EShop Data Lake Project

Sources

A python script running on local machine handles data ingestion from the following sources:

- **SQL Server:**
  Data is extracted from SQL Server tables using the `extract_from_sqlserver` function. This leverages the `pyodbc` library, which establishes a connection using a Windows-authenticated connection string. Queries such as `SELECT * FROM CustomerTable` can retrieve structured data for analysis.
- **CSV Files:**
  Raw transactional data, like purchase history, is read using the `extract_from_csv` function, which uses `pandas.read_csv`. This efficiently parses flat files and converts them into a DataFrame for processing.
- **JSON Files:**
  Hierarchical and semi-structured data is processed via the extract_from_json function. The json.load and pandas.json_normalize methods are employed to flatten nested JSON into a tabular format suitable for storage and querying or upload_json_to_s3 method was used which sent the file to s3 directly without converting to csv.
- **Parquet files:**
  Read using pandas.read_parquet, which efficiently loads the columnar data into a DataFrame. Parquet files are optionally transformed (e.g., cleansing or type conversions) using the same methods as CSV or JSON data. The processed Parquet data can be uploaded by converting it into s3 for easier handling.

Ingestion Tools

To process data from these sources python script was embedded in the AWS Lambda function, which was triggered when a file was uploaded to eshopdataraw s3 bucket. The file was then streamed through and converted into csv format for better data analytics and ease for Amazon Glue crawler to create tables. Then, the file was uploaded to another s3 bucket eshopdatadpdb The following tools and libraries are utilized:

1. **`pyodbc`:**
   Facilitates connection to SQL Server, allowing the script to execute queries and extract data efficiently. This is particularly useful for accessing relational databases in real-time.
2. **`pandas`:**
   Serves as the core data manipulation library. It parses CSV and JSON files into DataFrame structures, enabling seamless data transformation and cleansing. Its versatility allows for handling both tabular and hierarchical data formats.
3. **`boto3`:**
   AWS SDK for Python, `boto3`, is used to interact with Amazon S3. It allows uploading processed DataFrames as CSVs and JSON objects to the specified S3 bucket. Raw

data is categorized into folders (e.g., `eshopdataraw.csv` for CSV, `productdata.json` for JSON).

Analytics:

AWS Glue was used for Transforming data further operations for analytics

- Cleansing raw data (removing nulls, standardizing formats).
- Transforming data (e.g., converting JSON to Parquet for efficient querying).
- Writing processed data back to S3 in a structured manner for analytics.

Querying Data with AWS Athena

- Athena was used to perform serverless SQL querying directly on processed S3 data.
- It was used for
    - Joining and combining transaction logs with customer data.
    - Analysing clickstream data as a function of customer data.
    - Mapping clickstream to product data.

Amazon QuickSight

We utilized Amazon QuickSight to create interactive dashboards that visualized query results directly from AWS Athena. These visual were designed to monitor performance of the ecommerce website eshop. Multiple tables were joined as two datasets. Then, on these datasets, analytics were performed visually, like which customers are spending the highest by mapping customer and purchase data.

Data Handling of Multiple formats:

Using Pandas and pyodbc library, Json and CSV, data was processed as part of Lambda function script.

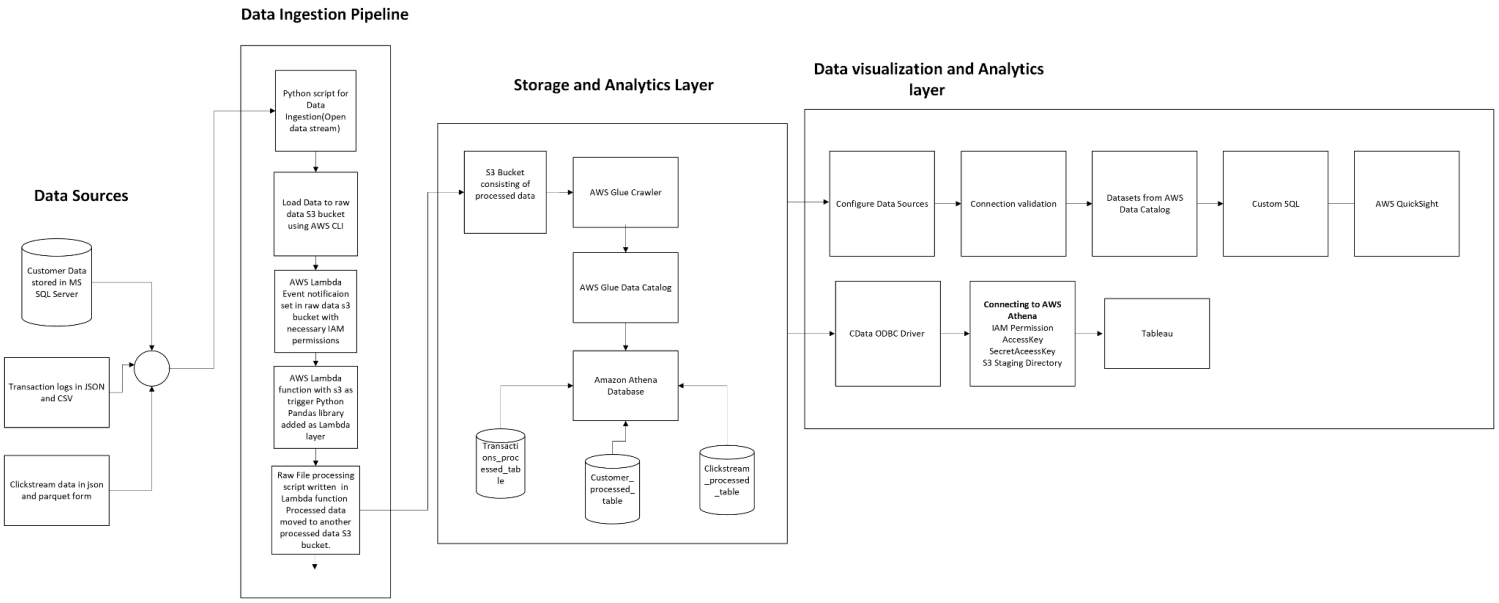
AWS Glue's schema registry and Apache Spark's(which is embedded in AWS Glue) flexibility will manage Parquet, CSV, JSON, and SQL data transformations.

Scalability:

Amazon S3 is the data lake, offering virtually unlimited storage with high durability. Its ability to support multi-format datasets allowed me to scale data ingestion without the need for manual tasks. I optimized AWS Athena's performance by partitioning data in S3 and converting it into efficient formats like Parquet. This will improve execution times, even with large data. Glue Crawlers automat the schema detection and data cataloging process for datasets stored in

Amazon S3, for formats like JSON, CSV, and Parquet. This will make sure that as data sources grow in complexity and size, the metadata remained consistent and query-ready.

Architecture diagram for Datalake:



Task 1:

A python script was run on local machine to get data from following sources:

- Customer data from a MySQL database.
- Transaction data from a CSV file located on local machine.
- Product information from a JSON file on local machine.

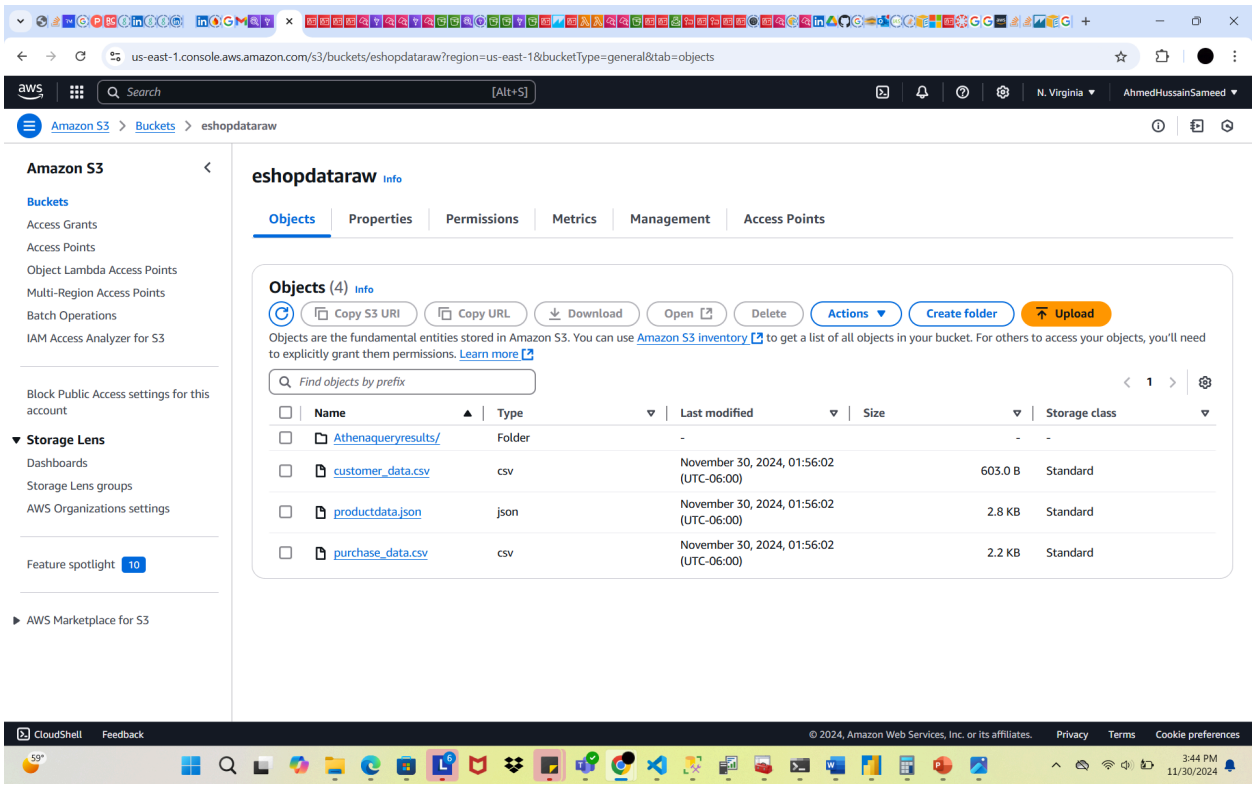
The following functions were used to load data into Amazon S3 in the ETL pipeline:

load_to_s3(data, s3_bucket, s3_key):

