

Title: Build an Analytical Platform for eCommerce using AWS Services

B. Ajith Kumar Reddy - 2200080242



CLOUD SERVERLESS COMPUTING-22CEC3305A

Section 32

Under the guidance of

Dr. S.Kavitha

Abstract

The project, titled "Build an Analytical Platform for eCommerce using AWS Services," aims to develop a modern, cloud-based data analytics solution that enables eCommerce businesses to make faster, smarter, and more informed decisions. In today's highly competitive digital retail landscape, companies must continually analyze vast amounts of data to understand customer behavior, monitor sales performance, and optimize their operations. This platform utilizes a combination of **AWS services** to build a robust and scalable analytics workflow.

Data generated from various e-commerce activities—such as order transactions, customer clicks, product searches, inventory updates, and user reviews—is collected and stored in **Amazon S3**, serving as a cost-effective and durable data lake. Using **AWS Glue**, the raw data is automatically crawled, transformed, and cleaned through ETL (Extract, Transform, Load) jobs. The transformed data is then loaded into **Amazon Redshift**, a powerful data warehousing service that supports complex queries and large-scale data analytics.

To make the insights easily understandable and accessible, **Amazon QuickSight** is used for building dynamic dashboards and visual reports. These visualizations provide key metrics such as top-selling products, customer purchase patterns, sales trends, and regional performance, allowing stakeholders across marketing, finance, and operations teams to track and act on business performance in real time.

Additionally, **AWS Lambda** is used to trigger automation tasks within the pipeline—such as data updates or scheduled jobs—eliminating the need for dedicated servers and reducing operational overhead. The solution ensures high scalability, fault tolerance, and performance while keeping infrastructure management minimal.

Modules

1. Data Ingestion Module

This module is responsible for capturing and pushing e-commerce data into the analytical pipeline. Data is simulated using AWS CloudShell with Python scripts and Boto3 SDK to mimic customer activity such as browsing, cart additions, purchases, etc.

Tools Used: AWS CloudShell, Boto3, Amazon S3, Amazon Kinesis Data Streams

Functionalities:

- Simulates real-time transactional and behavioral data.
- Uploads static files (CSV) into S3 for batch analytics.
- Streams live data to Kinesis for real-time analytics

2. Real-Time Analytics and Stream Processing Module

This module handles real-time analytics using streaming services to process customer activity on-the-fly. It allows near-instant detection of user trends, fraudulent behavior, inventory depletion, and promotional responses.

Tools Used: Amazon Kinesis Data Analytics (Flink), AWS Lambda

Functionalities:

- Applies real-time filters, aggregations, and pattern detection using Apache Flink.
- Processes data instantly and invokes AWS Lambda functions.
- Helps trigger alerts, store analytics in DynamoDB, and update logs.

3. ETL and Batch Processing Module

In this module, larger datasets collected over time are cleaned, transformed, and stored in a more analytically friendly format for in-depth analysis. This enables long-term trend analysis, customer segmentation, and behavior insights.

Tools Used: AWS Glue, Amazon S3

Functionalities:

- AWS Glue performs automated schema detection and data transformation.
- Stores final datasets in formats like Parquet in S3 for efficient querying.

4. Storage and Query Module

Data, both structured and unstructured, is stored securely and is readily accessible for analysis. Different storage types support a range of queries from real-time lookups to batch reports.

Tools Used: Amazon S3 (raw + transformed), Amazon DynamoDB

Functionalities:

- Stores simulation logs, transaction data, and analytic output.
- Uses DynamoDB for fast key-value storage and quick retrievals.
- Acts as the backbone for reporting and visualization modules.

5. Monitoring, Logging & Alerting Module

This module ensures system health and performance are consistently monitored. It provides visibility into failures, usage spikes, or abnormal activities. It also integrates alerting mechanisms for real-time response.

Tools Used: Amazon CloudWatch, AWS Lambda, Amazon SNS

Functionalities:

- Lambda sends logs and metrics to CloudWatch for tracking.
- SNS sends notifications (SMS/email) to stakeholders upon defined triggers (e.g., fraud detection, system errors).
- Helps in audit, debugging, and performance tuning.

6. Reporting & Visualization Module

This module is the final stage of the pipeline where insights are visually represented. Business intelligence dashboards and charts are created to help decision-makers understand trends and patterns.

Tools Used: Amazon QuickSight

Functionalities:

- Visualizes transformed data for dashboards on revenue, customer behavior, sales performance, etc.
- Allows scheduled report generation and sharing with stakeholders.
- Enables self-service BI without the need for deep technical skills.

Event-Driven Architecture Overview

The platform architecture provides a complete end-to-end pipeline starting from data simulation to business analytics, all within a serverless, scalable AWS environment. It supports both real-time and batch analytics.

1. Data Generation and Ingestion

- The simulation starts in **AWS CloudShell**, where Python scripts using Boto3 simulate eCommerce activities like orders, product views, and payments.
- Data is directed into:
 - **Amazon S3** for CSV-based batch data.
 - **Amazon Kinesis Data Streams** for continuous real-time ingestion.

2. Stream Processing & Analytics

- **Amazon Kinesis Data Analytics (Flink)** processes real-time data from Kinesis Streams.
- Flink jobs apply transformations (like windowed aggregations and pattern detection).
- Output is sent to:
 - **AWS Lambda** for further logic execution.
 - Lambda functions write to:
 - **Amazon DynamoDB** for structured storage.
 - **Amazon CloudWatch** for logs and monitoring.
 - **Amazon SNS** to notify about specific events (e.g., purchase spike, suspicious activity).

3. Batch Pipeline and ETL

- Raw streaming data is also buffered by **Kinesis Firehose**, which stores it in Amazon S3.
- **AWS Glue** crawls the S3 bucket, performs ETL, and writes transformed data in Parquet format into a separate S3 bucket for analytics.

