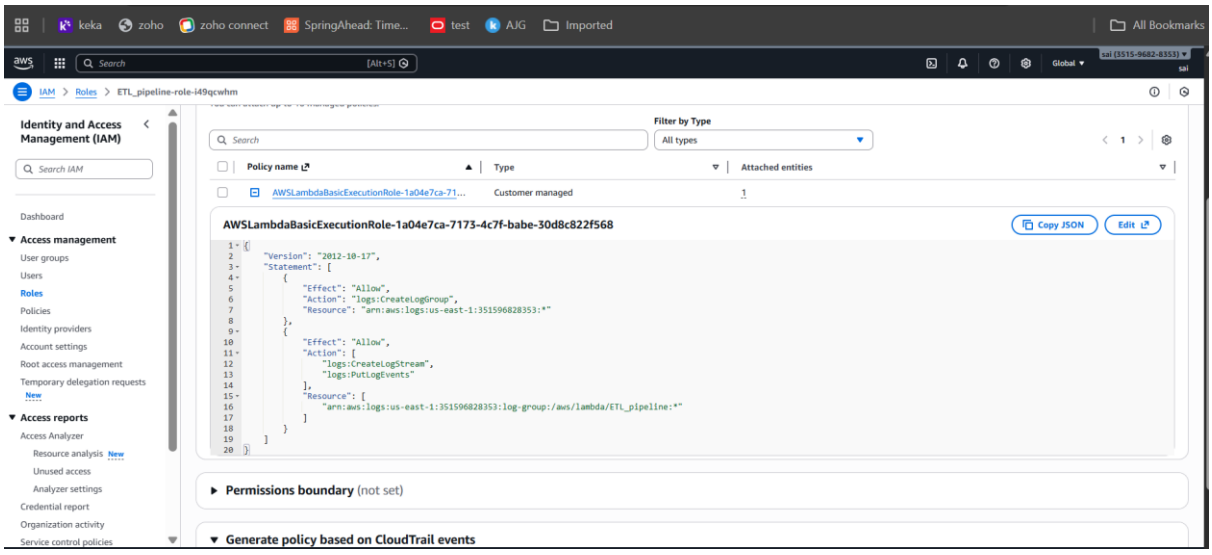


As this function is created, we want to read the function from s3

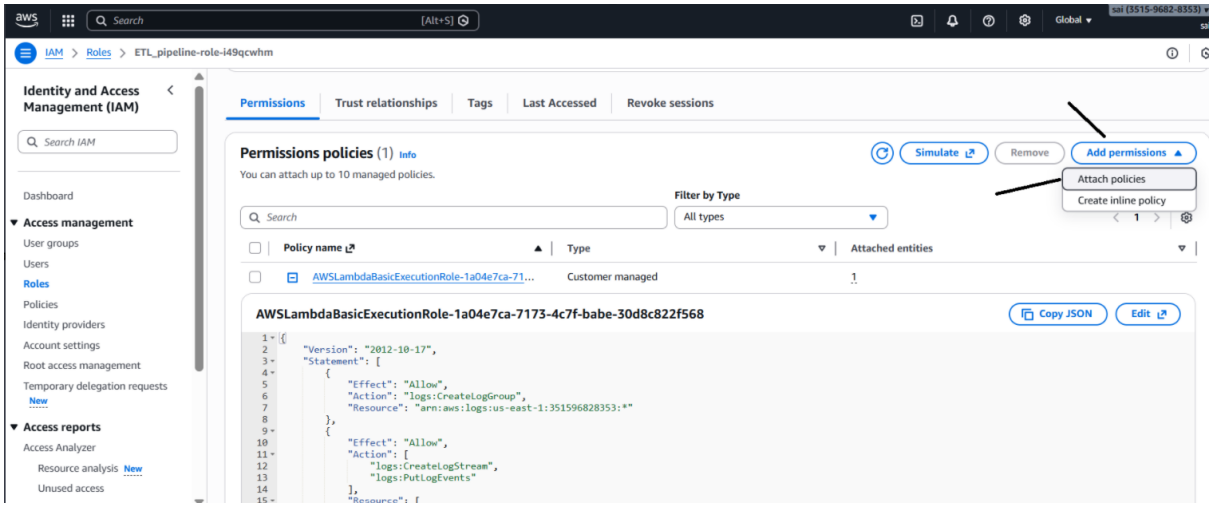


When we create a lamda function, it by default created a role as shown above i.e whatever permissions this role has, those only can be done by lambda function

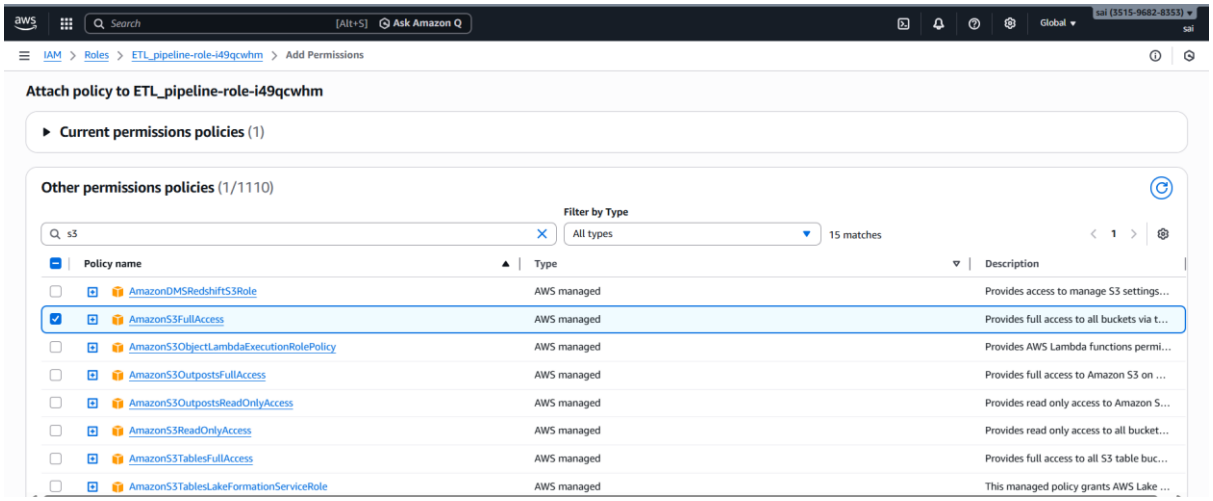
Lets click on that role and check



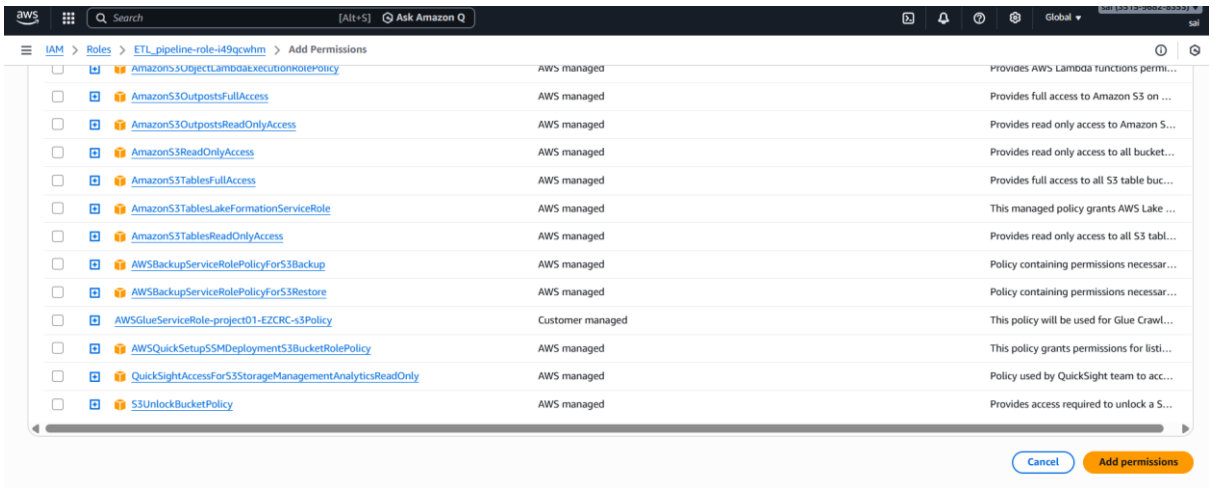
As we want to read data from s3, let us give s3 access as well

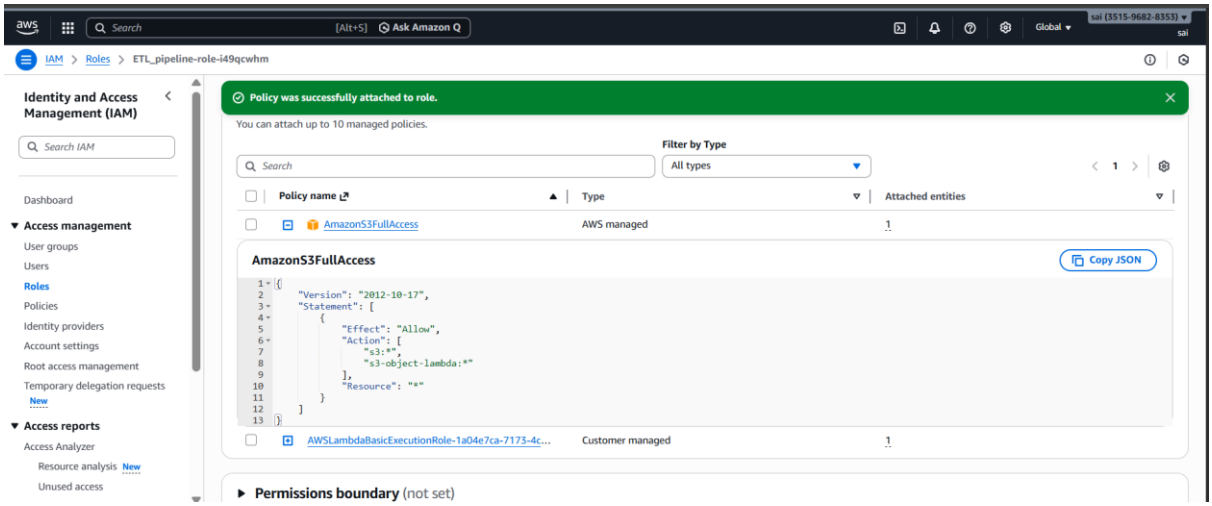


Giving full access to S3

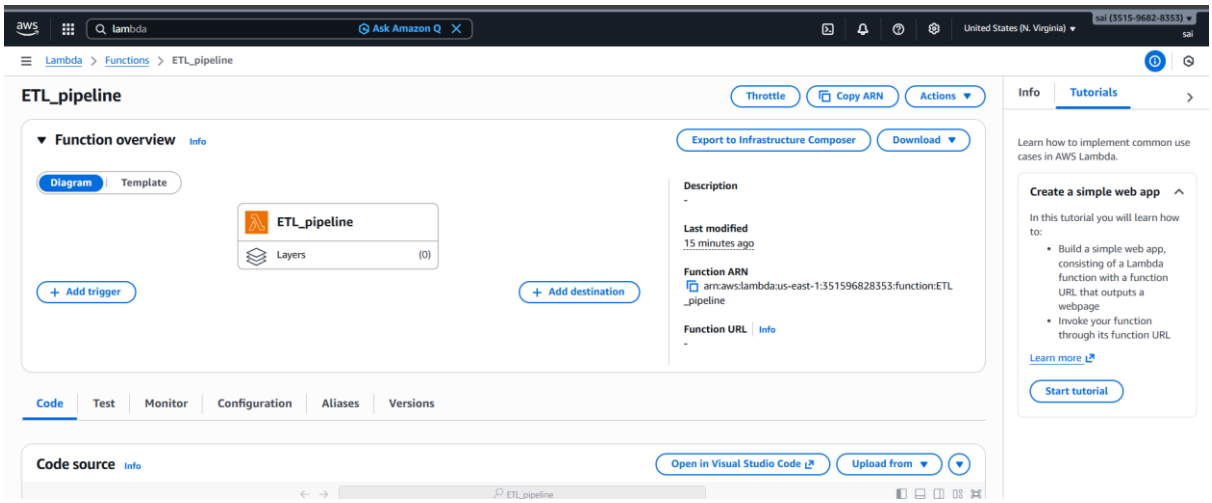


Click on add permission





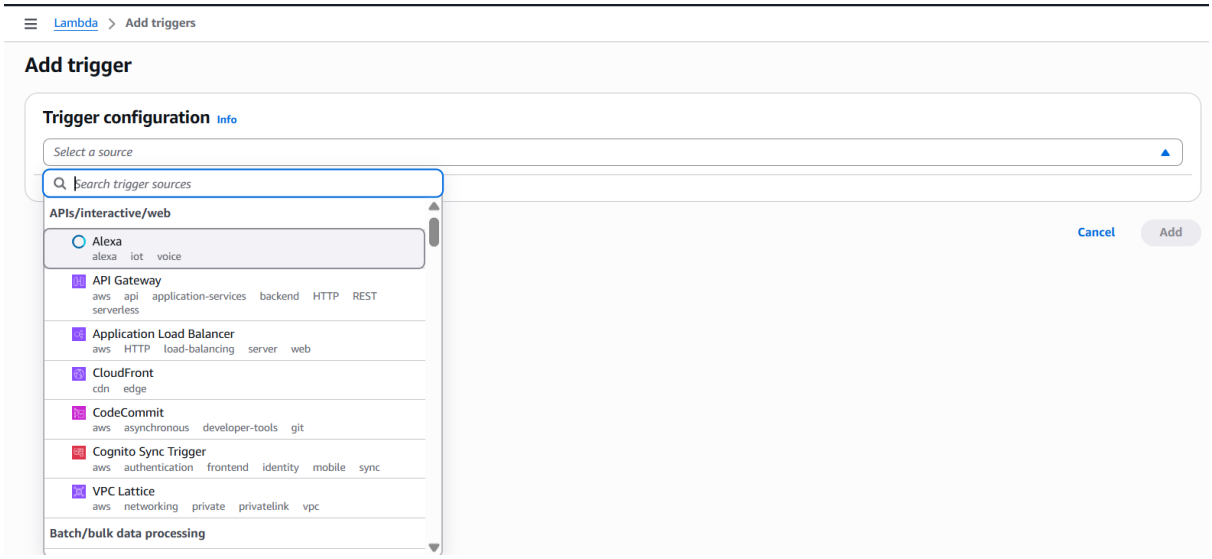
Lets go back to lambda

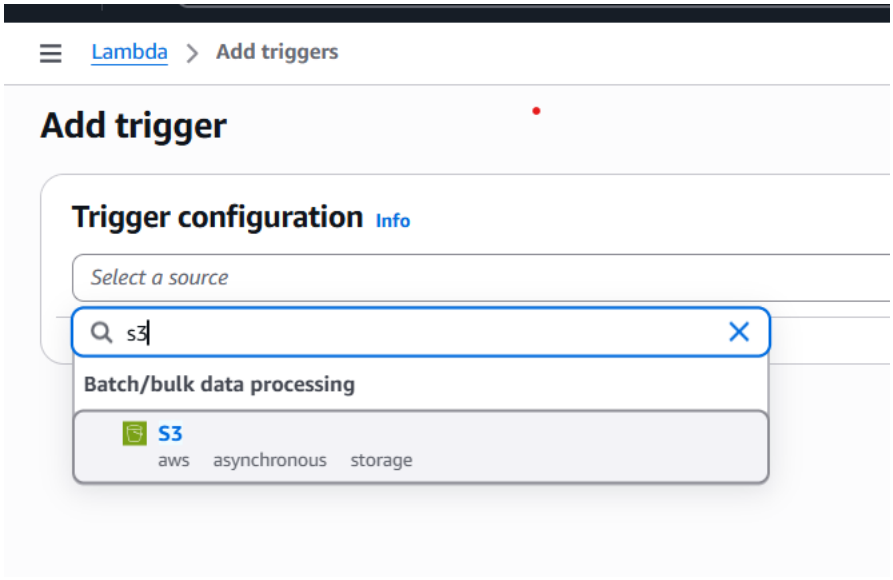


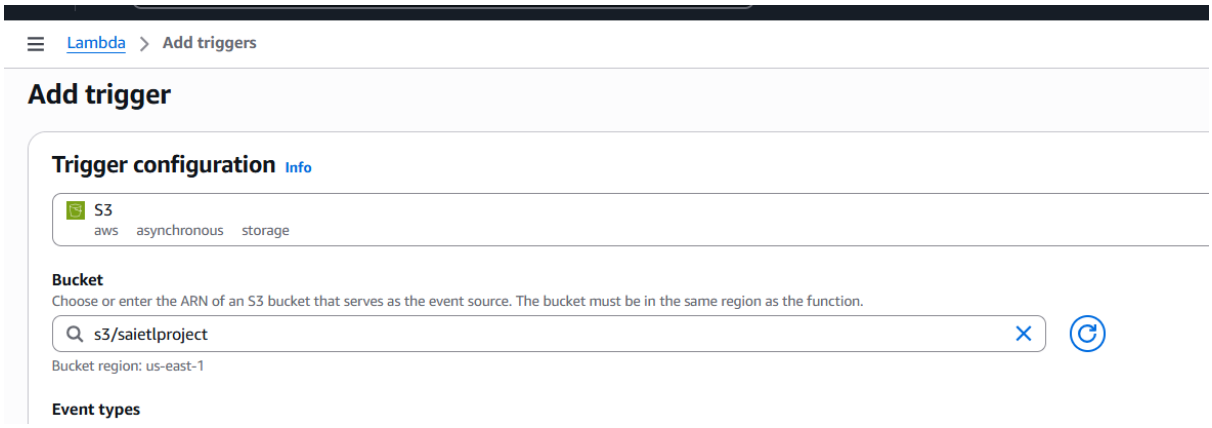
We need to add trigger becoz as soon as file available in s3, this lambda function has to be executed and we write a code in this function to flatten under code section