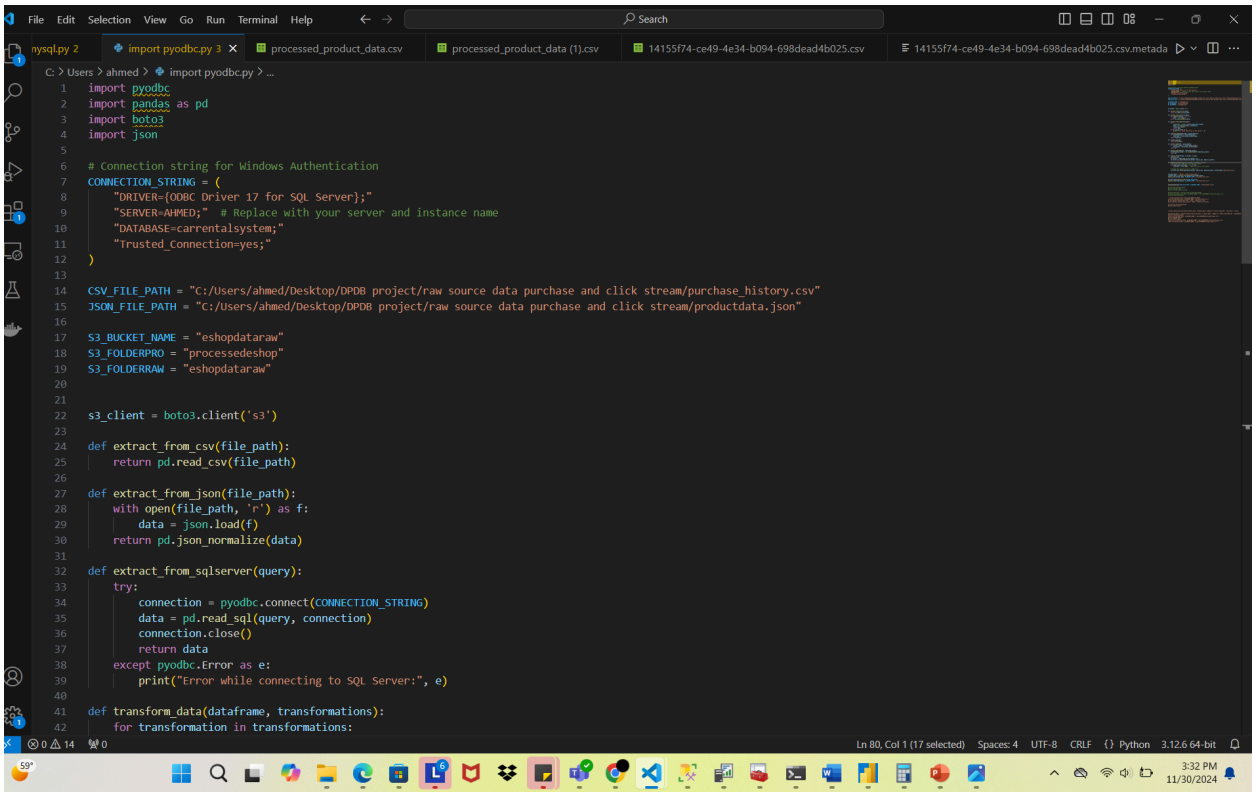
This function was used to upload data to S3. The data argument is the data that needs to be stored in s3.
The function converts the DataFrame into a CSV string using dataframe.to_csv(index=False) and then uploads it to the specified S3 bucket and key (path) using s3_client.put_object(). The s3_key represents the file path and name in the S3 bucket, ensuring that data is organized properly within the appropriate folder (either raw or processed).
upload_json_to_s3(local_file_path, s3_bucket, s3_key):

This function was specifically designed for uploading JSON files to S3. It reads the content of a local JSON file (specified by local_file_path) using open(local_file_path, 'r'), then reads the entire file into a string.
The content of the JSON file is then uploaded to the specified S3 bucket with the s3_client.put_object() method. The ContentType='application/json' is specified to ensure that the file is recognized as JSON. The s3_key specifies the path where the JSON file will be stored in the S3 bucket.

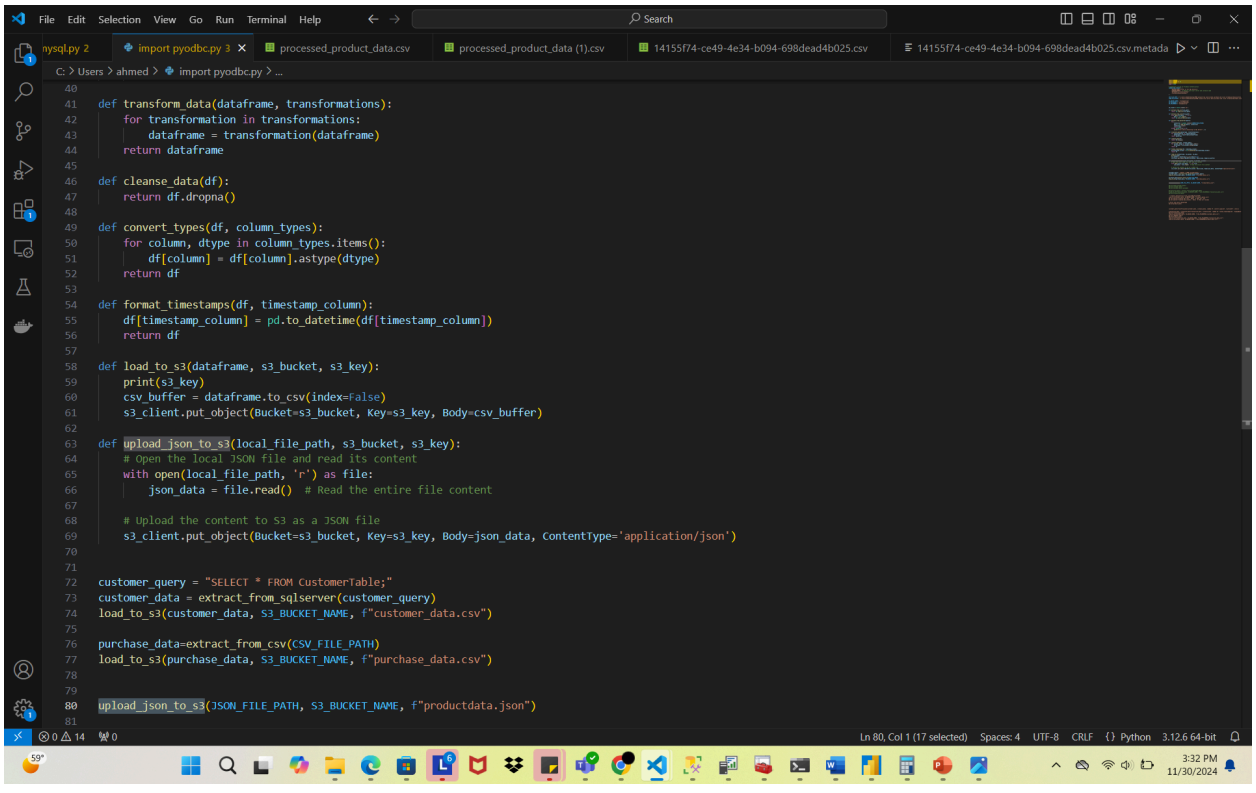
Raw Data S3 Bucket after loading data:



Python script to upload data from sources to S3 bucket.



```python
import pyodbc
import pandas as pd
import boto3
import json

# Connection string for Windows Authentication
CONNECTION_STRING = (
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=AHMED;"  # Replace with your server and instance name
    "DATABASE=carrentalsystem;"
    "Trusted_Connection=yes;"
)

CSV_FILE_PATH = "C:/Users/ahmed/Desktop/DPDB project/raw source data purchase and click stream/purchase_history.csv"
JSON_FILE_PATH = "C:/Users/ahmed/Desktop/DPDB project/raw source data purchase and click stream/productdata.json"

S3_BUCKET_NAME = "eshopdataraw"
S3_FOLDERPRO = "processedeshop"
S3_FOLDERRAW = "eshopdataraw"


s3_client = boto3.client('s3')

def extract_from_csv(file_path):
    return pd.read_csv(file_path)

def extract_from_json(file_path):
    with open(file_path, 'r') as f:
        data = json.load(f)
    return pd.json_normalize(data)

def extract_from_sqlserver(query):
    try:
        connection = pyodbc.connect(CONNECTION_STRING)
        data = pd.read_sql(query, connection)
        connection.close()
        return data
    except pyodbc.Error as e:
        print("Error while connecting to SQL Server:", e)

def transform_data(dataframe, transformations):
    for transformation in transformations:
```



```python
def transform_data(dataframe, transformations):
    for transformation in transformations:
        dataframe = transformation(dataframe)
    return dataframe

def cleanse_data(df):
    return df.dropna()

def convert_types(df, column_types):
    for column, dtype in column_types.items():
        df[column] = df[column].astype(dtype)
    return df

def format_timestamps(df, timestamp_column):
    df[timestamp_column] = pd.to_datetime(df[timestamp_column])
    return df

def load_to_s3(dataframe, s3_bucket, s3_key):
    print(s3_key)
    csv_buffer = dataframe.to_csv(index=False)
    s3_client.put_object(Bucket=s3_bucket, Key=s3_key, Body=csv_buffer)

def upload_json_to_s3(local_file_path, s3_bucket, s3_key):
    # Open the local JSON file and read its content
    with open(local_file_path, 'r') as file:
        json_data = file.read()  # Read the entire file content

    # Upload the content to S3 as a JSON file
    s3_client.put_object(Bucket=s3_bucket, Key=s3_key, Body=json_data, ContentType='application/json')


customer_query = "SELECT * FROM CustomerTable;"
customer_data = extract_from_sqlserver(customer_query)
load_to_s3(customer_data, S3_BUCKET_NAME, f"customer_data.csv")

purchase_data=extract_from_csv(CSV_FILE_PATH)
load_to_s3(purchase_data, S3_BUCKET_NAME, f"purchase_data.csv")


upload_json_to_s3(JSON_FILE_PATH, S3_BUCKET_NAME, f"productdata.json")
```

Data Transformation and Loading:

For both data transformation and data loading, we have used AWS Lambda Function. In the AWS lambda function, we have added layers to include libraries such as Pandas to process and clean the data.

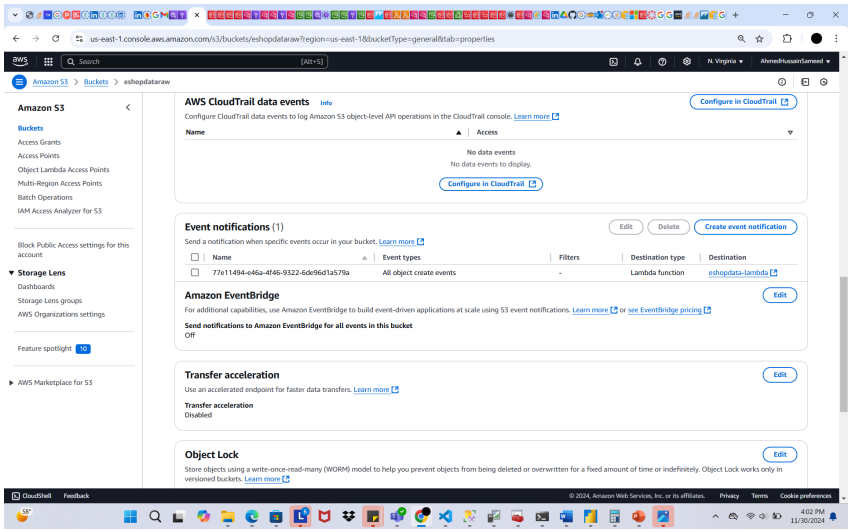
Data Transformation with AWS Lambda:
AWS Lambda was set up to automatically trigger whenever a file was uploaded to an S3 bucket, either a raw CSV or JSON file. Lambda would then process the data without the need for external servers or compute resources.
To transform the data, Python libraries, especially **Pandas**, were added as Lambda layers. This allowed the Lambda function to perform tasks like:

- **Data Cleansing**: Removing any rows with missing or null values.
- **Type Conversions**: Changing the data types of columns as needed (e.g., converting strings into integers for numerical columns).
- **Timestamp Formatting**: Standardizing date columns to a consistent timestamp format using the `pandas.to_datetime()` function.

