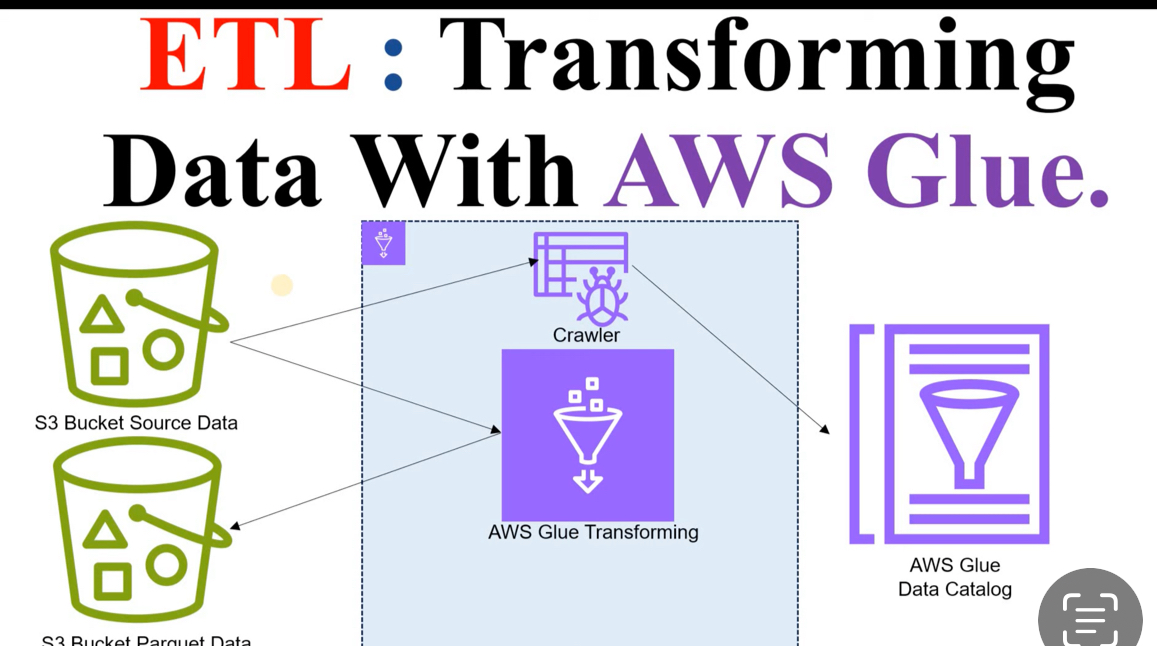
# High-level approach to solve this data engineering assignment.

**Step 1** – I uploaded test csv to S3 Bucket Source Data and then used Crawler to crawl data into AWS Glue Data Catalog (internally implemented SQL database). Then written an AWS Glue job to do data transformation and then upload to S3 Bucket Destination Data.



**Step 2** – As above step upload transformed data file to S3 Destination bucket which triggered below Lambda function to download data file and then insert into the PostgreSQL DB as show below.



**1. Code and Configuration:**

A - S3 Bucket creation:

[databucketsourcedemo](https://eu-north-1.console.aws.amazon.com/s3/buckets/databucketsourcedemo?region=eu-north-1&bucketType=general) - S3 bucket for input sales\_data.csv file

[databucketdestinationdemo](https://eu-north-1.console.aws.amazon.com/s3/buckets/databucketdestinationdemo?region=eu-north-1&bucketType=general) – S3 bucket for transformed file

A screenshot of a computer

Description automatically generated

**B – AWS Glue Script**

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from pyspark.sql.functions import udf, col, to\_date, year, month, sum as spark\_sum

from pyspark.sql.types import StringType, DoubleType

from awsglue.dynamicframe import DynamicFrame

from dateutil import parser

import datetime

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

def parse\_date(date\_string):

try:

return parser.parse(date\_string, dayfirst=True).strftime('%Y-%m-%d')

except:

return None

# Register UDF

parse\_date\_udf = udf(parse\_date, StringType())

# Read the input data

datasource = glueContext.create\_dynamic\_frame.from\_catalog(

database="demodatadb",

table\_name="databucketsourcedemo"

)

df = datasource.toDF()

# Drop duplicates

df = df.dropDuplicates()

# Apply the date parsing UDF

df = df.withColumn('sale\_date', parse\_date\_udf(col('sale\_date')))

# Convert the parsed string to date type

df = df.withColumn('sale\_date', to\_date(col('sale\_date')))

# Rename 'price' to 'total\_sales' as it already represents the total sale amount

df = df.withColumnRenamed('price', 'total\_sales')

# Ensure the total\_sales column is of DoubleType

df = df.withColumn('total\_sales', col('total\_sales').cast(DoubleType()))

# Extract year and month from sale\_date to group by month

df = df.withColumn('sale\_year', year(col('sale\_date')))

df = df.withColumn('sale\_month', month(col('sale\_date')))

