Serverless Data Analytics Pipeline Using AWS

Sem End Project Report

Department of Computer Science and Engineering

By

2200080221-M.Rehan Haneef

CLOUD & SERVERLESS COMPUTING-22CEC3305P

Section - **31**

Under the guidance of

Dr.K.V.Raviteja

Koneru Lakshmaiah Education Foundation

(Deemed to be University estd., u/s 3 of UGC Act 1956)

Green Fields, Vaddeswaram, Guntur (Dist.), Andhra Pradesh - 522302

December, 2024

CONTENTS

LIST OF FIGURES

Abstract

The need for real-time data analytics has become critical across various industries, enabling businesses to make faster and more informed decisions. Traditional infrastructure-based data pipelines often struggle with scalability, maintenance overhead, and cost efficiency. To address these challenges, this project explores the development of a serverless real-time analytics pipeline on AWS, leveraging fully managed services that eliminate infrastructure management while ensuring high availability, scalability, and cost optimization. The architecture integrates Amazon Kinesis Data Streams to ingest real-time data from sources such as IoT devices, logs, or stock market feeds. Data is then processed and transformed using AWS Lambda and AWS Glue, ensuring efficient and automated ETL (Extract, Transform, Load) operations. The processed data is stored in Amazon S3 for scalable object storage and Amazon DynamoDB for low-latency lookups. Real-time querying is enabled through Amazon Athena, allowing SQL-based analysis of the ingested data without requiring dedicated database infrastructure. Finally, insights are visualized using Amazon QuickSight, providing interactive dashboards for business intelligence and decision-making. This project demonstrates how AWS serverless technologies can be combined to build an end to-end real-time analytics solution that is cost-effective, highly available, and scalable. The pipeline is ideal for various use cases, including log monitoring, fraud detection, IoT data processing, stock market trend analysis, and customer behavior tracking. By eliminating the need for traditional servers and databases, this solution significantly reduces operational complexity while ensuring seamless data processing and analytics in real-time.

Introduction

In the modern digital age, organizations are continuously seeking solutions that allow them to process and analyze data in real-time to drive timely insights and decisions. With the increasing velocity, volume, and variety of data generated every second, traditional data processing models often struggle to scale efficiently without incurring high operational overhead. To address this challenge, serverless computing has emerged as a transformative approach that allows developers to build and deploy applications without the burden of managing underlying infrastructure.

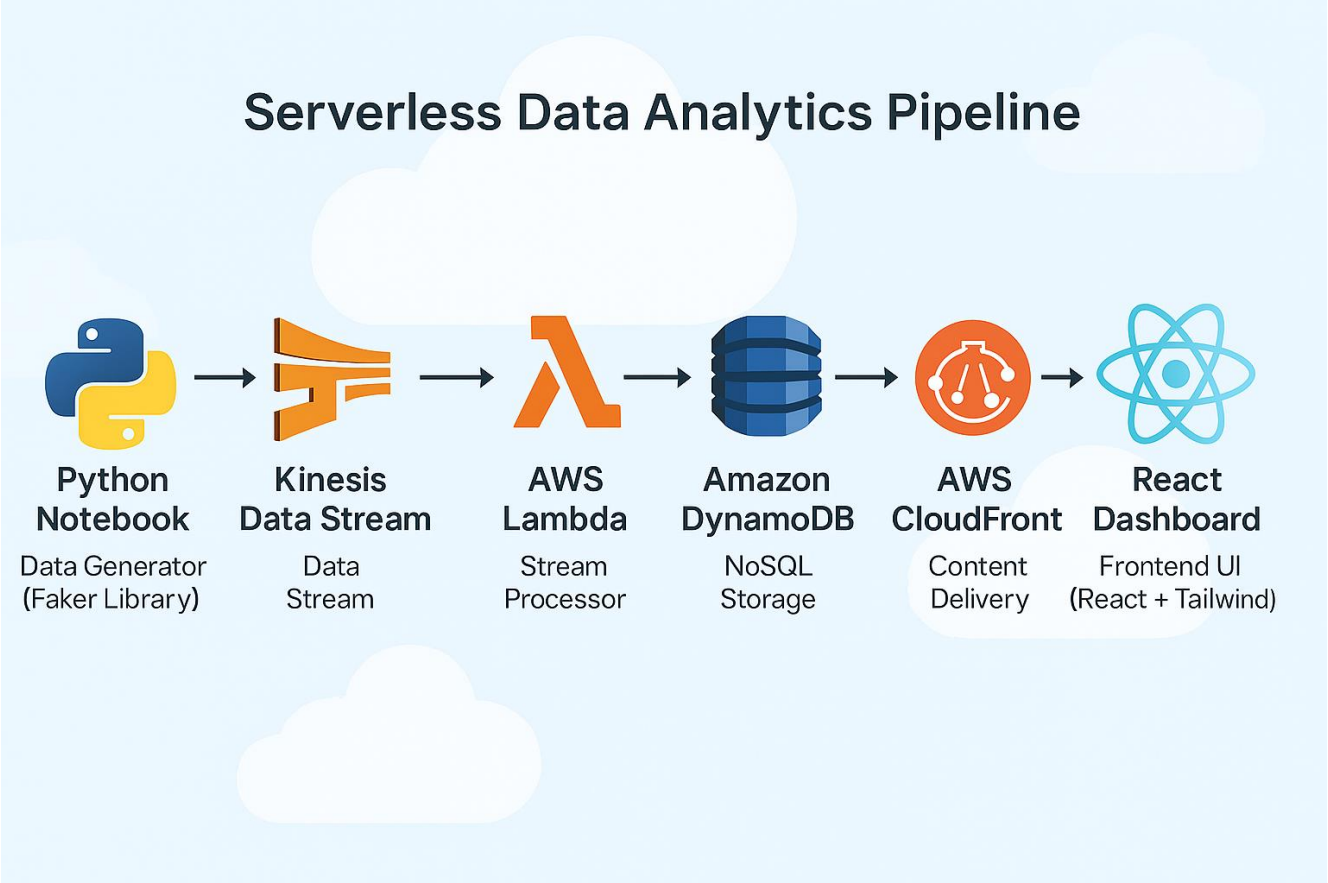
This project titled **"Serverless Data Analytics Pipeline using AWS and ReactJS"** was developed as a part of the academic curriculum for the course **Cloud and Serverless Computing**. The project demonstrates how to architect and implement a real-time analytics pipeline using a fully serverless model on **Amazon Web Services (AWS)**. The aim is to simulate a real-world data streaming scenario where live data is ingested, processed, stored, and visualized using cloud-native tools.

The architecture consists of multiple interconnected AWS services that work together in a streamlined and efficient manner. A **Python script** using the Faker library generates synthetic data which is continuously streamed into **Amazon Kinesis Data Streams**. This data is then processed in real-time by **AWS Lambda functions**, which transform and forward it to **Amazon DynamoDB**, a fast and flexible NoSQL database. On the frontend, a **React.js** web application is developed to fetch and display the data via **API Gateway**, allowing users to monitor real-time metrics through an intuitive dashboard interface.

To deliver this solution to end users in a reliable and scalable way, the React application is deployed to an **S3 bucket** and distributed globally using **Amazon CloudFront**, ensuring low-latency access with secure HTTPS connectivity.

This project not only highlights the benefits of using a **serverless architecture**, such as scalability, cost-efficiency, and reduced operational complexity, but also emphasizes modern software development practices including **CI/CD**, **cloud deployment**, and **real-time user interface design**. It enabled us to gain practical exposure to widely used cloud services and fostered a deeper understanding of how to build event-driven systems.

Through this documentation, we will walk through the system design, key components, implementation steps, and learnings acquired throughout the project. It reflects the skills acquired in full-stack development, cloud integration, and serverless deployment, and serves as a solid foundation for building more advanced and production-ready data processing solutions in the future.



Fig(1): Architecture Diagram

Literature Review

In recent years, the explosive growth of data and the need for responsive applications have led to a significant shift in how organizations architect their IT solutions. Traditional monolithic infrastructures are increasingly being replaced with scalable, event-driven architectures that support real-time data processing. Cloud computing has revolutionized this space, enabling on-demand resource provisioning, elasticity, and cost efficiency. Within this domain, **serverless computing** has emerged as a powerful paradigm that abstracts infrastructure management and allows developers to focus purely on application logic.

1. Serverless Computing

Serverless computing, often referred to as Function-as-a-Service (FaaS), eliminates the need for developers to provision or manage servers. Platforms such as **AWS Lambda**, **Azure Functions**, and **Google Cloud Functions** enable the execution of code in response to specific events. Literature shows that serverless computing enhances scalability, reduces costs due to pay-per-use pricing models, and supports rapid development cycles (Baldini et al., 2017). In our project, AWS Lambda is a critical component responsible for transforming data streamed via Amazon Kinesis and storing it in DynamoDB.

2. Data Streaming with Amazon Kinesis

Real-time data streaming has become essential in fields such as IoT, finance, and social media analytics. Amazon Kinesis is a fully managed service designed to handle massive real-time data streams. Research indicates that streaming architectures allow organizations to gain faster insights and improve responsiveness to dynamic changes (Gulisano et al., 2012). In our implementation, Amazon Kinesis ingests synthetic data in real time, simulating use cases like sensor monitoring or live user interaction logging.

3. NoSQL Databases: DynamoDB

Traditional relational databases often fail to keep up with the speed and scale required for real-time applications. **Amazon DynamoDB**, a NoSQL key-value and document database, offers high availability and single-digit millisecond latency at any scale. Literature supports DynamoDB's suitability for serverless architectures, as it seamlessly integrates with other AWS services like Lambda and API Gateway, providing scalability and performance without manual tuning (Sivasubramanian, 2012).

4. Frontend Technologies: React.js and Cloud Integration

React.js is a widely adopted JavaScript library for building responsive and component-based user interfaces. According to recent surveys (e.g., Stack Overflow Developer Survey), React continues to be one of the most preferred tools for frontend development due to its performance and flexibility. When paired with **Amazon S3** and **CloudFront** for hosting and content delivery, developers can build and deploy globally available apps with minimal effort.

5. Cloud-Native Architecture and DevOps

Cloud-native development promotes microservices, containerization, and CI/CD practices, ensuring rapid deployment and scalability. The integration of AWS Amplify, S3, and CloudFront for hosting the frontend aligns with these best practices. Numerous studies advocate for DevOps pipelines and cloud-native stacks as they lead to increased productivity, reduced failure rates, and faster recovery (Forsgren et al., 2018).

6. Conclusion of Review

The collective insights from academic papers and industry research confirm the value and relevance of serverless, cloud-native approaches to building scalable and resilient data analytics platforms. By integrating services like Kinesis, Lambda, DynamoDB, and React-based dashboards, our project effectively reflects the cutting edge of modern web development and cloud computing strategies.

Problem Statement

In the age of digital transformation, organizations across industries are increasingly reliant on real-time data to drive decisions, monitor systems, and deliver personalized experiences. From IoT sensors and financial transactions to user activity on web platforms, the volume and velocity of data being generated have grown exponentially. However, many traditional data pipelines are built on monolithic architectures that require significant infrastructure setup, manual scaling, and maintenance efforts. These legacy systems struggle with processing real-time streams efficiently and often incur high operational costs.

Moreover, building scalable, secure, and responsive data analytics solutions typically demands expertise in managing servers, databases, APIs, and frontends — a barrier for many developers and teams looking to focus more on innovation than infrastructure.

To overcome these limitations, serverless computing has emerged as a powerful paradigm, offering fully managed services that auto-scale, require no server maintenance, and enable rapid development and deployment. With the advent of tools like AWS Lambda, Amazon Kinesis, DynamoDB, and API Gateway, it is now possible to build real-time data processing pipelines that are scalable, cost-efficient, and highly responsive — without provisioning or managing any servers.

Data Gathering

In this project, the core focus is on building a real-time data analytics pipeline, which necessitates a steady and reliable stream of data for processing, storage, and visualization. The data gathering phase plays a crucial role in simulating real-time data ingestion for the analytics pipeline. This section outlines the process by which data is collected, generated, and fed into the system.

The data gathering process for the project was divided into the following key components:

- Customer Activity Logs: Simulated logs capturing customer interactions with the platform, including product searches, views, and cart activities.
- Order Details: Transaction data containing customer ID, product ID, quantity, pricing, and time of purchase.
- Inventory Records: Periodic data representing stock levels, product categories, and warehouse information.
- Feedback & Reviews: Textual data simulating customer feedback, intended for sentiment analysis and future insights.

Data gathering is the foundational step that allows the serverless pipeline to function in real time. By leveraging tools like the Faker library for synthetic data generation, Amazon Kinesis for real-time streaming, and AWS Lambda for processing, the project successfully simulates a dynamic, scalable data analytics pipeline. This setup provides valuable insights into building similar pipelines in real-world applications where real-time data is essential for decision-making.

Creativity & Originality

The creativity and originality in the data gathering process lie in the innovative use of serverless architecture and real-time data streaming, with a focus on scalability and performance. Here's how the approach stands out:

---

1. Synthetic Data Generation

By utilizing Python's Faker library to generate synthetic data, the project demonstrates a unique way of simulating real-world scenarios, allowing for endless variations of data without relying on actual user inputs. This approach offers flexibility in testing various edge cases and ensures a diverse range of data for analysis.

2. Real-Time Data Streaming via Amazon Kinesis

The integration of Amazon Kinesis for real-time data streaming is a modern, scalable solution that handles high-throughput data with minimal delay. This approach ensures that large volumes of data are processed continuously, allowing the pipeline to simulate real-time interactions at scale, providing a seamless user experience.

3. Processing and Transformation

Using AWS Lambda to automatically scale and process data as it enters Kinesis provides a creative, serverless solution that minimizes the need for manual intervention. The transformation of raw data into structured, enriched insights highlights an efficient approach to data processing in cloud-based environments.

4. Data Storage and Access

Storing processed data in DynamoDB is a thoughtful design choice, considering its scalability and low-latency retrieval capabilities. By choosing a NoSQL database optimized for performance at scale, the project ensures that real-time analytics can be accessed swiftly, even as the data volume increases.

5. Frontend Interaction via API Gateway

The seamless integration of API Gateway with the React.js frontend introduces a clean and efficient way of displaying real-time analytics. The use of RESTful API endpoints for data retrieval is a creative method to ensure smooth communication between the frontend and backend, supporting dynamic visualizations without bottlenecks.

6. Challenges and Considerations

The originality lies in the strategic consideration of potential challenges like data integrity, latency, and scalability. The proactive approach to testing and troubleshooting real-time data flow ensures the system can handle continuous streaming at scale, showing foresight in dealing with common cloud architecture pitfalls.

Analysis and Problem-Solving

Introduction

Modern serverless web applications require real-time responsiveness and scalable backend systems. This project showcases a serverless architecture built using AWS services and a React-based frontend. It enables users to fetch product price data using product IDs, while incorporating real-time analytics, data delivery, and performance monitoring using services such as Amazon S3, Lambda, API Gateway, Kinesis, and DynamoDB. CloudFront and CloudWatch ensure fast, secure delivery and observability.

---

2. Problem Identification and Solutions

Problem 1: Hosting and Frontend Delivery

Challenge: Efficient delivery of a dynamic web interface to users.
Solution: The React application is compiled and hosted in Amazon S3, providing static file hosting. Amazon CloudFront serves these assets via its CDN, caching content at edge locations globally, resulting in fast, secure delivery.

Problem 2: API Communication with Backend

Challenge: Enabling the frontend to query backend logic without managing servers.

Solution: Amazon API Gateway exposes RESTful endpoints for the React frontend to send product ID queries. It routes the requests to AWS Lambda for processing.

Problem 3: Serverless Business Logic Processing

Challenge: Responding to user requests dynamically without provisioning infrastructure.

Solution: AWS Lambda functions are triggered by API Gateway to process input, query DynamoDB using the product ID, and return real-time price data to the frontend.

Problem 4: Real-Time Data Streaming

Challenge: Simulating and handling real-time analytics data for usage metrics or user activity.
Solution: Amazon Kinesis Data Streams is used to ingest real-time data (e.g., search or access events from the React app). This stream can later be used for analytics, monitoring access trends, or integrating with dashboards or alerts.

Problem 5: Structured and Fast Data Retrieval

Challenge: Storing and accessing product pricing data quickly.
Solution: Amazon DynamoDB is used as the database to store pricing data indexed by product ID. It offers low-latency, scalable key-based lookups, essential for real-time apps.

Problem 6: Monitoring and Debugging

Challenge: Detecting failures or performance issues in real time.
Solution: Amazon CloudWatch captures Lambda logs, API usage metrics, and system errors. This provides detailed observability into the system's behavior and helps in debugging and performance tuning.