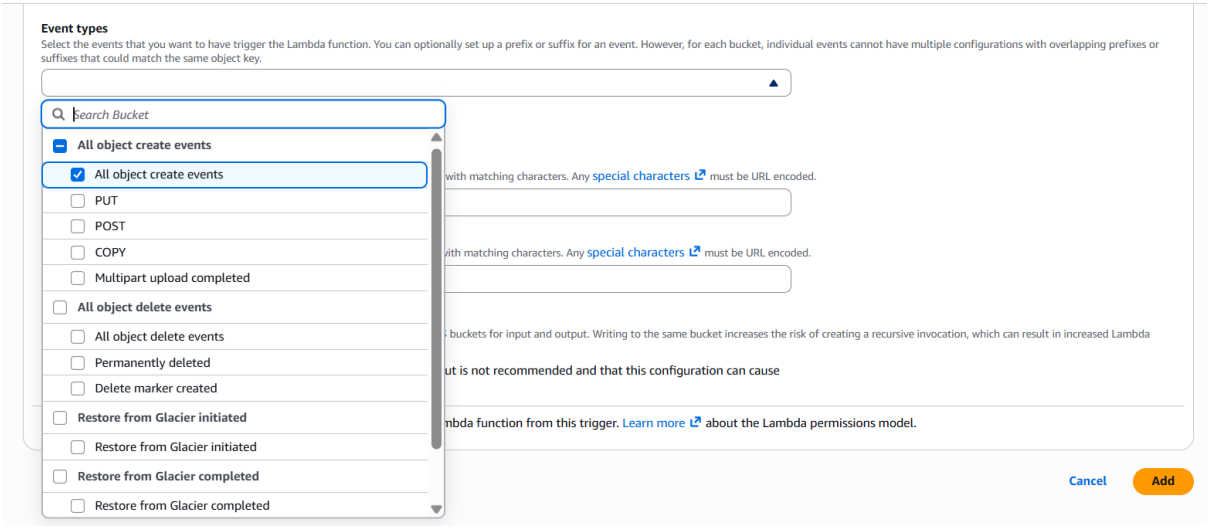
Click on add trigger

There are lot of ways to trigger lambda as follows but we choose s3 as shown

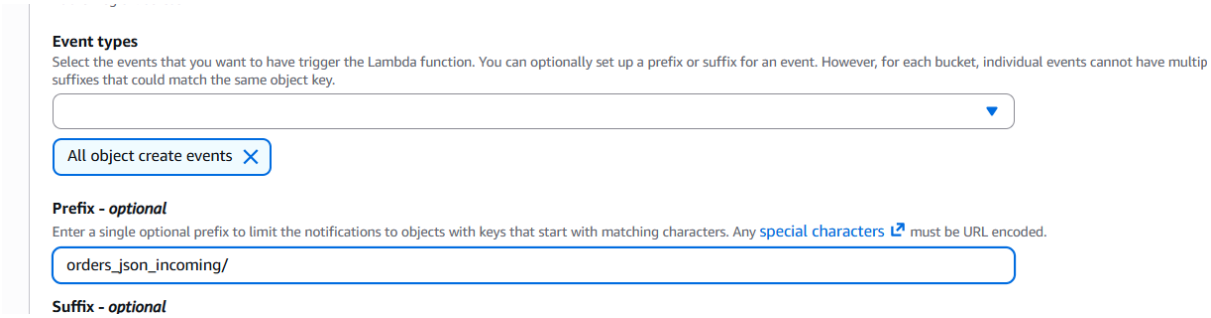

Lambda > Add triggers
Add trigger
Trigger configuration Info
Select a source
Search trigger sources
APIs/interactive/web
Alexa
alexa iot voice
API Gateway
aws api application-services backend HTTP REST serverless
Application Load Balancer
aws HTTP load-balancing server web
CloudFront
cdn edge
CodeCommit
aws asynchronous developer-tools git
Cognito Sync Trigger
aws authentication frontend identity mobile sync
VPC Lattice
aws networking private privatelink vpc
Batch/bulk data processing
Cancel
Add


Lambda > Add triggers
Add trigger
Trigger configuration Info
Select a source
s3
Batch/bulk data processing
S3
aws asynchronous storage


Lambda > Add triggers
Add trigger
Trigger configuration Info
S3
aws asynchronous storage
Bucket
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3/saietlproject
Bucket region: us-east-1
Event types

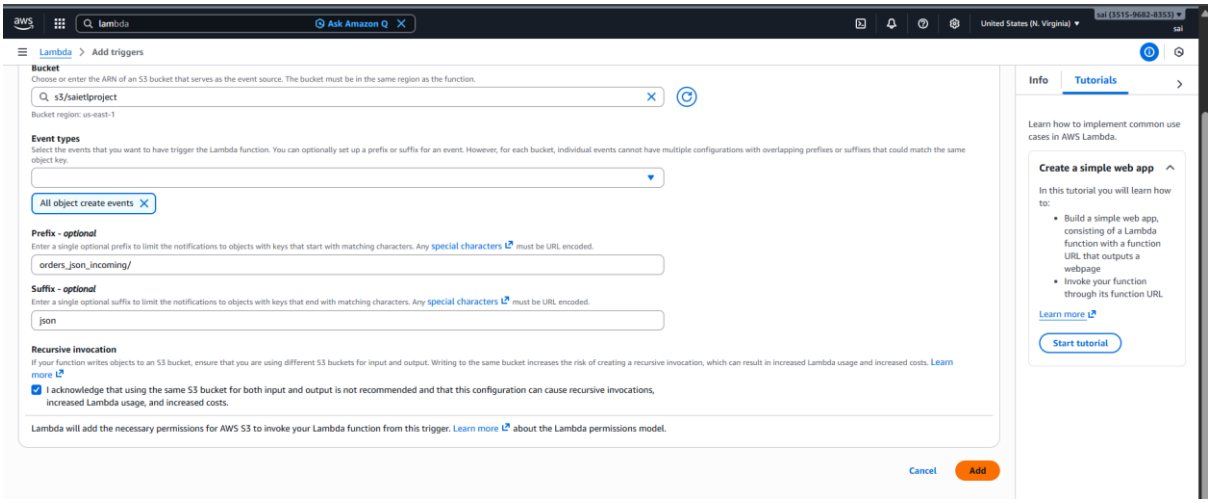


Whenever u give prefix implies when ever a file is available in that prefix, then lamda gets triggered

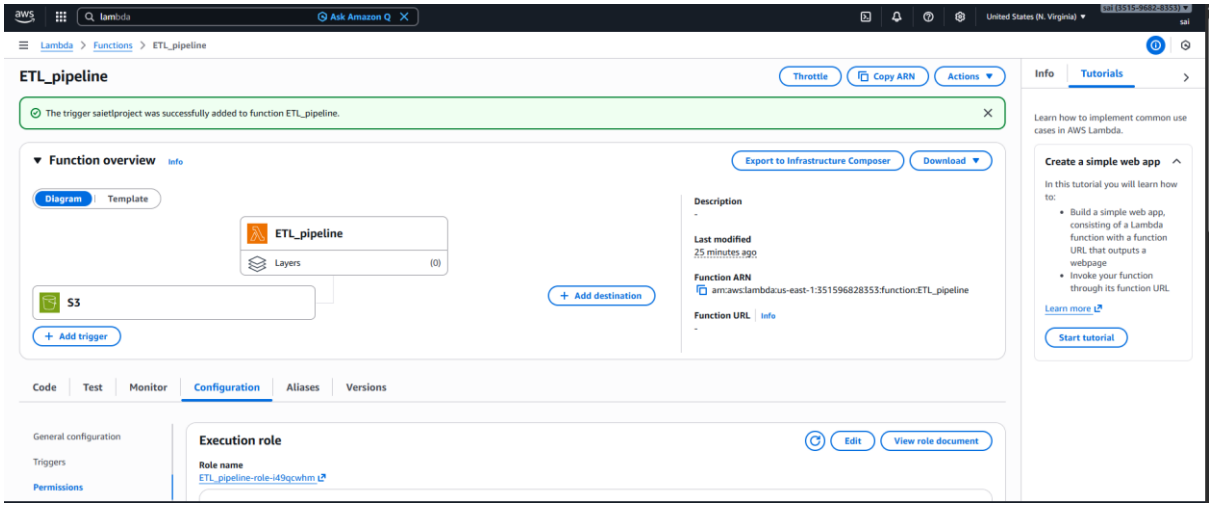


In suffix u can give json in our case as it is optional as the overall meaning is w.r.t to prefix or suffix lambda fn has to be triggered

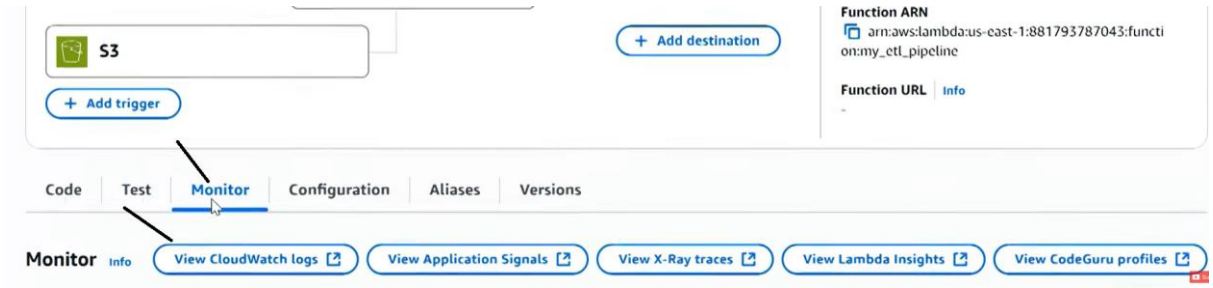
Acknowledge and add



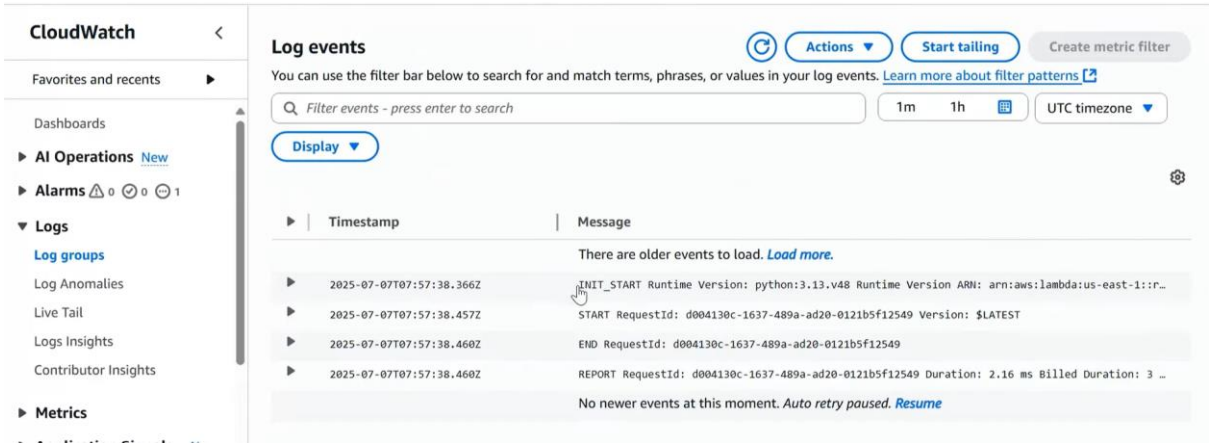
U see the interface as shown



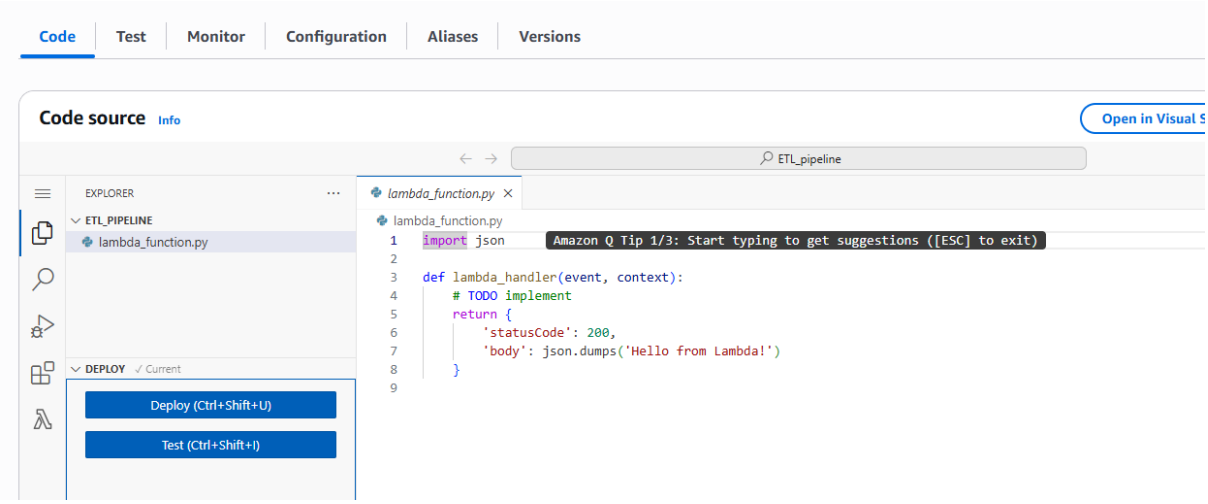
Lets test by uploading our orders file in that s3 path



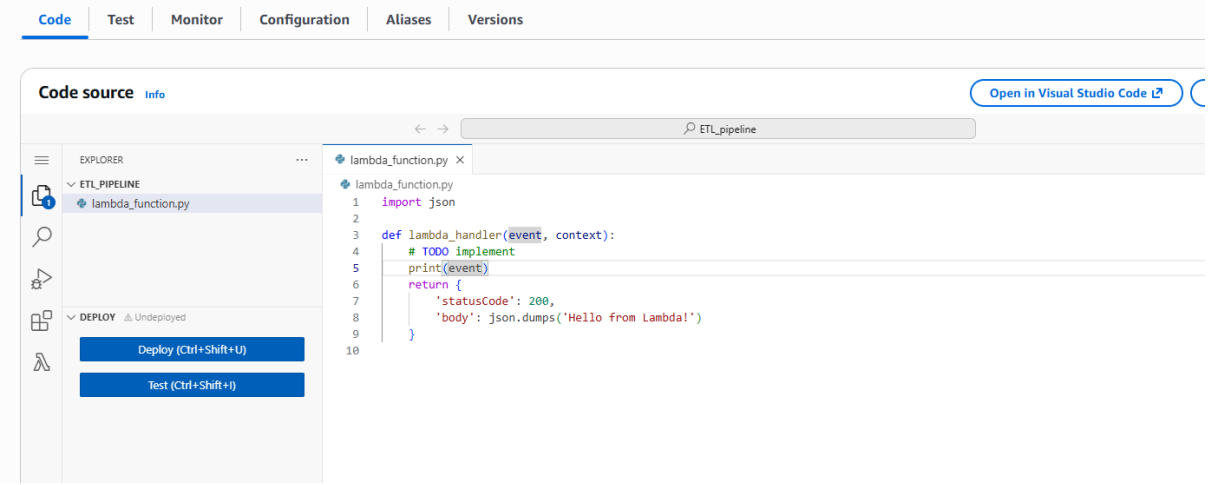
Go to monitor, click view cloudwatch logs



When u write a python code in lambda, the below function(lambda_handler) will automatically be called, u don’t need to write separate function