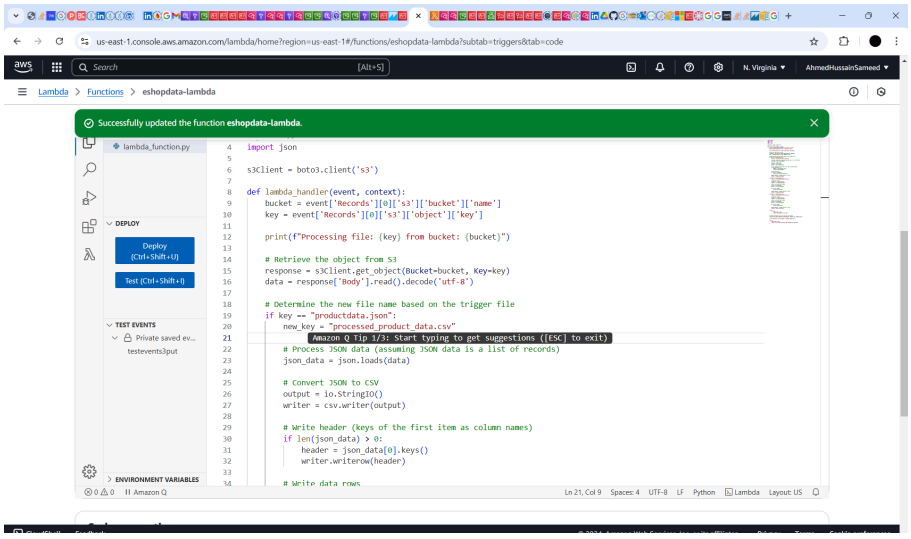
Data Loading:
Once the data was transformed inside Lambda, it was uploaded to a designated S3 bucket for processed data. The transformed files were stored in separate folders, making them ready for analysis. The raw data was placed in a different folder in the S3 bucket, keeping the original files for reference or reprocessing if needed.

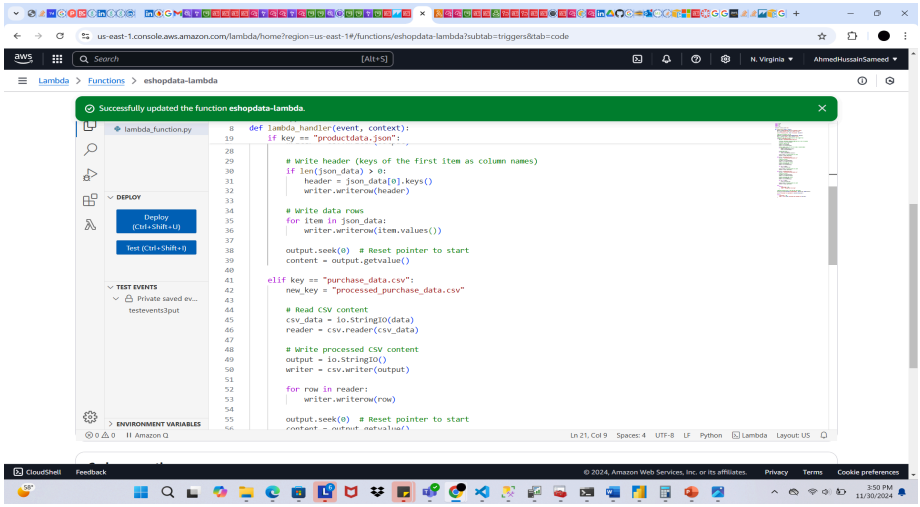
Automation and Monitoring:
The entire process was automated using **S3 event notifications**. Whenever a file was added to the raw data bucket, it would trigger the Lambda function to process the file and move it to the appropriate location in S3.
Logging and monitoring were set up with **Amazon CloudWatch**. This helped track errors or issues in the pipeline and allowed real-time visibility.

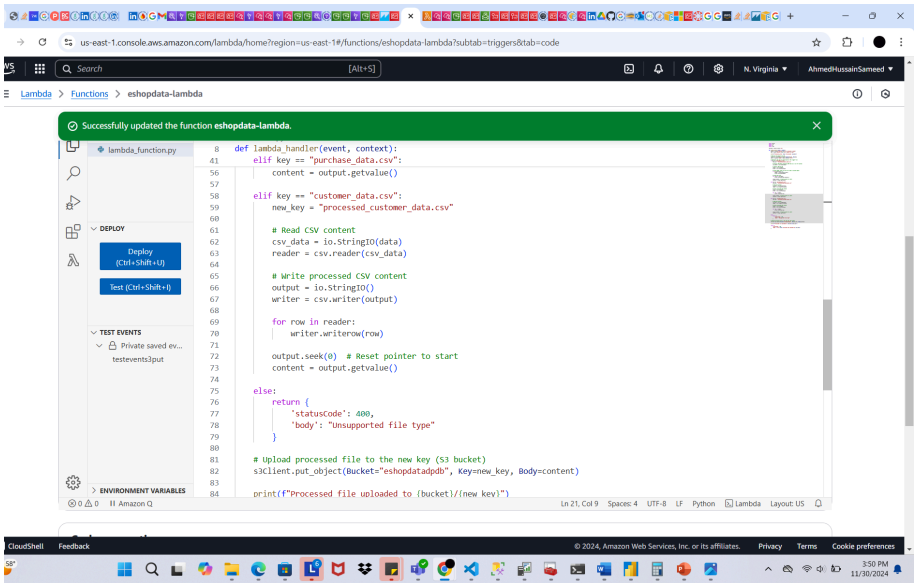
Event Notification in rawdata se bucket for lambda function:

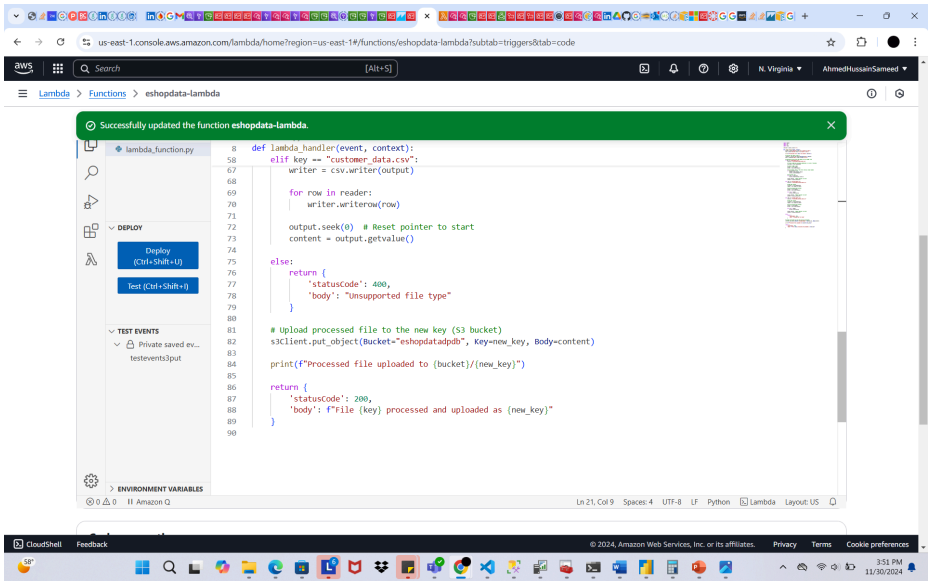


Lambda function for data transformation and loading into s3 bucket processed was written in Python 3.13 runtime:

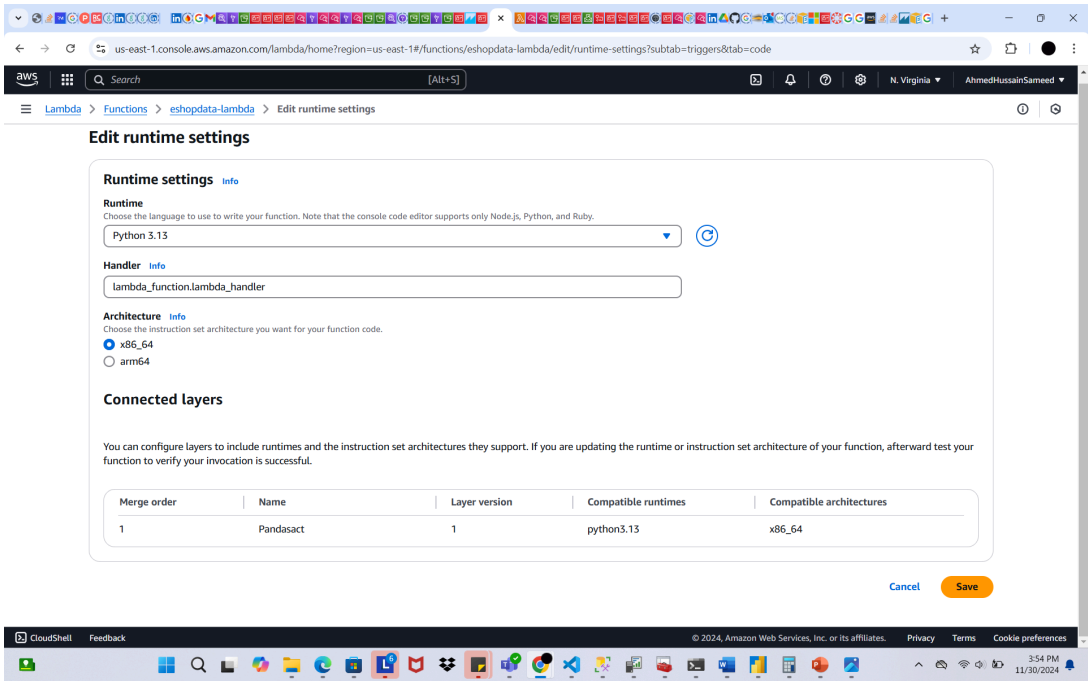


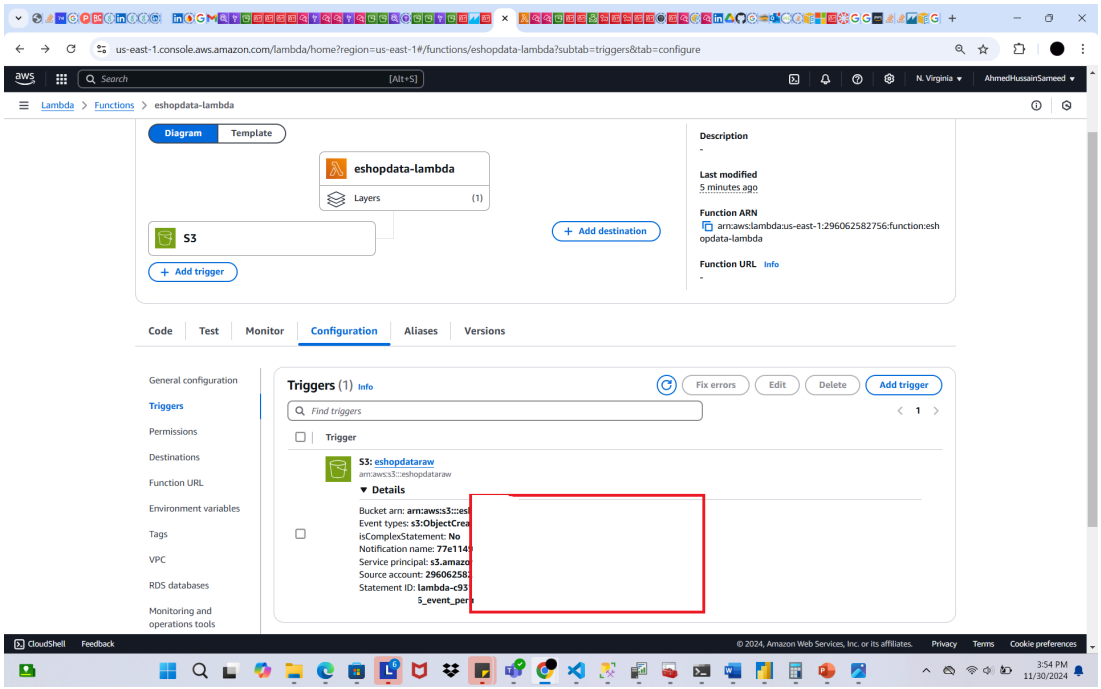

Successfully updated the function eshopdata-lambda.
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS
Private saved ev...
testevents3put
ENVIRONMENT VARIABLES
def lambda_handler(event, context):
if key == "productdata.json":
# Write header (keys of the first item as column names)
if len(json_data) > 0:
header = json_data[0].keys()
writer.writerow(header)
# Write data rows
for item in json_data:
writer.writerow(item.values())
output.seek(0) # Reset pointer to start
content = output.getvalue()
elif key == "purchase_data.csv":
new_key = "processed_purchase_data.csv"
# Read CSV content
csv_data = io.StringIO(data)
reader = csv.reader(csv_data)
# Write processed CSV content
output = io.StringIO()
writer = csv.writer(output)
for row in reader:
writer.writerow(row)
output.seek(0) # Reset pointer to start

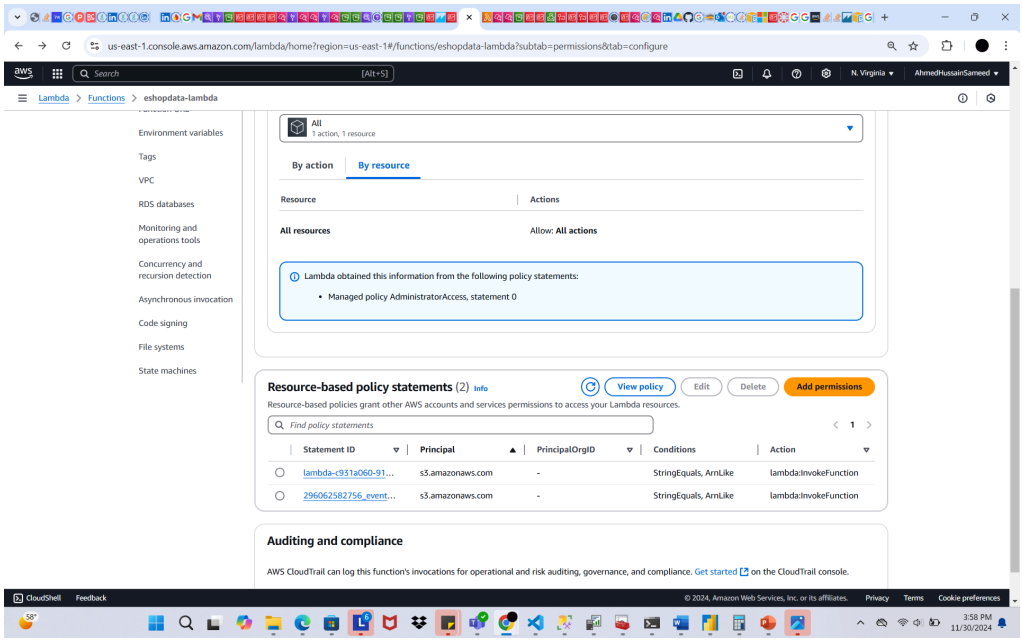

Successfully updated the function eshopdata-lambda.
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS
Private saved ev...
testevents3put
ENVIRONMENT VARIABLES
def lambda_handler(event, context):
elif key == "purchase_data.csv":
content = output.getvalue()
elif key == "customer_data.csv":
new_key = "processed_customer_data.csv"
# Read CSV content
csv_data = io.StringIO(data)
reader = csv.reader(csv_data)
# Write processed CSV content
output = io.StringIO()
writer = csv.writer(output)
for row in reader:
writer.writerow(row)
output.seek(0) # Reset pointer to start
content = output.getvalue()
else:
return {
'statusCode': 400,
'body': "Unsupported file type"
}
# Upload processed file to the new key (S3 bucket)
s3Client.put_object(Bucket="eshopdatadpdb", Key=new_key, Body=content)
print(f"Processed file uploaded to {bucket}/{new_key}")


Successfully updated the function eshopdata-lambda.
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS
Private saved ev...
testevents3put
ENVIRONMENT VARIABLES
def lambda_handler(event, context):
elif key == "customer_data.csv":
writer = csv.writer(output)
for row in reader:
writer.writerow(row)
output.seek(0) # Reset pointer to start
content = output.getvalue()
else:
return {
'statusCode': 400,
'body': "Unsupported file type"
}
# Upload processed file to the new key (S3 bucket)
s3Client.put_object(Bucket="eshopdatadpdb", Key=new_key, Body=content)
print(f"Processed file uploaded to {bucket}/{new_key}")
return {
'statusCode': 200,
'body': f"File {key} processed and uploaded as {new_key}"
}

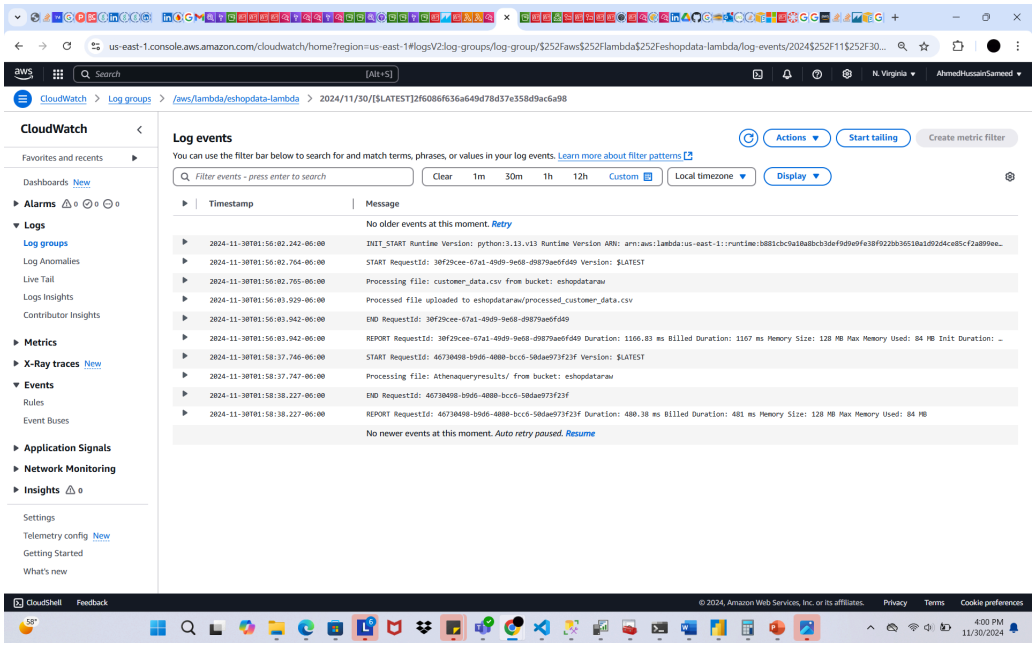
Configuration and Runtime Settings for Lambda function:



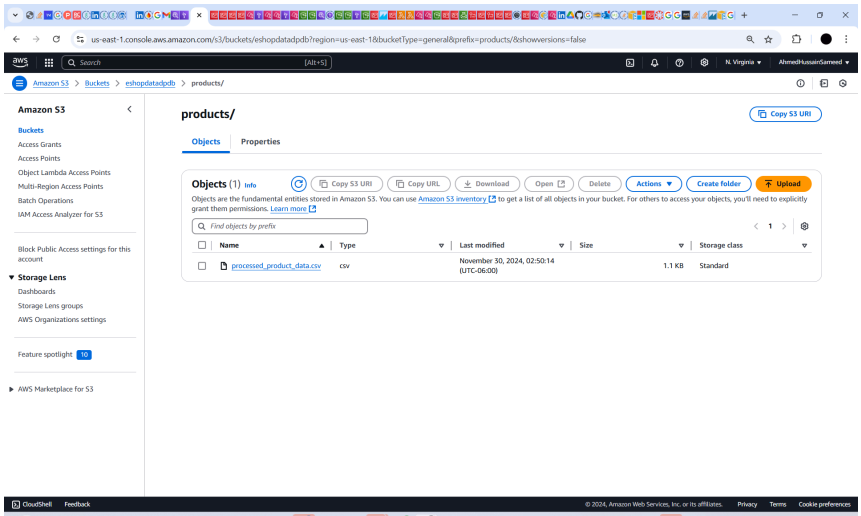


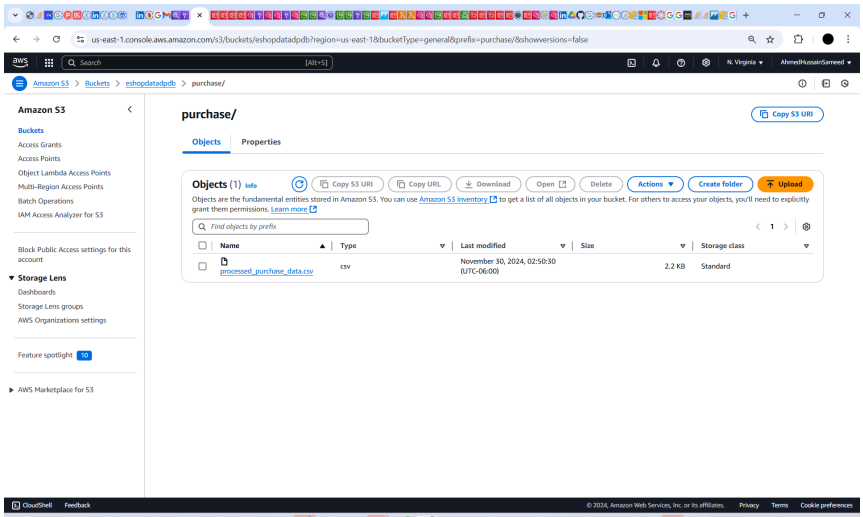


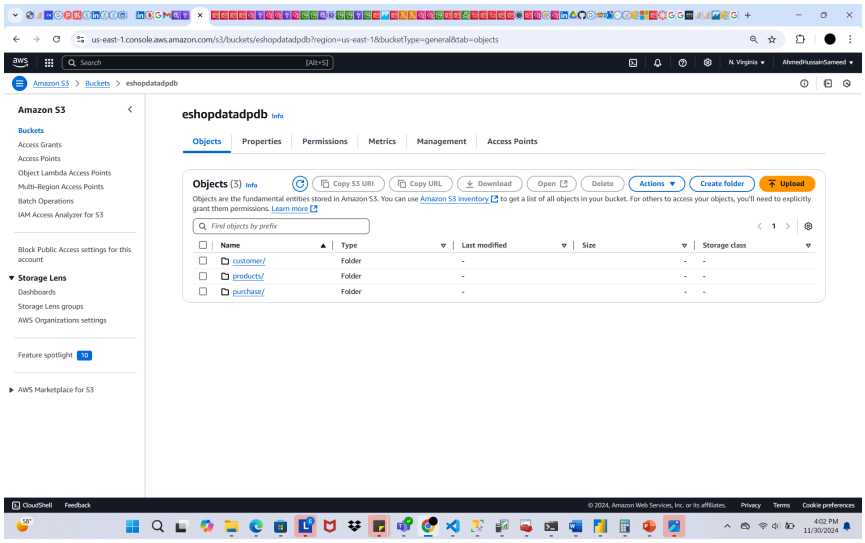
Logs for AWS Lambda in Amazon Cloudwatch:



Data Loaded onto Datalake:


us-east-1.console.aws.amazon.com/s3/buckets/eshopdatadpdb?region=us-east-1&bucketType=general&prefix=products/&showversions=false
Amazon S3 > Buckets > eshopdatadpdb > products/
products/
Copy S3 URI
Objects
Properties
Objects (1) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
processed_product_data.csv
csv
November 30, 2024, 02:50:14 (UTC-06:00)
1.1 KB
Standard
Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight
AWS Marketplace for S3
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

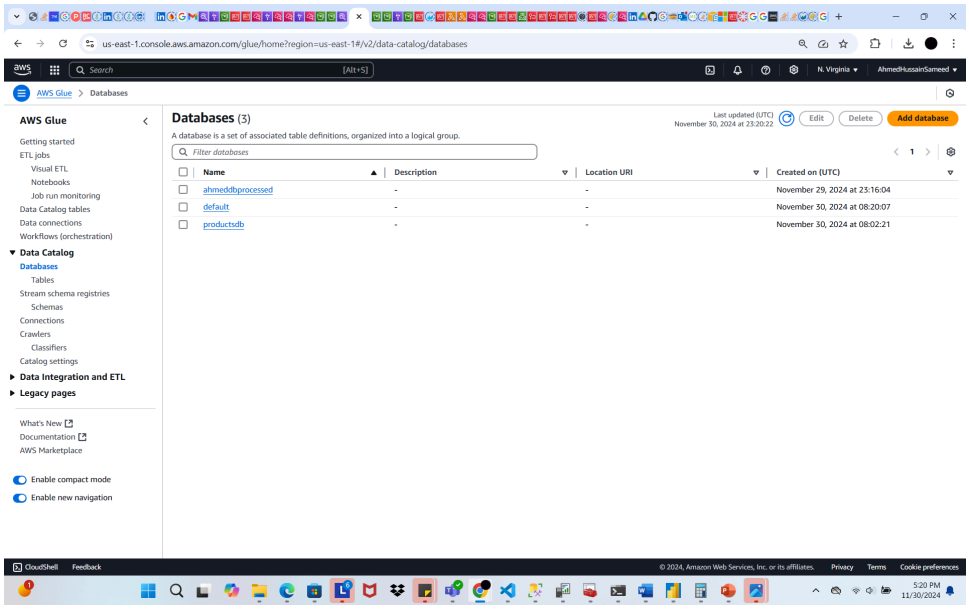

us-east-1.console.aws.amazon.com/s3/buckets/eshopdatadpdb?region=us-east-1&bucketType=general&prefix=purchase/&showversions=false
Amazon S3 > Buckets > eshopdatadpdb > purchase/
purchase/
Copy S3 URI
Objects
Properties
Objects (1) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
processed_purchase_data.csv
csv
November 30, 2024, 02:50:30 (UTC-06:00)
2.2 KB
Standard
Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight
AWS Marketplace for S3
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


us-east-1.console.aws.amazon.com/s3/buckets/eshopdatadpdb?region=us-east-1&bucketType=general&tab=objects
Amazon S3 > Buckets > eshopdatadpdb
eshopdatadpdb Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (3) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
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
Find objects by prefix
Name
Type
Last modified
Size
Storage class
customer/
Folder
products/
Folder
purchase/
Folder
Amazon S3
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings
Feature spotlight
AWS Marketplace for S3
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Data Querying and Analysis

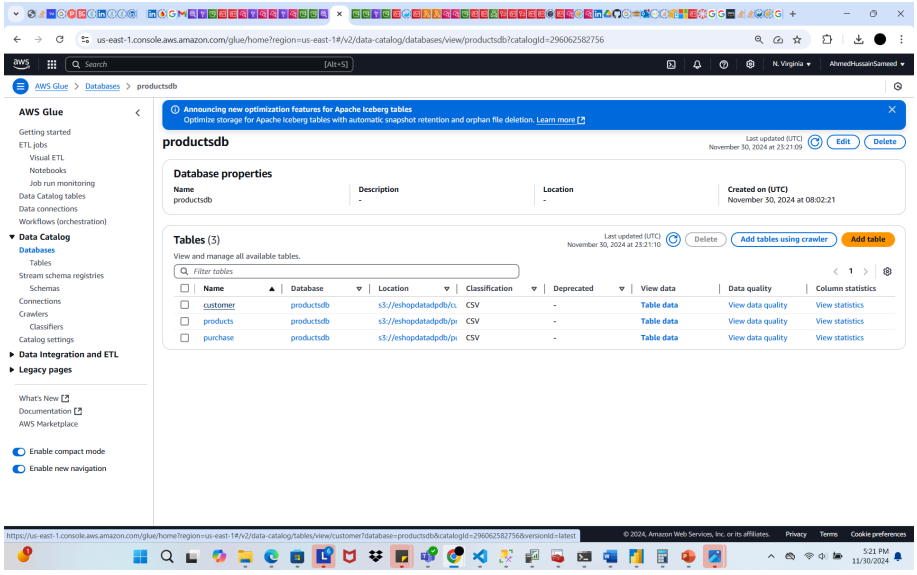
For data querying performing SQL-like queries, we have used Amazon Athena.
We have used AWS Glue to create tables.
AWS Glue Crawler Creation: A Glue Crawler was created to scan the data in the processed data S3 bucket. The crawler was configured to automatically detect the structure of the data and create corresponding table definitions in the Glue Data Catalog.
Crawler Execution: Once the crawler was executed, it created metadata tables in the Glue Data Catalog, which included details like column names, data types, and partitioning based on the data format and structure.
Athena Queries: With the metadata stored in the Glue Data Catalog, queries were executed in Amazon Athena. The tables created by Glue were used to run SQL queries for data analysis.
Automation: The Glue crawler was scheduled to run periodically, enabling automatic detection of new data in the S3 bucket. This ensured that the Athena tables remained up-to-date without manual intervention.

Databases created in AWS Glue:



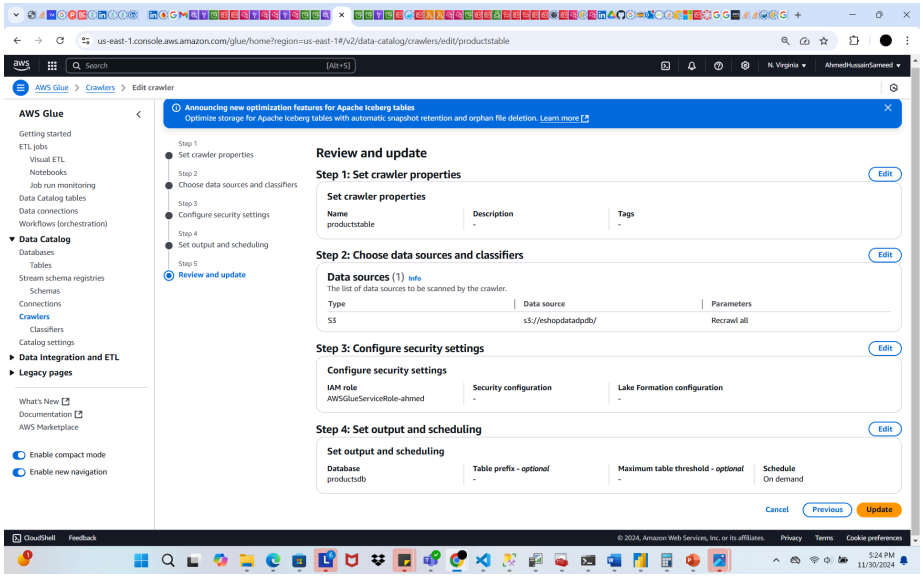
Tables consisting of processed Data:

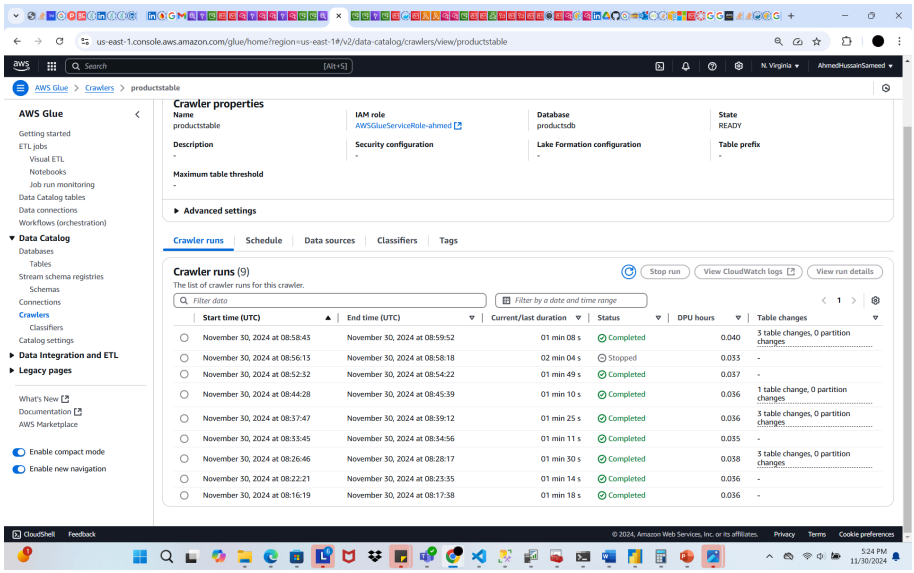
All the table columns and properties were created by AWS Glue. The folder path to the S3 directory was given as input. In the excluded fields, all other directories were added. Difficulty was encountered when specific files were given as input to the Data Catalog. Upon debugging, we found that it is ideal to give the folder path of an s3 file rather than a file name consisting of the data.