---

3. Outcome and Benefits

- Real-Time Functionality: Real-time API requests and responses provide users with immediate access to product data.
- Frontend Performance: React + CloudFront ensures quick and smooth user experience with a responsive UI.
- Scalability: Kinesis, Lambda, and DynamoDB scale automatically with user demand.
- Event Streaming Ready: Real-time events (like search actions) can be logged using Kinesis for further insight.
- Security and Speed: CloudFront and API Gateway ensure secure access and fast response times.
- Observability: CloudWatch makes it easy to debug, trace, and improve system performance.

4. Modules

1. Frontend Hosting & Delivery Module
Tools: React, Amazon S3, Amazon CloudFront
Key Functions:

- Build React SPA using Vite or Create React App.
- Host static files in S3.
- Deliver via CloudFront with HTTPS and caching.

2. API Communication Module
Tools: API Gateway, AWS Lambda
Key Functions:

- API Gateway exposes HTTP endpoints.
- Triggers Lambda to process requests and return responses to frontend.

3. Backend Logic & Data Access Module
Tools: AWS Lambda, Amazon DynamoDB
Key Functions:

- Lambda queries DynamoDB using product ID.
- Returns pricing info and store details.
- Ensures real-time, low-latency access.

4. Real-Time Event Streaming Module
Tools: Amazon Kinesis
Key Functions:

- Captures frontend events like page views or searches.
- Streams data to Kinesis for future analytics (optional integration with Lambda or S3).
- Enables tracking user behavior or triggering alerts.

5. Monitoring & Logging Module
Tools: CloudWatch
Key Functions:

- Captures logs and errors from Lambda/API Gateway.
- Helps debug issues and monitor system performance.
- Supports alerting with metric thresholds.

5. Tools and Technologies Used

Category	*Tool/Service*	*Description*
Frontend Framework	React	Builds modern, interactive web interface
Frontend Hosting	Amazon S3	Stores static files for React app
Content Delivery	Amazon CloudFront	Serves assets globally with low latency
API Management	Amazon API Gateway	Exposes REST APIs for communication with backend
Serverless Compute	AWS Lambda	Processes API requests and queries DynamoDB
Database	Amazon DynamoDB	Stores and retrieves pricing data
Event Streaming	Amazon Kinesis	Captures frontend events in real time
Monitoring	Amazon CloudWatch	Logs Lambda executions, tracks performance and errors

Discussions & Results

The **Serverless Data Analytics Pipeline using AWS** effectively demonstrates how modern cloud-native technologies can be utilized for handling real-time data ingestion, processing, and visualization while minimizing operational overhead. By integrating AWS services like Lambda, Kinesis, DynamoDB, API Gateway, and S3, this solution provided low-latency performance, scalability, and cost-efficiency, all powered by a React-based frontend.

System Performance

The system performed well throughout the various stages of the analytics pipeline. Data ingestion via Amazon Kinesis ensured near real-time processing, handling large data volumes efficiently with minimal delays. Amazon DynamoDB automatically scaled during high traffic periods, ensuring smooth data storage and access. The React frontend, served via S3 and CloudFront, displayed near-instant feedback from the backend, with API Gateway facilitating the flow of data between the client and server. Real-time monitoring through CloudWatch provided insights into system health and allowed for performance optimization when needed.

Data Quality

AWS Lambda functions played a key role in maintaining high data quality. Before storing incoming data in DynamoDB, Lambda functions performed data validation and cleaning, ensuring that only accurate and structured data were retained. This preprocessing step helped reduce errors and inconsistencies, resulting in cleaner outputs and more reliable analytics for the end-users.

Scalability & Cost Efficiency

Scalability was achieved through the use of AWS's serverless offerings. DynamoDB's auto-scaling feature ensured that the database could handle varying workloads without manual intervention. The serverless compute model of AWS Lambda reduced the need to provision fixed resources, allowing the system to scale up or down as required. Costs were optimized by using serverless services that only charged for actual usage, while S3 and CloudFront minimized storage and bandwidth costs for serving static assets.

Real-Time Insights

The integration of Amazon Kinesis enabled the processing of streaming data in real time, ensuring that business metrics were always up-to-date. For example, when new data was uploaded, the system immediately reflected these changes in the frontend through API Gateway, Lambda, and DynamoDB. Real-time monitoring through CloudWatch helped maintain the system's responsiveness and provided alerts for any

anomalies, contributing to rapid issue detection and resolution.

Challenges

During development and testing, several challenges were encountered:

- **Data Latency**: Minor delays were observed during high data loads, particularly when Lambda cold starts occurred or when large batches of data were ingested.
- **Data Transformation Complexity**: Developing transformation logic for different data structures and edge cases added complexity and required extensive testing to ensure accuracy.
- **Cost Management**: DynamoDB's throughput-based pricing required careful management to avoid overprovisioning resources, especially when simulating peak loads or high volumes of data.

Future Enhancements

Looking ahead, there are several opportunities to further enhance the platform. One key improvement would be integrating **Amazon SageMaker** to build predictive models that can forecast pricing trends, analyze customer behavior, and detect anomalies in real-time. Another enhancement could involve enabling **historical analytics** by utilizing **AWS Glue** and **Athena**, which would allow users to query archived data stored in S3 for long-term trend analysis and reporting. To strengthen security and access management, **Amazon Cognito** could be integrated, providing role-based access control for users with different permissions such as admins and viewers. Additionally, the frontend could be upgraded with advanced charting libraries like **Recharts** or **Chart.js**, allowing for richer, more interactive data visualizations. For improved data integrity and compliance, stricter validation pipelines and enhanced logging could be implemented, ensuring better data governance and auditability. The platform could also benefit from global accessibility by deploying resources across multiple AWS regions to reduce latency and improve availability for users worldwide. Lastly, integrating **Amazon SNS** or third-party alerting services like **PagerDuty** would allow for more sophisticated alerting and escalation workflows, ensuring quicker responses to critical events.

Lambda Function:

1)process-kinesis-stream:

```
import json
import boto3

import base64


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

def lambda_handler(event, context):
    for record in event['Records']:
        # Decode and parse the Kinesis data
        payload = json.loads(base64.b64decode(record['kinesis']['data']))

        # Log the incoming record (optional)
        print(f"Received: {payload}")

        # Insert into DynamoDB
        table.put_item(Item={
            'id': str(payload['id']),
            'name': payload['name'],
            'country': payload['country'],
            'source': payload['source']
        })
    return {'statusCode': 200, 'body': 'Processed'}
```

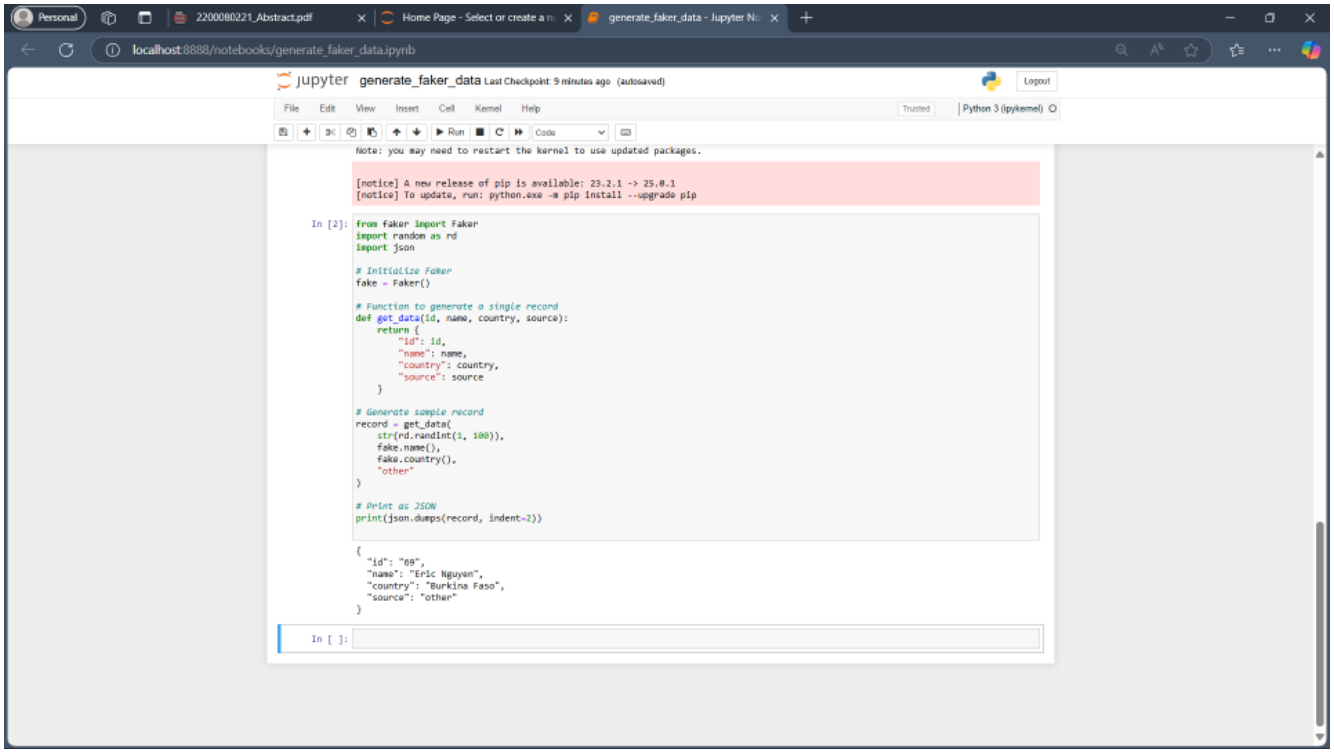
2)getDynamoData

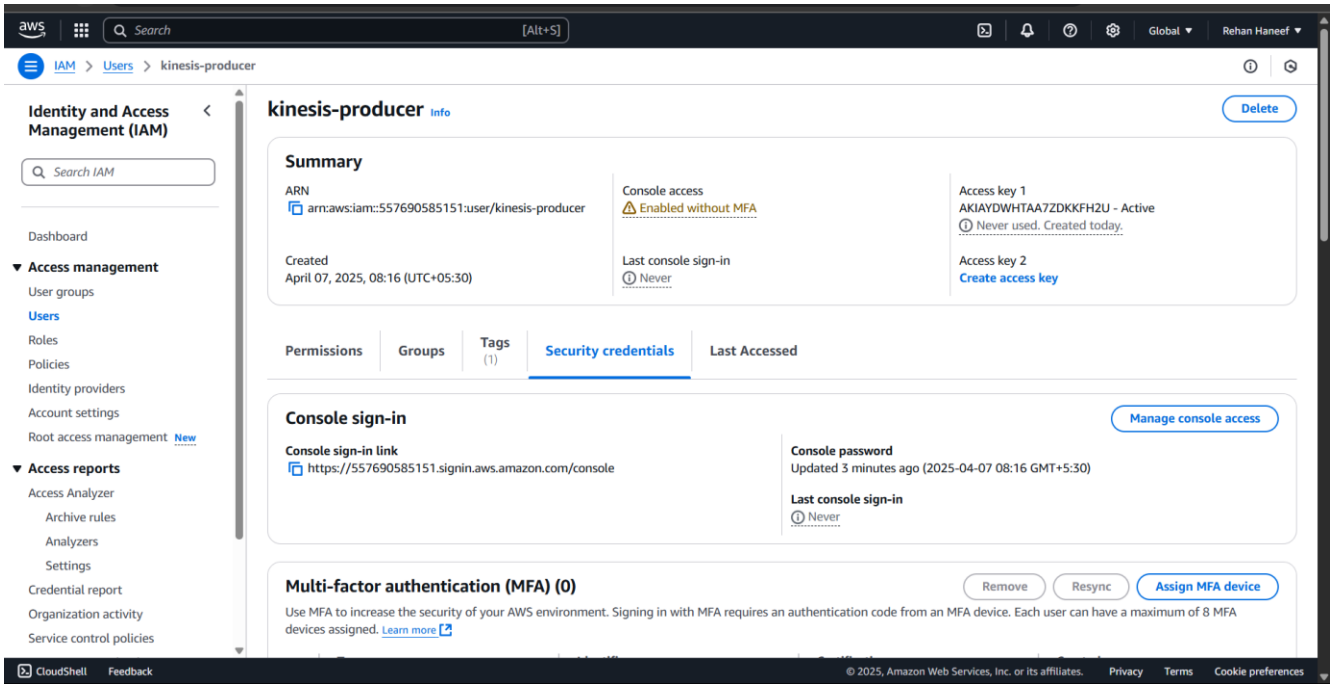
```
import boto3
import json

# Initialize DynamoDB resource
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('KinesisDataTable')

def lambda_handler(event, context):
    try:
        # Scan entire DynamoDB table
        response = table.scan()
        items = response.get('Items', [])

        return {
            'statusCode': 200,
            'headers': {
                "Access-Control-Allow-Origin": "*", # Allow all domains
                "Access-Control-Allow-Methods": "GET,OPTIONS",
                "Access-Control-Allow-Headers": "Content-Type",
                "Content-Type": "application/json"
            },
            'body': json.dumps(items)
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                "Access-Control-Allow-Origin": "*"
            },
            'body': json.dumps({'error': str(e)})
        }
```

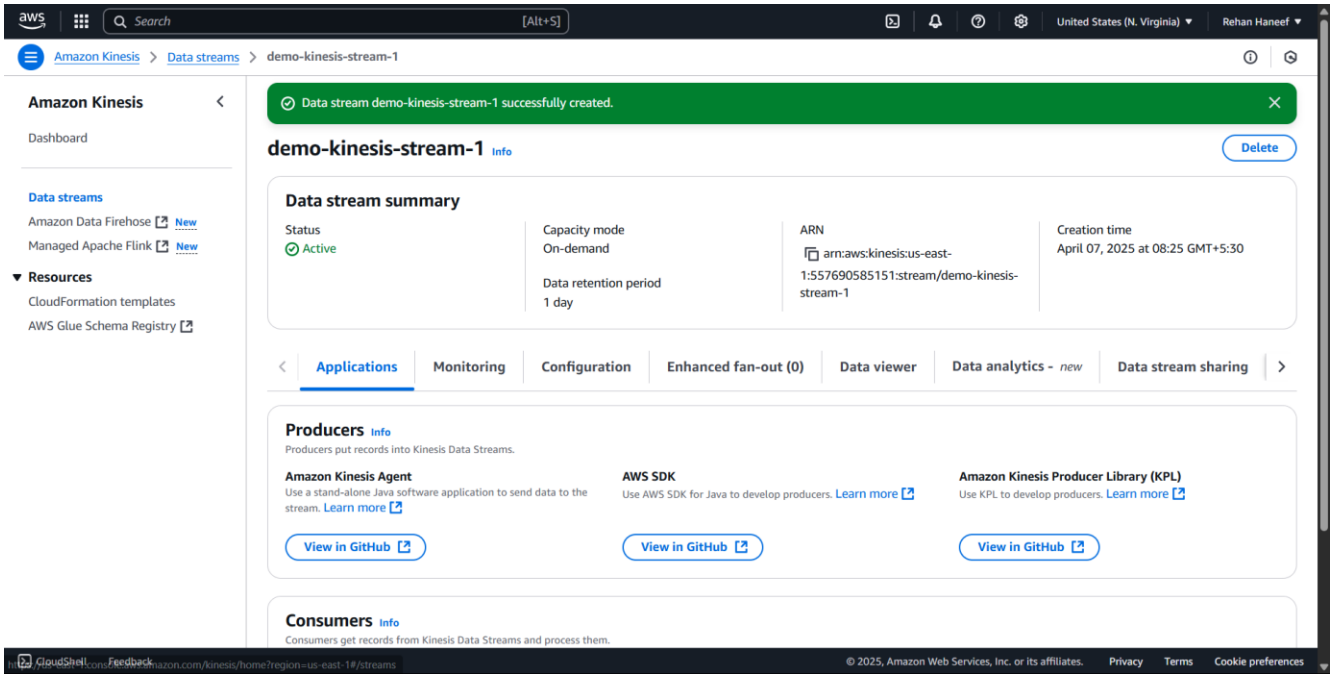
generate_faker_data.ipynb:

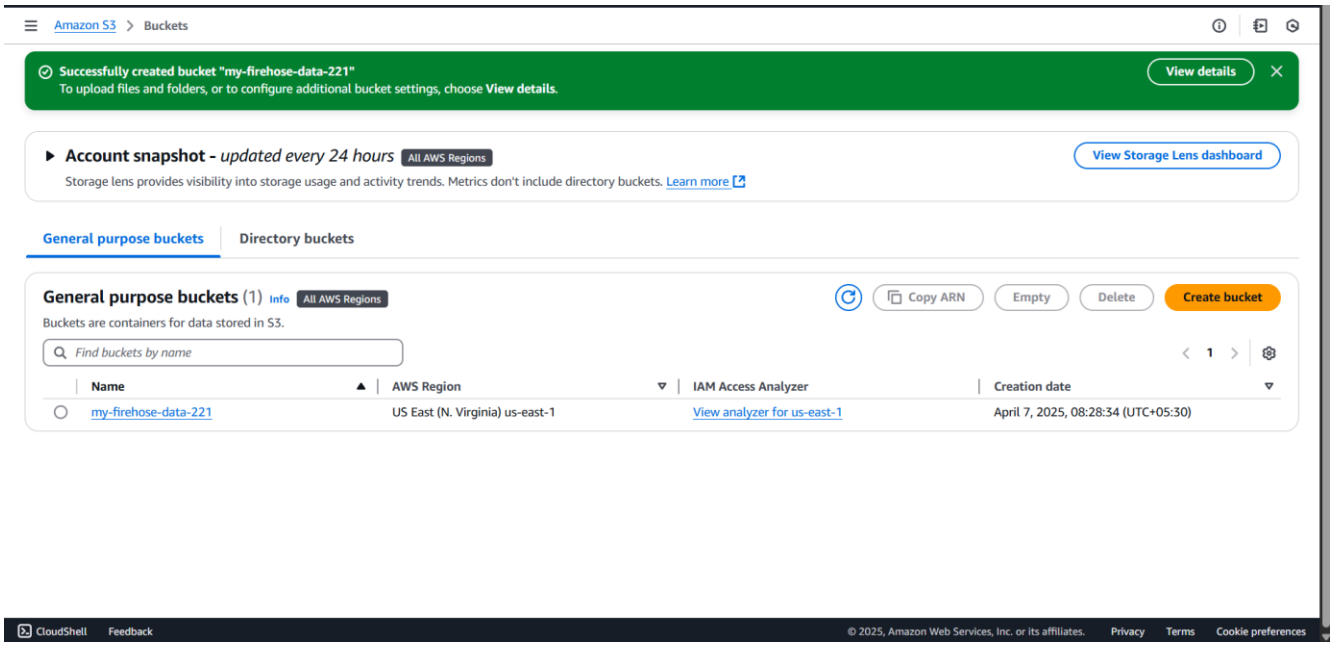
```
from faker import Faker
import random as rd
import json
import boto3

access_key = "AKIAYDWHTAA7ZDKKFH2U"
secret_key = "w/Vhioh3j+WEKn2L9LC00MjYQz/ZlhRTkc7dNfJ9"
region = "us-east-1"

session = boto3.Session(aws_access_key_id=access_key,
aws_secret_access_key=secret_key, region_name=region)

client = session.client('kinesis')
kinesis_stream = "demo-kinesis-stream-1"
sample = Faker()

def getData(id, name, country, source):
 return {
  "id": id,
  "name": name,
  "country": country,
  "source": source
 }

temp = 0
while temp < 50:
 source_type = 'local_system' if temp <= 25 else 'other'
 record = json.dumps(getData(str(rd.randint(1, 100)), sample.name(),
sample.country(), source_type))
 client.put_record(StreamName=kinesis_stream, Data=record, PartitionKey="1")
 print(record)
 temp += 1
```

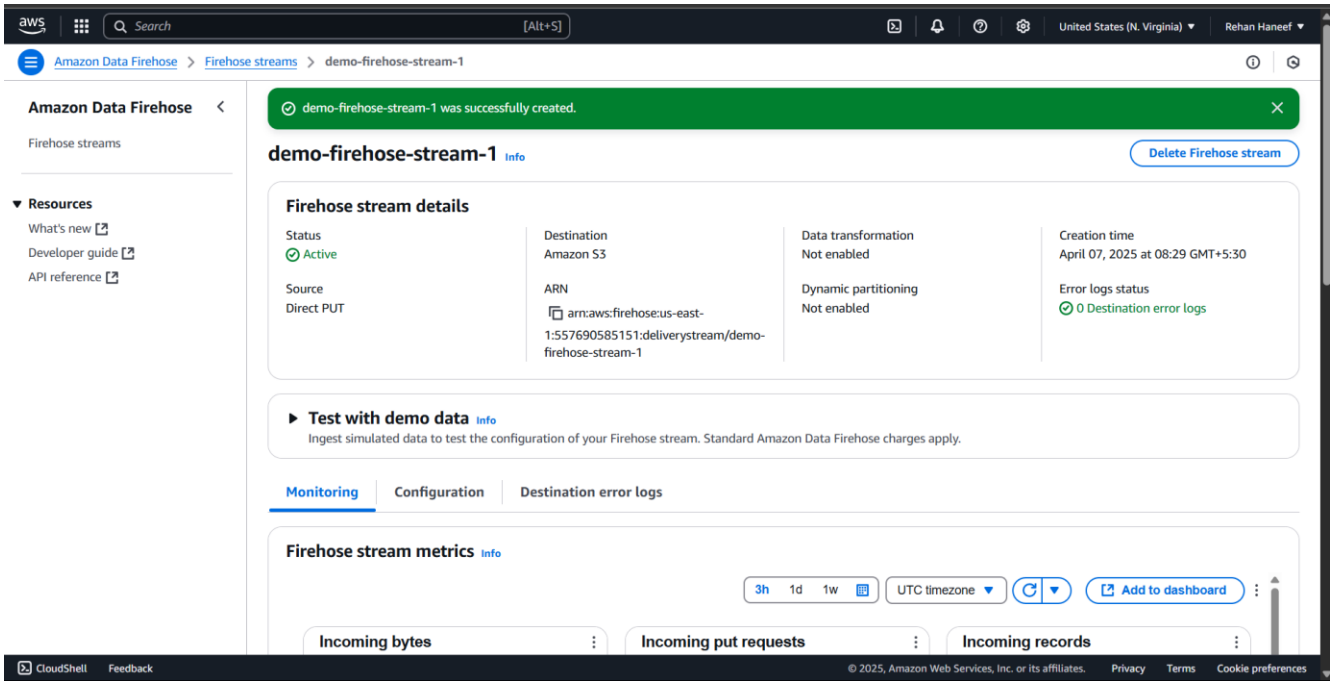
ScreenShots:



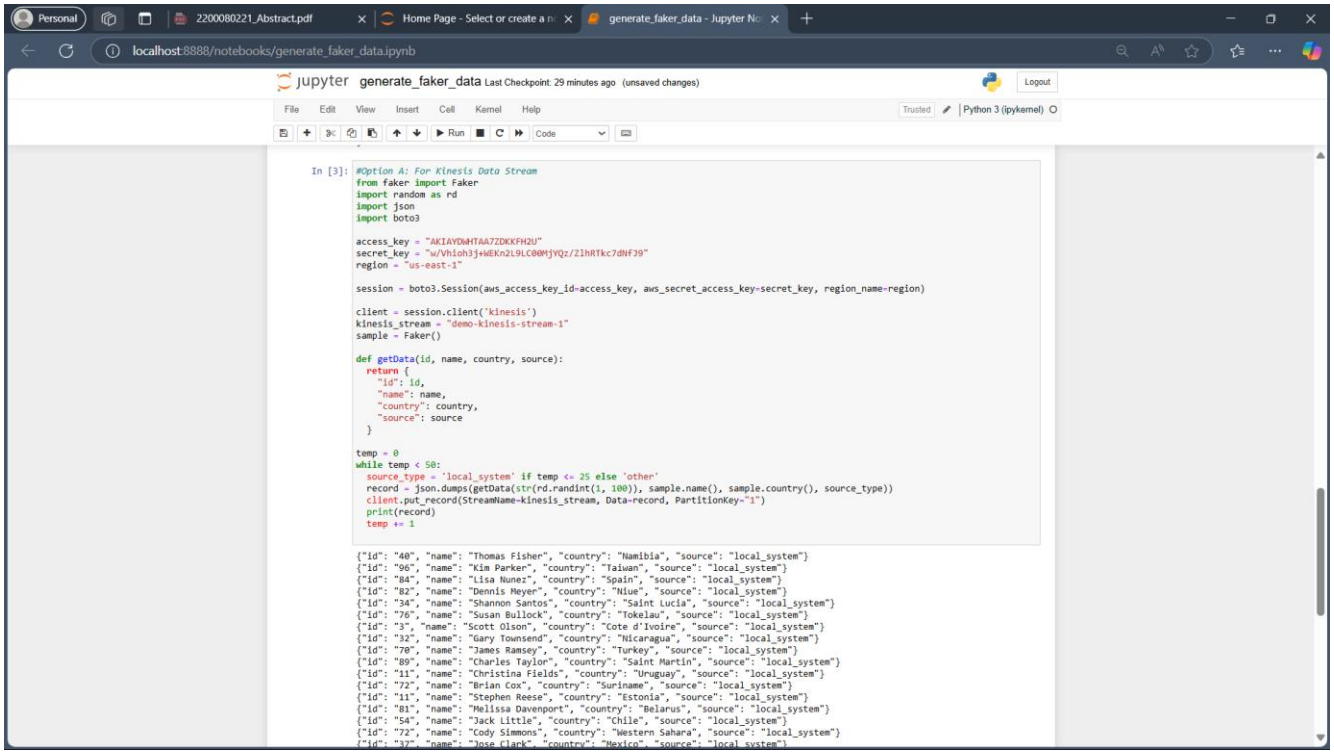



Search [Alt+S]
United States (N. Virginia)
Rehan Haneef
Amazon Kinesis > Data streams > demo-kinesis-stream-1
Amazon Kinesis
Dashboard
Data streams
Amazon Data Firehose New
Managed Apache Flink New
Resources
CloudFormation templates
AWS Glue Schema Registry
Data stream demo-kinesis-stream-1 successfully created.
demo-kinesis-stream-1 Info
Delete
Data stream summary
Status
Active
Capacity mode
On-demand
Data retention period
1 day
ARN
arn:aws:kinesis:us-east-1:557690585151:stream/demo-kinesis-stream-1
Creation time
April 07, 2025 at 08:25 GMT+5:30
Applications
Monitoring
Configuration
Enhanced fan-out (0)
Data viewer
Data analytics - new
Data stream sharing
Producers Info
Producers put records into Kinesis Data Streams.
Amazon Kinesis Agent
Use a stand-alone Java software application to send data to the stream. Learn more
View in GitHub
AWS SDK
Use AWS SDK for Java to develop producers. Learn more
View in GitHub
Amazon Kinesis Producer Library (KPL)
Use KPL to develop producers. Learn more
View in GitHub
Consumers Info
Consumers get records from Kinesis Data Streams and process them.
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

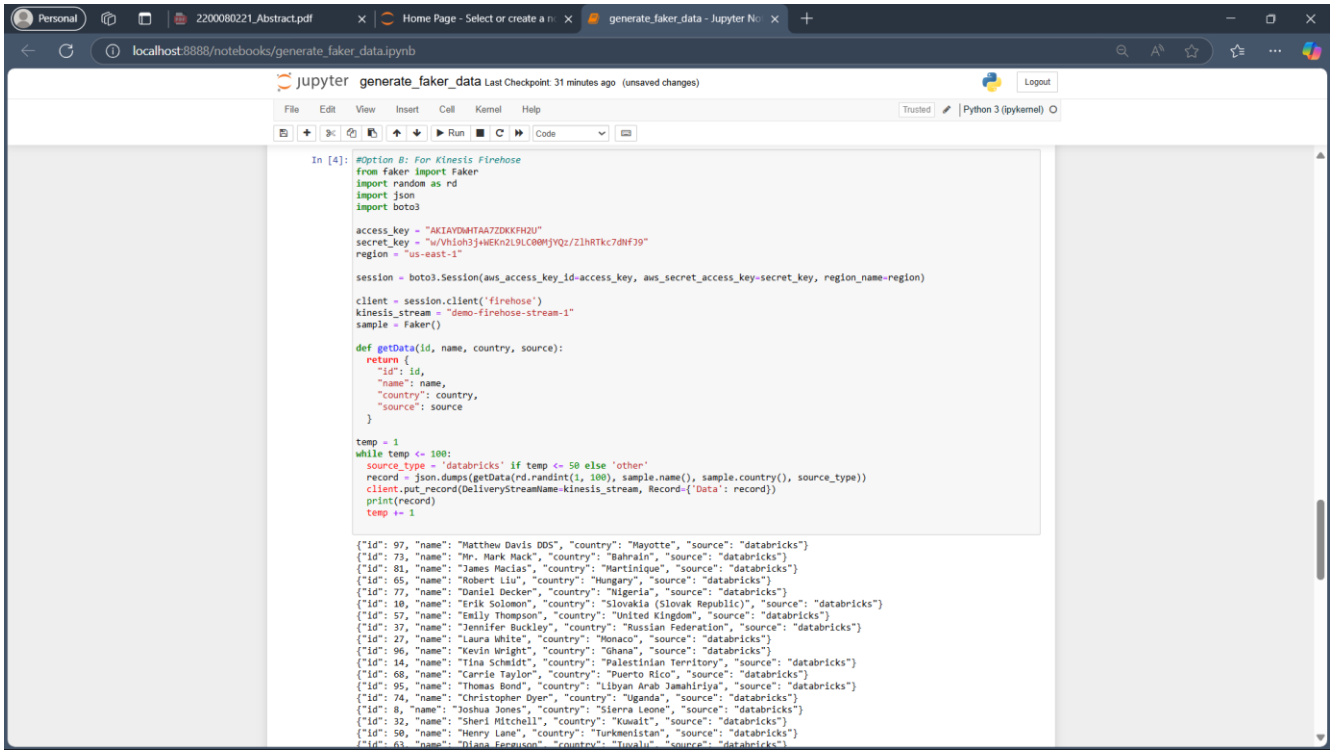

Amazon S3 > Buckets
Successfully created bucket "my-firehose-data-221"
To upload files and folders, or to configure additional bucket settings, choose View details.
View details
Account snapshot - updated every 24 hours All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info All AWS Regions
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
my-firehose-data-221
US East (N. Virginia) us-east-1
View analyzer for us-east-1
April 7, 2025, 08:28:34 (UTC+05:30)
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