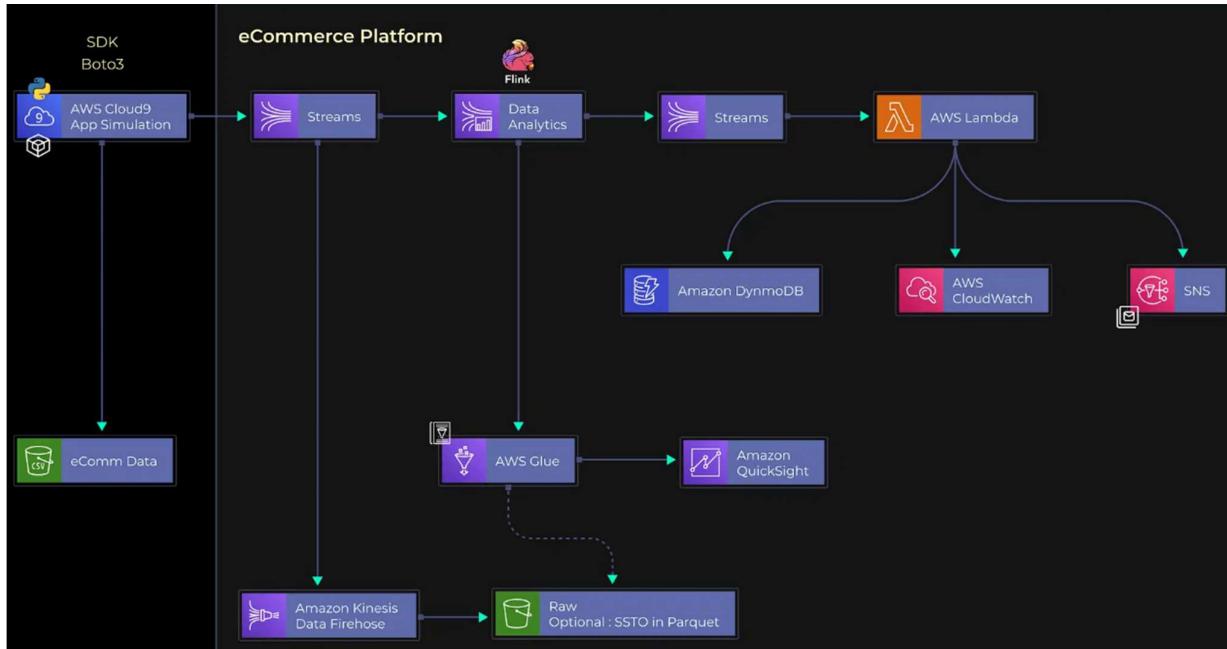
4. Business Intelligence & Visualization

- The final transformed data in S3 is connected to **Amazon QuickSight**.
- QuickSight dashboards present interactive charts and reports on key business KPIs:
 - Sales trends
 - Customer engagement
 - Inventory performance
 - Conversion rates

5. Key Architectural Features

- **Fully Serverless**: No manual provisioning; scalable architecture.
- **Real-Time + Batch Integration**: Handles both high-velocity streams and bulk data.
- **Secure and Reliable**: Uses AWS IAM, encryption (SSE), and CloudWatch for logging.
- **Cost-Effective**: Pay-as-you-go model; uses managed services like Glue and QuickSight

Architecture diagram



Services Used in the Project:

1. Amazon Kinesis Data Streams

Used to ingest high-speed streaming data from eCommerce platforms, such as user clicks, product views, and transactions. It enables real-time data collection and further processing.

Example: Capturing user browsing behavior as soon as it happens on the platform.

2. Amazon Kinesis Data Analytics

This service is used to analyze and transform streaming data on the fly using SQL. It processes incoming data from Kinesis Streams and extracts meaningful insights, such as top-selling products or spikes in user traffic.

Example: Detecting sudden increases in demand for a product in real time.

3. AWS CloudShell

A browser-based terminal used to write and execute scripts for simulating real-time data generation using AWS CLI and Python. This eliminated the need for EC2 or S3-triggered uploads during development.

Used for convenience to simulate data instead of deploying separate compute resources.

4. AWS Lambda

Lambda is triggered by Kinesis Analytics to act on processed data. It helps route the analyzed data to storage or notification systems, allowing for event-driven automation without provisioning servers.

Example: Automatically inserting processed data into DynamoDB or logging alerts to CloudWatch.

5. Amazon DynamoDB

A fast and flexible NoSQL database service used to store real-time metrics such as user activity stats, order counts, or inventory summaries. Ideal for quick reads/writes in milliseconds.

Enables rapid querying of up-to-date analytics for the admin panel or monitoring.

6. Amazon CloudWatch

Used for monitoring system behavior, collecting logs, and setting up alarms. It plays a crucial role in observing data flow and troubleshooting errors within Lambda, Kinesis, and Glue.

Example: Alerting if the incoming data rate exceeds the threshold or if Lambda fails.

7. Amazon S3 (Simple Storage Service)

Used for storing batch data files in formats like CSV or JSON. These files represent historical sales records, customer databases, or transaction logs.

Acts as the central storage for offline analytics and large datasets.

8. AWS Glue

A serverless data integration service used to crawl, catalog, and prepare batch data stored in S3. It helps define a unified schema, making the data ready for querying via Athena or visualization via QuickSight.

It bridges S3 storage and Athena querying by creating a data catalog.

9. Amazon Athena

Used to run SQL queries directly on S3 data without the need to set up a traditional database. It provides fast, serverless querying capability for batch analytics.

Example: Running queries to identify monthly top-performing product categories.

10. Amazon QuickSight

A powerful business intelligence tool used to create dashboards and visualizations for stakeholders. It pulls data from Athena or DynamoDB to present interactive insights.

Final visual output to the end-user – showing trends, patterns, and predictions.

Lambda Function Code:

```
import base64
import json
import boto3
from decimal import Decimal # Add this import

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('EcommOrders')

sns = boto3.client('sns')
SNS_TOPIC_ARN = 'arn:aws:sns:your-region:your-account-id:ecomm-alerts' # Replace if using alerts

def lambda_handler(event, context):
    for record in event['Records']:
        payload =
base64.b64decode(record['kinesis']['data']).decode('utf-8')
        data = json.loads(payload)

        if not data.get('InvoiceNo'):
            continue

        try:
            item = {
                'InvoiceNo': str(data.get('InvoiceNo')),
                'StockCode': str(data.get('StockCode', "")),
                'Description': str(data.get('Description', "")),
                'Quantity': int(float(data.get('Quantity', 0))),
                'InvoiceDate': str(data.get('InvoiceDate', "")),
                'UnitPrice': Decimal(str(data.get('UnitPrice', 0.0))), #
Use Decimal
                'CustomerID': str(data.get('CustomerID', "")),
                'Country': str(data.get('Country', ""))
            }

            table.put_item(Item=item)

            if item['UnitPrice'] > Decimal('1000'):
                sns.publish(
```

```
TopicArn=SNS_TOPIC_ARN,
Message=f"High value order detected:
{item['InvoiceNo']}",
Subject="Alert: High Order"
)

except Exception as e:
    print(f"Error inserting: {e}")
```

Process and Screenshots

➤ Upload Dataset to S3

- Uploaded the Online Retail.csv file to an S3 bucket (ecomm-data-bucket-1) for storage and access.