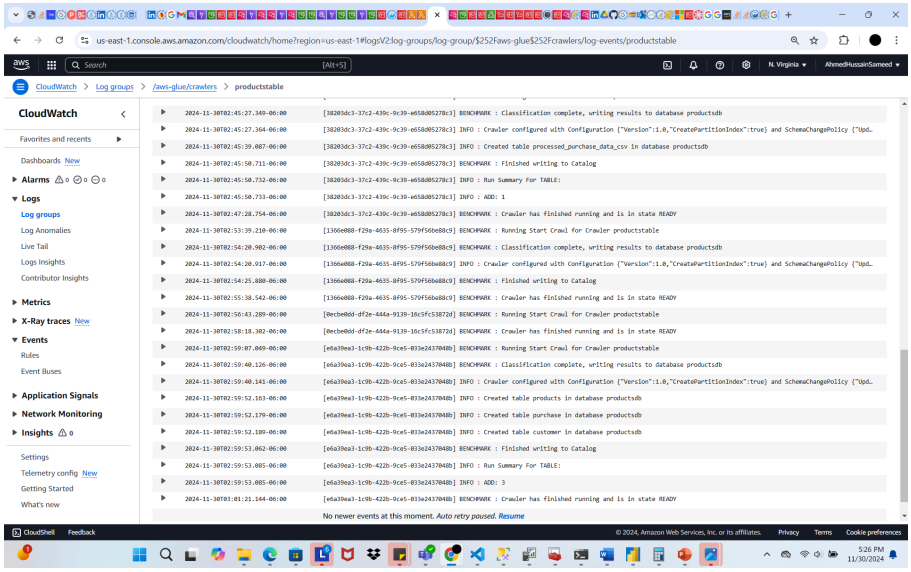
AWS Glue Crawler properties and configuration:

Frequency of running crawler is set to onDemand.

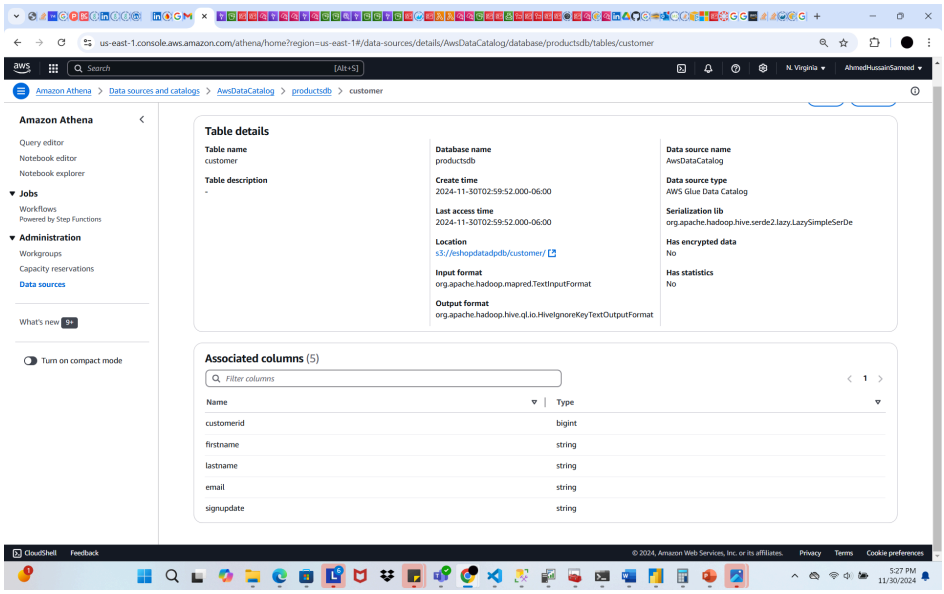




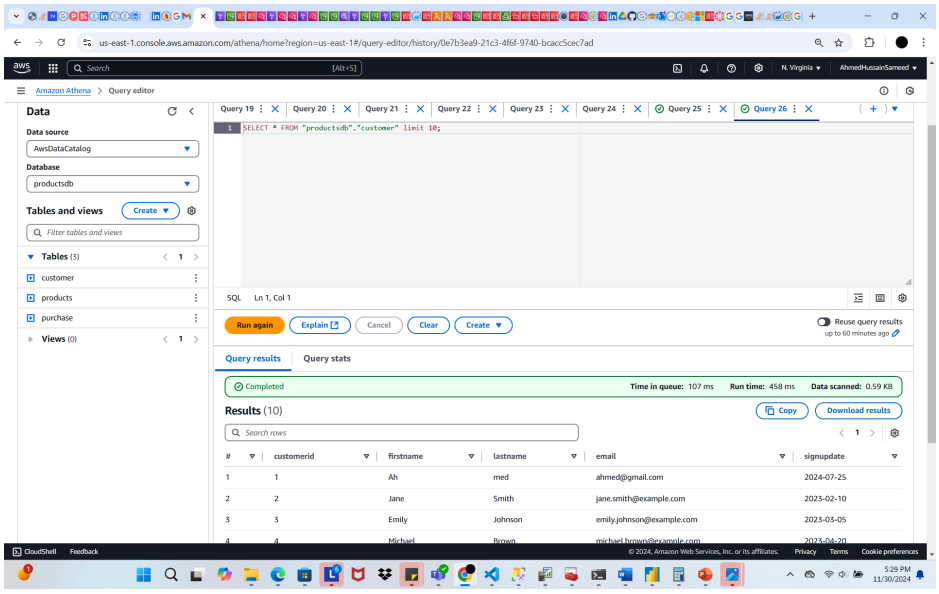
AWS Cloudwatch logs when Crawler was running:



Sample Table created after Crawler run was successful:

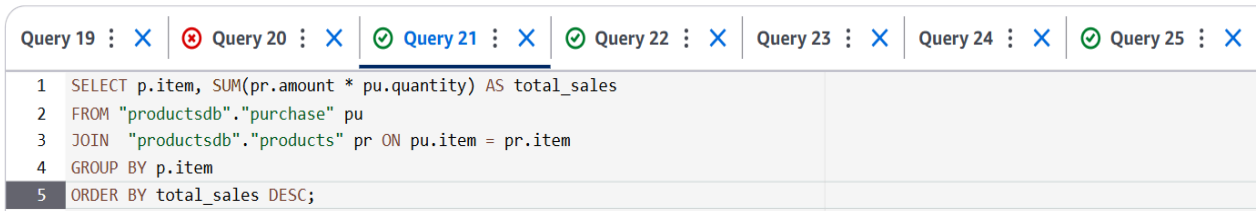


Sample query result



Queries:

Total Sales by Product



Top Selling Products

Query 19 | Query 20 | Query 21 | Query 22 | Query 23 | Query 24 | Query 25

```sql
SELECT p.item, SUM(pu.quantity) AS total_quantity_sold
FROM "productsdb"."purchase" pu
JOIN "productsdb"."products" p ON pu.item = p.item
GROUP BY p.item
ORDER BY total_quantity_sold DESC
LIMIT 10;
```

Average Purchase Amount by Payment Method

Query 19 | Query 20 | Query 21 | Query 22 | Query 23 | Query 24 | Query 25

```sql
SELECT paymentmethod, AVG(amount) AS avg_purchase_amount
FROM "productsdb"."purchase"
GROUP BY paymentmethod
ORDER BY avg_purchase_amount DESC;
```

Sales by category

Query 21 | Query 22 | Query 23 | Query 24 | Query 25 | Query 26 | Query 27

```sql
SELECT pr.category, SUM(pu.amount * pu.quantity) AS total_sales
FROM "productsdb"."purchase" pu
JOIN "productsdb"."products" pr ON pu.item = pr.item
GROUP BY pr.category
ORDER BY total_sales DESC;
```

Query to segregate customers based on purchase data. Low value vs high value

Query 29 | Query 20 | Query 21 | Query 22 | Query 23 | Query 24 | Que

```sql
WITH CustomerPurchaseSummary AS (
    SELECT
        c.customerid,
        c.firstname,
        c.lastname,
        c.email,
        SUM(p.amount * p.quantity) AS total_spent
    FROM "productsdb"."purchase" p
    JOIN "productsdb"."customer"  c ON p.customerid = c.customerid
    GROUP BY c.customerid, c.firstname, c.lastname, c.email
)
SELECT
    cps.customerid,
    cps.firstname,
    cps.lastname,
```

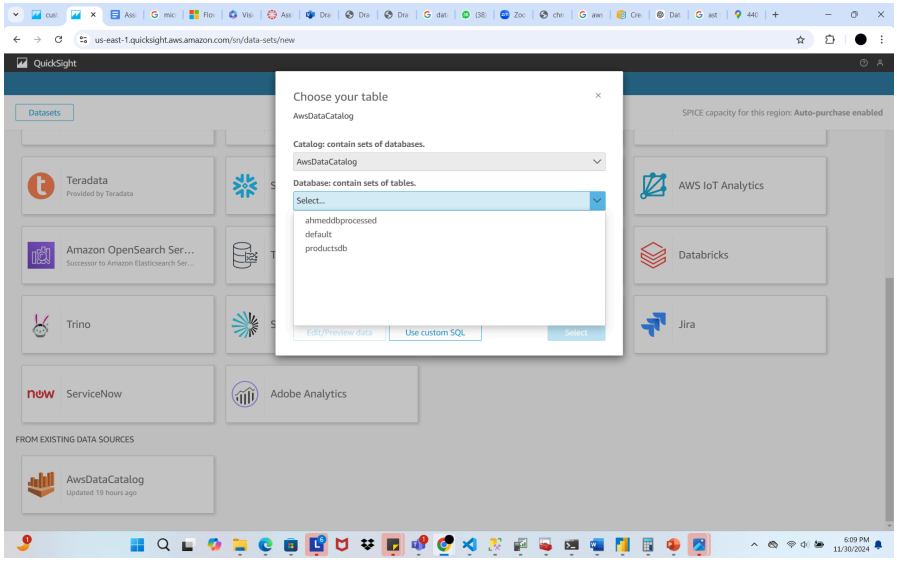
Query 29 | Query 20 | Query 21 | Query 22 | Query 23 | Query 24 | Que

