

Serverless Real-time Analytics Dashboard with AWS Kinesis Analytics

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY (22SDCS07A)

by

Nalluri Tulasi

(2210030388)

Under the esteemed guidance of

**Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

*Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075*

April 2025

K L Deemed to be UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Certificate

This is Certified that the project entitled "**Serverless Real Time Analytics Dashboard with AWS Kinesis Analytics**" which is a experimental &/ Simulation work carried out by Nalluri Tulasi (2210030388), in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi

Course Coordinator

Dr. Arpita Gupta

Head of the Department

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

	Page No.
1. Introduction	1
2. AWS Services Used as part of the project	2
3. Steps involved in solving project problem statement	5
4. Stepwise Screenshots with brief description	8
5. Learning Outcomes	20
6. Conclusion	21
7. Future Enhancement	22
8. References	23

1. INTRODUCTION

In today's fast-paced digital world, businesses need real-time insights to stay competitive. Traditional analytics systems often struggle with scalability, latency, and operational costs, making serverless solutions a more efficient alternative. The Serverless Real-time Analytics Dashboard with AWS Kinesis Analytics leverages AWS Kinesis to process streaming data efficiently, offering a scalable, cost-effective, and high-performance solution [9].

AWS Kinesis Analytics enables businesses to analyze real-time data instantly without managing complex infrastructure. This is particularly valuable in industries like e-commerce, healthcare, gaming, and fraud detection, where immediate insights improve decision-making, security, and customer engagement [4][5][7]. In e-commerce, real-time data helps create AI-driven recommendation systems, improving user experience and conversion rates [1][2]. Fraud detection systems use AWS Kinesis to monitor transactions in real-time, identifying anomalies and preventing financial losses [7][9]. The gaming industry benefits from real-time analytics by detecting unusual player behavior and optimizing game mechanics dynamically [6].

By integrating services like Kinesis Firehose, Amazon S3, AWS Lambda, and QuickSight, this project aims to develop a serverless analytics dashboard that provides real-time data visualization and actionable insights. With high availability, fault tolerance, and scalability, this architecture ensures seamless integration into modern business environments [9].

This project demonstrates how AWS-native services can power real-time analytics, helping businesses process and interpret large volumes of streaming data efficiently. By leveraging serverless technologies, organizations can reduce infrastructure overhead while gaining deep insights into their operations in real time.

2. AWS SERVICES USED AS PART OF THE PROJECT

Amazon Kinesis – Real-time Data Streaming & Analytics

Amazon Kinesis is the backbone of the analytics pipeline, handling real-time data ingestion and transformation.

- Kinesis Data Streams (KDS) → Captures real-time data from various sources (e.g., logs, transactions, IoT devices). KDS provides high-throughput and low-latency streaming.
- Kinesis Data Firehose → A fully managed service that automatically loads streaming data into Amazon S3 for storage and further analysis.
- Kinesis Data Analytics (KDA) → Performs real-time transformations and analysis on streaming data using SQL-based processing before forwarding it to storage or further processing [9].



AWS Lambda – Serverless Compute for Data Processing

AWS Lambda processes streaming data in real time without managing servers. It can:

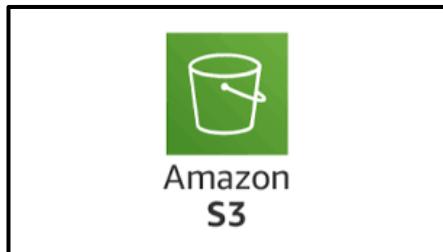
- Filter, aggregate, and transform data from Kinesis streams before storage.
- Trigger automated actions based on real-time analytics results.
- Seamlessly integrate with Kinesis, Firehose, and S3 for event-driven processing [9].



Amazon S3 – Scalable Storage for Streaming Data

Amazon S3 acts as the primary storage layer for raw and processed data. It provides:

- Highly durable and scalable storage for real-time event logs and processed analytics data.
- Seamless integration with other AWS services for further processing.
- Low-cost archival options for long-term data retention [9].



AWS IAM (Identity & Access Management) – Security & Access Control

AWS IAM ensures secure access and permissions across all AWS services in the project:

- Role-based access control (RBAC) restricts data access to authorized users and services.
- Encryption and authentication policies protect streaming and stored data.
- Integration with AWS CloudTrail allows tracking and auditing of access logs [9].



Amazon CloudWatch – Real-time Monitoring & Logging

Amazon CloudWatch collects logs, metrics, and alerts to monitor system performance and detect anomalies:

- Tracks Kinesis, Lambda, and Firehose performance to ensure seamless data flow.
- Detects failures, anomalies, and latency issues in real-time analytics.
- Triggers alerts for operational failures or unusual data patterns [9].