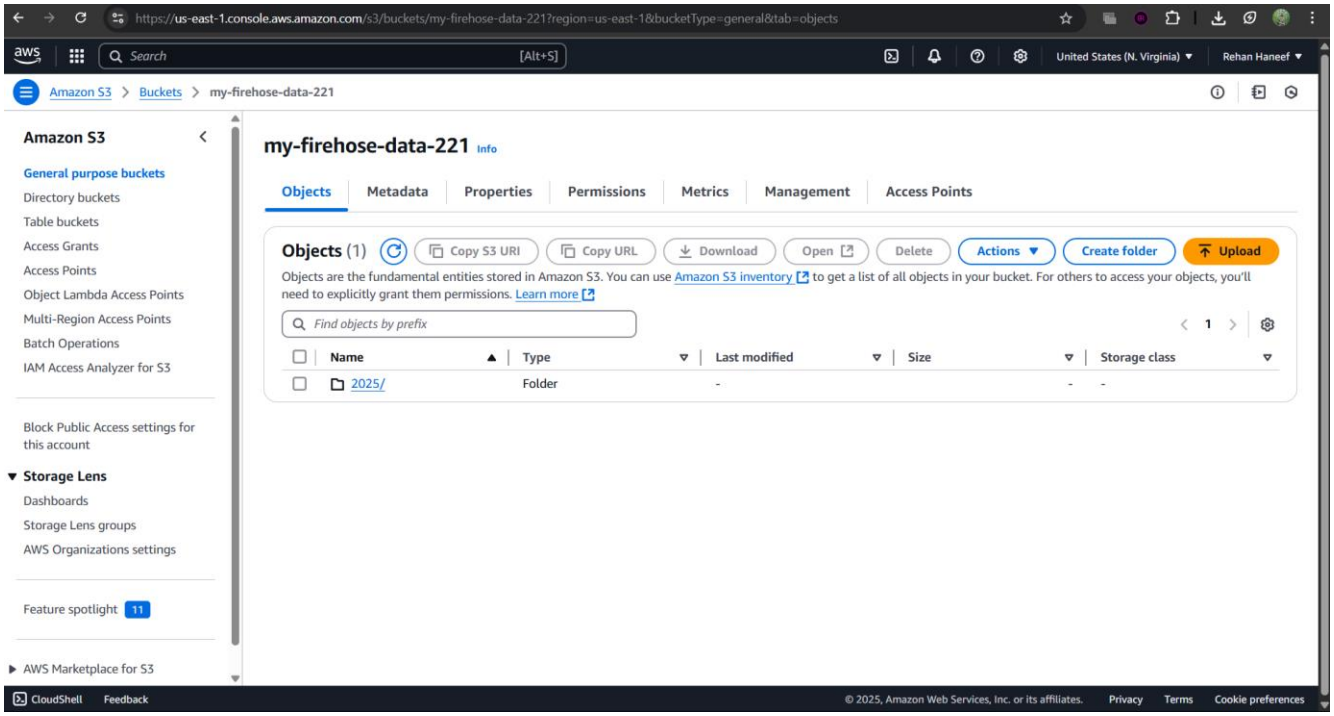


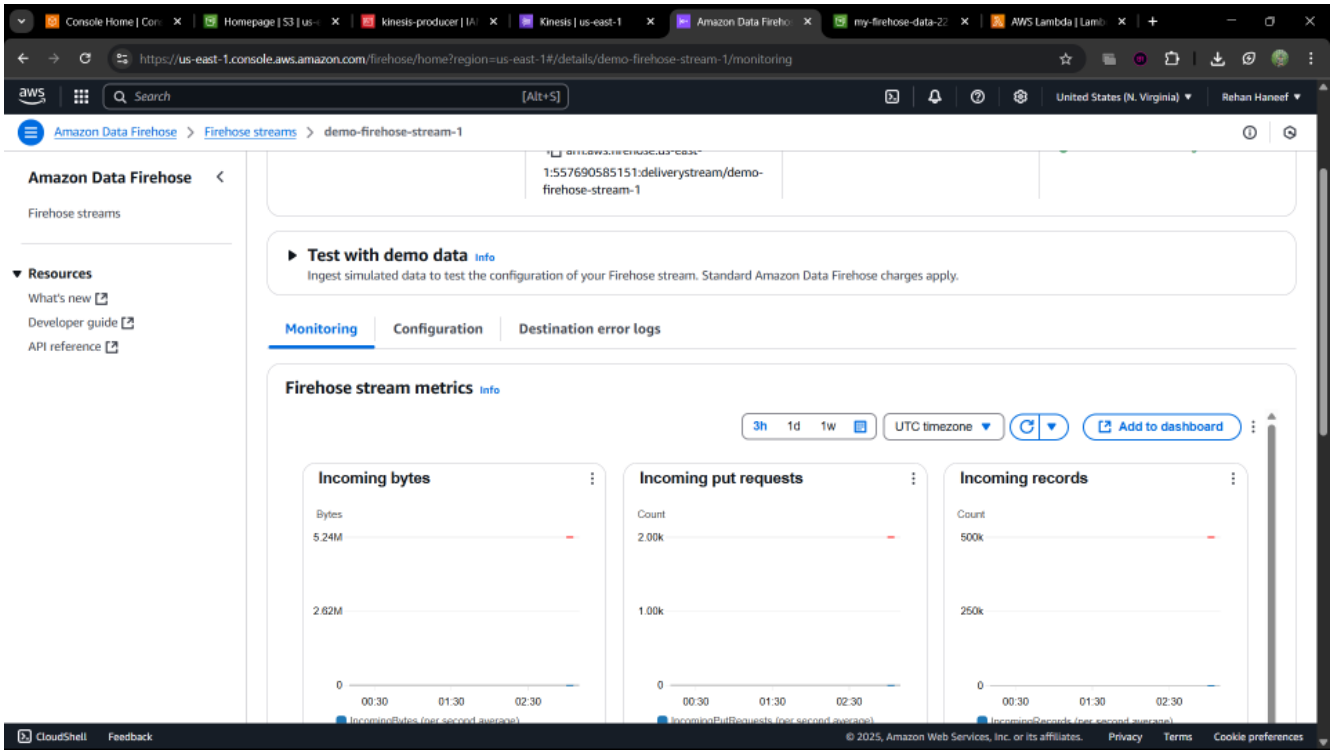
Fig(2): Data Stream by Kinesis

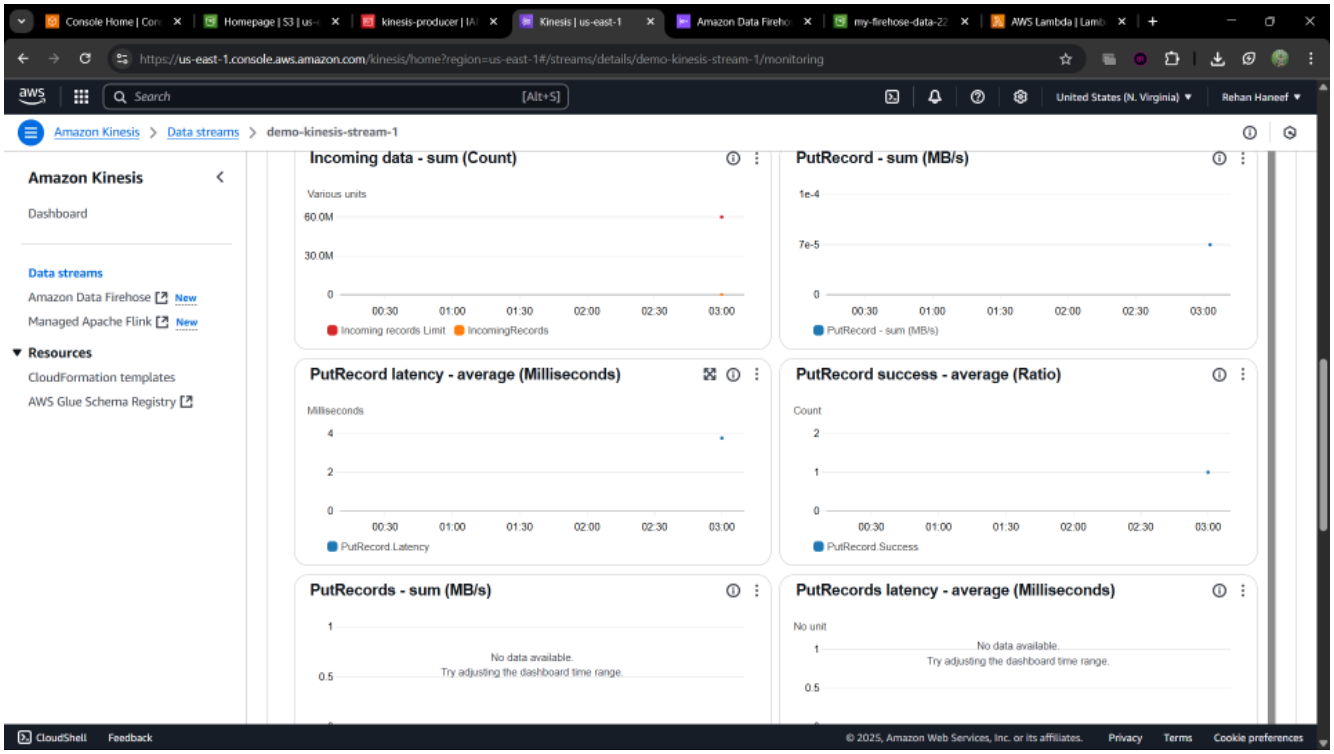


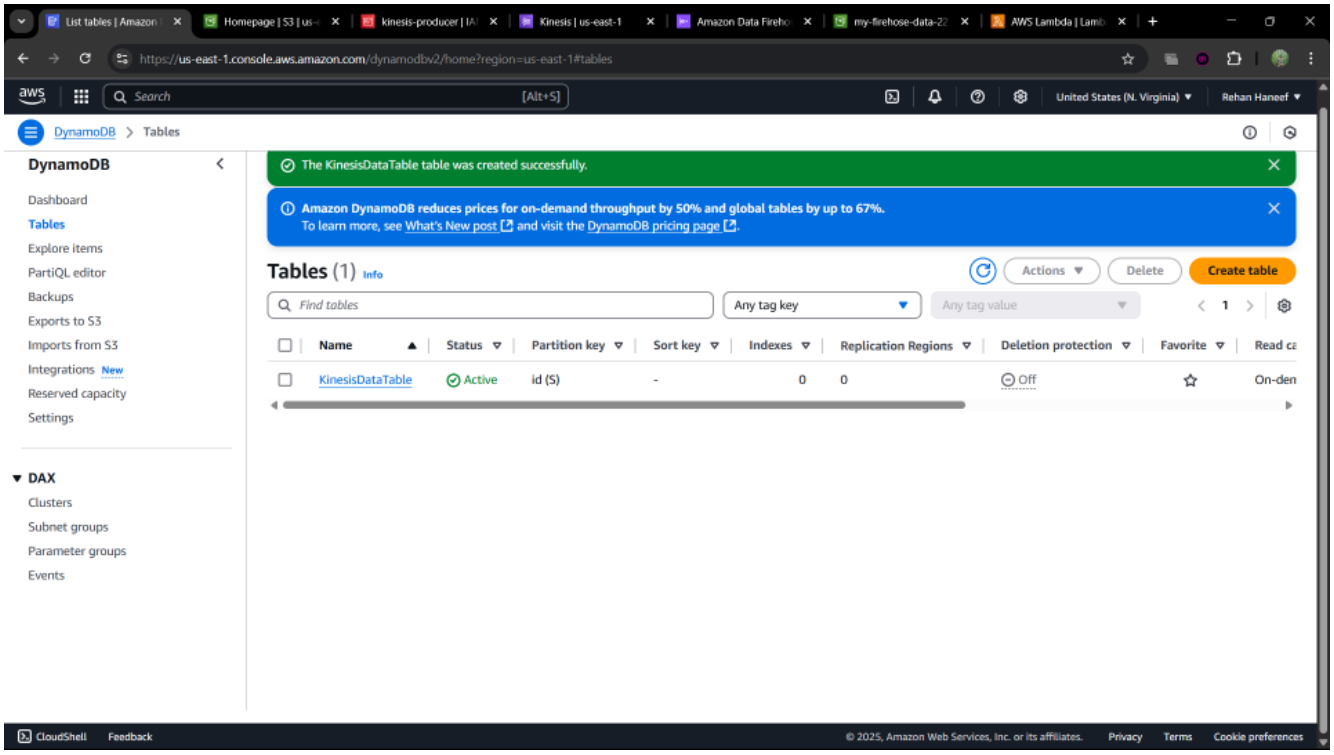
Fig(3): Generate faker Data


localhost:8888/notebooks/generate_faker_data.ipynb
jupyter generate_faker_data Last Checkpoint: 31 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
In [4]:
#Option B: For Kinesis Firehose
from faker import Faker
import random as rd
import json
import boto3
access_key = "AKIAYDWHTAA7ZDKKFH2U"
secret_key = "w/Vhioh3j+WEKn2L9LC00MjYQz/ZlhRTkc7dNfJ9"
region = "us-east-1"
session = boto3.Session(aws_access_key_id=access_key, aws_secret_access_key=secret_key, region_name=region)
client = session.client('firehose')
kinesis_stream = "demo-firehose-stream-1"
sample = Faker()
def getData(id, name, country, source):
return {
"id": id,
"name": name,
"country": country,
"source": source
}
temp = 1
while temp <= 100:
source_type = 'databricks' if temp <= 50 else 'other'
record = json.dumps(getData(rd.randint(1, 100), sample.name(), sample.country(), source_type))
client.put_record(DeliveryStreamName=kinesis_stream, Record={'Data': record})
print(record)
temp += 1
{"id": 97, "name": "Matthew Davis DDS", "country": "Mayotte", "source": "databricks"}
{"id": 73, "name": "Mr. Mark Mack", "country": "Bahrain", "source": "databricks"}
{"id": 81, "name": "James Macias", "country": "Martinique", "source": "databricks"}
{"id": 65, "name": "Robert Liu", "country": "Hungary", "source": "databricks"}
{"id": 77, "name": "Daniel Decker", "country": "Nigeria", "source": "databricks"}
{"id": 10, "name": "Erik Solomon", "country": "Slovakia (Slovak Republic)", "source": "databricks"}
{"id": 57, "name": "Emily Thompson", "country": "United Kingdom", "source": "databricks"}
{"id": 37, "name": "Jennifer Buckley", "country": "Russian Federation", "source": "databricks"}
{"id": 27, "name": "Laura White", "country": "Monaco", "source": "databricks"}
{"id": 96, "name": "Kevin Wright", "country": "Ghana", "source": "databricks"}
{"id": 14, "name": "Tina Schmidt", "country": "Palestinian Territory", "source": "databricks"}
{"id": 68, "name": "Carrie Taylor", "country": "Puerto Rico", "source": "databricks"}
{"id": 95, "name": "Thomas Bond", "country": "Libyan Arab Jamahiriya", "source": "databricks"}
{"id": 74, "name": "Christopher Dyer", "country": "Uganda", "source": "databricks"}
{"id": 8, "name": "Joshua Jones", "country": "Sierra Leone", "source": "databricks"}
{"id": 32, "name": "Sheri Mitchell", "country": "Kuwait", "source": "databricks"}
{"id": 50, "name": "Henry Lane", "country": "Turkmenistan", "source": "databricks"}

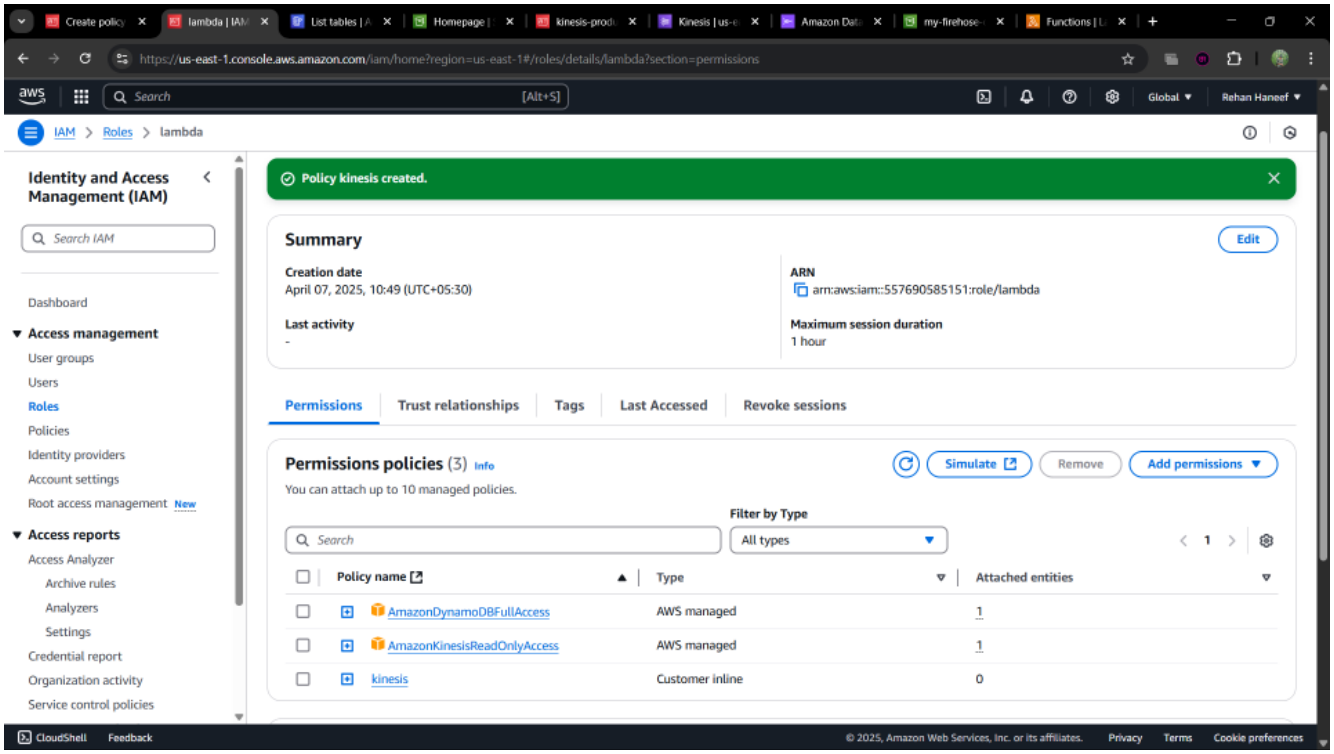

https://us-east-1.console.aws.amazon.com/s3/buckets/my-firehose-data-221?region=us-east-1&bucketType=general&tab=objects
aws
Search
[Alt+S]
United States (N. Virginia)
Rehan Haneef
Amazon S3 > Buckets > my-firehose-data-221
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
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 11
AWS Marketplace for S3
my-firehose-data-221 Info
Objects
Metadata
Properties
Permissions
Metrics
Management
Access Points
Objects (1)
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
2025/
Folder
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