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Amazon S3', 'General purpose buckets', 'Storage Lens', and 'AWS Marketplace for S3'. The main area displays the 'ecomm-data-bucket-1' bucket. It shows one object named 'manifest.json' which is a json file last modified on April 6, 2025, at 12:07:56 UTC+05:30, with a size of 265.0 B and a storage class of Standard. There's also a folder named 'online-retail/'. At the top right, there are buttons for 'Actions' (with options like Copy 53 URI, Copy URL, Download, Open, Delete, Create folder, and Upload), and a search bar. The URL in the browser is <https://us-east-1.console.aws.amazon.com/s3/buckets/ecomm-data-bucket-1?region=us-east-1&bucketType=general>.

➤ Catalog Data using AWS Glue

- Created a Glue Crawler to scan the S3 data and catalog it into the AWS Glue Data Catalog for querying.

The screenshot shows the AWS Glue console. The left sidebar includes sections for 'AWS Glue', 'Data Catalog', and 'Data Integration and ETL'. The main content area is titled 'Crawlers' and describes what it does: 'A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.' It shows a table of crawlers with one entry: 'ecomm-crawler' which is 'Ready' and has 'Succeeded' in the 'Last run' column, with the last run timestamp being April 6, 2025, at 14:17:40. There are buttons for 'Action' (with options like Run, Create crawler, and Delete), 'Run', and 'Create crawler'. The URL in the browser is <https://us-east-1.console.aws.amazon.com/glue/home?region=us-east-1#/v2/data-catalog/crawlers>.

- create database and table while creating the crawler and select the s3 bucket correctly

The screenshot shows the AWS Glue Data Catalog Tables page. On the left, there's a navigation sidebar with sections like AWS Glue, Data Catalog tables, Data Catalog, Data Integration and ETL, and Zero-ETL Integrations. The main content area is titled 'Tables' and shows a table named 'online_retail' in the 'ecomm_db' database. The table is located at 's3://ecomm-data-bucket-1' and is defined as 'CSV'. There are buttons for 'Delete', 'Add tables using crawler', and 'Add table'.

#	Column name	Data type	Partition key	Comment
1	invoiceno	string	-	-
2	stockcode	string	-	-
3	description	string	-	-
4	quantity	bigint	-	-
5	invoicedate	string	-	-
6	unitprice	double	-	-
7	customerid	bigint	-	-
8	country	string	-	-

➤ Query Data with Amazon Athena

- Used Athena to run SQL queries on the data directly from S3 for batch data analysis.
- Open the table and click on the action-> view data

The screenshot shows the Amazon Athena Query Editor interface. At the top, a query is written in SQL:

```
1 SELECT * FROM "AwsDataCatalog"."ecomm_db"."online_retail_csv" limit 10;
```

Below the query, the status bar indicates "SQL Ln 1, Col 1".

Control buttons include "Run again", "Explain", "Cancel", "Clear", and "Create". A note says "Reuse query results up to 60 minutes ago".

The "Query results" tab is selected, showing a green "Completed" status bar with "Time in queue: 109 ms", "Run time: 344 ms", and "Data scanned: 1.41 MB".

The results table has one row under "Results (0)".

At the bottom, there's a search bar, a footer with copyright information, and links for "Privacy", "Terms", and "Cookie preferences".

- After clicking on the run you can see the results

The screenshot shows the Amazon Athena Query Editor interface with the results of the previous query.

The results table displays 10 rows of data from the "online_retail" table:

#	invoiceno	stockcode	description	quantity	invoicedate	unitprice	customerid	country
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom
3	536365	84069	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850	United Kingdom
7	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850	United Kingdom
8	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850	United Kingdom
9	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850	United Kingdom
10	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047	United Kingdom

➤ Set up the cloudshell

- uploaded the dataset to AWS CloudShell and used a Python script with boto3 to send data to the Kinesis Data Stream.
The script read the CSV, converted rows to JSON or string, and streamed them into Kinesis in real-time.

```

CloudShell
Actions ▾
us-east-1 + simulate_stream.py Modified

File Name to Write: simulate_stream.py
^G Help
^C Cancel
M-D DOS Format
M-H Mac Format
M-A Append
M-P Prepend
M-B Backup File
^T Browse

# CloudShell
us-east-1

GNU nano 5.8
import boto3
import pandas as pd
import json
import time

df = pd.read_csv("OnlineRetail.csv") # CSV file must be in same directory
kinesis = boto3.client('kinesis', region_name='us-east-1') # Update region if needed

for _, row in df.iterrows():
    record = row.to_dict()
    clean = {k: v for k, v in record.items() if not pd.isnull(v) or k in record.items()}
    kinesis.put_record(
        StreamName='ecommerce-stream', # Ensure this stream exists
        Data=json.dumps(clean),
        PartitionKey='1'
    )
    print("Sent:", clean)
    time.sleep(0.5)