Amazon EC2 – Compute Environment for Code Execution

Amazon EC2 provides a flexible compute environment to write, deploy, and execute code for real-time analytics processing.

- Code Execution: EC2 instances serve as the primary development and execution environment for analytics scripts interacting with streaming data.
- Data Integration: EC2 integrates with Kinesis streams, Lambda, and S3 to process, analyze, and visualize real-time data efficiently [9].
- Scalability: EC2 allows running additional workloads, such as machine learning models for anomaly detection or predictive analytics [7].



3. STEPS INVOLVED IN SOLVING PROJECT PROBLEM STATEMENT

1. Understanding the Problem Statement

- Identify the data source – a temperature sensor continuously generating real-time temperature readings.
- Define real-time analytics requirements, such as monitoring temperature variations, triggering alerts for anomalies, and storing data for further analysis.
- Determine the scalability and latency constraints to ensure efficient real-time processing.

2. Designing the AWS Architecture

- Use Amazon Kinesis Data Streams for ingesting continuous real-time temperature data from IoT sensors [9].
- Implement Amazon Kinesis Data Firehose to load the streaming data into Amazon S3 for storage and further analysis [10].
- Deploy AWS Lambda to process incoming temperature data, check for anomalies (e.g., extremely high or low temperatures), and trigger alerts [9].
- Utilize Amazon EC2 instances to execute and test custom analytics scripts that process the temperature data in real time [7].
- Use Amazon CloudWatch to monitor system performance and log sensor data for insights [9].

3. Data Ingestion & Processing

- Simulate real-time temperature data from a temperature sensor.
- Stream the data into Kinesis Data Streams, ensuring low-latency ingestion.
- Use AWS Lambda to process and filter temperature readings, identifying extreme values.
- Store the processed temperature data in Amazon S3 for further analysis.

4. Security & Access Control

- Set up AWS IAM roles and policies to restrict access to AWS services (Kinesis, Lambda, EC2, and S3) [9].
- Implement encryption and secure authentication mechanisms to protect sensor data.

5. Monitoring & Optimization

- Use Amazon CloudWatch to monitor logs, track anomalies, and generate system alerts [9].
- Optimize AWS resource utilization (e.g., configuring EC2 auto-scaling, adjusting Kinesis shard capacity) to enhance performance.

6. Deployment & Testing

- Deploy the temperature monitoring system and verify real-time data flow.
- Test anomaly detection and alert triggers by simulating temperature variations.
- Optimize performance based on CloudWatch metrics.

7. Real-time Visualization & Alerts

- Monitor real-time temperature data via the AWS Dashboard.
- Trigger alerts when temperature readings exceed predefined thresholds.

4. STEPWISE SCREENSHOTS WITH BRIEF DESCRIPTION

Step 1: Create a IAM user to securely manage access to AWS services and resources.

- click on create user → enter the name → Attach policies required
 - AmazonS3FullAccess
 - AmazonAthenaFullAccess
 - AmazonKinesisFullAccess
- click on create user
- go to → security credentials → create Access Key

The screenshot shows the AWS IAM 'Users' page. On the left, there's a sidebar with 'Identity and Access Management (IAM)' and 'Access management' sections. The main area displays a message about Identity Center and lists 'Users (1)'. A table shows one user: 'kinesis_admin' with a checkmark in the 'User name' column. The table includes columns for Path, Group, Last activity, MFA, and Password age. At the top right, there are 'Delete' and 'Create user' buttons.

Fig 4.1.1 Creating the IAM user and also create the access key

The screenshot shows the 'Permissions' tab of the 'kinesis_admin' user details page. The sidebar on the left is identical to Fig 4.1.1. The main area shows 'Permissions policies (5)' attached to the user. A table lists five policies:

Policy name	Type	Attached via
AmazonAthenaFullAccess	AWS managed	Directly
AmazonKinesisFullAccess	AWS managed	Directly
AmazonQuickSight	Customer managed	Directly
AmazonS3FullAccess	AWS managed	Directly
IAMUserChangePassword	AWS managed	Directly

Fig 4.1.2 Adding the permissions/policies required for the project.

Step 2 : Create a EC2 Instance to Simulate Real-Time Streaming Data

- Click on launch instance → name the instance → select Amazon Linux → Instance type is t2.micro → create a key pair → click launch
- Instance is successfully created.
- After Creating the EC2 Instance click on connect and copy the SHH client

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with options like Dashboard, EC2 Global View, Events, Instances (selected), Images, and Elastic Block Store. The main area displays 'Instances (1/2)' with a search bar and filters for Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. Two instances are listed:

- kinesis-data-stream (i-053d2b0836dd9d1c3): Running, t2.micro, 2/2 checks passed. This instance is selected.
- kinesis-data-strea... (i-08cd04a9aa22af62a): Terminated, t2.micro.