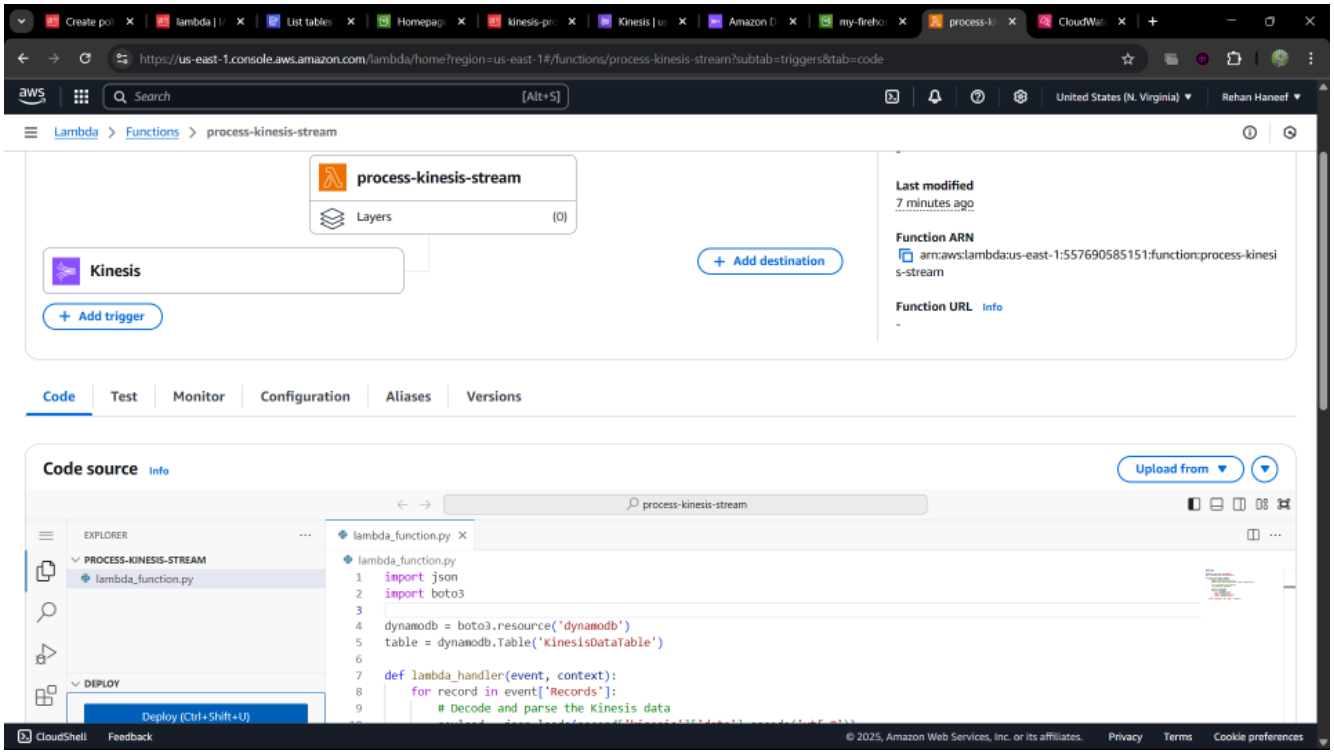

Amazon Data Firehose
Firehose streams
demo-firehose-stream-1
Test with demo data
Info
Ingest simulated data to test the configuration of your Firehose stream. Standard Amazon Data Firehose charges apply.
Monitoring
Configuration
Destination error logs
Firehose stream metrics
Info
3h
1d
1w
UTC timezone
Add to dashboard
Incoming bytes
Incoming put requests
Incoming records
Bytes
Count
5.24M
2.62M
2.00k
1.00k
500k
250k
0
00:30
01:30
02:30
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

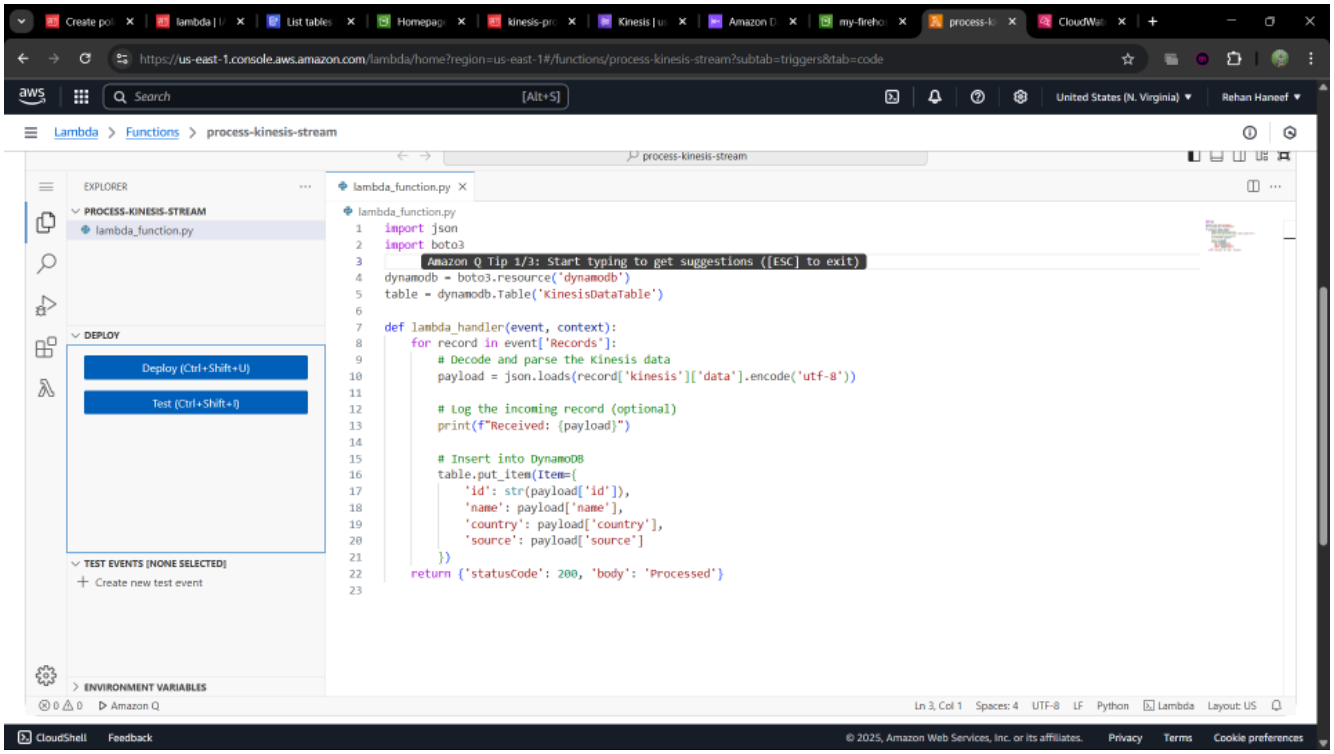

Amazon Kinesis
Data streams
demo-kinesis-stream-1
Incoming data - sum (Count)
PutRecord - sum (MB/s)
PutRecord latency - average (Milliseconds)
PutRecord success - average (Ratio)
PutRecords - sum (MB/s)
PutRecords latency - average (Milliseconds)
No data available.
Try adjusting the dashboard time range.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