```

- You can see the uploaded data in the kinesis stream using the Cloud Shell

```

CloudShell
us-east-1 + simulate_stream.py

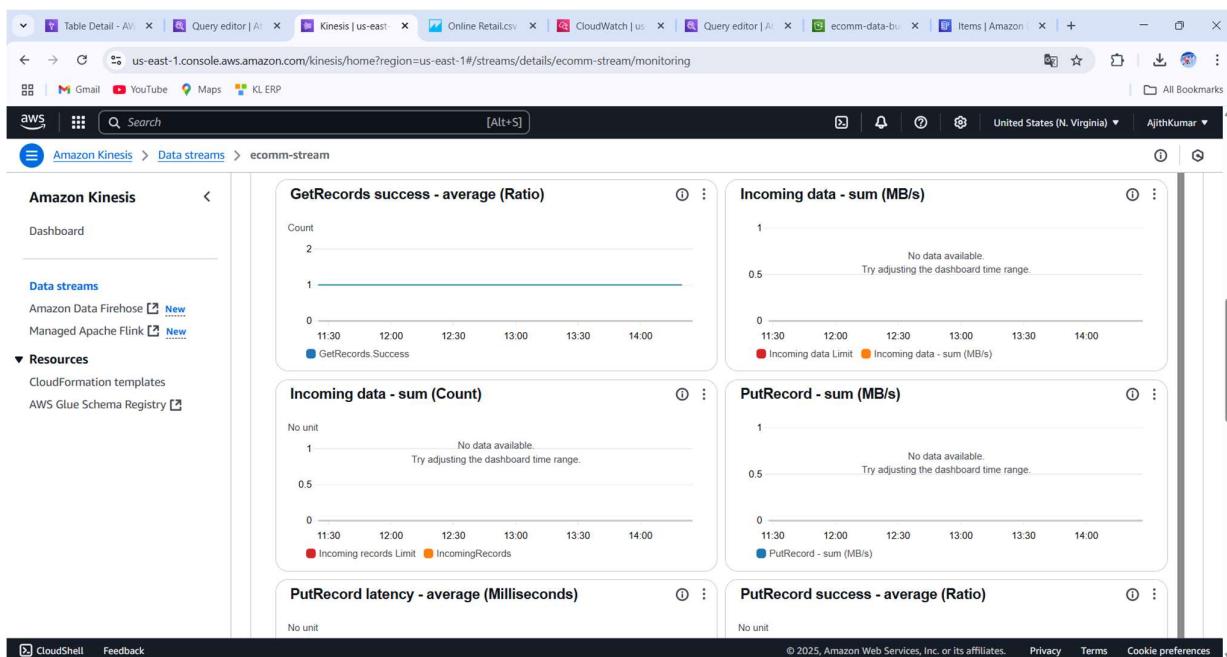
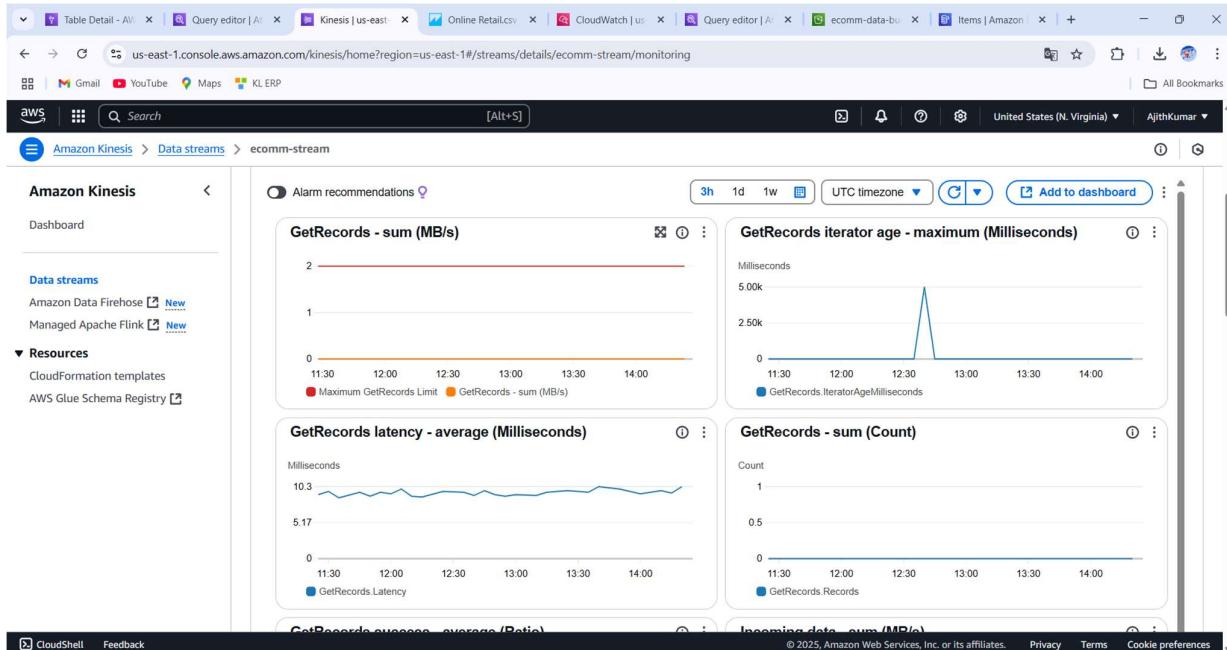
FileNotFoundError: [Errno 2] No such file or directory: 'OnlineRetail.csv'
~ $ nano simulate_stream.py
~ $ python3 simulate_stream.py
Sent: {'InvoiceNo': '536365', 'StockCode': '85123A', 'Description': 'WHITE HANGING HEART T-LIGHT HOLDER', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 2.55, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '27252', 'Description': 'WHITE METAL LANTERN', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 3.39, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '84406R', 'Description': 'CREAM CUPID HEARTS COAT HANGER', 'Quantity': 8, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 2.75, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '84029C', 'Description': 'KNITTED UNION FLAG HOTTE BOTTLE', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 3.39, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '84029E', 'Description': 'RED WOOLY HOTTE WHITE HEART', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 3.39, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '22752', 'Description': 'SET 7 BABUSHKA NESTING BOXES', 'Quantity': 2, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 7.65, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '21730', 'Description': 'GLASS STAR FROSTED T-LIGHT HOLDER', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 4.25, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '22753', 'Description': 'ASSORTED COLOURED GLASS T-LIGHT HOLDERS', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:26', 'UnitPrice': 4.25, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536365', 'StockCode': '22632', 'Description': 'HAND WARMER RED POLKA DOT', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:28', 'UnitPrice': 1.85, 'CustomerID': 17850.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '84879', 'Description': 'ASSORTED COLOUR BIRD ORNAMENT', 'Quantity': 32, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 1.69, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22745', 'Description': 'POPPY'S PLAYHOUSE BEDROOM', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 2.1, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22748', 'Description': 'POPPY'S PLAYHOUSE KITCHEN', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 2.1, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22750', 'Description': 'FIREPLACE FIREWOOD LOGS', 'Quantity': 8, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 3.75, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22310', 'Description': 'EVERY KNITTED MUG COSY', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 1.65, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '84969', 'Description': 'BOX OF 6 ASSORTED COLOUR TEASPOONS', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.25, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22623', 'Description': 'BOX OF VINTAGE JIGSAW BLOCKS', 'Quantity': 3, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '22754', 'Description': 'HOME BUILDING BLOCK WORD', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 5.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '21755', 'Description': 'LOVE BUILDING BLOCK WORD', 'Quantity': 3, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 5.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '21777', 'Description': 'RECIPE BOX WITH METAL HEART', 'Quantity': 4, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 7.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536367', 'StockCode': '48187', 'Description': 'DOORMAT NEW ENGLAND', 'Quantity': 4, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 7.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536368', 'StockCode': '22960', 'Description': 'JAM MAKING SET WITH JARS', 'Quantity': 6, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.25, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536368', 'StockCode': '22913', 'Description': 'RED COAT RACK PARIS FASHION', 'Quantity': 3, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536368', 'StockCode': '22912', 'Description': 'YELLOW COAT RACK PARIS FASHION', 'Quantity': 3, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'}
Sent: {'InvoiceNo': '536368', 'StockCode': '22914', 'Description': 'BLUE COAT RACK PARIS FASHION', 'Quantity': 3, 'InvoiceDate': '12/1/2010 8:34', 'UnitPrice': 4.95, 'CustomerID': 13047.0, 'Country': 'United Kingdom'

File "/home/cloudshell-user/simulate_stream.py", line 19, in <module>
    time.sleep(0.5)
KeyboardInterrupt
~ $ 