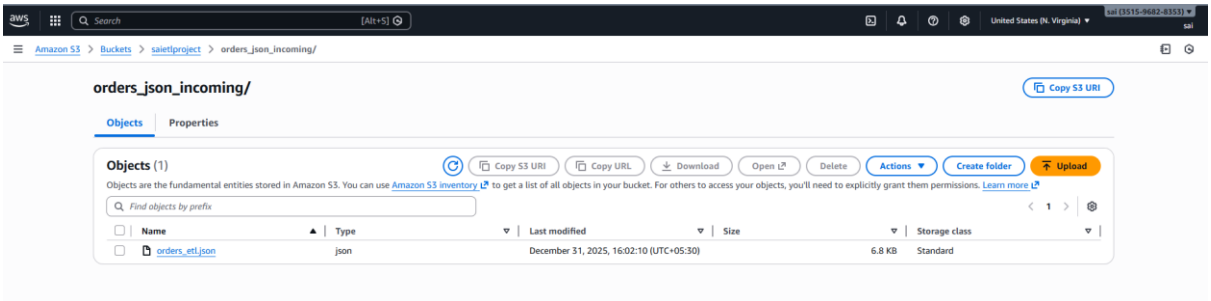


As we are testing lets write a print statement as shown

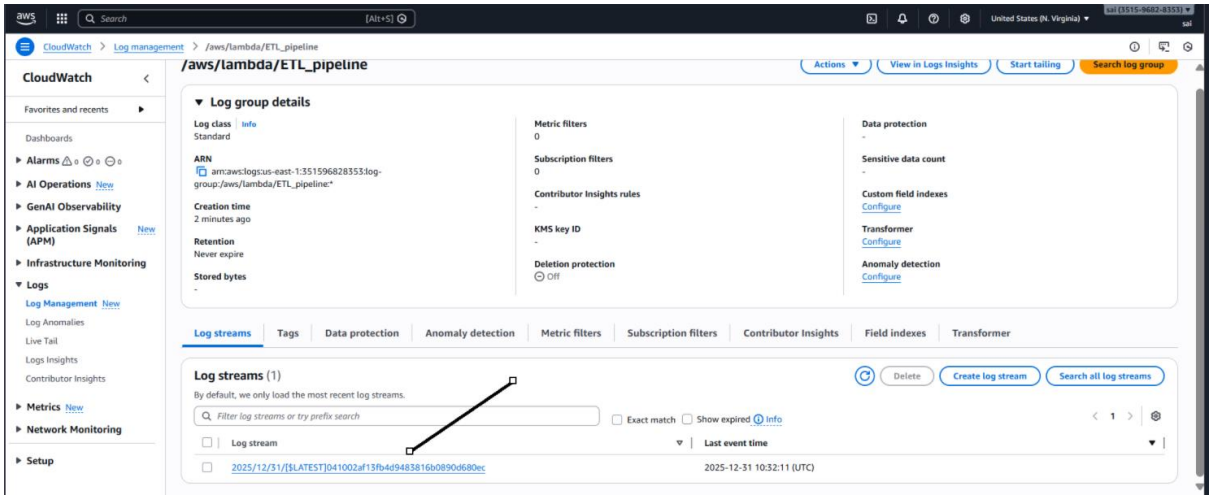


Click on on deploy

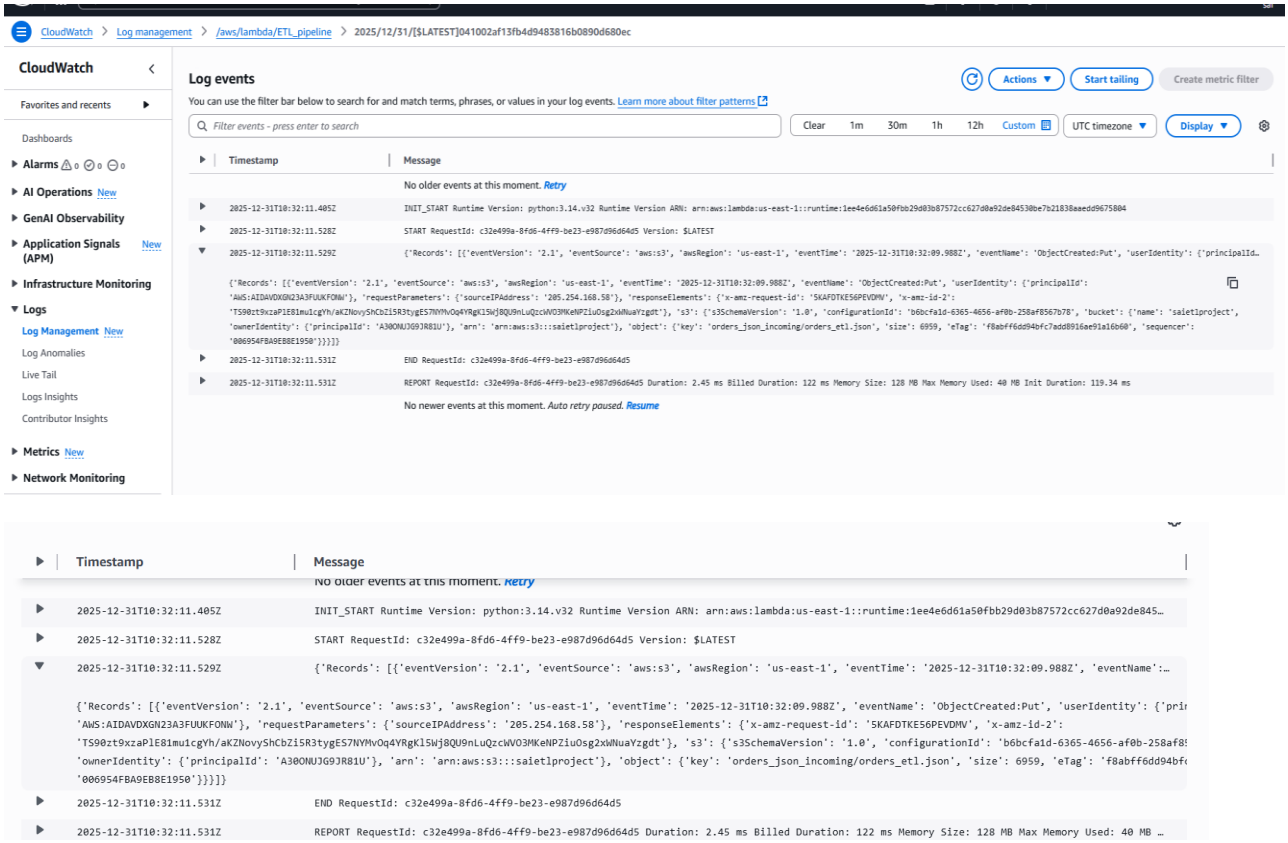
Now upload the file in the s3 bucket



Lets go to cloud watch and check



I print the event info and it is giving me as shown



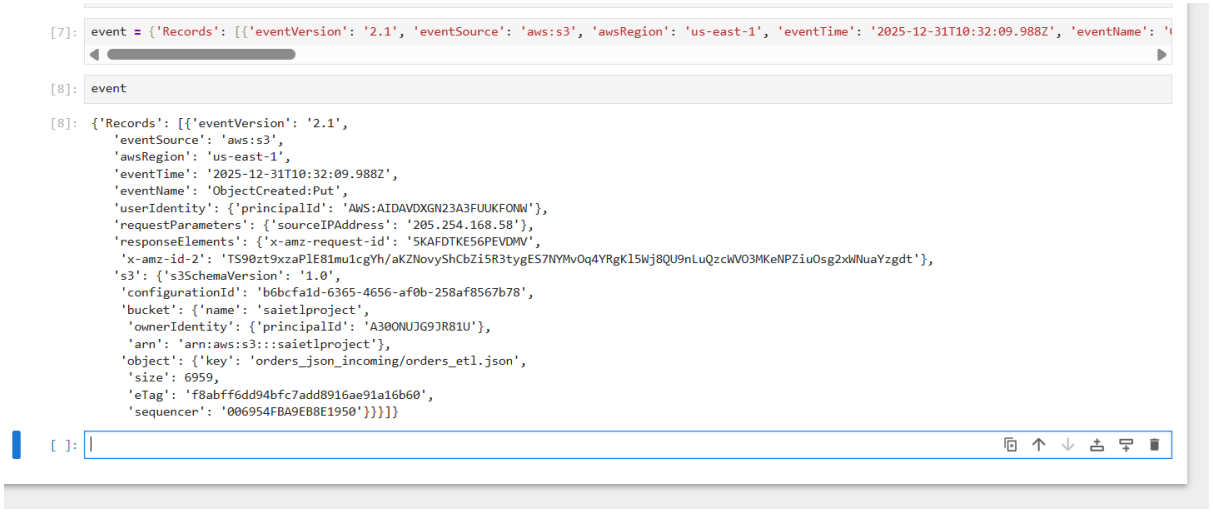
The event info is about bucket name,file etc

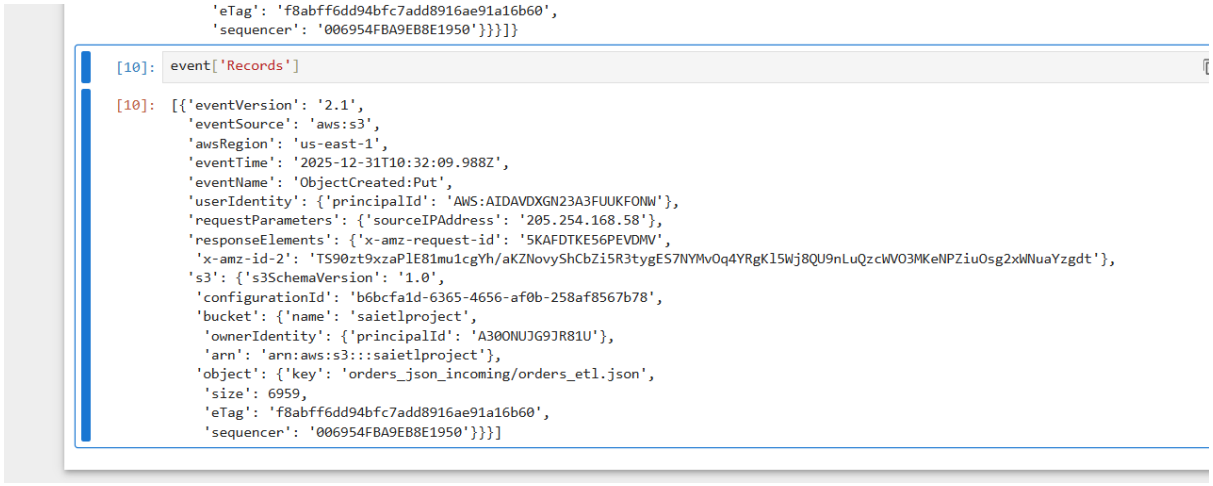
Now let us not hardcode our file name to flatten in the above lambda python code

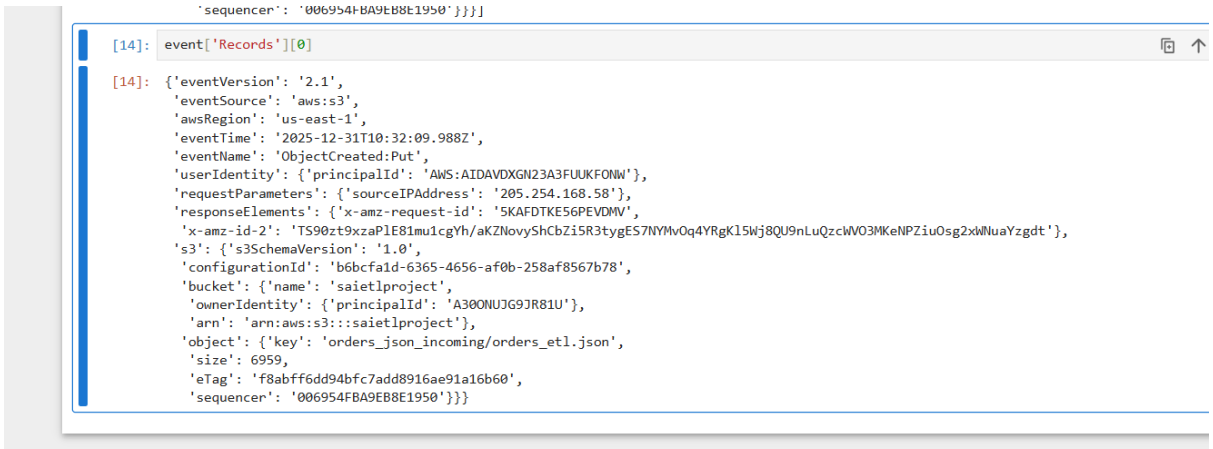
Becoz tomorrow due to incremental load the file name might be different

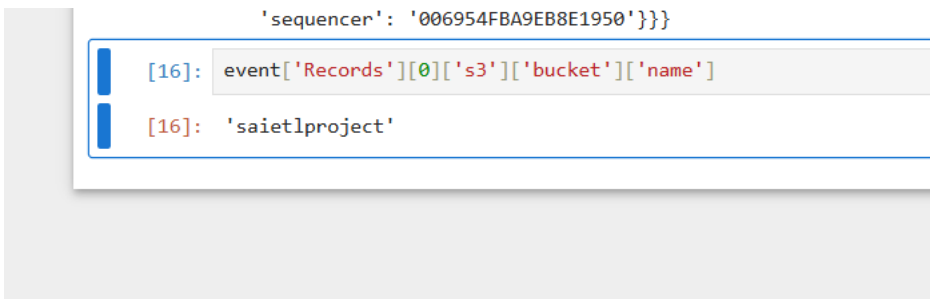
Lets copy that event information as per the above image