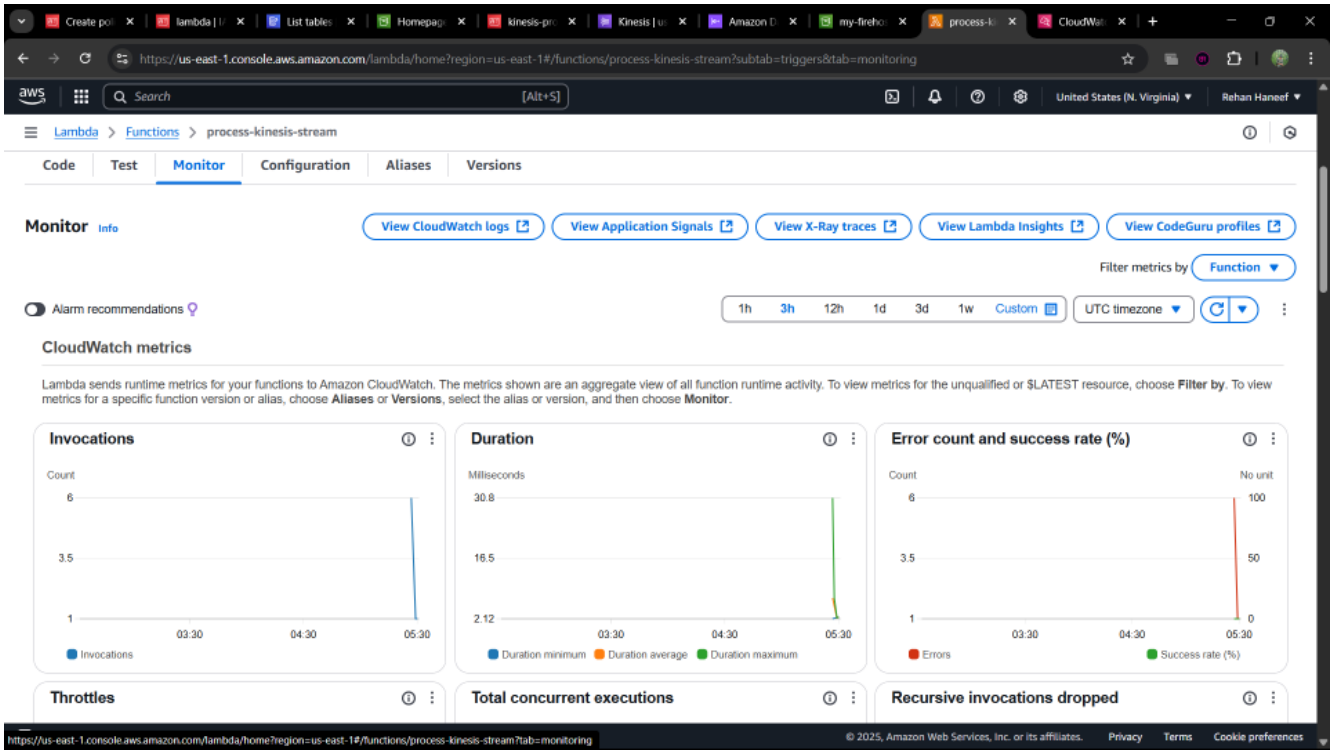
Fig(4): DynamoDB Table created

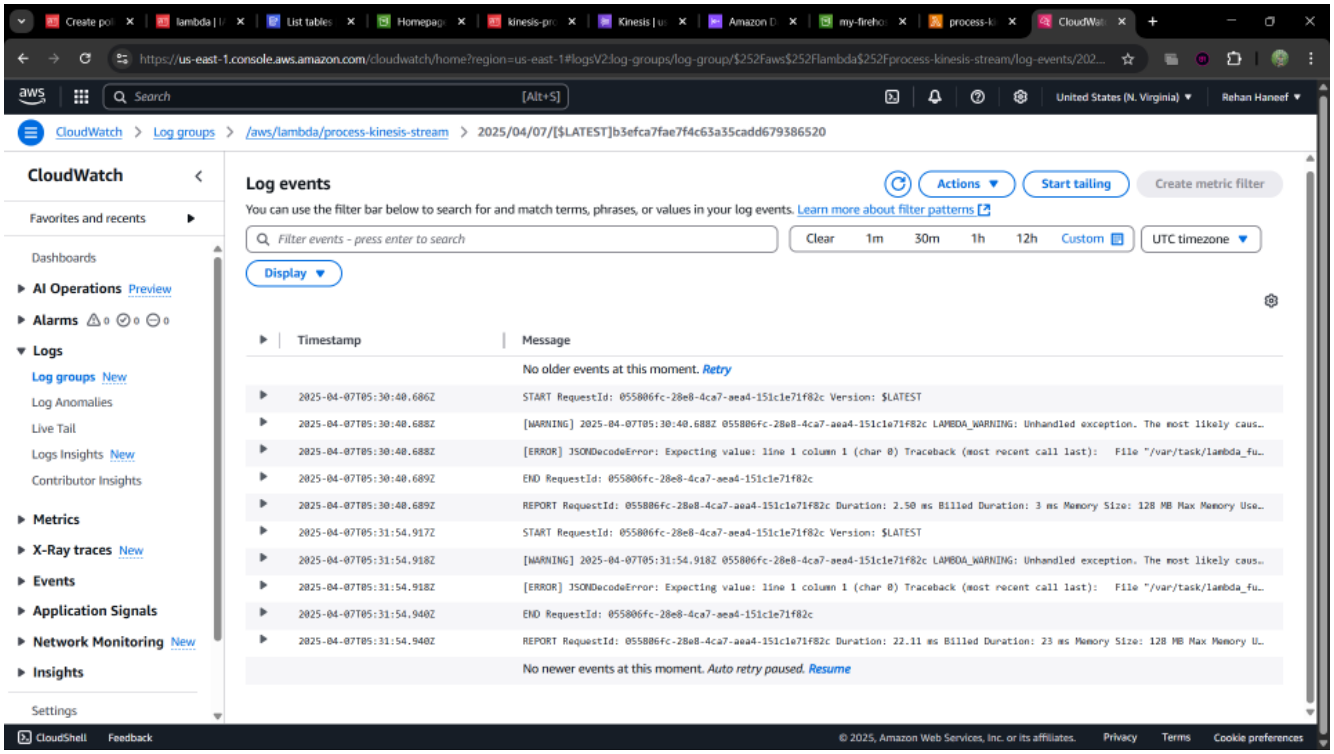


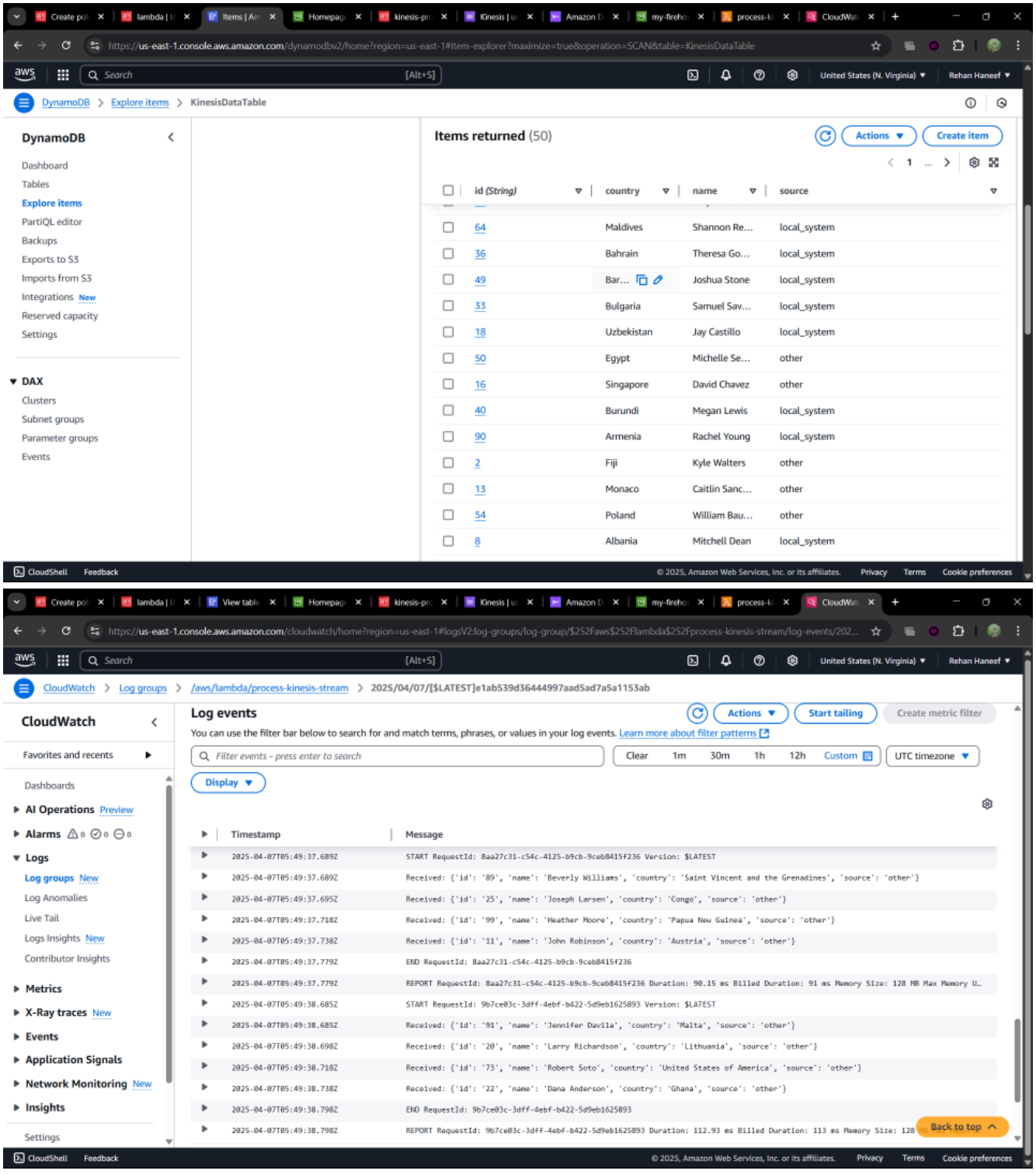




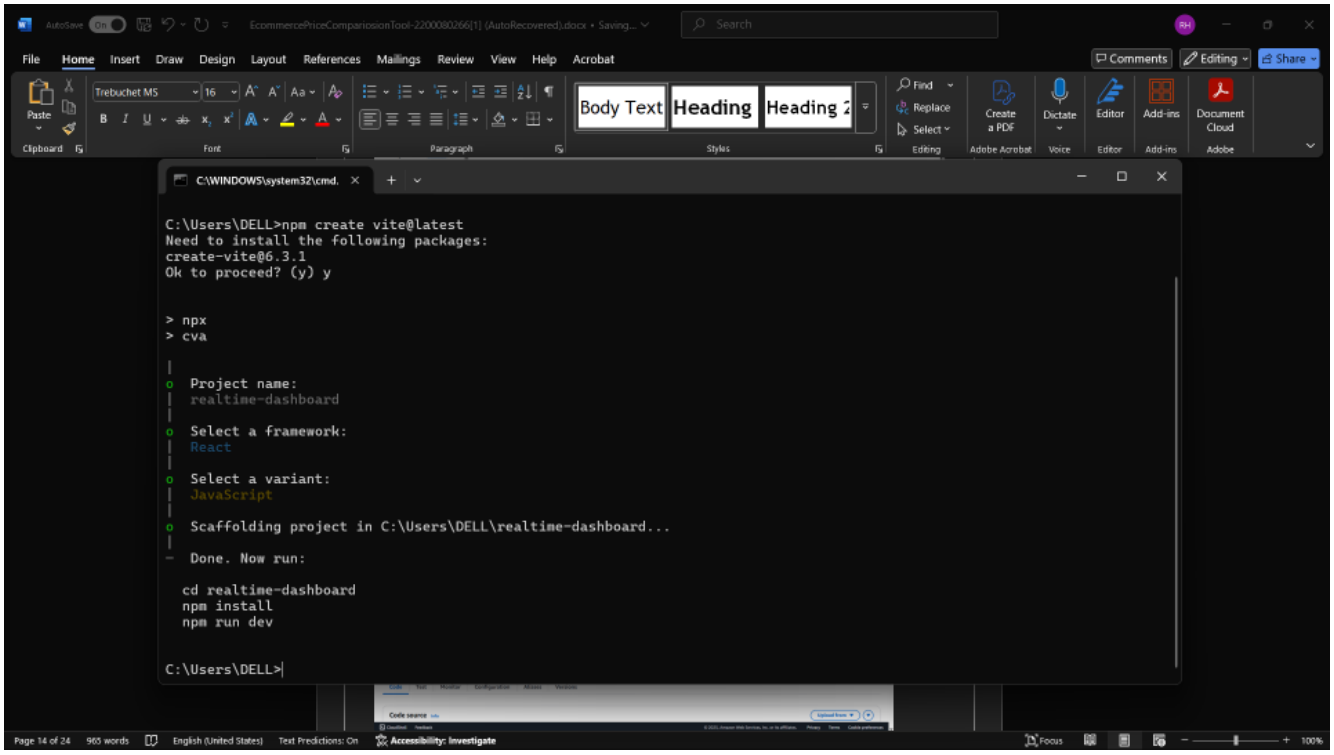
Fig(5): Lambda function-1

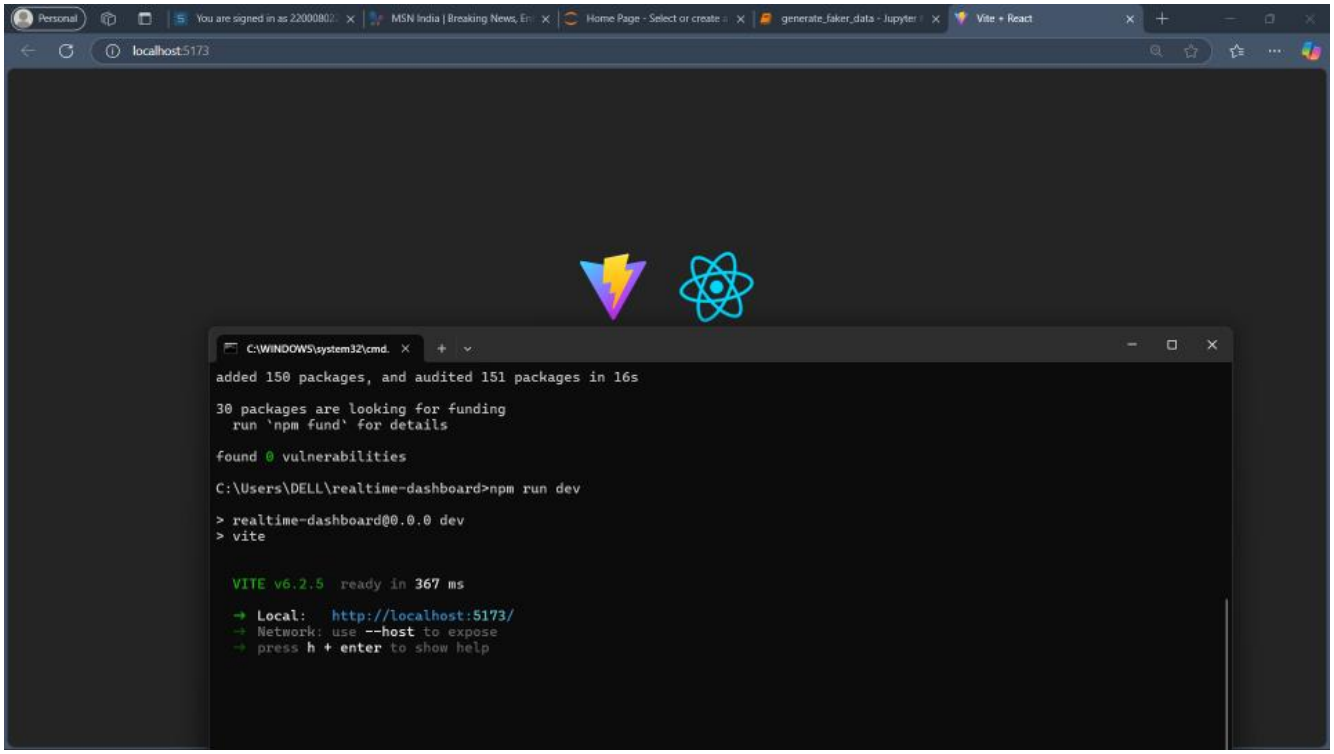

Lambda > Functions > process-kinesis-stream
Code
Test
Monitor
Configuration
Aliases
Versions
Monitor Info
View CloudWatch logs
View Application Signals
View X-Ray traces
View Lambda Insights
View CodeGuru profiles
Filter metrics by
Function
Alarm recommendations
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
CloudWatch metrics
Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose Filter by. To view metrics for a specific function version or alias, choose Aliases or Versions, select the alias or version, and then choose Monitor.
Invocations
Count
Invocations
Duration
Milliseconds
Duration minimum
Duration average
Duration maximum
Error count and success rate (%)
Count
No unit
Errors
Success rate (%)
Throttles
Total concurrent executions
Recursive invocations dropped


CloudWatch > Log groups > /aws/lambda/process-kinesis-stream > 2025/04/07/[$LATEST]b3efca7fae7f4c63a35cadd679386520
CloudWatch
Favorites and recents
Dashboards
AI Operations Preview
Alarms
Logs
Log groups New
Log Anomalies
Live Tail
Logs Insights New
Contributor Insights
Metrics
X-Ray traces New
Events
Application Signals
Network Monitoring New
Insights
Settings
Log events
Actions
Start tailing
Create metric filter
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
No older events at this moment. Retry
2025-04-07T05:30:40.686Z START RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c Version: $LATEST
2025-04-07T05:30:40.688Z [WARNING] 2025-04-07T05:30:40.688Z 055806fc-28e8-4ca7-aea4-151c1e71f82c LAMBDA_WARNING: Unhandled exception. The most likely caus…
2025-04-07T05:30:40.688Z [ERROR] JSONDecodeError: Expecting value: line 1 column 1 (char 0) Traceback (most recent call last): File "/var/task/lambda_fu…
2025-04-07T05:30:40.689Z END RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c
2025-04-07T05:30:40.689Z REPORT RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c Duration: 2.50 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Use…
2025-04-07T05:31:54.917Z START RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c Version: $LATEST
2025-04-07T05:31:54.918Z [WARNING] 2025-04-07T05:31:54.918Z 055806fc-28e8-4ca7-aea4-151c1e71f82c LAMBDA_WARNING: Unhandled exception. The most likely caus…
2025-04-07T05:31:54.918Z [ERROR] JSONDecodeError: Expecting value: line 1 column 1 (char 0) Traceback (most recent call last): File "/var/task/lambda_fu…
2025-04-07T05:31:54.940Z END RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c
2025-04-07T05:31:54.940Z REPORT RequestId: 055806fc-28e8-4ca7-aea4-151c1e71f82c Duration: 22.11 ms Billed Duration: 23 ms Memory Size: 128 MB Max Memory U…
No newer events at this moment. Auto retry paused. Resume
CloudShell
Feedback

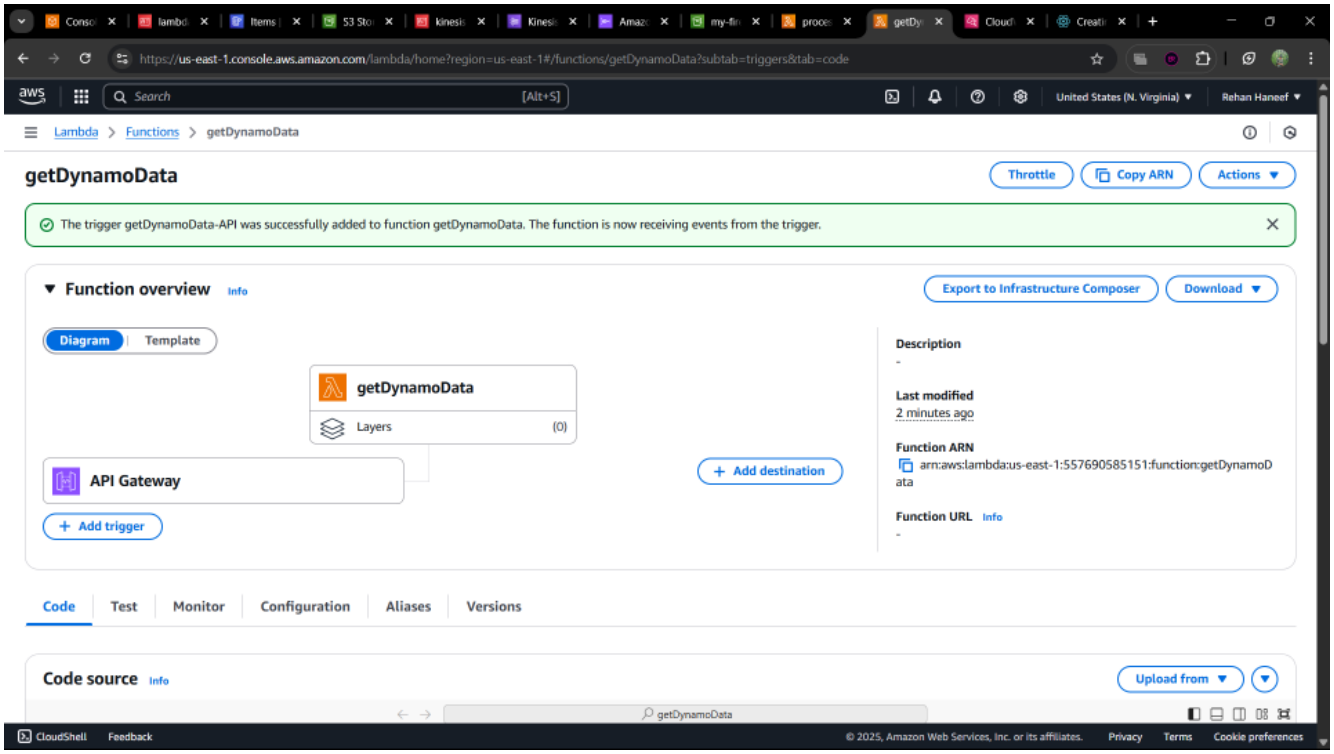


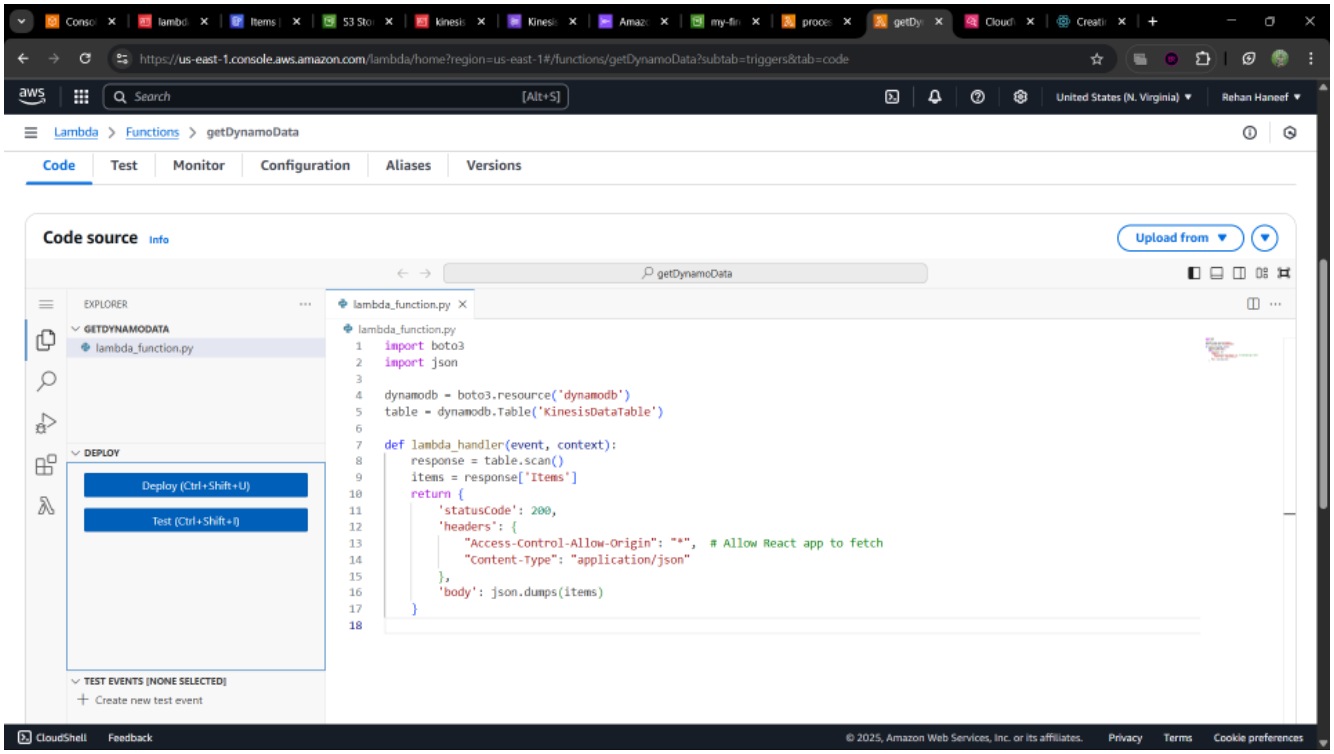
Fig(6): CloudWatch Metrics And Table in dynamDB