```

➤ Set Up Real-Time Ingestion with Kinesis Data Streams

- Streamed eCommerce events (e.g., product views, purchases) into Kinesis for real-time processing.



➤ Process Live Data with Kinesis Data Analytics (Flink)

- Built a Flink-based app to process and analyze streaming data for real-time dashboards and insights.

The screenshot shows the AWS CloudFormation blueprint creation interface for a Kinesis Data Analytics (Flink) application. At the top, there's a navigation bar with tabs like 'Table Detail', 'Query editor', 'Kinesis', 'Create stream', 'Online Retail', 'CloudWatch', 'Query editor', 'ecomm-data', 'Items', and 'YouTube'. Below the navigation is a search bar and a user profile for 'AjithKumar'.

The main content area has a diagram showing the flow from 'Kinesis data stream' to 'Amazon Managed Service for Apache Flink' which then processes the data into an 'S3 bucket'. A note states: 'Flink reads the data from the Kinesis Data Stream and processes it.'

A large box labeled 'Basic information' contains fields for 'Application name' (set to 'flink-demo-1743950585019'), 'Apache Flink version' (set to 'Apache Flink 1.20'), 'Amazon Kinesis Data Streams source' (set to 'flink-demo-1743950585019-stream'), and 'S3 bucket destination' (set to 'flink-demo-008971663175-us-east-1-1743950585019-bucket').

Below this is a section for deployment details, showing two options: 'flink-demo-bootstrap-1743950585019-stack' and 'flink-demo-1743950585019-stack'. A note says: 'The process takes about 10 minutes. If you reload this tab or navigate away your blueprint might not deploy properly. You can continue to use the console in a new tab.'

At the bottom right are buttons for 'Cancel', 'Deploy blueprint', and links to 'CloudShell', 'Feedback', and 'Cookie preferences'.

This screenshot shows the same AWS CloudFormation blueprint creation interface, but with a different configuration. The 'Create from scratch' button is now selected, while the 'Use a blueprint' button is unselected.

The 'Apache Flink configuration' section includes a note: 'To run a Python Managed Service for Apache Flink application, specify your code files by configuring your application properties after creating your application.' A link to 'Learn more' is provided.

The 'Application configuration' section includes fields for 'Application name' (set to 'ecomm-analytics-app') and 'Description - optional' (with a placeholder 'Enter description').

The 'Access to application resources' section includes a note: 'Create or choose IAM role with the required permissions. Learn more' and three options: 'Create / update IAM role kinesis-analytics-ecomm-analytics-app-us-east-1 with required policies' (selected), 'Choose from IAM roles that Managed Service for Apache Flink can assume' (unselected), and 'Create or choose IAM role with the required permissions' (unselected).

At the bottom right are buttons for 'Cancel', 'Deploy blueprint', and links to 'CloudShell', 'Feedback', and 'Cookie preferences'.

The screenshot shows the AWS Managed Apache Flink application configuration interface. On the left, there's a sidebar with links like 'Managed Apache Flink' (selected), 'Dashboard', 'Apache Flink applications', 'Resources' (with 'What's new', 'Streaming data solution using Kinesis Data Streams', 'Streaming data solution using Amazon MSK', 'Release versions', and 'In-place version upgrade'), and 'Customer survey'. The main content area has three steps: Step 1 'Configure data source' (with a note about using a custom data source or Kinesis Data Stream), Step 2 'Configure application to upload code' (with a note about building application code to point to a source stream and upload to an S3 bucket), and Step 3 'View application code in Amazon S3' (with a note about viewing application code in Amazon S3). Below these steps is an 'Application details' section with fields for 'Status' (Ready), 'ARN' (arn:aws:kinesisanalytics:us-east-1:008971663175:application:ecomm-analytics-app), 'Runtime' (Apache Flink 1.20), 'IAM role' (kinesis-analytics-ecomm-analytics-app-us-east-1), 'Last updated' (April 06, 2025, 20:08 GMT+5:30), and 'Description' (empty). At the bottom, tabs for 'Monitoring', 'Snapshots', 'Configuration' (selected), 'Tags', 'Versions', and 'Operations - new' are visible. A 'Code location and runtime properties' section follows, with a note about destination for code in Amazon S3. Finally, a 'Runtime properties (0)' section is shown.

The screenshot shows the Apache Flink Dashboard interface. The left sidebar has sections for 'Overview', 'Jobs' (selected), 'Running Jobs' (selected), 'Completed Jobs', 'Task Managers', and 'Job Manager'. The main content area is titled 'Running Jobs' and displays a table with one row:

Job Name	Start Time	Duration	End Time	Tasks	Status
Kinesis Data Streams to S3 Flink Streaming App	2025-04-06 20:18:51.177	2m 3s 681ms	-	4 / 4	RUNNING

At the top right, it says 'Version: 1.20.0' and 'Message: 0'.

➤ Store Processed Events in DynamoDB

- Streamed important transactional data to DynamoDB for persistent, low-latency access.