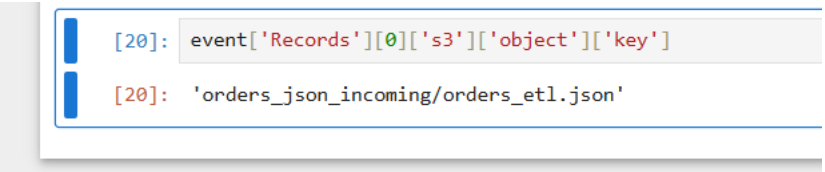
Now go to notebook as shown



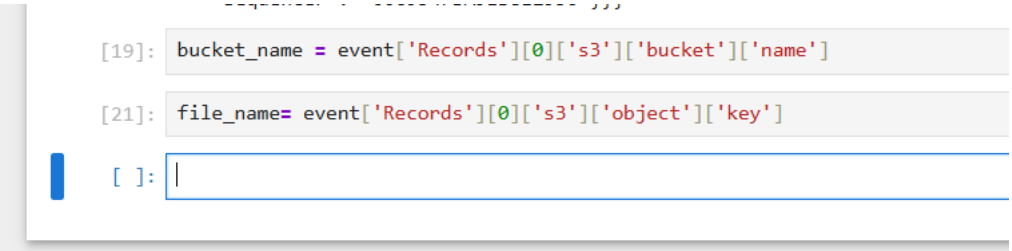




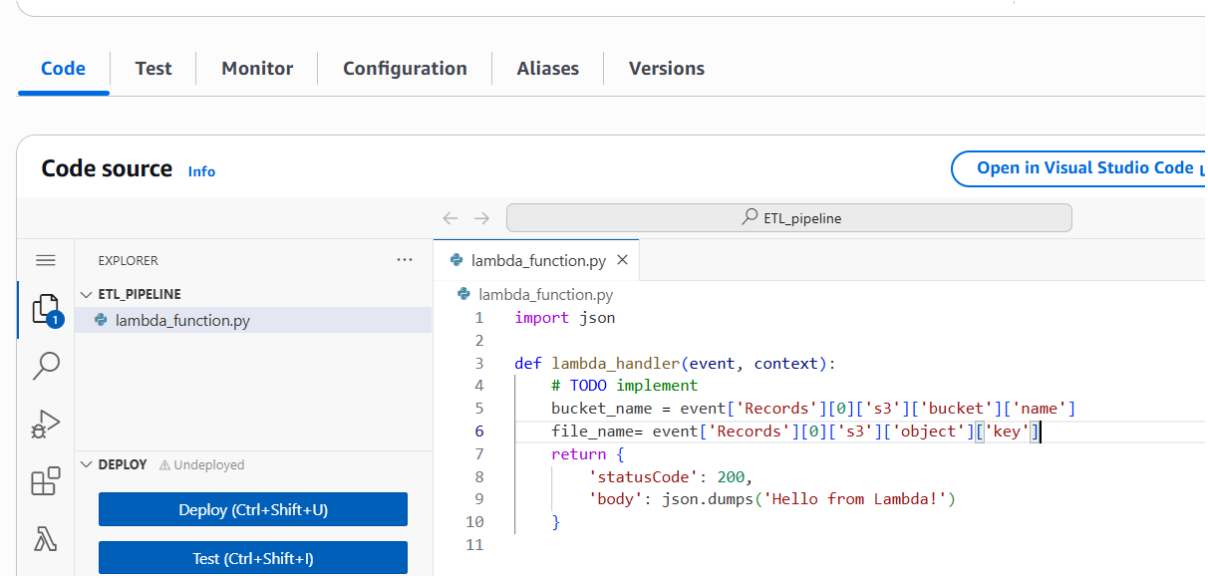




Lets store in variables



Now remove print and paste as follows

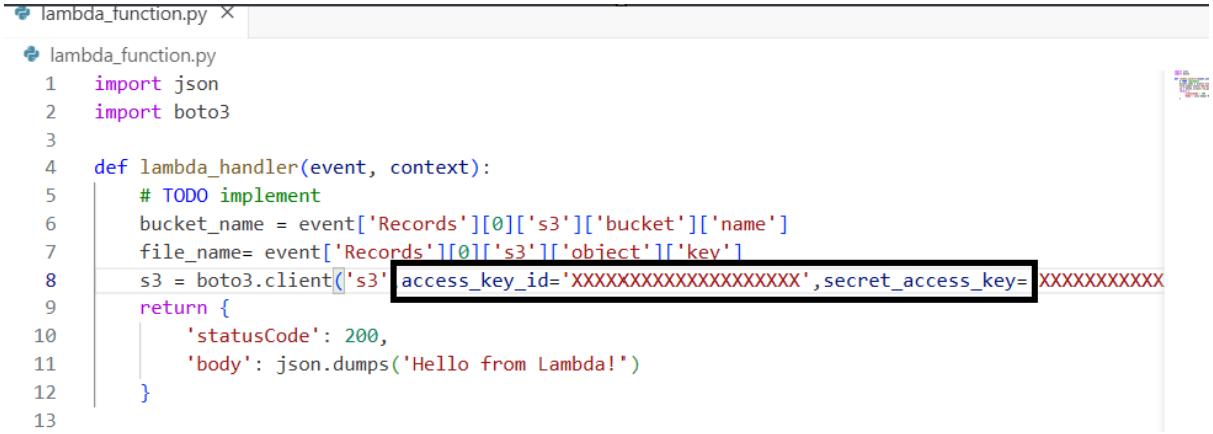


Now lets process/read the data

For that we require boto3 library which helps to read data from s3

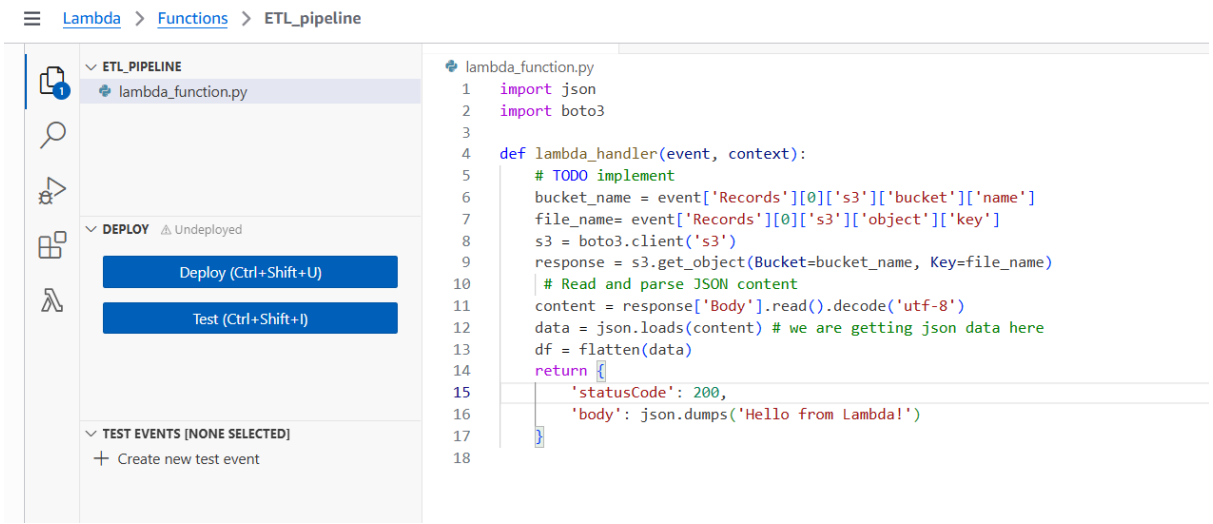
Here we don’t have to pass secret keys as whatever role that we are using to trigger the lamda has already access to s3 which we had set,

If u r running from ur local like Jupiter notebook trying to read data from s3 then these secret keys have to be mentioned as below



As we want to read data from s3 no need of secret keys

Why we decode is becoz that ‘response’ variable have the data in the form of binary form i.e bytes. So converting to string format while reading the body we use decode(‘utf-8’)



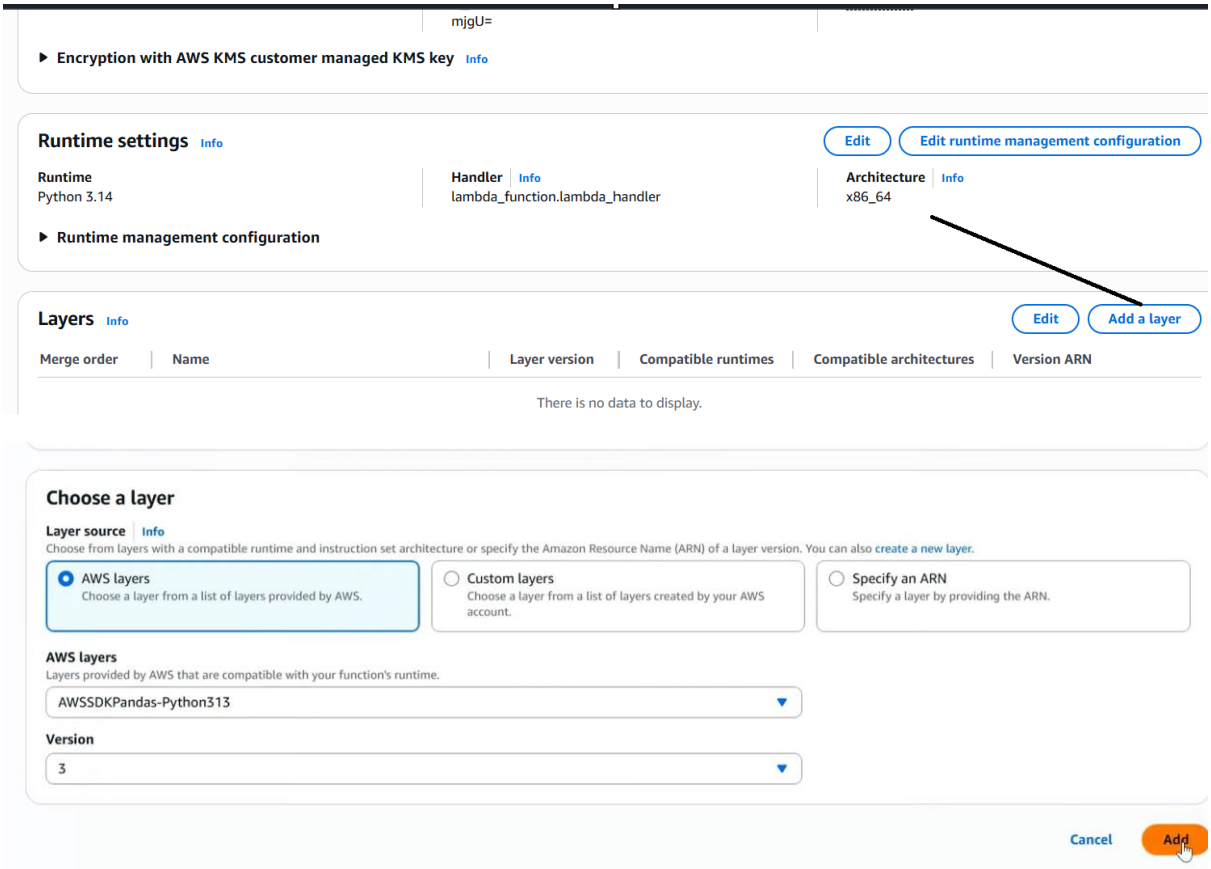
Now lets create flatten function, just paste the code we had written in notebook

lambda_function.py

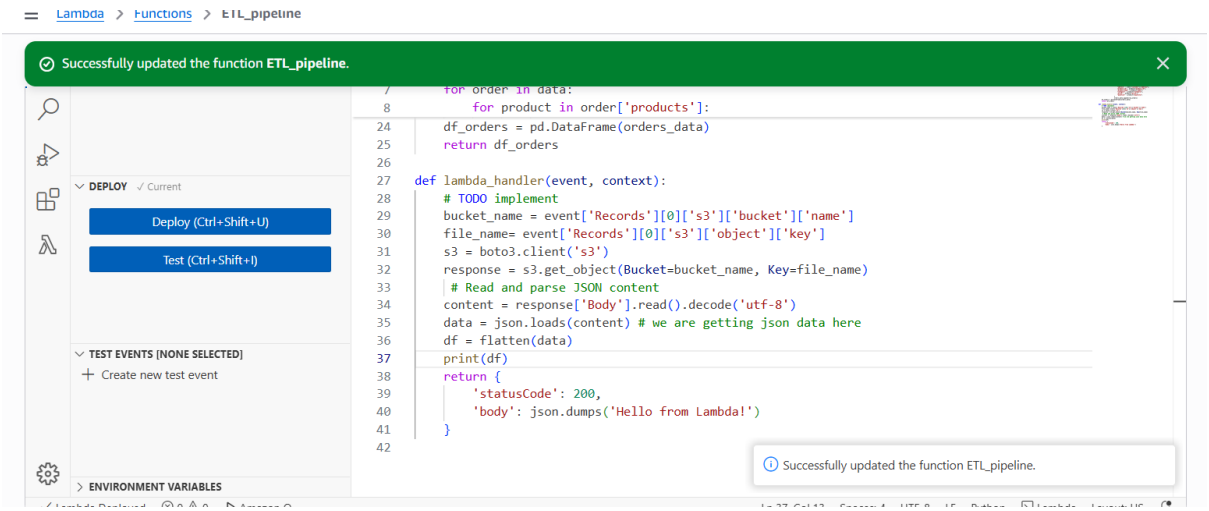
```python
import json
import boto3

def flatten(data):
    orders_data=[]
    for order in data:
        for product in order['products']:
                    row_orders = {
                        "order_id" : order["order_id"],
                        "order_date" : order["order_date"],
                        "total_amount" : order["total_amount"],
                        "customer_id" : order["customer"]["customer_id"],
                        "customer_name" : order["customer"]["name"],
                        "email" : order["customer"]["email"],
                        "address" : order["customer"]["address"],
                        "product_id" : product["product_id"],
                        "product_name" : product["name"],
                        "category" : product["category"],
                        "price" : product["price"],
                        "quantity" : product["quantity"]
                    }
                    orders_data.append(row_orders)
    df_orders = pd.DataFrame(orders_data)
    return df_orders
```