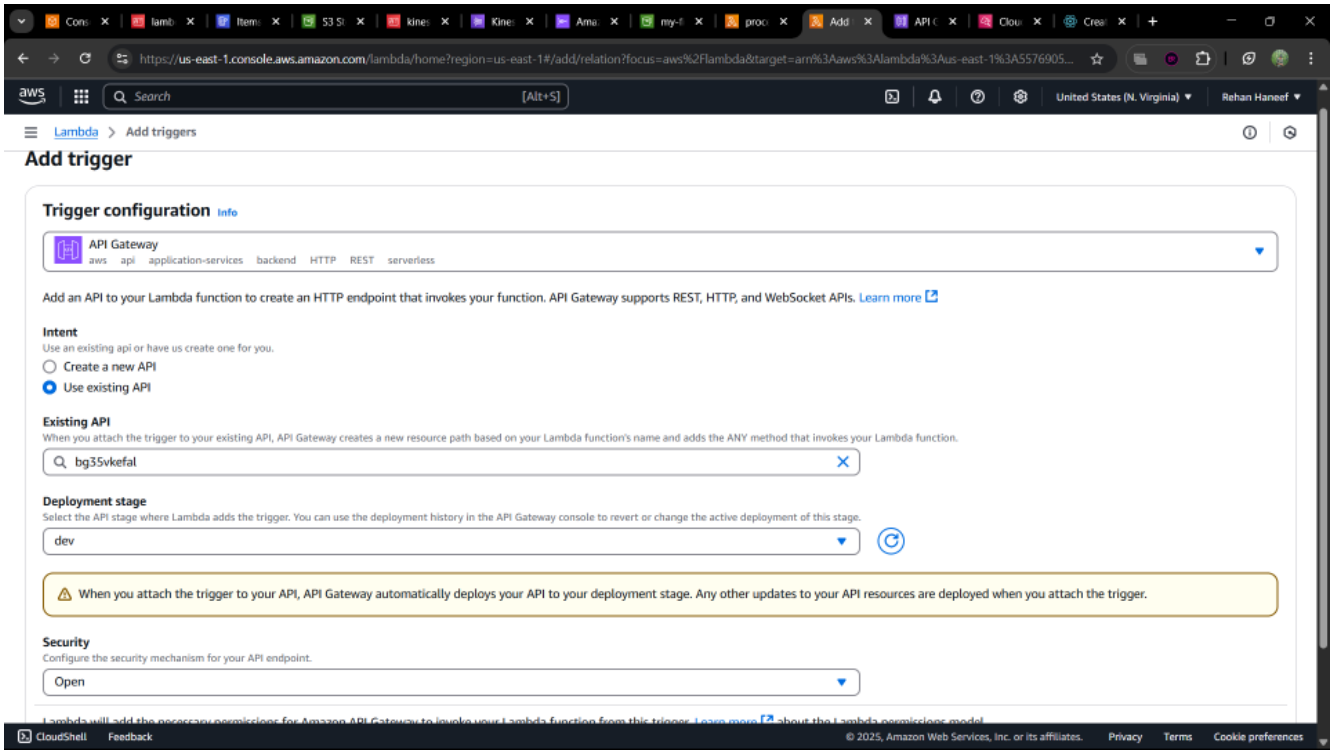


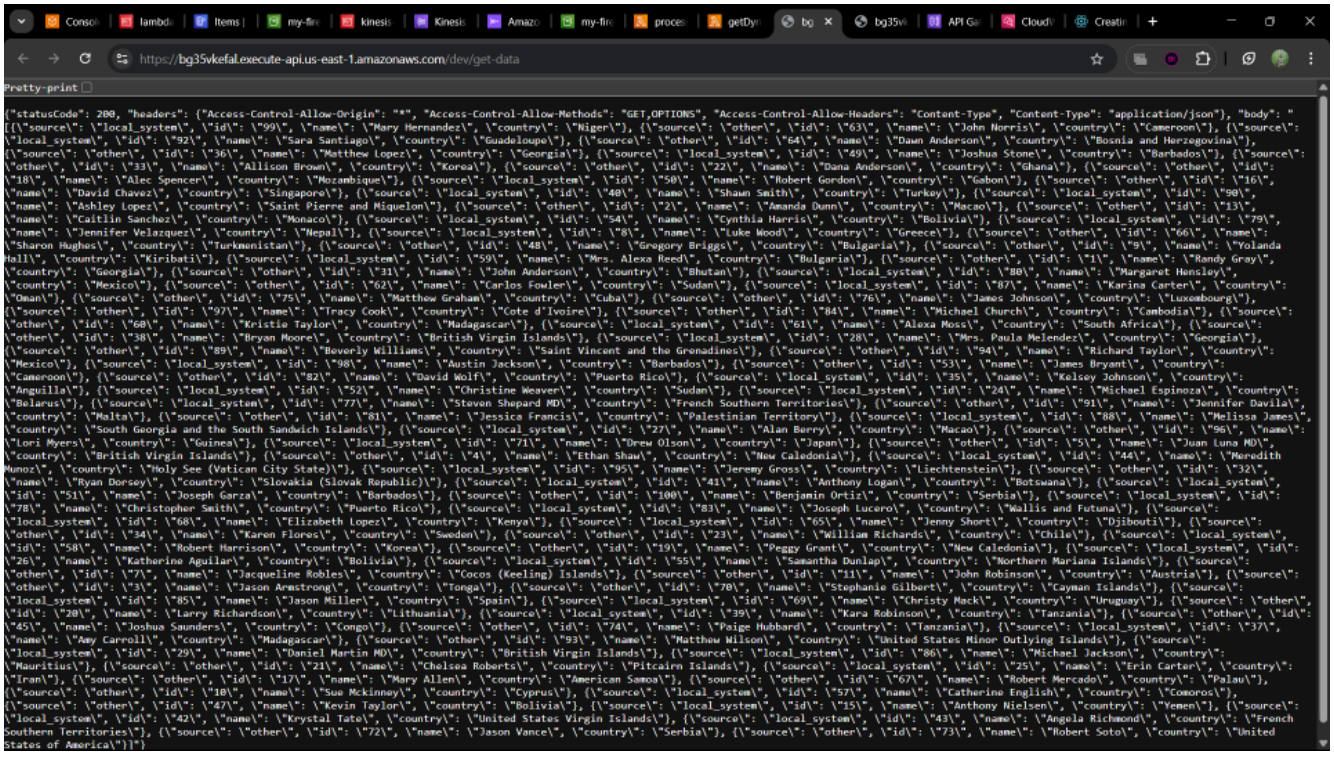
Fig(7): React Frontend Set up



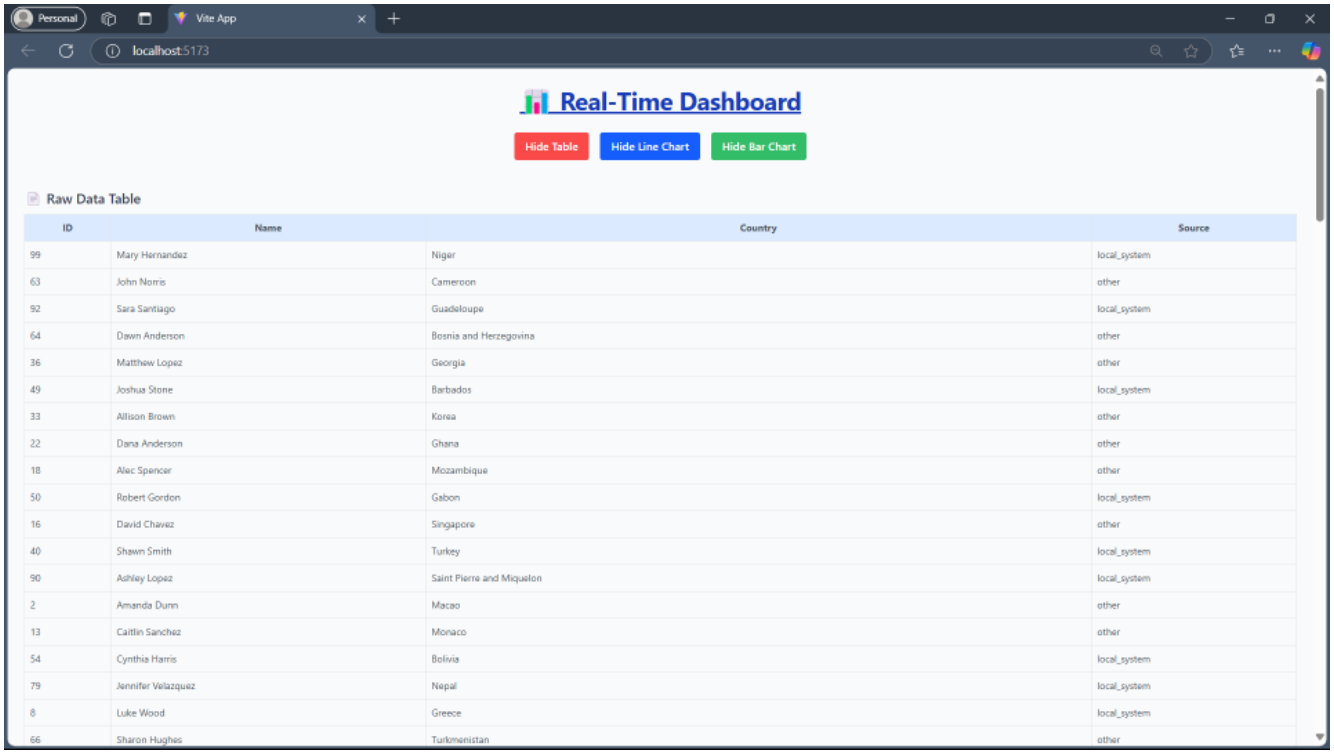


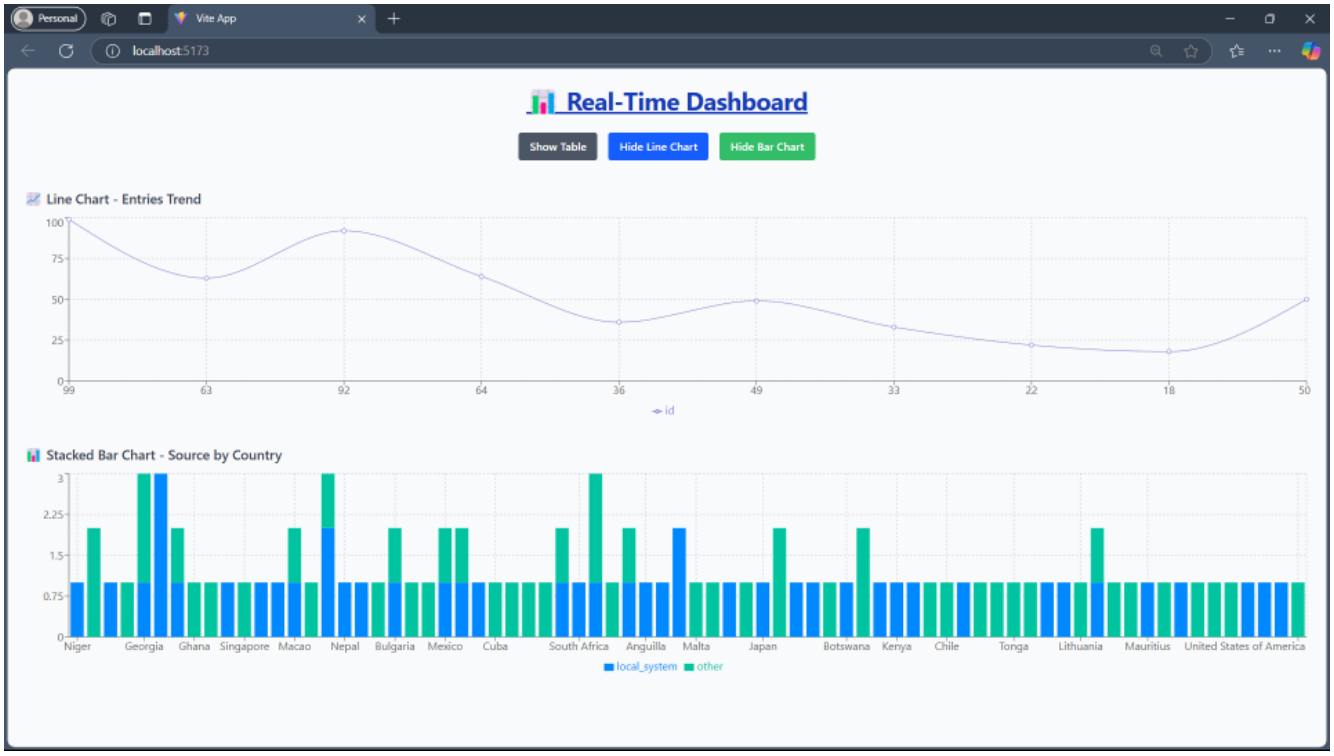
Fig(8): Lambda function-2



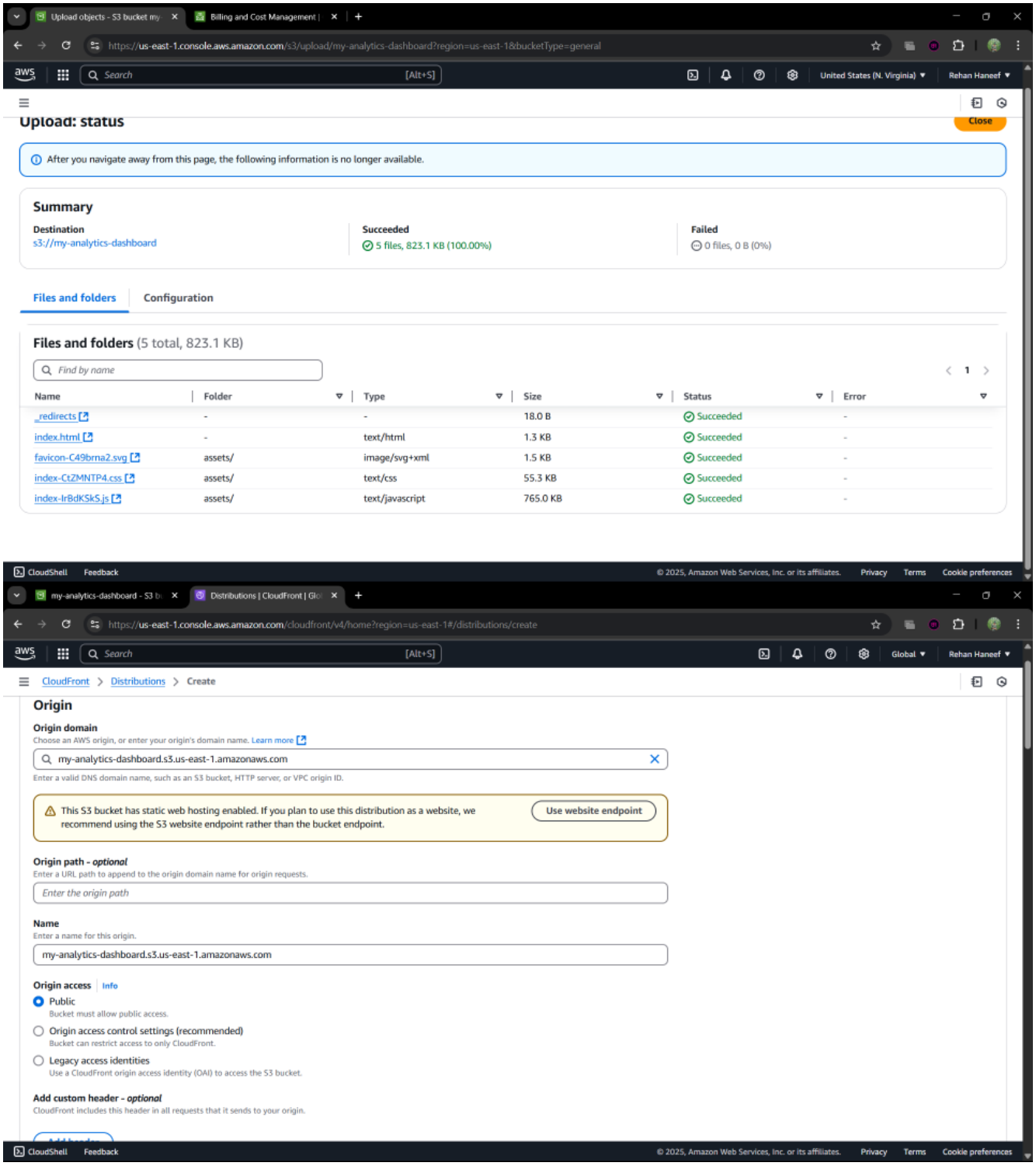


Fig(9): API gateway

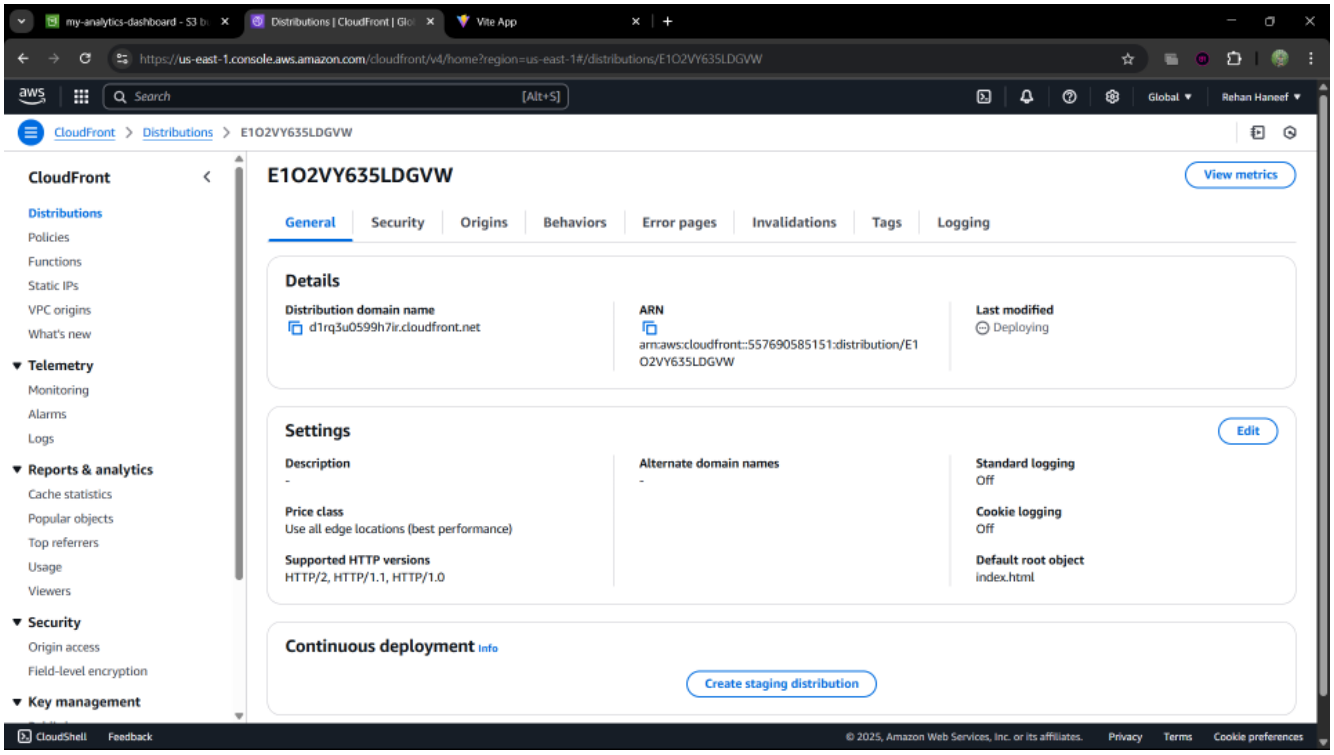


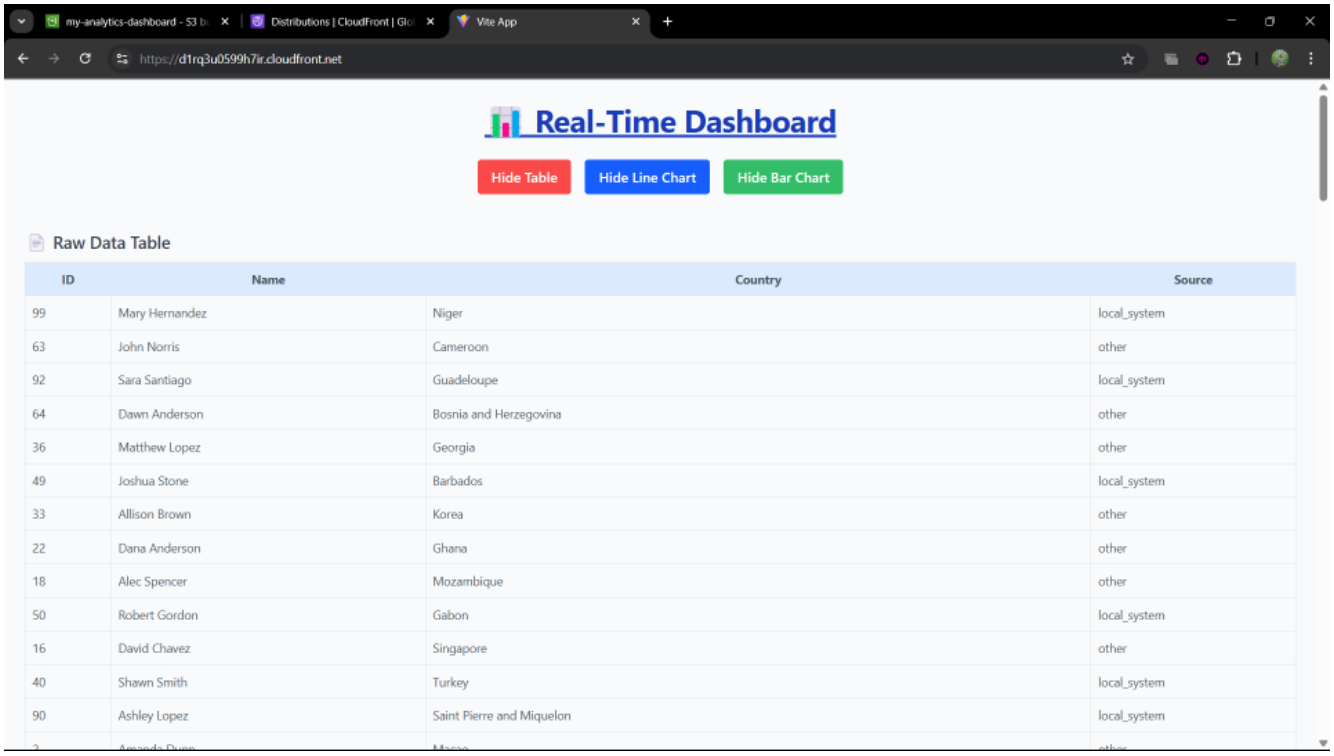


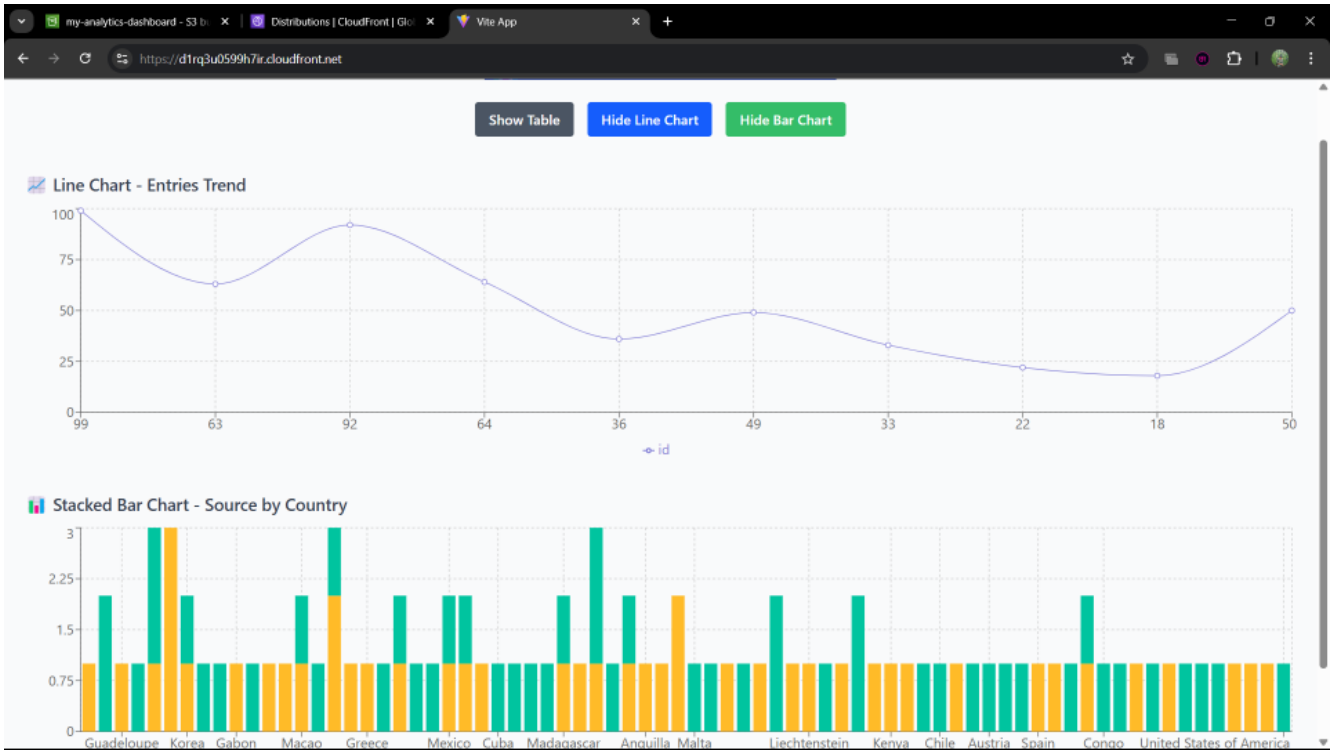
Fig(10): React Frontend Dashboard Output


Upload objects - S3 bucket my
Billing and Cost Management |
https://us-east-1.console.aws.amazon.com/s3/upload/my-analytics-dashboard?region=us-east-1&bucketType=general
Search
[Alt+S]
United States (N. Virginia)
Rehan Haneef
Upload: status
Close
After you navigate away from this page, the following information is no longer available.
Summary
Destination
s3://my-analytics-dashboard
Succeeded
5 files, 823.1 KB (100.00%)
Failed
0 files, 0 B (0%)
Files and folders
Configuration
Files and folders (5 total, 823.1 KB)
Find by name
Name
Folder
Type
Size
Status
Error
_redirects
index.html
favicon-C49brna2.svg
index-CtZMNTP4.css
index-IrBdKSkS.js
assets/
18.0 B
1.3 KB
text/html
image/svg+xml
1.5 KB
text/css
55.3 KB
text/javascript
765.0 KB
Succeeded
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

my-analytics-dashboard - S3 b
Distributions | CloudFront | Glo
https://us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/create
Search
[Alt+S]
Global
Rehan Haneef
CloudFront > Distributions > Create
Origin
Origin domain
Choose an AWS origin, or enter your origin's domain name. Learn more
my-analytics-dashboard.s3.us-east-1.amazonaws.com
Enter a valid DNS domain name, such as an S3 bucket, HTTP server, or VPC origin ID.
This S3 bucket has static web hosting enabled. If you plan to use this distribution as a website, we recommend using the S3 website endpoint rather than the bucket endpoint.
Use website endpoint
Origin path - optional
Enter a URL path to append to the origin domain name for origin requests.
Enter the origin path
Name
Enter a name for this origin.
my-analytics-dashboard.s3.us-east-1.amazonaws.com
Origin access
Info
Public
Bucket must allow public access.
Origin access control settings (recommended)
Bucket can restrict access to only CloudFront.
Legacy access identities
Use a CloudFront origin access identity (OAI) to access the S3 bucket.
Add custom header - optional
CloudFront includes this header in all requests that it sends to your origin.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


my-analytics-dashboard - S3 b
Distributions | CloudFront | Glo
Vite App
https://us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/E1O2VY635LDGVW
Search
[Alt+S]
Global
Rehan Haneef
CloudFront > Distributions > E1O2VY635LDGVW
CloudFront
Distributions
Policies
Functions
Static IPs
VPC origins
What's new
Telemetry
Monitoring
Alarms
Logs
Reports & analytics
Cache statistics
Popular objects
Top referrers
Usage
Viewers
Security
Origin access
Field-level encryption
Key management
E1O2VY635LDGVW
View metrics
General
Security
Origins
Behaviors
Error pages
Invalidations
Tags
Logging
Details
Distribution domain name
d1rq3u0599h7ir.cloudfront.net
ARN
arn:aws:cloudfront::557690585151:distribution/E1O2VY635LDGVW
Last modified
Deploying
Settings
Edit
Description
-
Alternate domain names
-
Standard logging
Off
Price class
Use all edge locations (best performance)
Cookie logging
Off
Supported HTTP versions
HTTP/2, HTTP/1.1, HTTP/1.0
Default root object
index.html
Continuous deployment Info
Create staging distribution
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


my-analytics-dashboard - S3 b
Distributions | CloudFront | Glo
Vite App
https://d1rq3u0599h7ir.cloudfront.net
Real-Time Dashboard
Hide Table
Hide Line Chart
Hide Bar Chart
Raw Data Table
ID
Name
Country
Source
99 Mary Hernandez Niger local_system
63 John Norris Cameroon other
92 Sara Santiago Guadeloupe local_system
64 Dawn Anderson Bosnia and Herzegovina other
36 Matthew Lopez Georgia other
49 Joshua Stone Barbados local_system
33 Allison Brown Korea other
22 Dana Anderson Ghana other
18 Alec Spencer Mozambique other
50 Robert Gordon Gabon local_system
16 David Chavez Singapore other
40 Shawn Smith Turkey local_system
90 Ashley Lopez Saint Pierre and Miquelon local_system



Fig(11): Cloudfront Deployed by build in React

Conclusion