InvoiceNo	Country	CustomerID	Description	InvoiceDate	Quantity	StockCode	UnitPrice
536365	United Kingdom	17850.0	GLASS STAR F...	12/1/2010 8:26	6	21730	4.25
536368	United Kingdom	13047.0	BLUE COAT R...	12/1/2010 8:34	3	22914	4.95
536369	United Kingdom	13047.0	BATH BUILDING...	12/1/2010 8:35	3	21756	5.95
536367	United Kingdom	13047.0	DOORMAT NEW...	12/1/2010 8:34	4	48187	7.95
536366	United Kingdom	17850.0	HAND WARMER...	12/1/2010 8:28	6	22632	1.85
511030	Germany	17920	Charms Race	2010-08-01 10:00:00	2	81487	19.3

➤ Trigger Notifications using AWS Lambda + SNS

- Configured Lambda functions to process DynamoDB streams and send alerts via Amazon SNS.
- Add the trigger to the kinesis

The screenshot shows the AWS Lambda Function Overview page for the "ProcessEcommStream" function. The function is triggered by a Kinesis stream. The "Code source" tab is selected, showing the Python code in "lambda_function.py". The code is as follows:

```
EXPLORER
PROCESSCOMMSTREAM
lambda_function.py
```

```

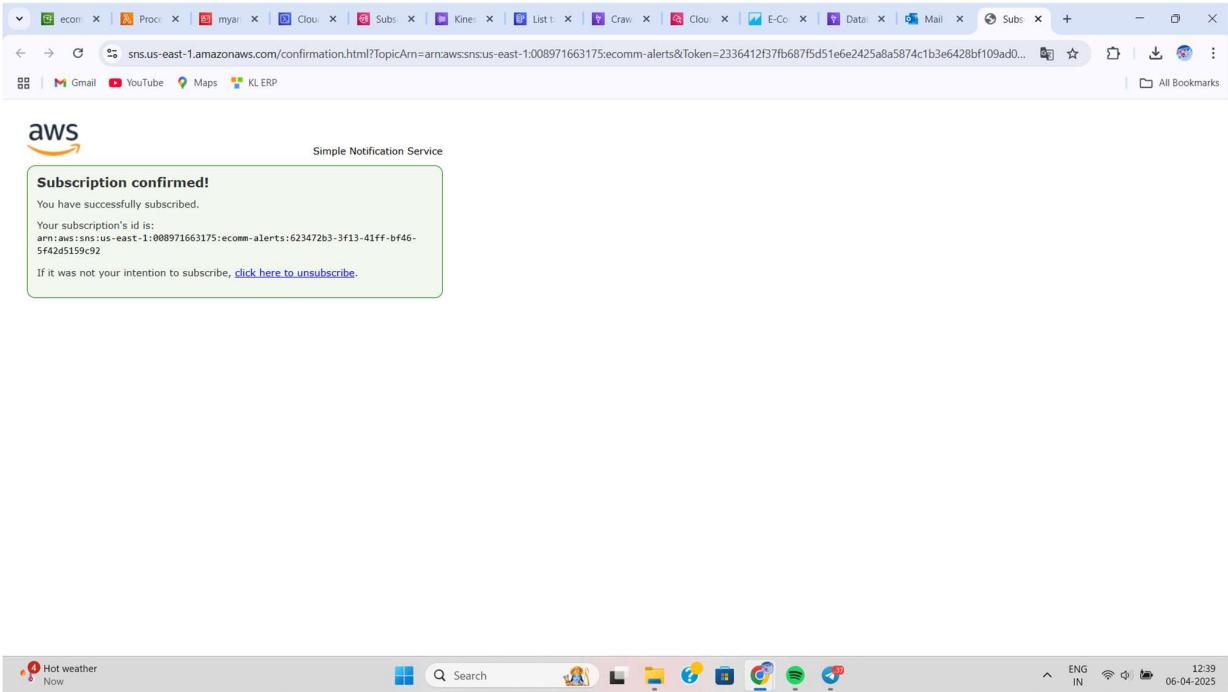
 10     SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:008971663175:ecomm-alerts' # Replace if using alerts
 11
 12     def lambda_handler(event, context):
 13         for record in event['Records']:
 14             payload = base64.b64decode(record['kinesis']['data']).decode('utf-8')
 15             data = json.loads(payload)
 16
 17             if not data.get('InvoiceNo'):
 18                 continue
 19
 20             try:
 21                 item = {
 22                     'InvoiceNo': str(data.get('InvoiceNo')),
 23                     'StockCode': str(data.get('StockCode', '')),
 24                     'Description': str(data.get('Description', '')),
 25                     'Quantity': Decimal(str(data.get('Quantity', 0))),
 26                     'InvoiceDate': str(data.get('InvoiceDate', '')),
 27                     'UnitPrice': Decimal(str(data.get('UnitPrice', 0.0))), # Use Decimal
 28                     'CustomerID': str(data.get('CustomerID', '')),
 29                 }
 30
 31             except Exception as e:
 32                 print(f"Error processing item: {e}")
 33
 34             # Add item to the queue
 35             sqs_client.send_message(QueueUrl=SQS_QUEUE_URL, MessageBody=json.dumps(item))
 36
 37     return {
 38         'statusCode': 200,
 39         'body': 'Success'
 40     }

```

- Create a SNS topic and add the protocol type to the email

Name	Type	ARN
ecomm-alerts	Standard	arn:aws:sns:us-east-1:008971663175:ecomm-alerts

- Confirm the subscription



Visualize Results / Monitor Outputs

- Analyzed output files from the **Flink output S3 bucket**, checked streaming logs, visualized data using **Amazon QuickSight**
- Flink send the output to another bucket

Amazon S3

flink-demo-008971663175-us-east-1-1743950585019-bucket

Objects (3)

Name	Type	Last modified	Size	Storage class
app-mdf-kafka-to-s3job_start-1743950921801/	Folder	-	-	-
kds-to-s3-datastream-java-1.0.1.jar	jar	April 6, 2025, 20:14:23 (UTC+05:30)	192.6 MB	Standard
kds-to-s3-datastream-java.json	json	April 6, 2025, 20:14:38 (UTC+05:30)	37.0 KB	Standard