Pls import pandas as well but pandas module is not available in lambda

When u scroll down in the same page u see an option add layer



To save it we need to deploy every time

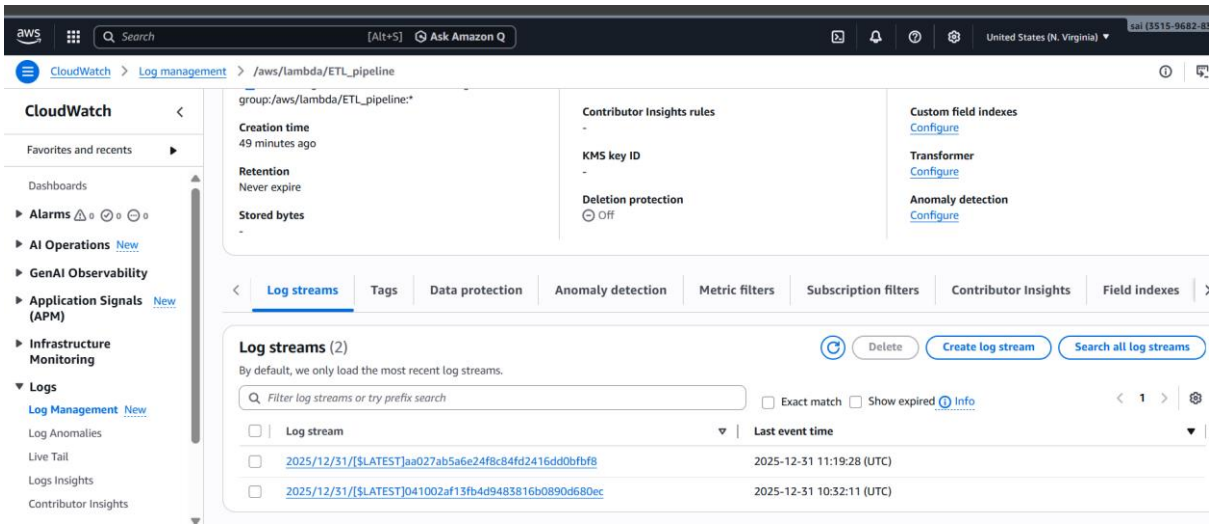


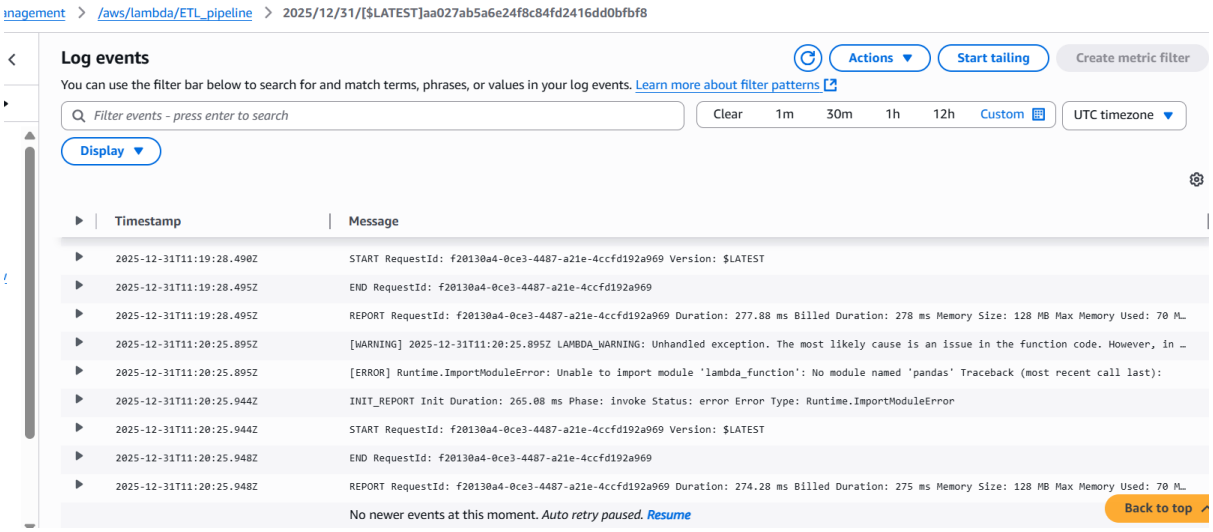
We just printed to df to check

To run it, we need to load the file in s3 one more time

Lets go to cloud watch and check

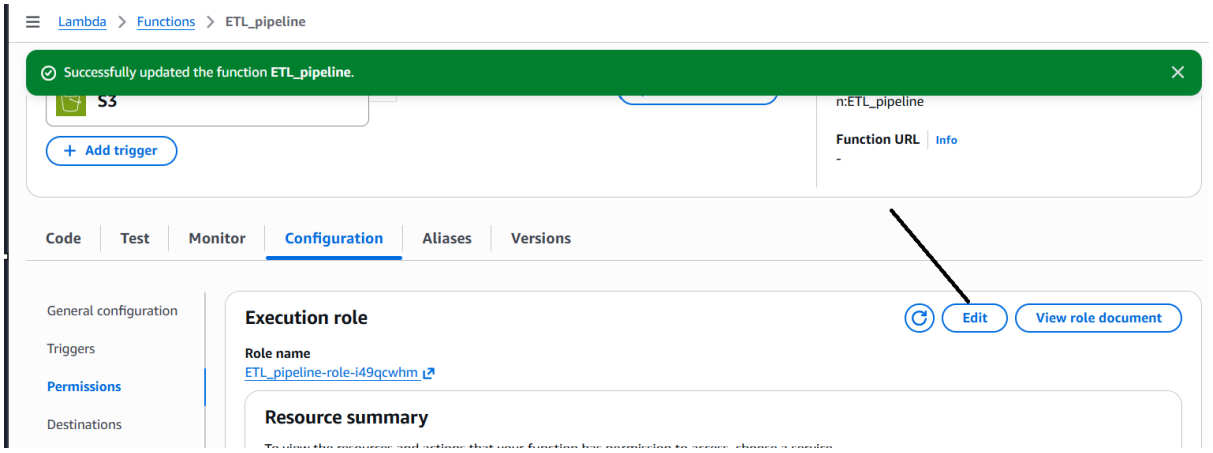
One more entry as shown,click on that



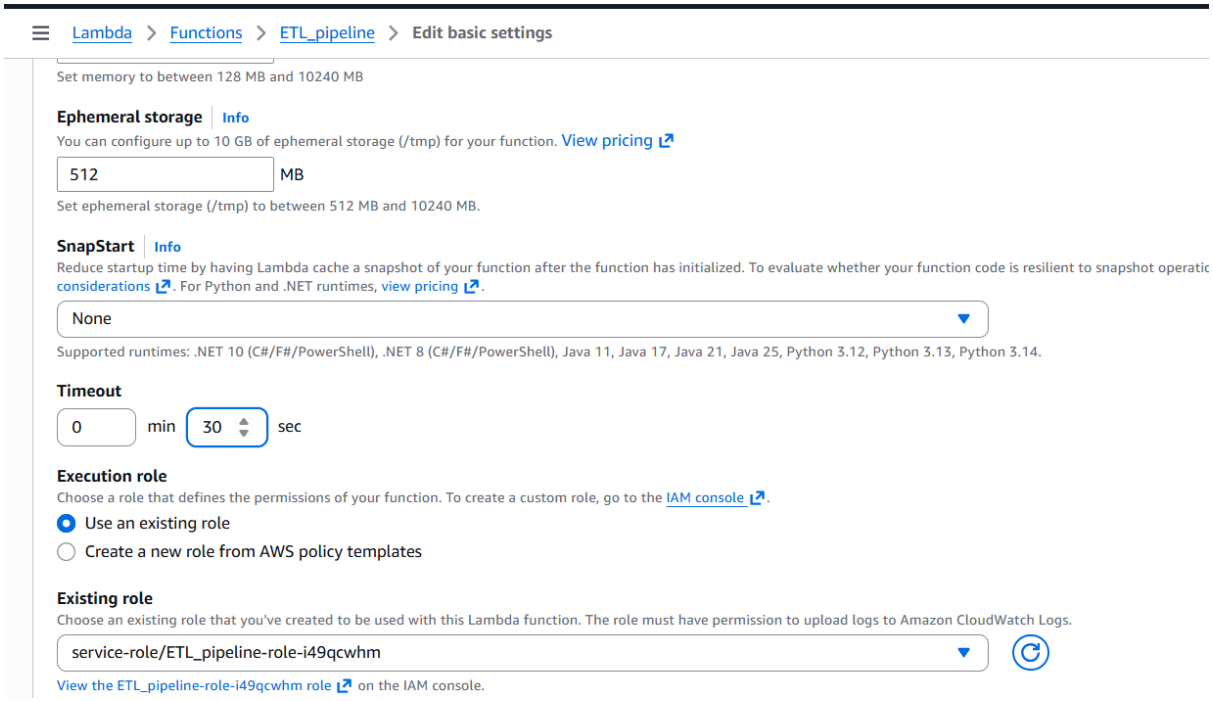


I don’t see any change…so to fix this issue we need to change timeout

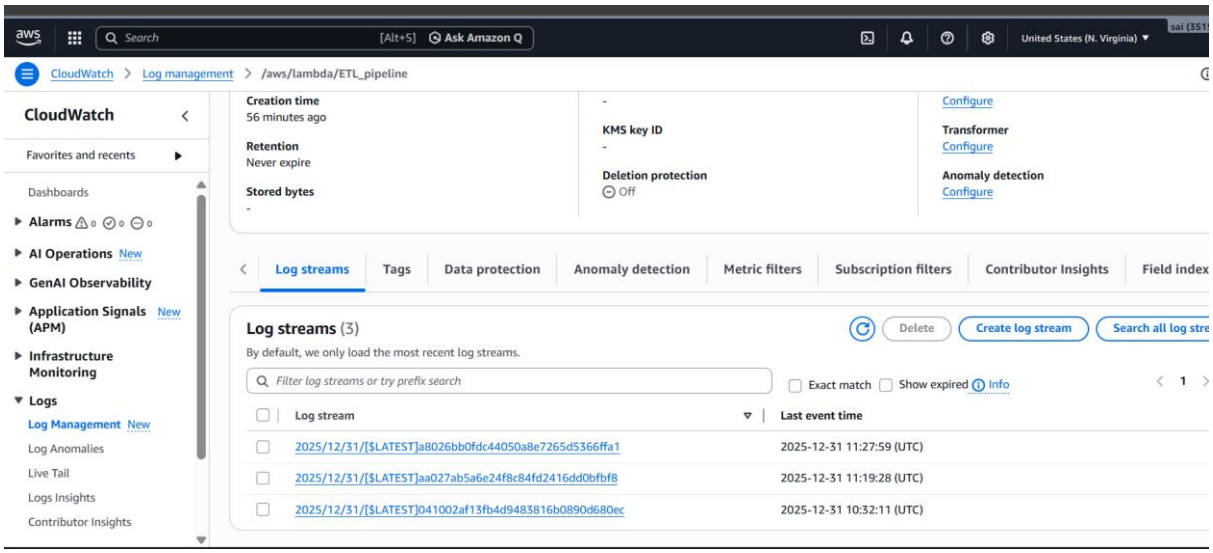
Ur lambda will timeout under 3 seconds, hence change by going to configuration



Increased to 30 sec

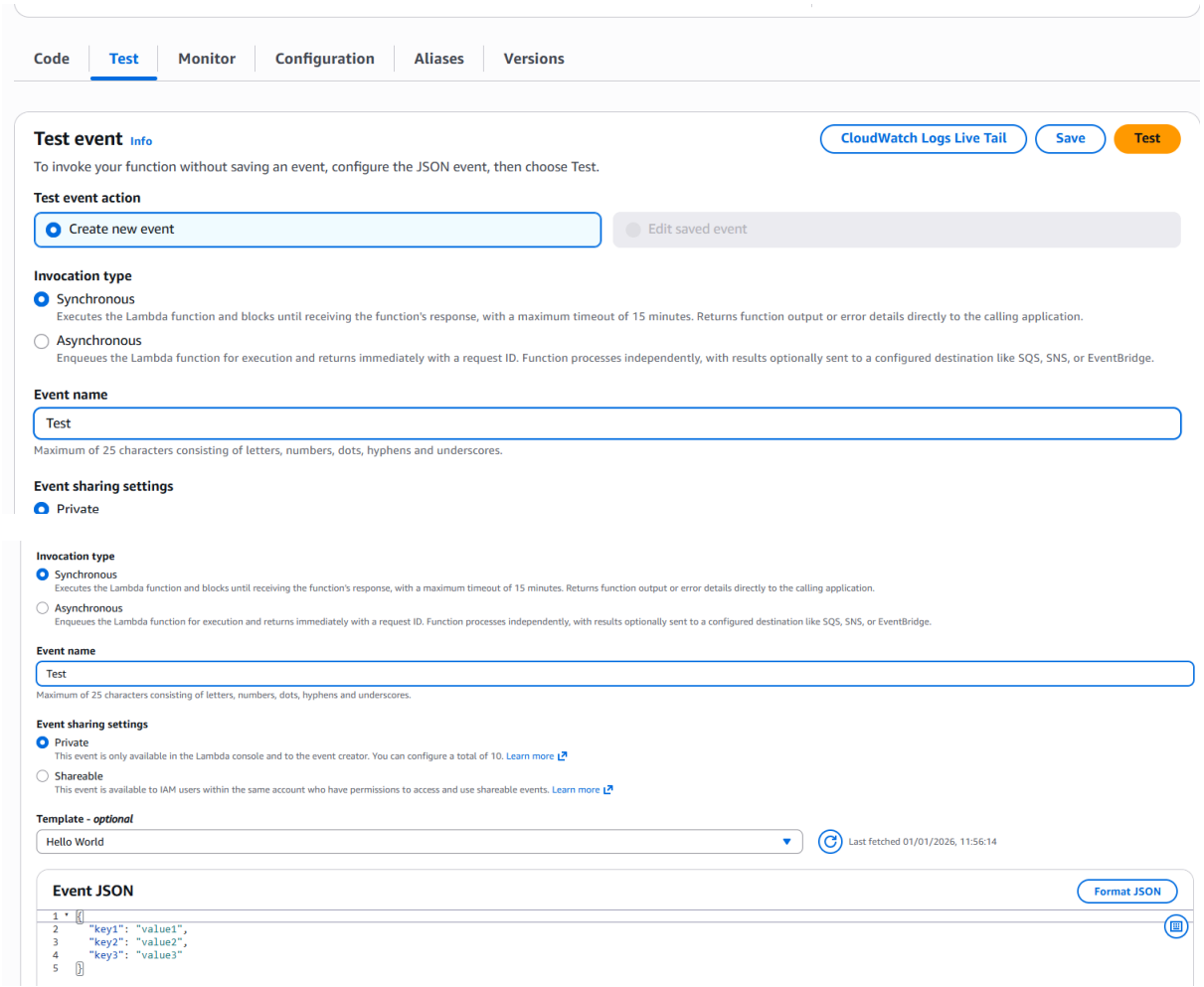


Lets upload file once again and check



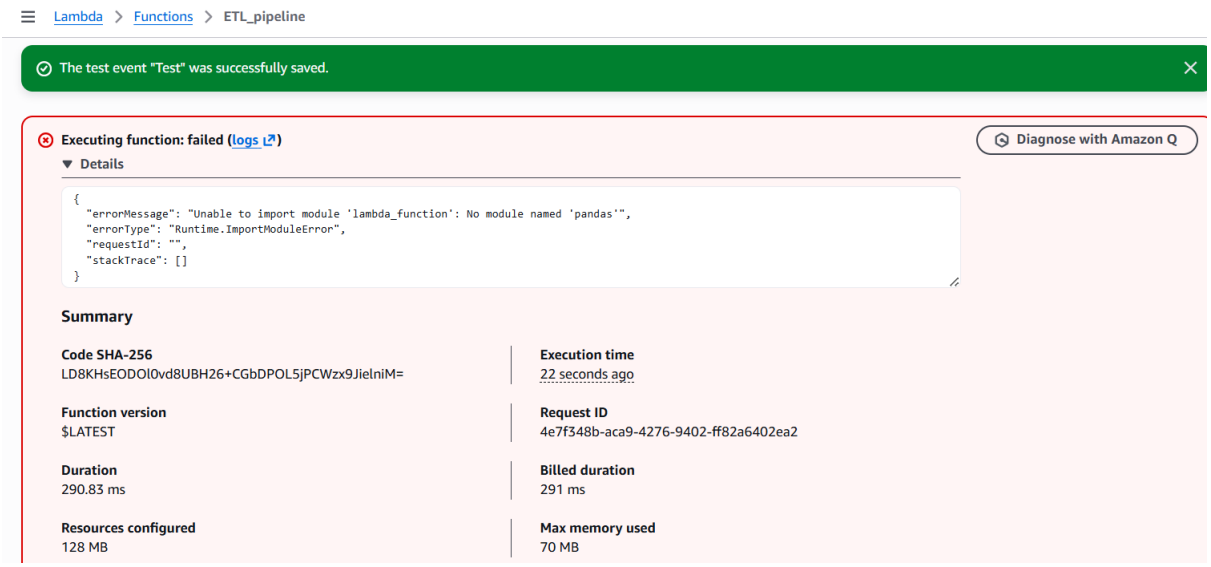
U need to see ur dataframe

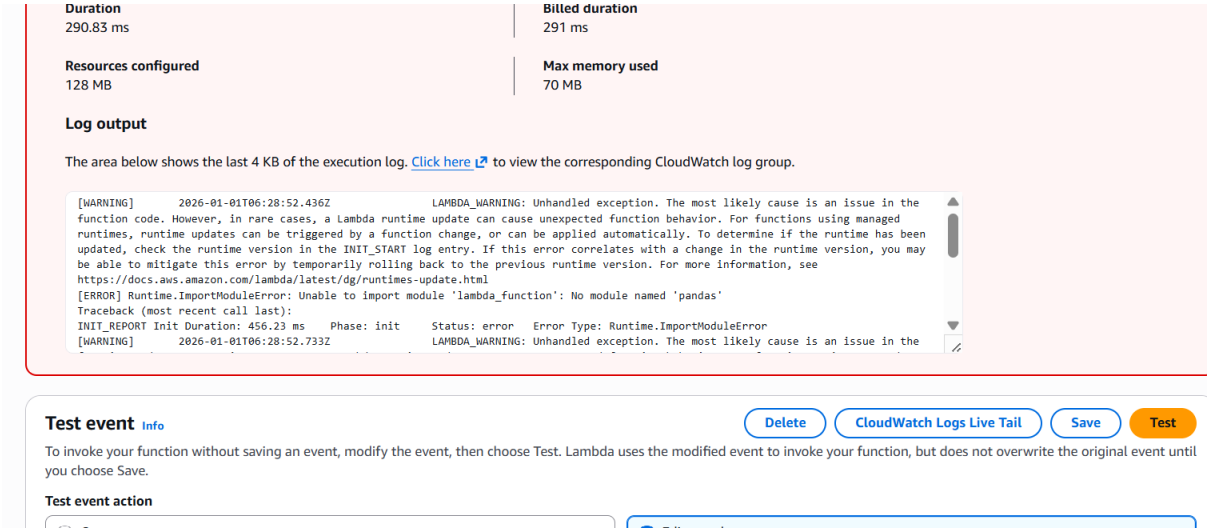
But testing can be done in other way as shown:



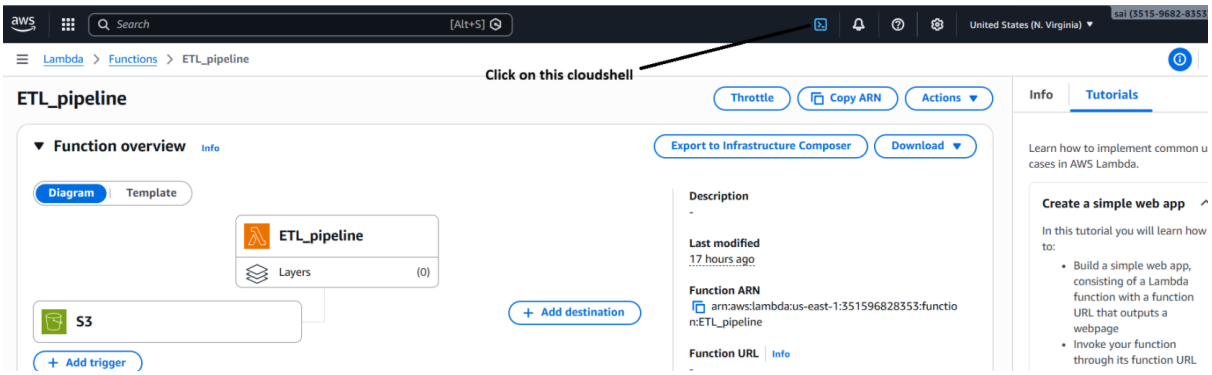
Click on save on top

When u click on test, u can see as follows

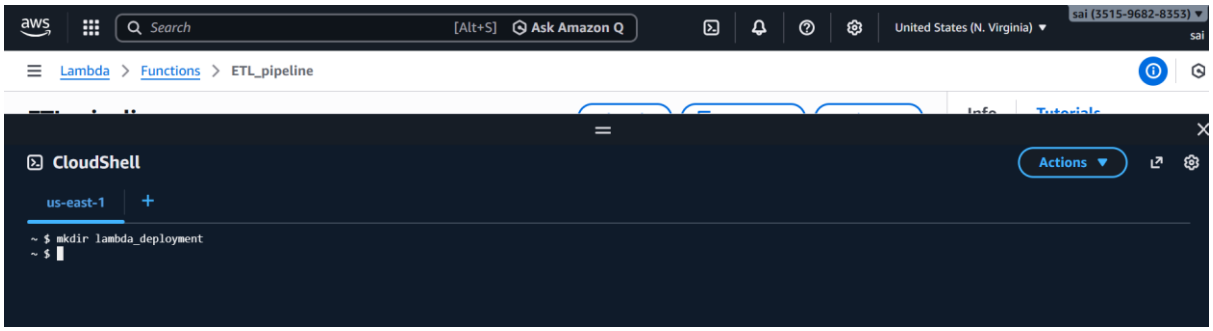




Now lets integrate the python packages with aws console



Here u can execute all the required details or commands here

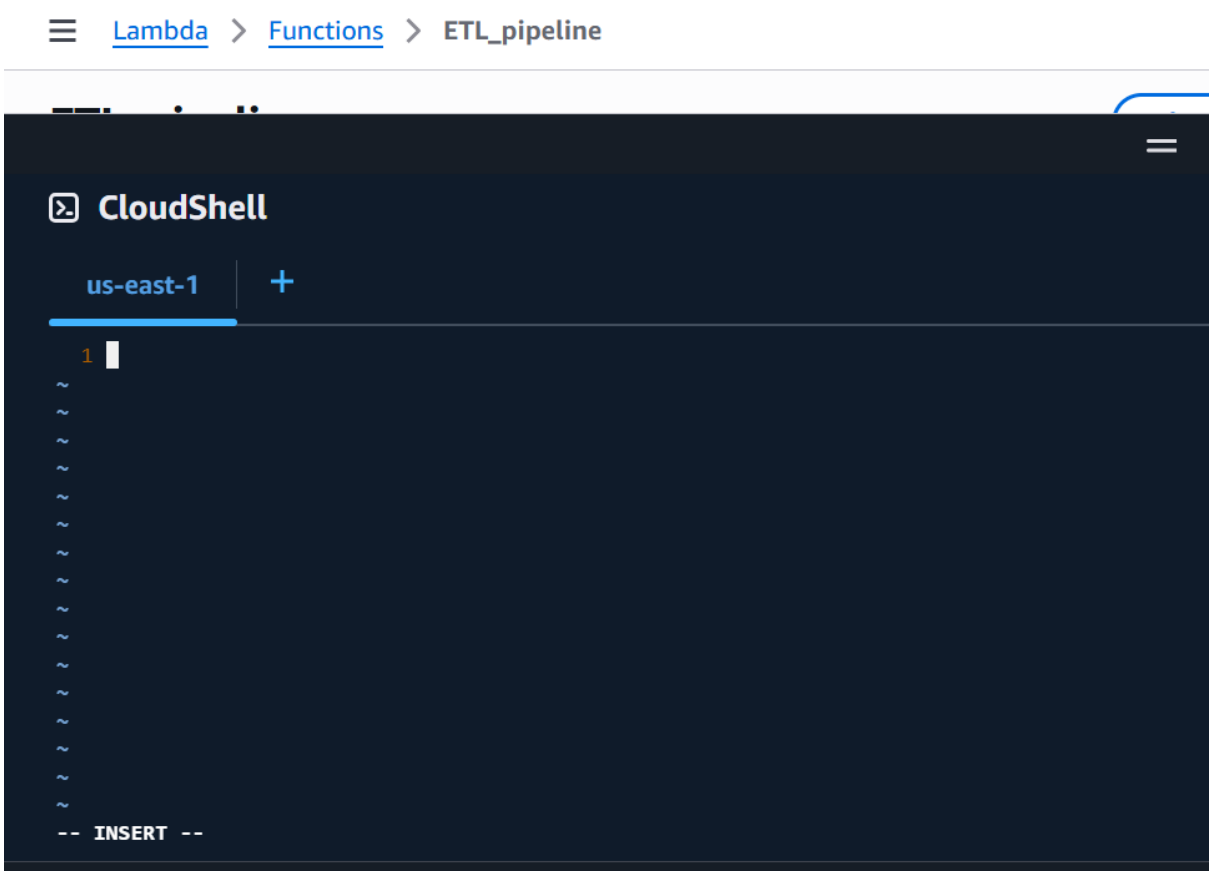


mkdir lambda_deployment

cd lambda_deployment/

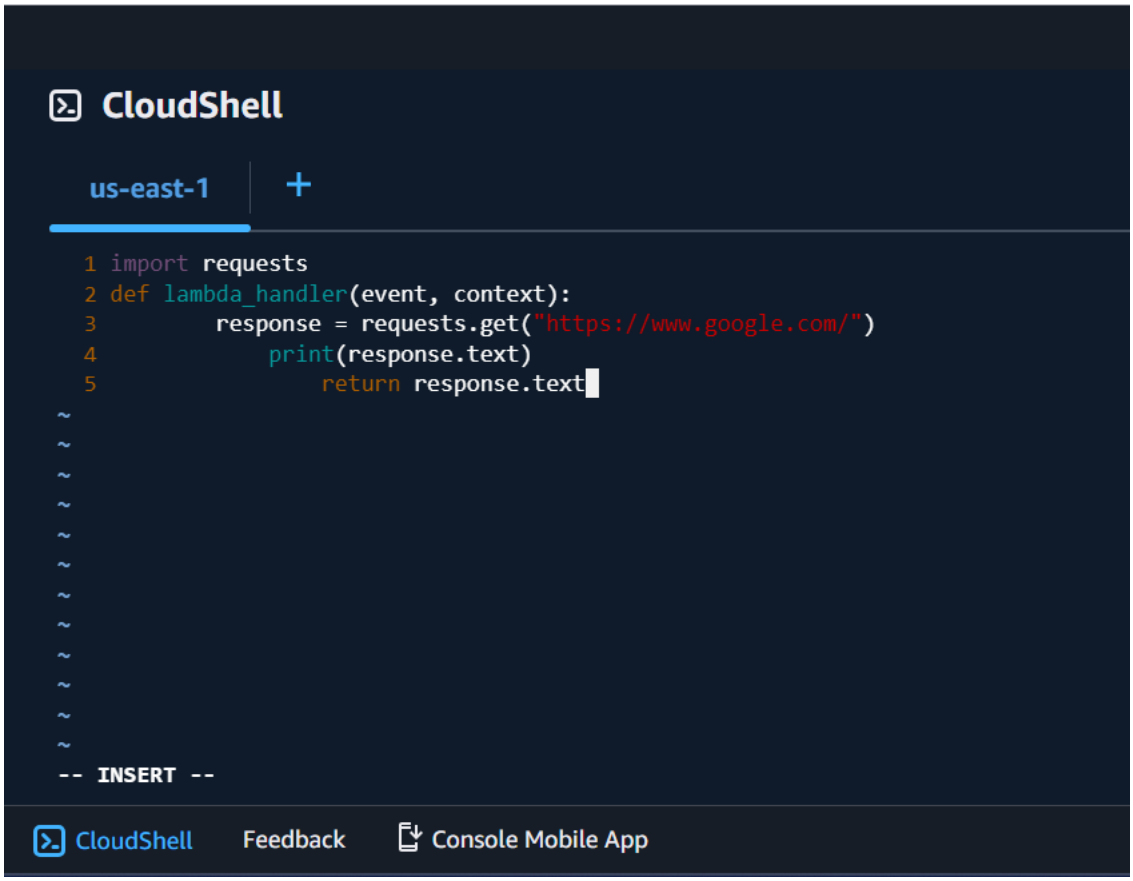
vi lambda_function.py

After this type “I” to be in insert mode as shown



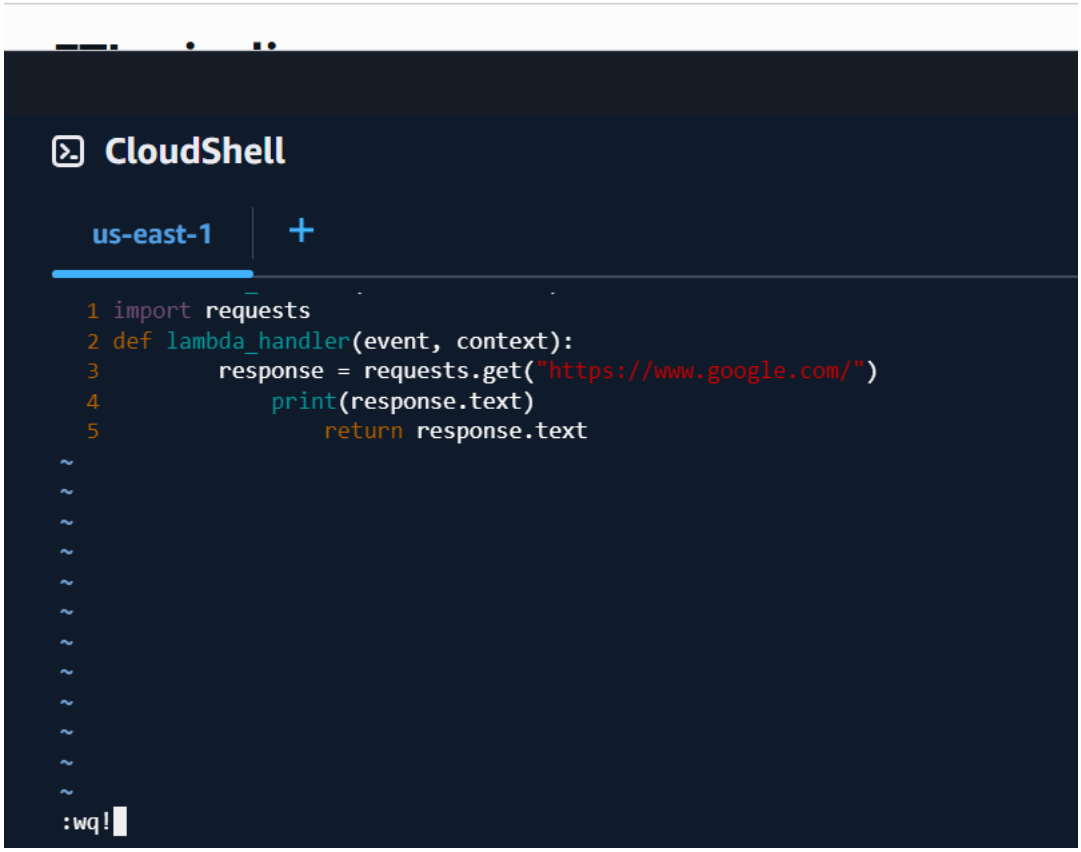
Copy the code and paste as shown:

```
import requests
def lambda_handler(event, context):
    response = requests.get("https://www.google.com/")
    print(response.text)
    return response.text
```

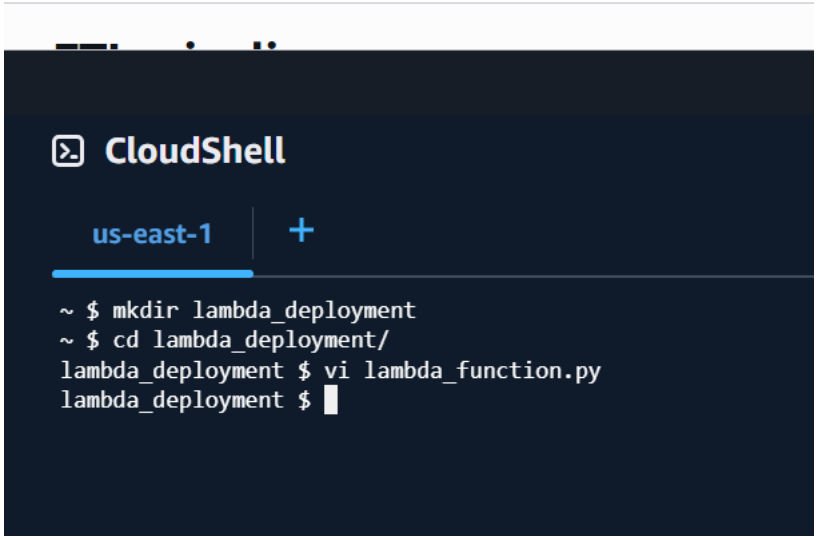


Now click esc

Then type “:wq!” wq stands for write and quick



Click enter

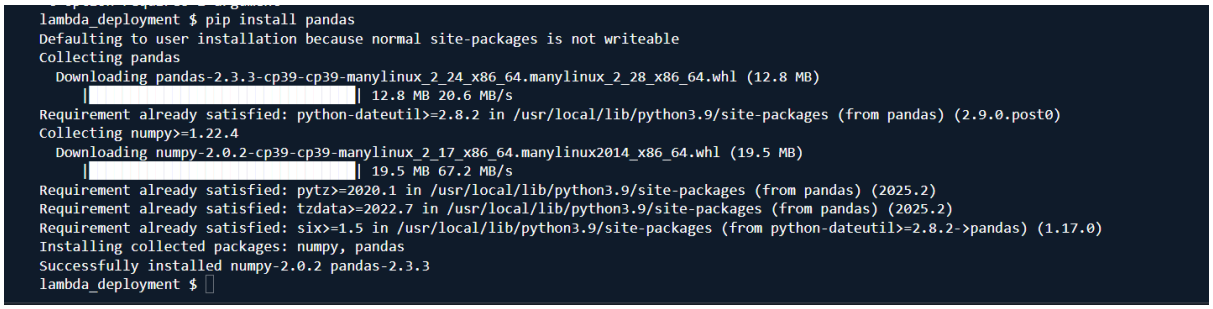


Now u have lambda_function.py in ur cloud shell

Now install the dependent packages

Type “pip install requests -t . ”

I also run “pip install pandas”



Whatever we download we need to zip it, so use this code

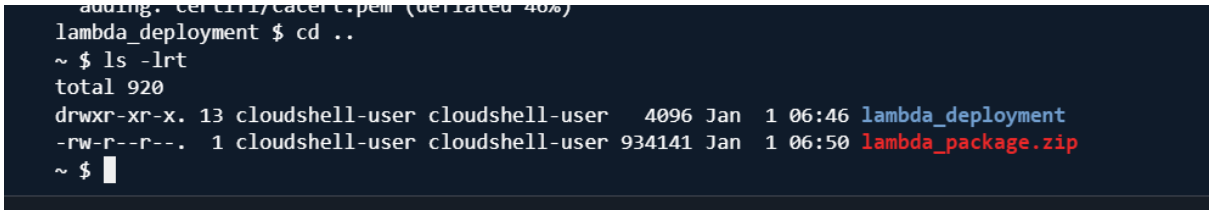
“zip -r ../lambda_package.zip . ”

Lambda_package is the name of the zip file we are creating

Go to one step backward as shown

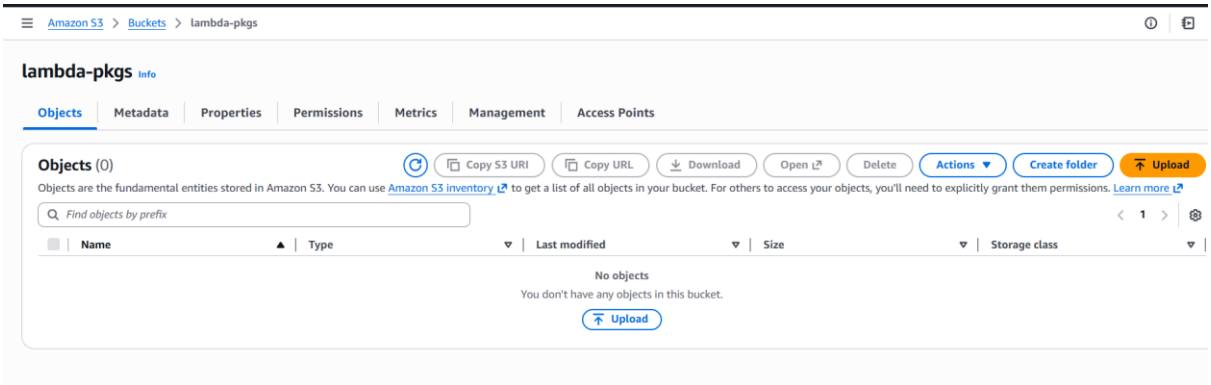
cd ..

ls -lrt



Now let us import this zip in lamda function using s3 bucket

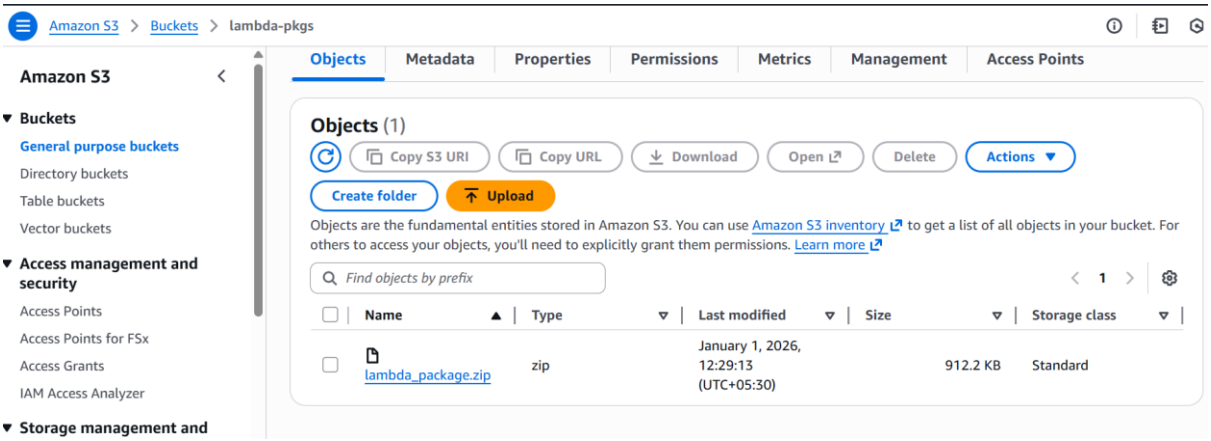
Lets create a bucket for this



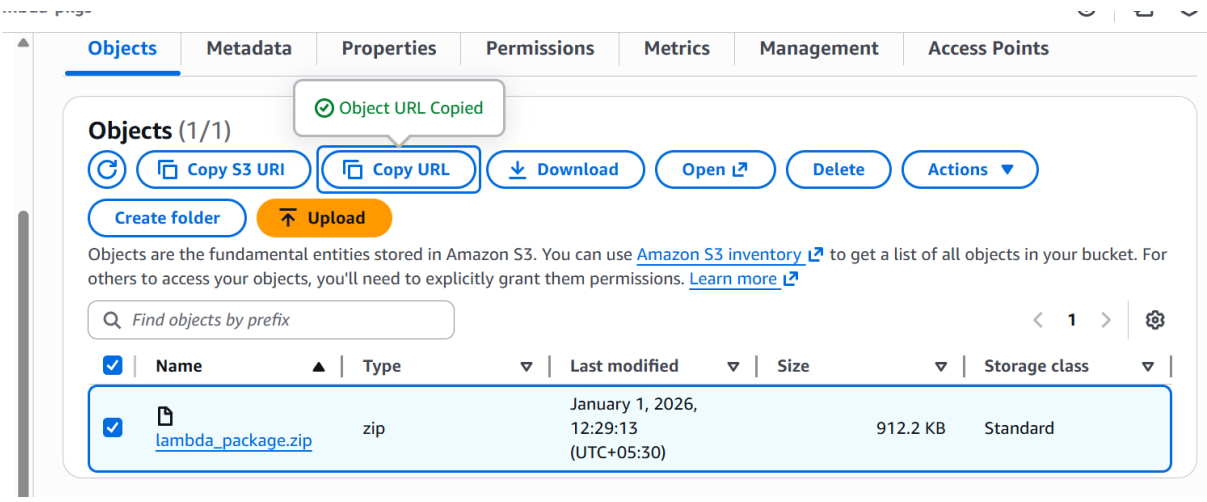
Now type as shown “aws s3 cp lambda_package.zip s3://{bucket_name}”