# Group by product\_id, year, and month, and calculate total sales per product per month

aggregated\_df = df.groupBy(col('product\_id'), col('sale\_year'), col('sale\_month')) \

.agg(spark\_sum('total\_sales').alias('monthly\_total\_sales'))

# Order by product\_id, sale\_year, and sale\_month

aggregated\_df = aggregated\_df.orderBy(col('product\_id'), col('sale\_year'), col('sale\_month'))

# Coalesce the DataFrame to a single partition

aggregated\_df = aggregated\_df.coalesce(1)

# Convert back to DynamicFrame

aggregated\_dynamic\_frame = DynamicFrame.fromDF(aggregated\_df, glueContext, "aggregated\_dynamic\_frame")

# Write the aggregated DynamicFrame to S3 as a single CSV file

glueContext.write\_dynamic\_frame.from\_options(

frame=aggregated\_dynamic\_frame,

connection\_type="s3",

connection\_options={

"path": "s3://databucketdestinationdemo/",

"partitionKeys": []

},

format="csv",

format\_options={

"quoteChar": -1,

"writeHeader": True,

"separator": ",",

"compression": "none"

},

transformation\_ctx="write\_data"

)

job.commit()

C – Lamda function script

import json

import psycopg2

import os

import boto3

import csv

## This is the tool

def lambda\_handler(event, context):

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 = "/tmp/{}".format(s3\_key)

print("from path {}".format(from\_path))

#initiate s3 client

s3 = boto3.client('s3')

#Download object to the file

s3.download\_file(s3\_bucket, s3\_key, from\_path)

print("donwloaded successfully....")

dbname = os.getenv('dbname')

host = os.getenv('host')

user = os.getenv('user')

password = os.getenv('password')

tablename = os.getenv('tablename')

connection = psycopg2.connect(dbname = dbname,

host = host,

port = '5432',

user = user,

password = password)

print('after connection....')

curs = connection.cursor()

print('after cursor....')

# opening the CSV file

with open(from\_path, mode ='r')as file:

# reading the CSV file

csvFile = csv.reader(file)

# displaying the contents of the CSV file

for lines in csvFile:

print(type(lines))

print(lines[0])

print(lines[1])

print(lines[2])

querry = "INSERT INTO sales (product\_id,sale\_year,sale\_month,monthly\_total\_sales) VALUES ('{}', '{}', '{}', '{}');".format(lines[0], lines[1], lines[2], lines[3])

print("query is {}".format(querry))

print('after querry....')

curs.execute(querry)

connection.commit()

#print(curs.fetchmany(3))

print('after execute....')

curs.close()

print('after curs close....')

connection.close()

print('after connection close....')

print('wow..executed....')

os.remove(from\_path)

print('file removed from lambda storage....')

D– PostgreSQL Script

**CREATE** **TABLE** sales (

product\_id **VARCHAR**(50),

sale\_year **VARCHAR**(10),

sale\_month **VARCHAR**(10),

monthly\_total\_sales **VARCHAR**(20)

);

**2. Documentation:**

1. S3 bucket – Uploaded input file by using AWS S3 UI
2. Glue Job:

Create a new AWS Glue job named SalesDataETLJob using option “Author code with a script editor” by using above glue script

**Step-1**

Above scrip read data from table name databucketsourcedemo of Glue catalog database named demodatadb which is created by crawler after readying input file sales\_data.csv from [databucketsourcedemo](https://eu-north-1.console.aws.amazon.com/s3/buckets/databucketsourcedemo?region=eu-north-1&bucketType=general) S3 bucket.

**Step-2**

It does transformation as below:

* Remove duplicate record
* Formatting date in correct format
* Aggregation

**Step – 3**

It upload transform data to [databucketdestinationdemo](https://eu-north-1.console.aws.amazon.com/s3/buckets/databucketdestinationdemo?region=eu-north-1&bucketType=general)

A screenshot of a computer

Description automatically generated

**Lambda function**

* Created a Lambda functional named StoreTransformedData with python 3.11
* Add trigger on S3 bucket named databucketdestinationdemo for any event like GET, POST and PUT etc
* In configuration, increase Memory and Ephemeral storage to 1024MB and timeout to 15 min
* In configuration, add environment variables dbname, host, password, tablename, user so that lambda script read these value to connect to the Postgres DB
* In configuration, define default VPC and choose all 3 existing subnet, also choose default security group. In this group we allowed communication with out RDS DB.
* In VPC Dashboard, I have created one endpoint to communicate between S3.
* In code, we created a zip file which include lamda\_funtion.py and other modules like psycopg2. This lamda\_funtion.py logic includes to read S3 and connect to DB to insert these records.

A screenshot of a computer

Description automatically generated

**Verification:**

1. Glue job execution

A screenshot of a computer

Description automatically generated

1. **Lambda function logging**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **PostgreSQL SQL query result**

A screenshot of a computer

Description automatically generated