• Checking CloudWatch logs

CloudWatch

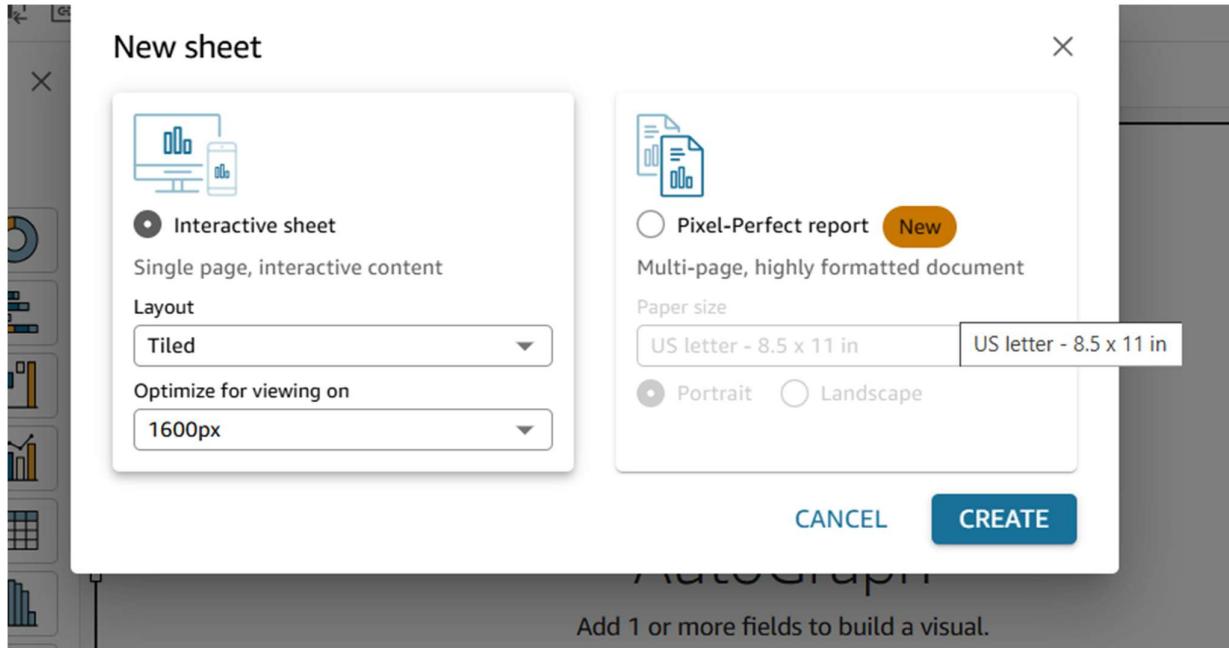
Log groups (23)

Log group	Log class	Anomaly d...	Data protection	Sensitive data co...	Retention	Metric filters	Contributor Ins...
/aws/dynamodb/imports	Standard	Configure	-	-	3 months	-	-
/aws/glue/crawlers	Standard	Configure	-	-	Never expire	-	-
/aws/kinesis-analytics/ecommerce-analytics-app	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/80242function	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/80242function234	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/80242functionjhdfshgdfhasdgh	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/PowerCalculator	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/ProcessEcommStream	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/chatbotcreating	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/flink-demo-1743950585019-Deployan...	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/flink-demo-1743950585019-Singleton...	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/flink-demo-1743950585019-StarthSF...	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/flink-demo-bootstrap-1743-SingletonL...	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/hvhdjdhvhvhvbjhvhv80242	Standard	Configure	-	-	Never expire	-	-

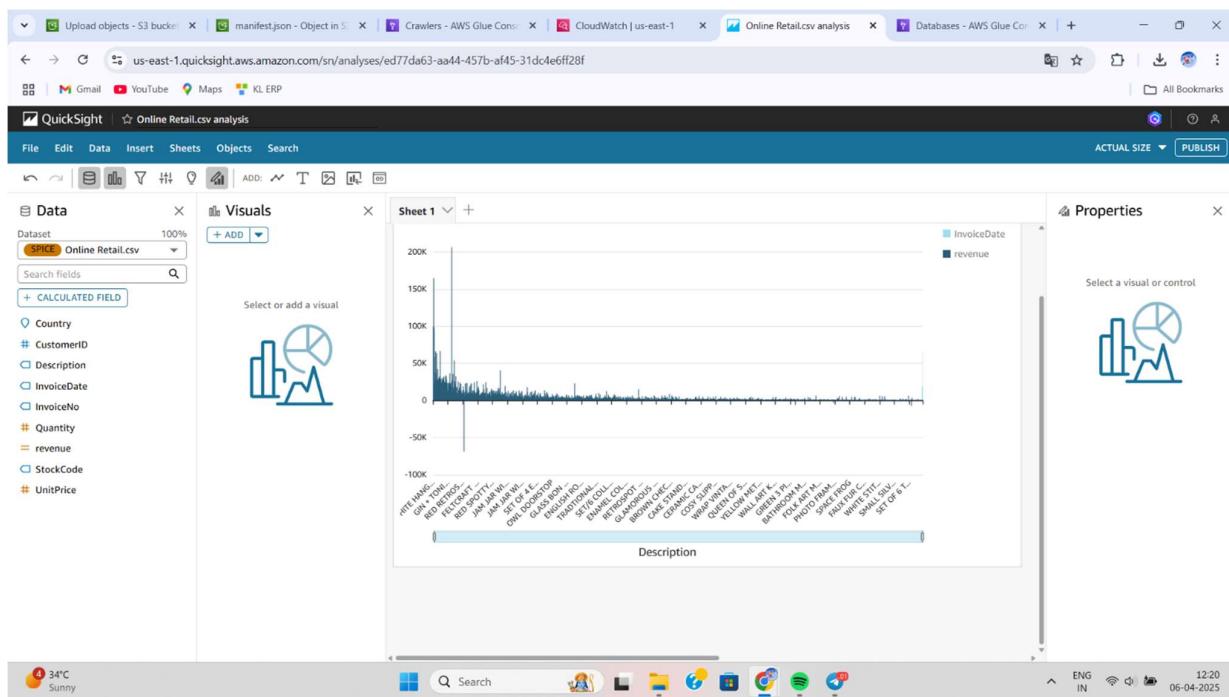
- Accessed Amazon QuickSight to visualize the streaming output data stored in the S3 bucket by connecting to the existing dataset, enabling insightful analysis and real-time dashboards