In conclusion, the serverless data analytics pipeline built using AWS services for the Real-Time Data Analytics Pipeline using AWS project demonstrates a highly scalable, efficient, and cost-effective architecture for handling real-time data ingestion, storage, and processing. By leveraging key AWS components such as Amazon Kinesis, Lambda, DynamoDB, API Gateway, and S3, the platform achieved low-latency data processing and near-instant insights. The integration of a React frontend enhanced user experience with dynamic, real-time visualizations. Data quality was ensured through robust validation and preprocessing mechanisms using AWS Lambda, guaranteeing clean and accurate data for analysis. The platform's scalability, supported by DynamoDB's auto-scaling and Lambda's serverless compute, allowed it to handle varying workloads efficiently, while keeping operational costs low. Additionally, real-time insights were enabled by Kinesis, with alerts and monitoring provided by CloudWatch for proactive issue resolution. While challenges such as data latency during high-volume periods and cost management for DynamoDB throughput occurred, these were mitigated through thoughtful system design. Looking ahead, potential enhancements such as machine learning integration with Amazon SageMaker, historical data analytics with AWS Glue and Athena, and improved visualizations will further expand the capabilities of the platform, making it even more robust and adaptable to future requirements.

Future Work

The **Serverless Data Analytics Pipeline using AWS** can be enhanced in several ways to improve its scalability, security, and functionality:

- **Machine Learning Integration**: The integration of **Amazon SageMaker** for building predictive models could enhance the platform's capability to forecast future trends, such as customer purchasing behavior or product pricing changes. This could lead to more proactive decision-making and automation of certain processes.
- **Historical Data Analysis**: To extend the functionality of the pipeline, **AWS Glue** and **Amazon Athena** could be used to enable querying of archived data stored in **Amazon S3**. This would allow for long-term trend analysis, deeper insights into past data, and improved reporting over time.
- **Enhanced Visualizations**: Incorporating advanced charting libraries such as **Recharts** or **Chart.js** into the **React** frontend would provide real-time, interactive visualizations, making the data more accessible and actionable for users.
- **Role-Based Access Control (RBAC)**: By leveraging **Amazon Cognito**, secure user authentication and role-based access can be added to ensure that only authorized users have access to specific parts of the system, such as admin dashboards, product details, or pricing data.
- **Global Deployment**: For improved availability and performance, deploying the pipeline across multiple **AWS regions** would ensure low latency and high availability for users across the globe. This would allow the system to cater to a more global audience with optimized performance.
- **Advanced Monitoring and Alerting**: Integrating **Amazon SNS** or third-party services like **PagerDuty** for alerting and escalation would help ensure that critical operational issues are swiftly addressed. This integration would enable richer notifications and more advanced alerting workflows, improving system reliability and user experience.
- **Data Governance and Compliance**: To ensure better data quality, integrity, and compliance with industry regulations, implementing enhanced **data validation pipelines** and **logging** mechanisms would be crucial. This would provide a solid foundation for maintaining high standards of data governance and ensuring the accuracy and reliability of data across the pipeline.

References

1. **Amazon Web Services (AWS)**. (n.d.). *Amazon Kinesis Data Streams*. Retrieved from https://aws.amazon.com/kinesis/data-streams/
2. **Amazon Web Services (AWS)**. (n.d.). *AWS Lambda*. Retrieved from https://aws.amazon.com/lambda/
3. **Amazon Web Services (AWS)**. (n.d.). *Amazon DynamoDB*. Retrieved from https://aws.amazon.com/dynamodb/
4. **Amazon Web Services (AWS)**. (n.d.). *Amazon API Gateway*. Retrieved from https://aws.amazon.com/api-gateway/
5. **Amazon Web Services (AWS)**. (n.d.). *Amazon S3*. Retrieved from https://aws.amazon.com/s3/
6. **Amazon Web Services (AWS)**. (n.d.). *Amazon CloudFront*. Retrieved from https://aws.amazon.com/cloudfront/
7. **Amazon Web Services (AWS)**. (n.d.). *Amazon CloudWatch*. Retrieved from https://aws.amazon.com/cloudwatch/
8. **Amazon Web Services (AWS)**. (n.d.). *Amazon QuickSight*. Retrieved from https://aws.amazon.com/quicksight/
9. **Amazon Web Services (AWS)**. (n.d.). *AWS Glue*. Retrieved from https://aws.amazon.com/glue/
10. **React**. (n.d.). *React Documentation*. Retrieved from https://reactjs.org/docs/getting-started.html