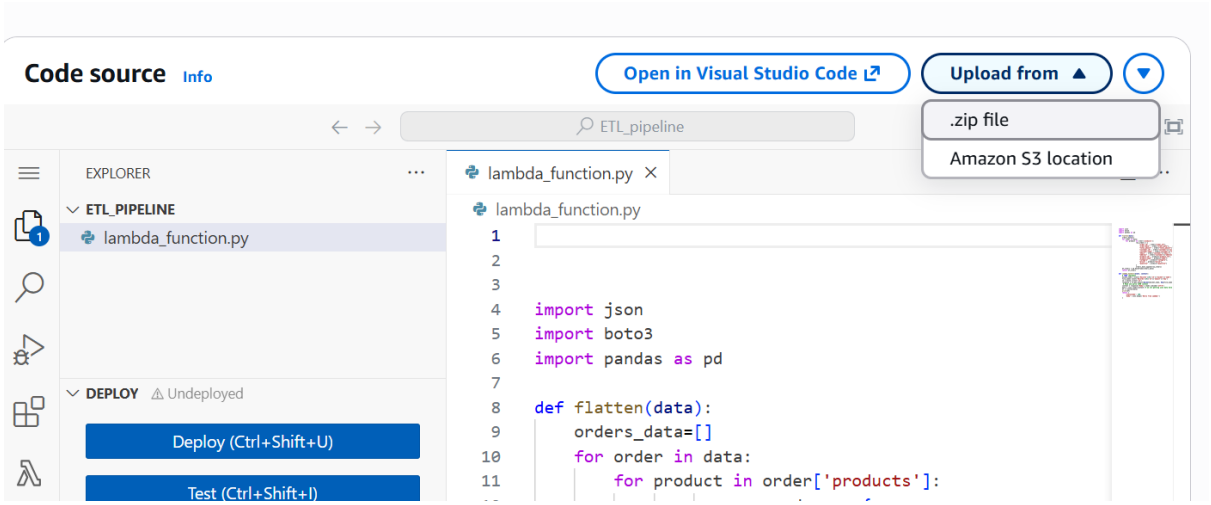
```
~ $ ls -lrt
total 920
drwxr-xr-x. 13 cloudshell-user cloudshell-user   4096 Jan  1 06:46 lambda_deployment
-rw-r--r--.  1 cloudshell-user cloudshell-user 934141 Jan  1 06:50 lambda_package.zip
~ $ aws s3 cp lambda_package.zip s3://lambda-pkgs
upload: ./lambda_package.zip to s3://lambda-pkgs/lambda_package.zip
~ $
```

Upload has been completed as shown above

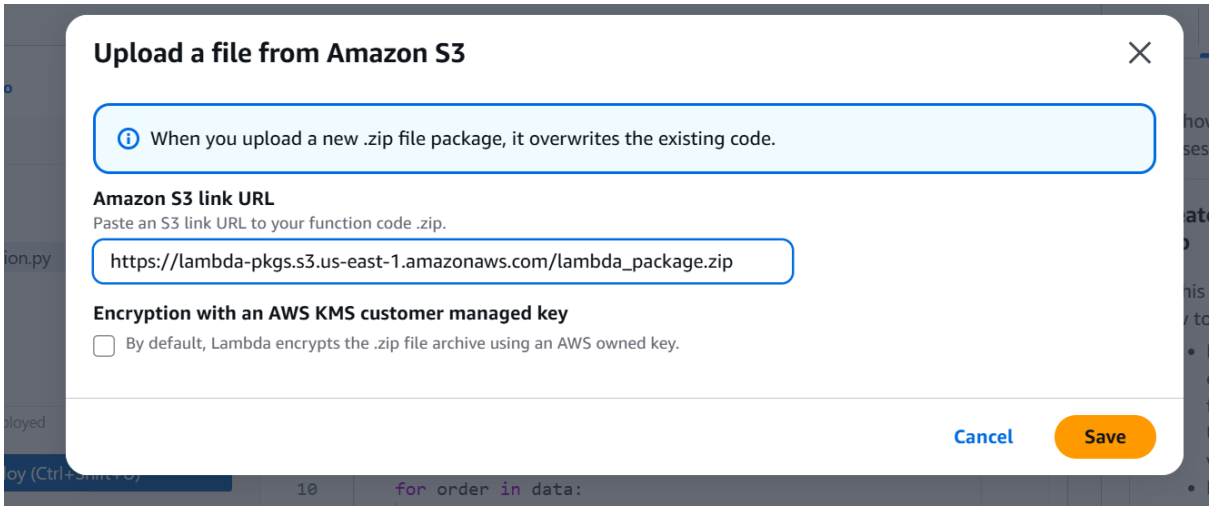


Copy url





Choose amazon s3 location



Save

Now u can test

We are able to read the data now

Now, lets convert this dataframe into parquet

```
40
41      # in-memory binary buffer to hold data like a file, but without writing to disk.
42      parquet_buffer = io.BytesIO()
43      df.to_parquet(parquet_buffer, index=False, engine='pyarrow')
44
45      now = datetime.now()
46      timestamp = now.strftime("%Y%m%d_%H%M%S")
47
48      #key_staging=f'orders_csv/orders_ETL_{timestamp}.csv'
49      key_staging=f'orders_parquet/orders_ETL_{timestamp}.parquet'
50
51      #s3.put_object(Bucket=bucket_name, Key=key_staging, Body=csv_buffer.getvalue())
52      s3.put_object(Bucket=bucket_name, Key=key_staging, Body=parquet_buffer.getvalue())
```

42 - we are converting to inmemory bytes i.e holds the data in the form of bytes. Created a buffer

43 – write by dataframe to this buffer (note: import io library), now parquet_buffer holds my data in byte format

49 – I can't have a fixed name as if multiple files are coming I cant say all file names as order_etl.so it overwrites all the time if u have incremental load. So adding a timestamp to my file so that doesn't overwrite

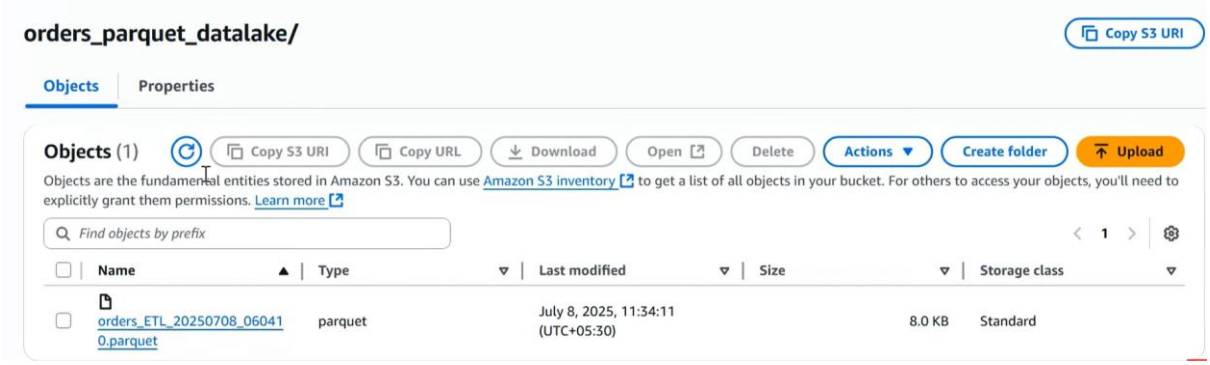
52 – key_staging is where u want to get the files into

In our case it is as shown

Getvalue() gives me the value in byte format

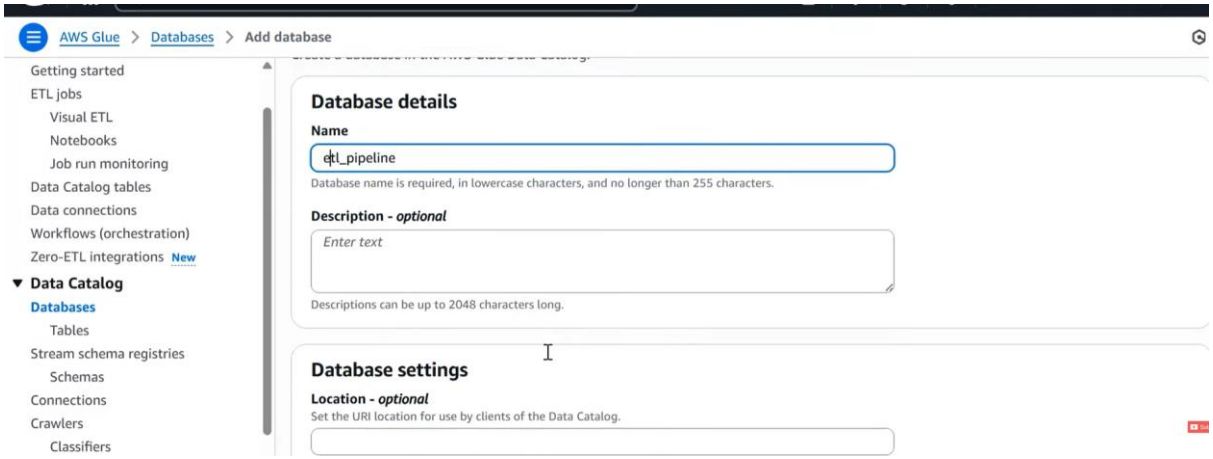
Lets deploy and test like upload ur file in s3 and check

U can see ur parquet file now

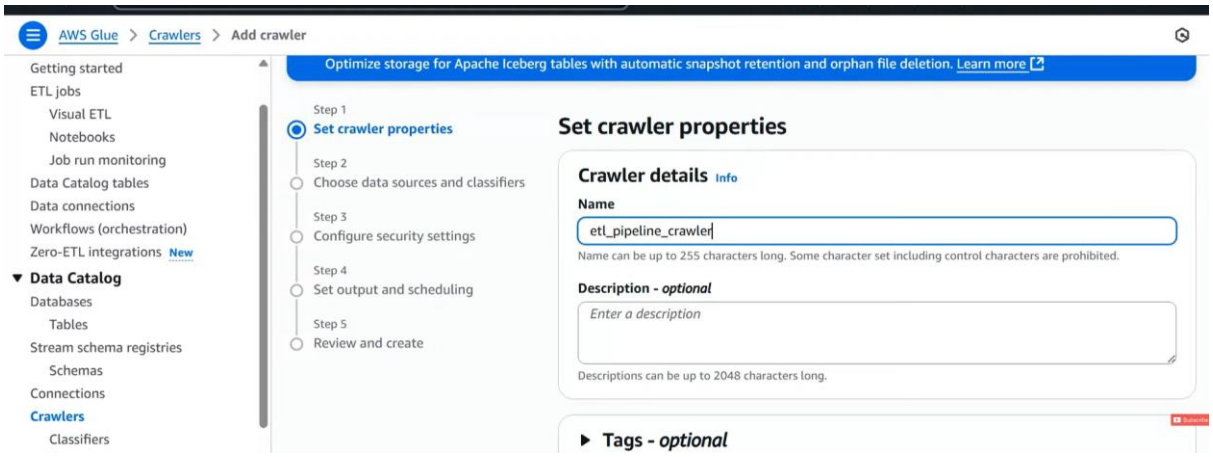


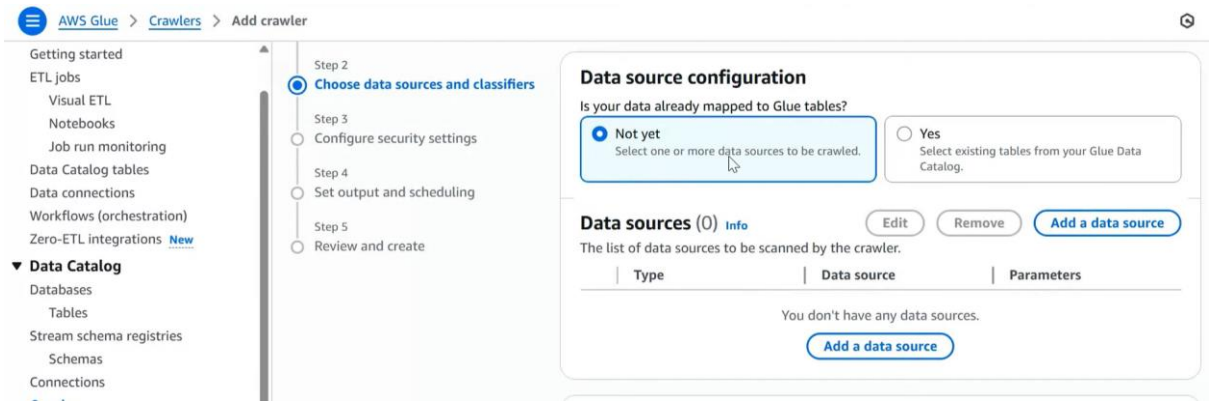
Now lets create a data catalog for it

Create a database

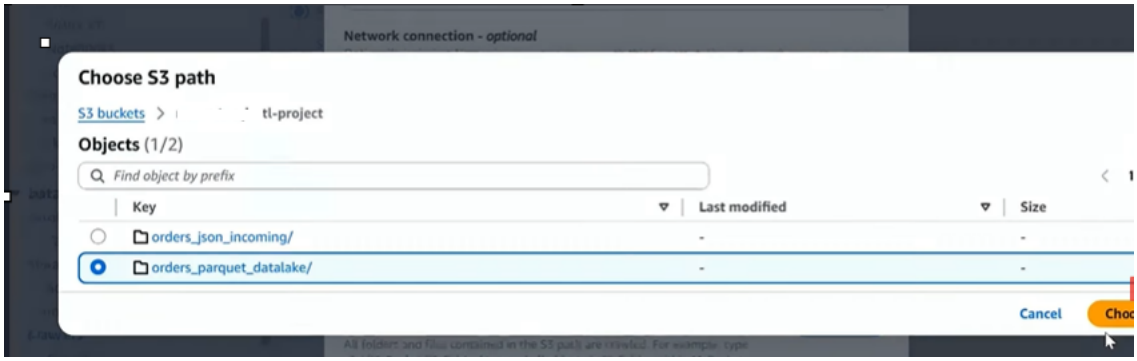


Create a crawler

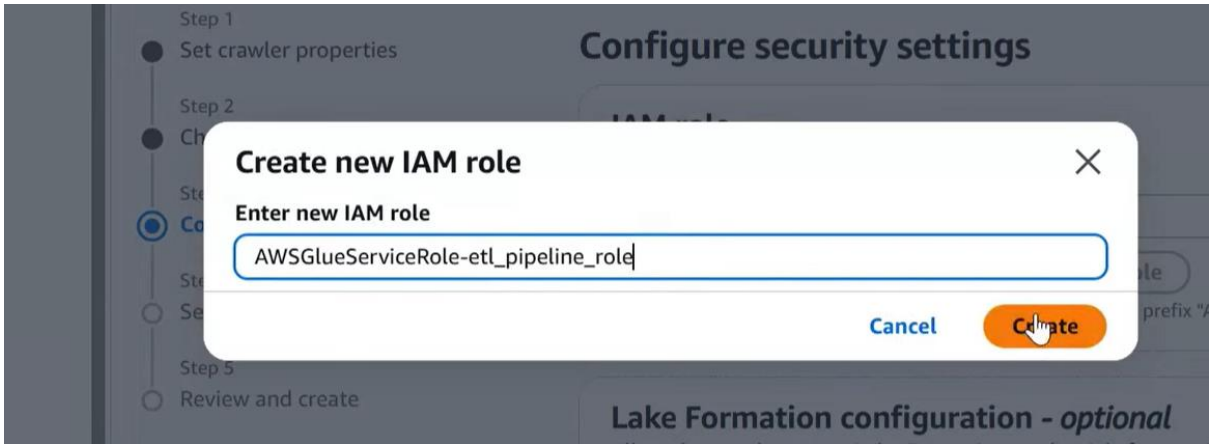


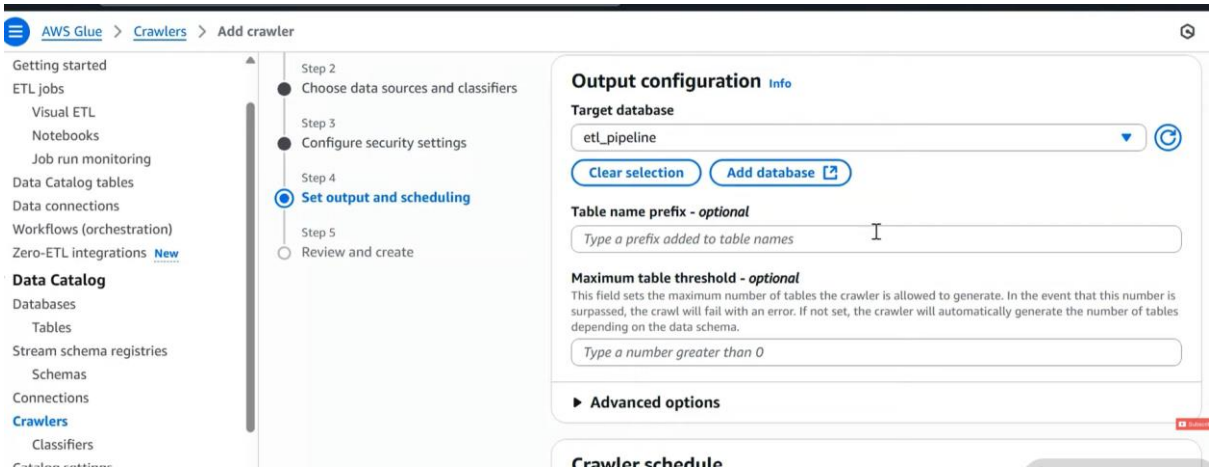


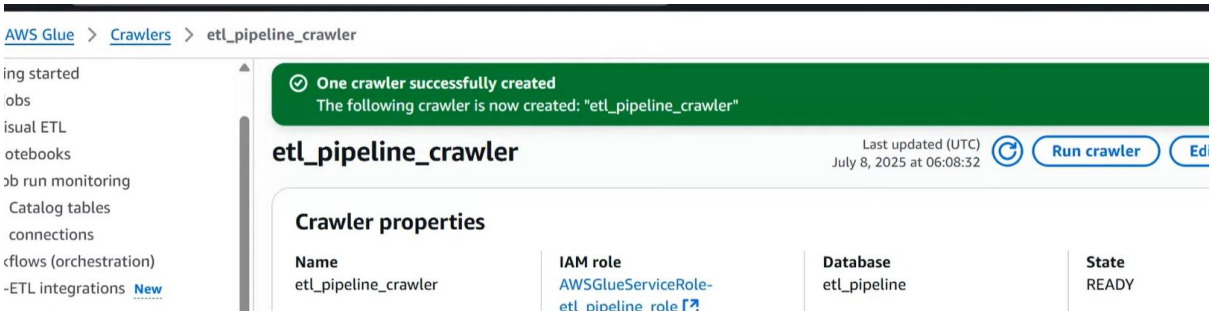
Add datasource and select ur data source which is s3 location