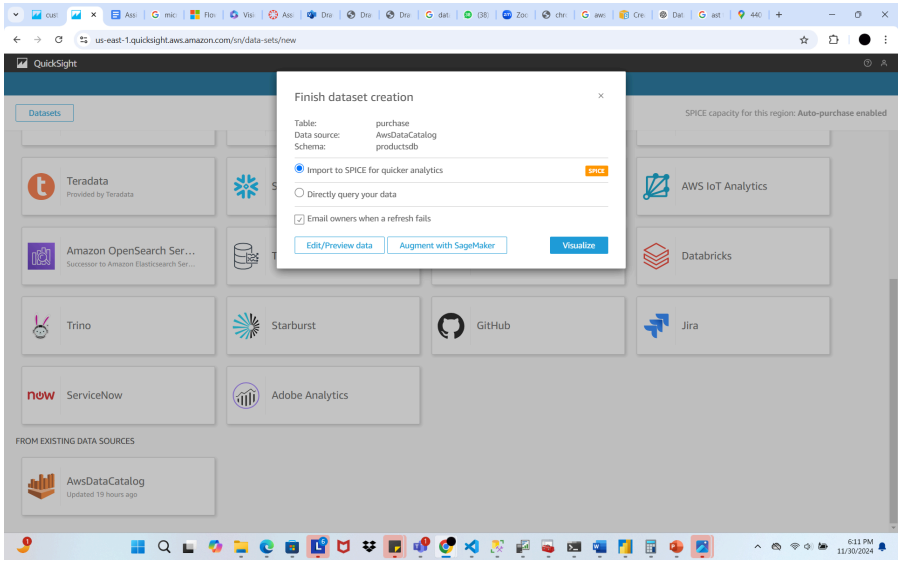
```sql
)
SELECT
    cps.customerid,
    cps.firstname,
    cps.lastname,
    cps.email,
    cps.total_spent,
    CASE
        WHEN cps.total_spent >= 1000 THEN 'High-Value'
        WHEN cps.total_spent >= 500 AND cps.total_spent < 1000 THEN 'Medium-Value'
        ELSE 'Low-Value'
    END AS customer_segment
FROM CustomerPurchaseSummary cps
ORDER BY cps.total_spent DESC;
```

Data Visualization:

We have used Amazon Quicksight for visualization of datasets in the datalake in s3 bucket. First, data is made in table format using AWS Glue and Crawlers. Then, it is loaded onto Amazon Athena from AWS DataCatalog. Quicksight also loads data from AWA DataCatalog.

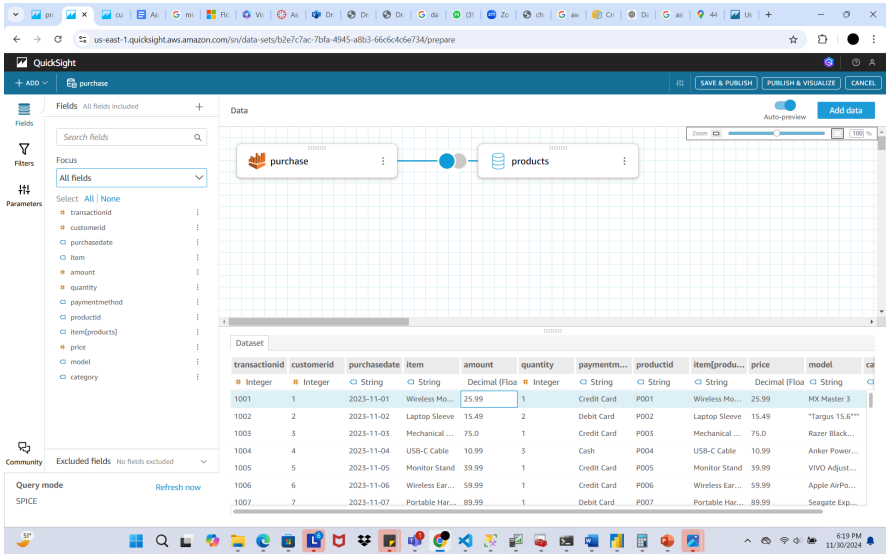
Creating datasets in Amazon Quicksight using AWS DataCatalog

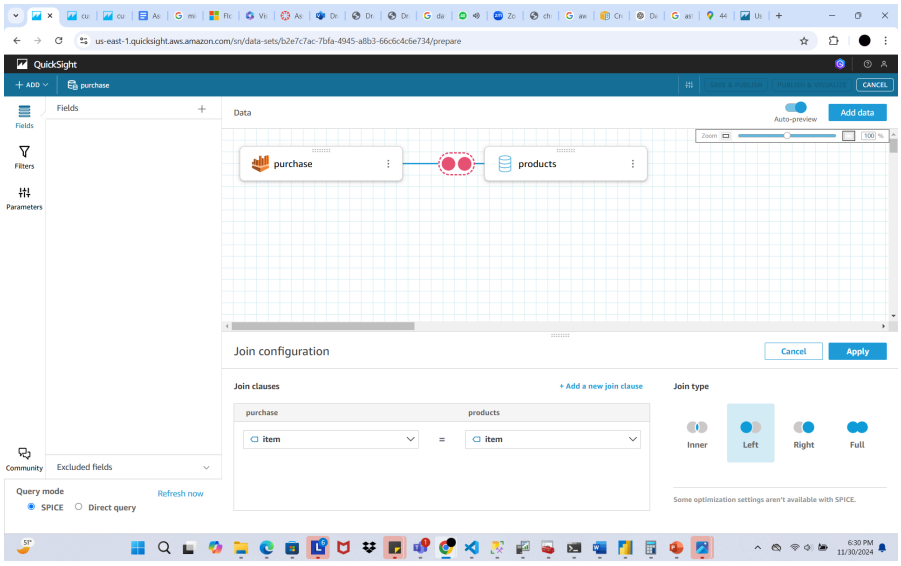




Visualization:

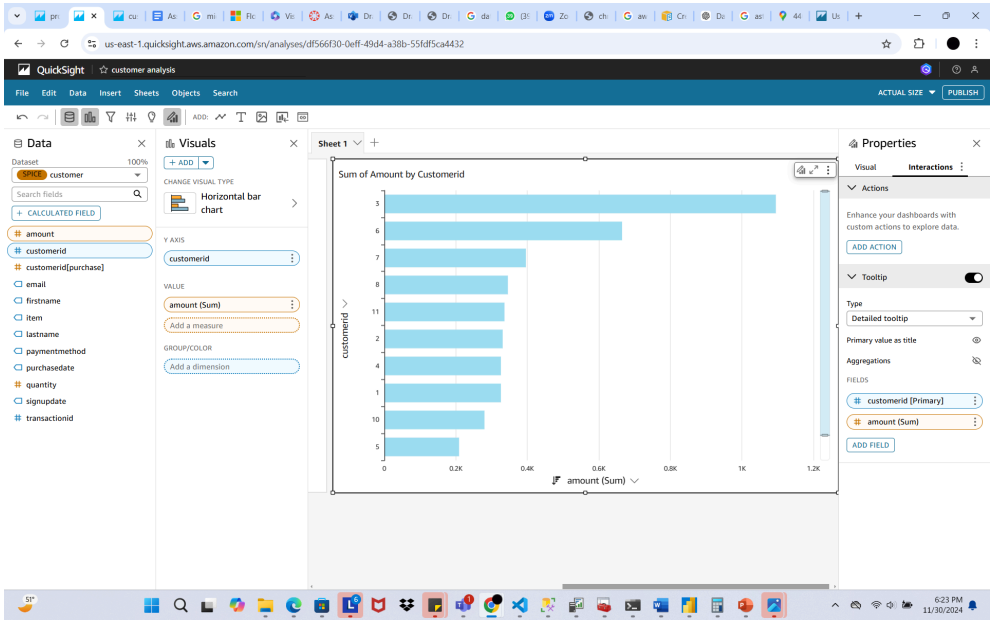
Joining Tables in Amazon Quick Sight:





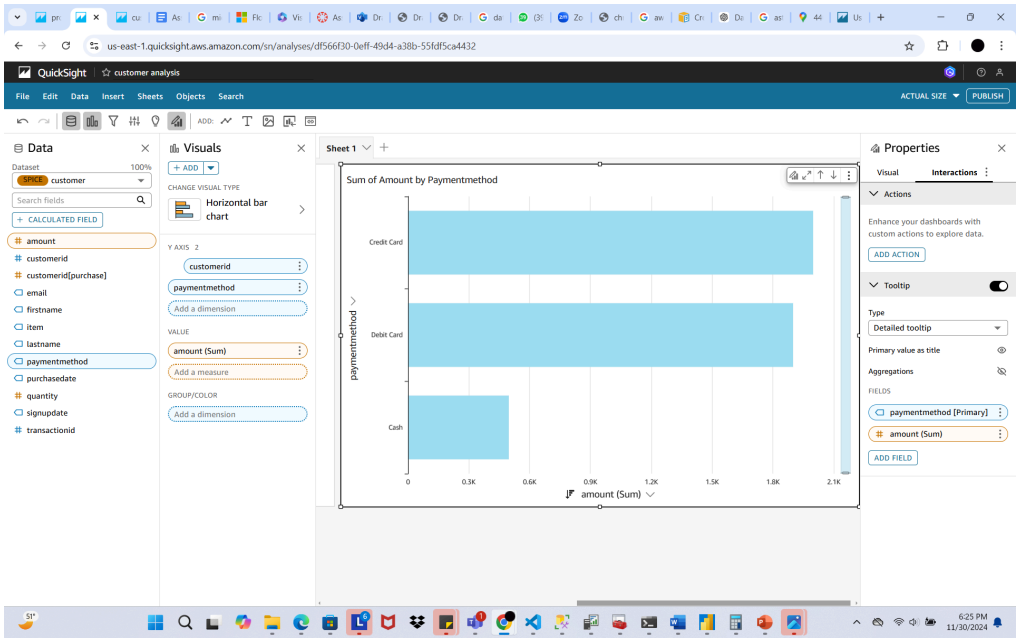
Ranking customers in accordance with their purchase amounts

This horizontal bar graph shows the customer spending habits, depicting the total of purchase amounts against each customer ID. This will help E-Shop identify its most lucrative customers, targeting those whose contribution to revenue is very high. Insights from this might drive the development of loyalty programs, personalized marketing, and VIP offers in order to retain these top customers. Additionally, customers with lower spending could be allowed promotional offers or discounts that would help in repeat purchases. By understanding the trend in spending, E-Shop can segment the customer base and optimize marketing budgets to draw up strategies to maximize customer lifetime value and overall revenue growth.