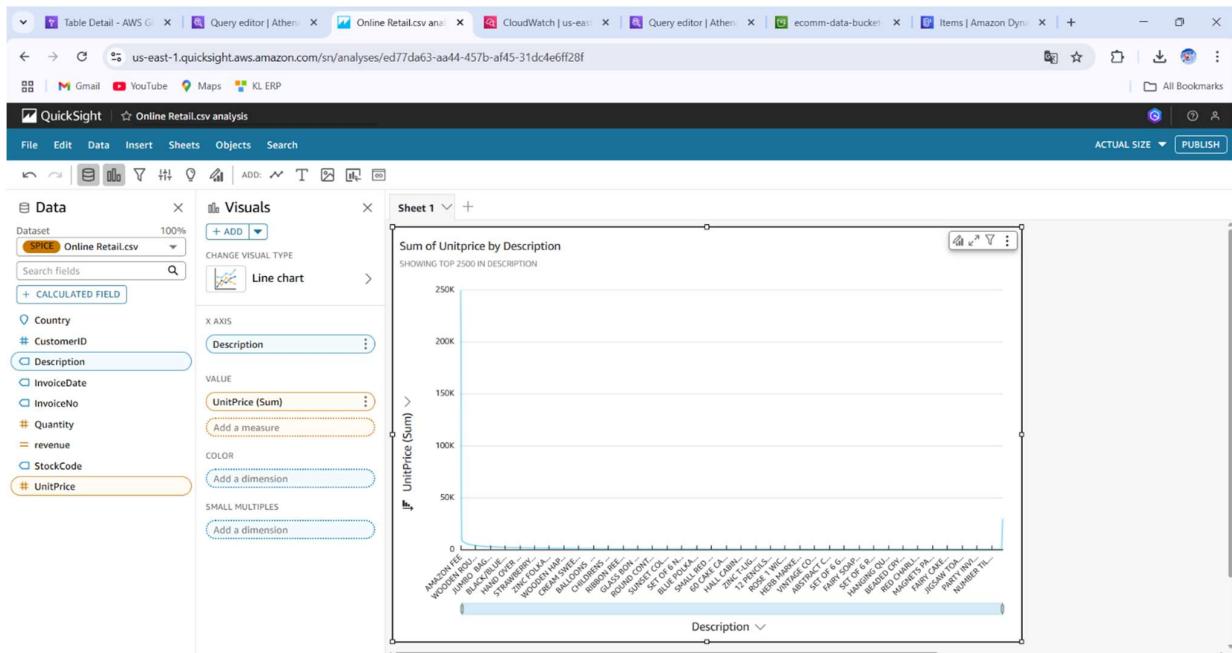
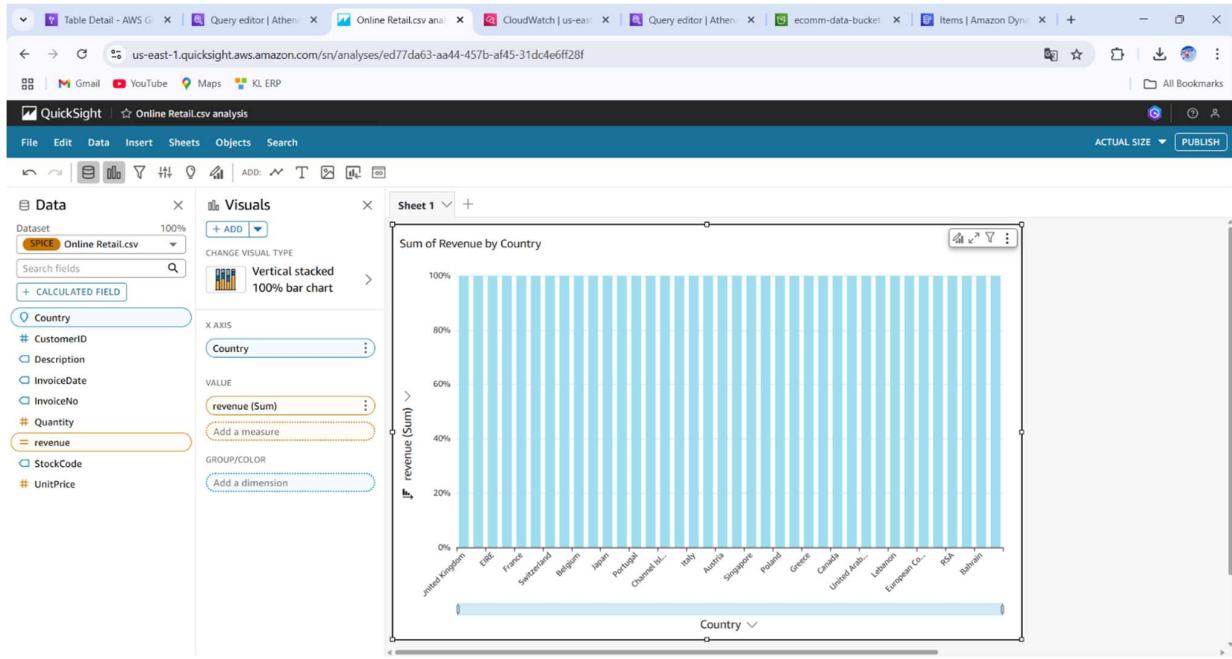
- Click on the new analysis and click on the new dataset, select the S3 URL and upload it

- Create an interactive sheet



- Now u can visualize and change the types like bar chart, vertical bar, line graph etc After that click on the publish, and it will create a dashboard





Conclusion

In this project, we successfully designed and implemented a scalable, secure, and highly available analytical platform tailored for an e-commerce environment using AWS services. The primary goal was to simulate and analyze user activity data such as product views, cart events, and purchase histories, in both batch and real-time scenarios, enabling powerful insights and timely decision-making.

We started by ingesting historical eCommerce data into Amazon S3 and used AWS Glue to catalog the dataset and make it queryable via Amazon Athena. Glue Crawlers automated schema detection, while Athena provided serverless SQL querying capabilities for batch analytics.

For real-time data processing, Amazon Kinesis Data Streams was integrated to simulate live clickstream data. The stream was further analyzed in real-time using Kinesis Data Analytics, powered by Apache Flink, enabling near-instant insights and alert generation.

Additionally, AWS Lambda and DynamoDB Streams were leveraged to build a reactive notification system. As new orders were inserted into DynamoDB, Lambda functions triggered SNS (Simple Notification Service) to alert stakeholders or services about customer actions. This event-driven architecture demonstrated how microservices can react to business events without polling or delays.

To visualize and gain business insights from both batch and streaming outputs, **Amazon QuickSight** was used to build interactive dashboards by connecting directly to Athena queries and the output data stored in Amazon S3. This provided rich, real-time analytics and user-friendly visual reports for stakeholders.

Throughout the pipeline, security and best practices were maintained using IAM roles, S3 bucket policies, and fine-

grained access controls, ensuring data privacy and safe access to resources.

Ultimately, this project demonstrates how cloud-native services on AWS can be combined to build a powerful big data pipeline for eCommerce analytics, covering everything from data ingestion and transformation to real-time processing, visualization, querying, and notification. This end-to-end platform not only enables deep insights into customer behavior but also lays the foundation for predictive analytics, personalized marketing, inventory optimization, and better customer service – all critical for the success of modern digital businesses.