Create a new role else u can use existing one but need to know for which things u have access







Lets run the crawler

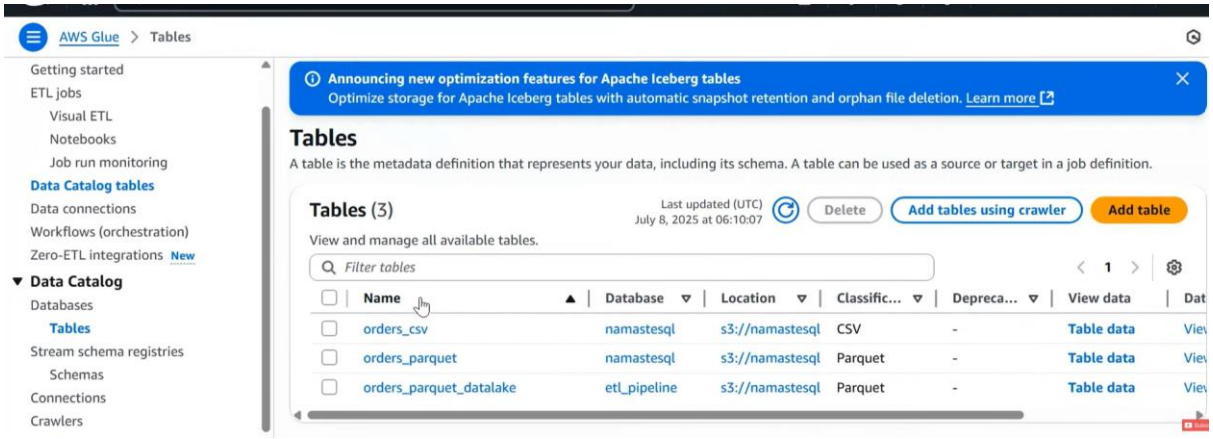
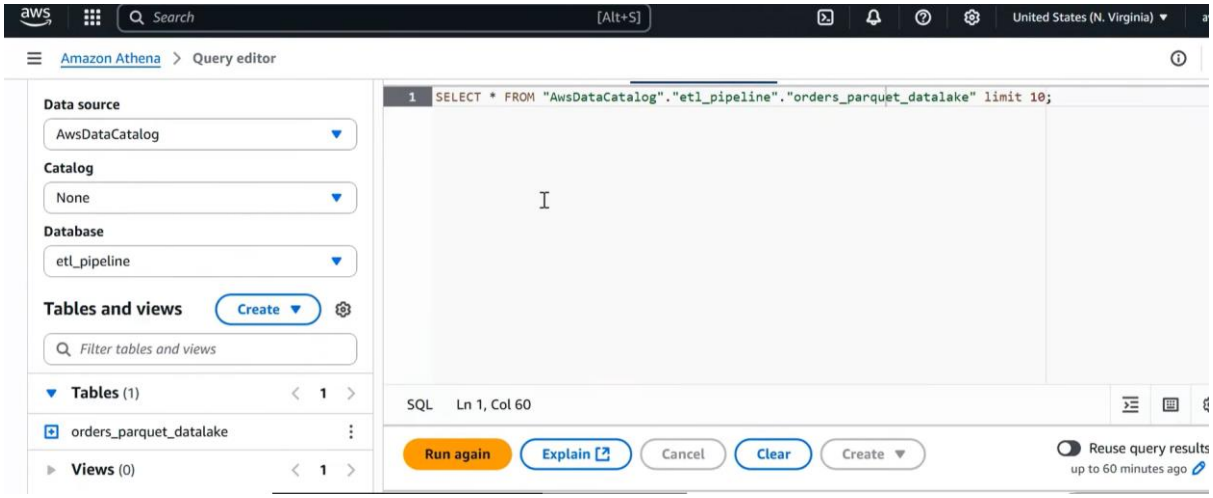
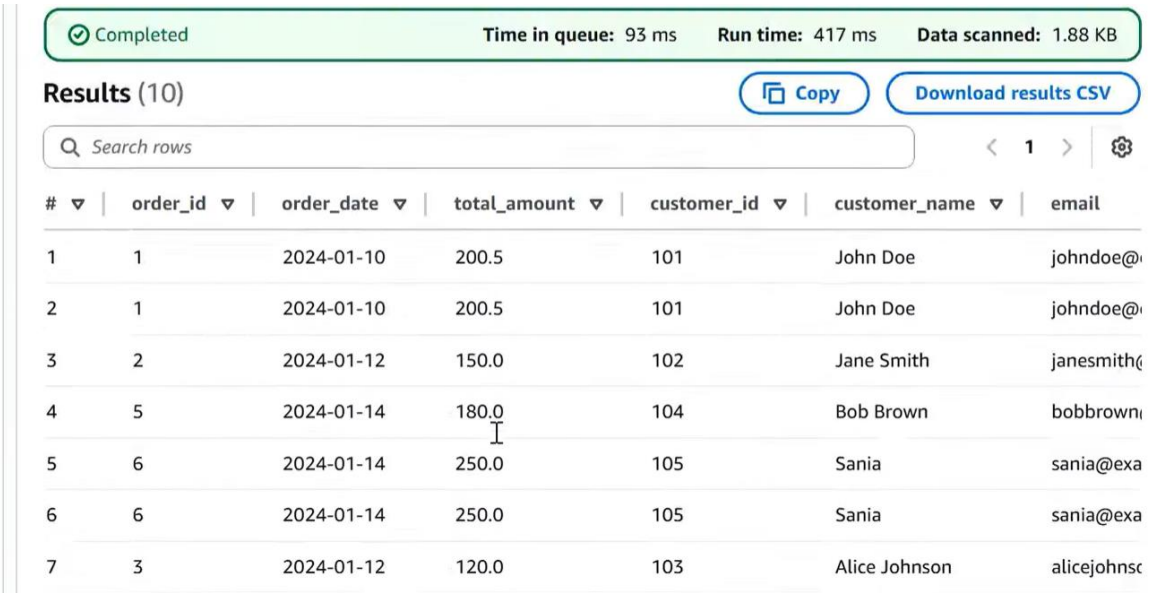


Table is created

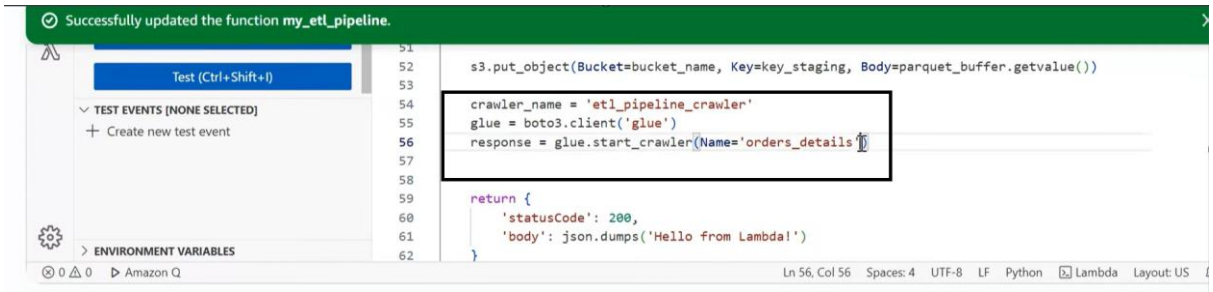
Note that it is not storing the data, data is in s3 only

Go to athen and query it

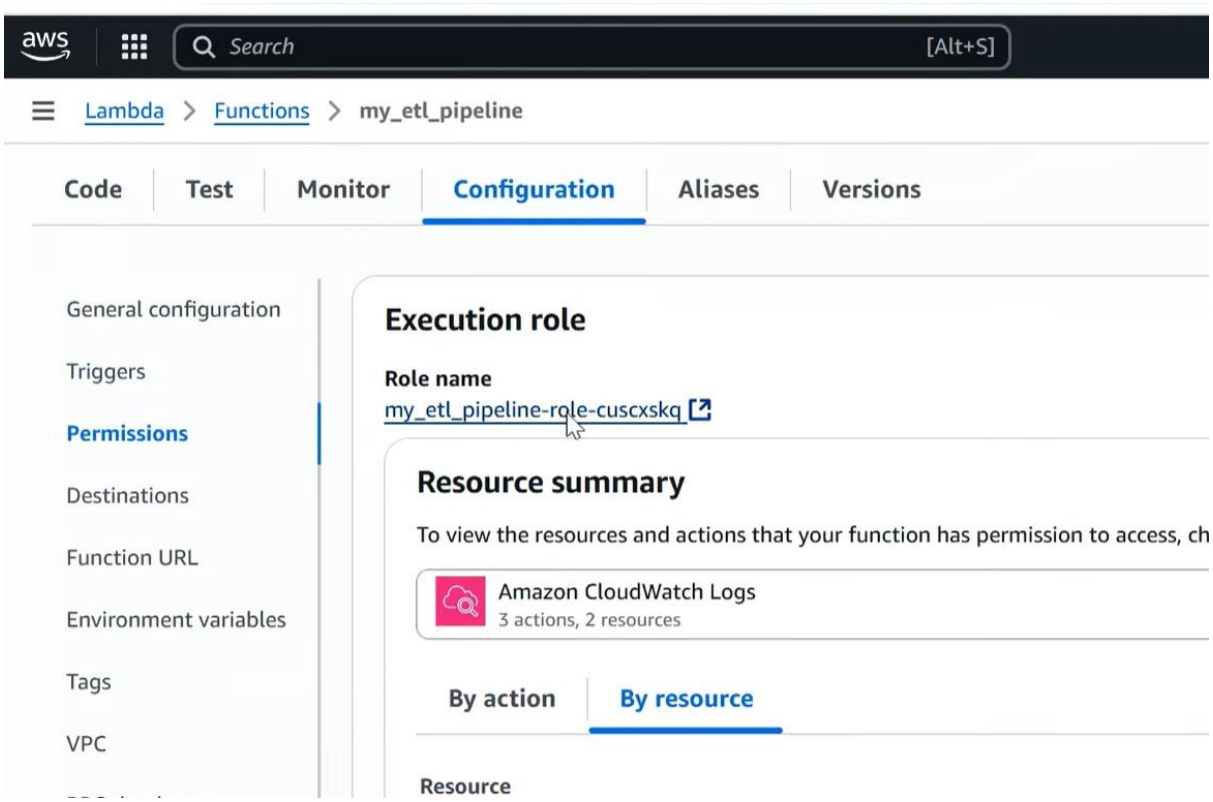




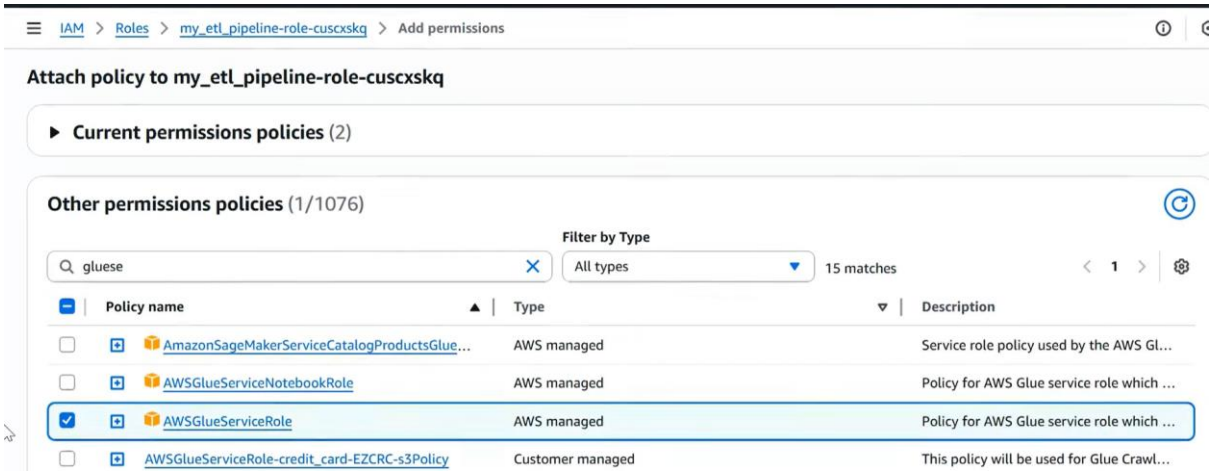
How tomake this crawler automated as once as per incremental load in incoming files, I automated them to convert to parquet files now I need to make this crawler run when ever I see my parquet files



This (lambda) IAM role doesn't have permission to run the crawler

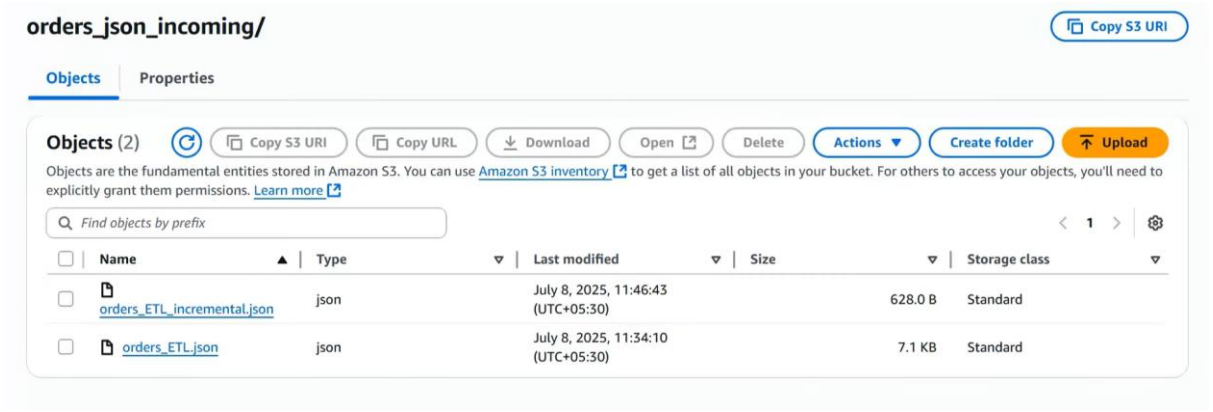


Click on the role name and add a new permission

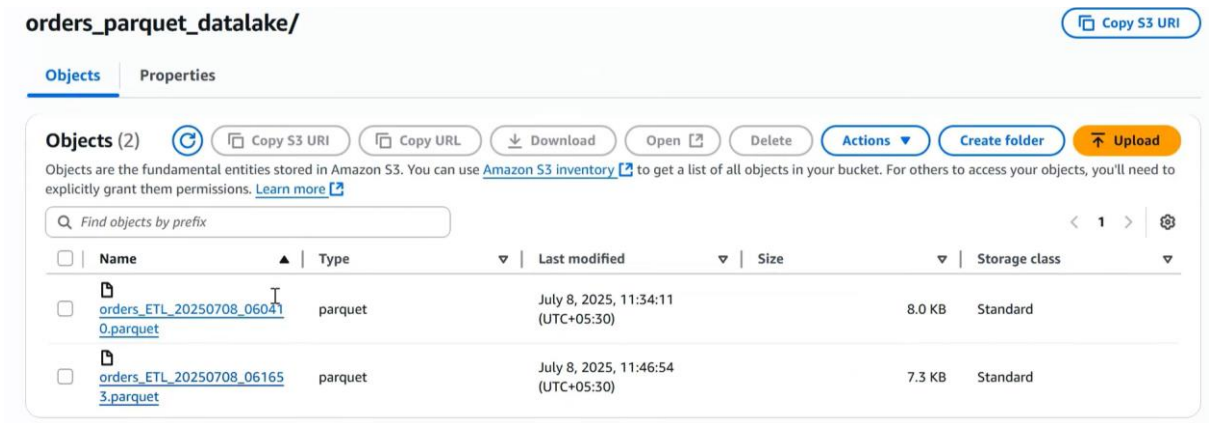


Now lets test it

Uploaded another file



Lets check



Run the query in athena and check the count