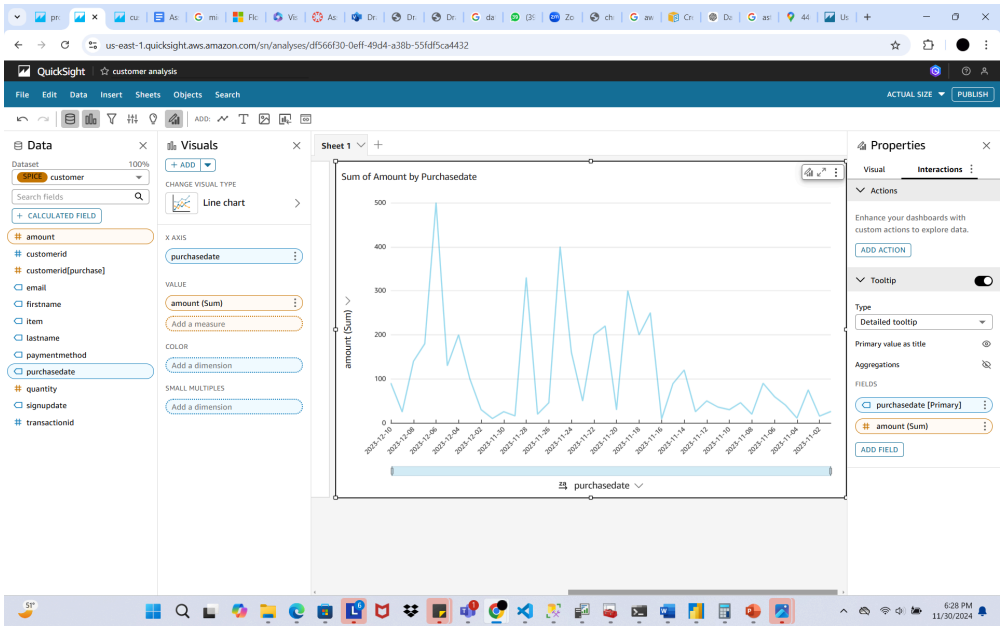
Grouping purchases as a function of payment method

This bar chart shows the total amount of transactions by different payment methods: Credit Card, Debit Card, and Cash. The chart provides important information about the preferences of customers regarding payments, showing that combined debit and credit card transactions comprise a far larger proportion than cash purchases. E-Shop shall use these findings to improve its payment infrastructure, ensuring seamless digital payment experiences. It also helps in understanding the lower cash reliance to make informed decisions that can further promote digital payment options and reduce operational costs related to cash handling. Strategic alliances with card providers will also allow for incentives on more card-based purchases, promoting convenience for customers and increasing overall revenue.

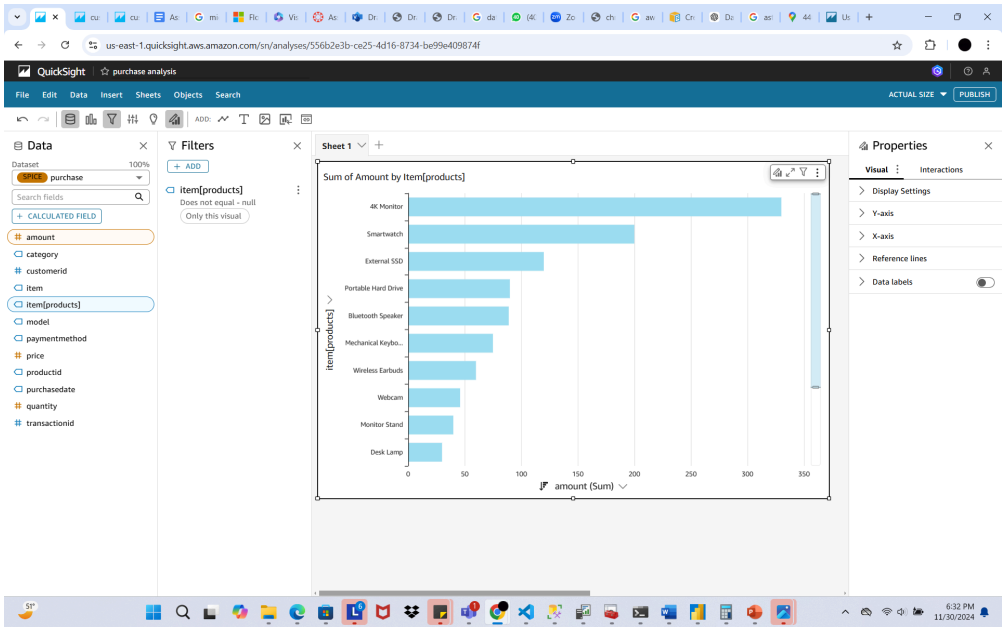


Line graph with purchase date on x axis and amount on y axis

This line chart shows the trend of purchase amounts over time, giving insight into sales performance for different dates. E-Shop can analyze these to identify high-revenue periods, such as during promotions, holidays, or product launches. This information will help optimize marketing efforts by focusing on dates with potential for higher sales. Similarly, low-revenue periods provide opportunities for targeted campaigns to drive customer engagement. Also, consistent analysis of sales trends can aid in inventory planning, ensuring stock availability during peak demand and reducing overstocking during slow periods.

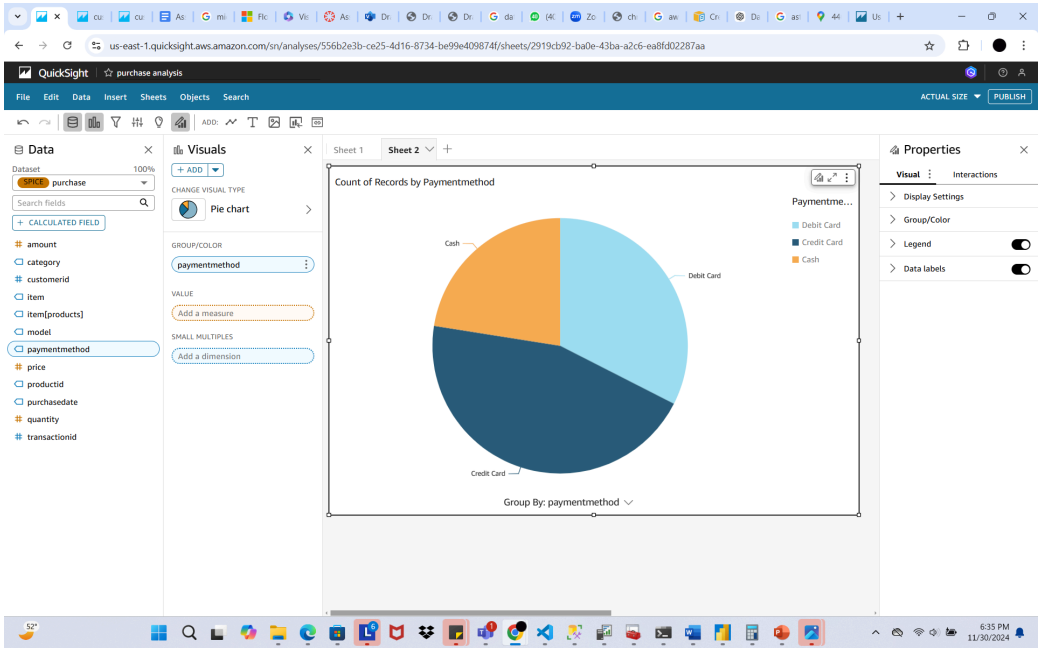


Horizontal Bar Graph showing products and amount sold:
The chart highlights the most and least popular products based on revenue. For instance, items like "4K Monitor" and "Smartwatch" generate the highest revenue, indicating their strong demand. E-Shop can prioritize stocking these items to ensure availability of high selling items and increase sales.



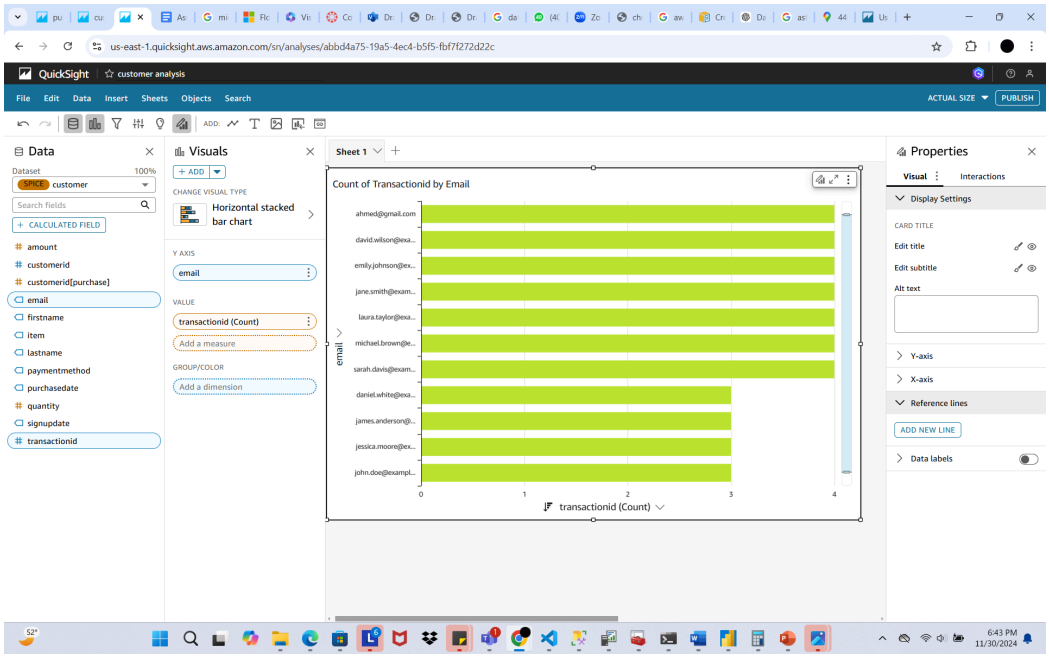
Pie chart depicting payment methods used in purchases:

The chart shows what proportion of transactions were completed using debit cards, credit cards, and cash. It helps E-Shop understand customer payment habits for better alignment with their preferred ways of paying. In case the share of debit and credit card settlements is higher, E-Shop can focus on enhancing its digital payment systems. In the case of high cash usage, it suggests improvement in cash handling procedures at physical stores or cash-on-delivery support for online orders.



Graph depicting transaction count with users(email address):

Through the monitoring of the transactions that are associated with particular email addresses, E-Shop can determine its top customers. This helps the business in identifying the most valuable customers and work on strategies to retain them such as loyalty programs, discounts, and exclusive offers. Using the active customer profiles (through email) E-Shop can enhance the effectiveness of its marketing strategies by creating targeted campaigns, sending out emails to encourage customers to make another purchase, or to promote specific products based on their interests or purchase history as well as to re-engage inactive customers with exclusive offers.


us-east-1.quicksight.aws.amazon.com/sn/analyses/abbd4a75-19a5-4ec4-b5f5-fbf7f272d22c
QuickSight
customer analysis
File Edit Data Insert Sheets Objects Search
ACTUAL SIZE
PUBLISH
Data
Dataset
100%
SPICE customer
Search fields
CALCULATED FIELD
amount
customerid
customerid[purchase]
email
firstname
item
lastname
paymentmethod
purchasedate
quantity
signupdate
transactionid
Visuals
ADD
CHANGE VISUAL TYPE
Horizontal stacked bar chart
Y AXIS
email
VALUE
transactionid (Count)
Add a measure
GROUP/COLOR
Add a dimension
Sheet 1
Count of Transactionid by Email
ahmed@gmail.com
david.wilson@exa...
emily.johnson@ex...
jane.smith@exam...
laura.taylor@exa...
michael.brown@e...
sarah.davis@exam...
daniel.white@exa...
james.anderson@...
jessica.moore@ex...
john.doe@exampl...
email
0
1
2
3
4
transactionid (Count)
Properties
Visual
Interactions
Display Settings
CARD TITLE
Edit title
Edit subtitle
Alt text
Y-axis
X-axis
Reference lines
ADD NEW LINE
Data labels
6:43 PM
11/30/2024