Below the instances, there's a detailed view for the selected instance (i-053d2b0836dd9d1c3). It shows the instance summary with Public IPv4 address (70.231.17.1) and Private IPv4 addresses (172.31.32.100). The details tab is selected, followed by Status and alarms, Monitoring, Security, Networking, Storage, and Tags.

Fig 4.2.1 Successfully created EC2 Instance

The screenshot shows the 'Connect to instance' page for the kinesis-data-stream instance. The URL is <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ConnectToInstance:instanceId=i-053d2b0836dd9d1c3>. The page has tabs for EC2 Instance Connect, Session Manager, SSH client (selected), and EC2 serial console. The Instance ID is i-053d2b0836dd9d1c3 (kinesis-data-stream). Below the tabs, there's a list of steps to connect using an SSH client:

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is kinesis-KeyPair.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 "kinesis-KeyPair.pem"
4. Connect to your instance using its Public DNS:
ec2-3-84-31-47.compute-1.amazonaws.com

There's also an example command:

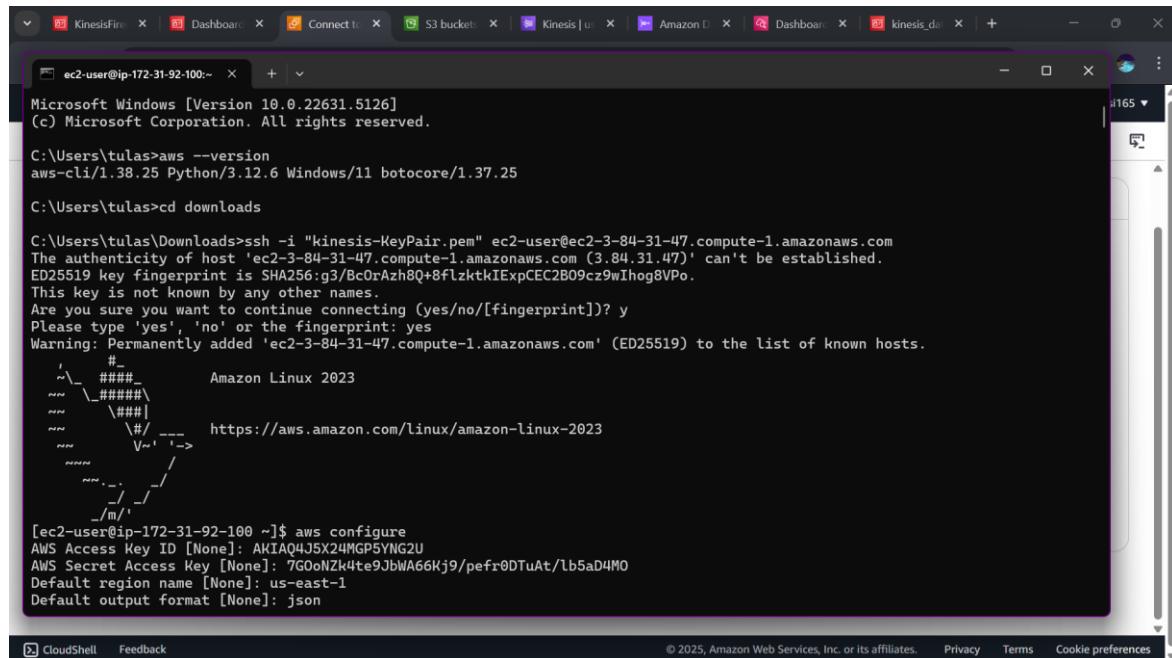
```
ssh -i "kinesis-KeyPair.pem" ec2-user@ec2-3-84-31-47.compute-1.amazonaws.com
```

A note at the bottom says: "Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username." The footer includes CloudShell, Feedback, and standard AWS links.

Fig 4.2.2 click on connect and copy the SHH client

Step 3 : Connecting to AWS CLI

- Open CMD → check for AWS CLI version → navigate to location where the keypair in downloaded → paste the SSH client → configure → enter access key , security key, region, output format.
- Install all the required packages for your project or code.



```

Microsoft Windows [Version 10.0.22631.5126]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tulas>aws --version
aws-cli/1.38.25 Python/3.12.6 Windows/11 botocore/1.37.25

C:\Users\tulas>cd downloads

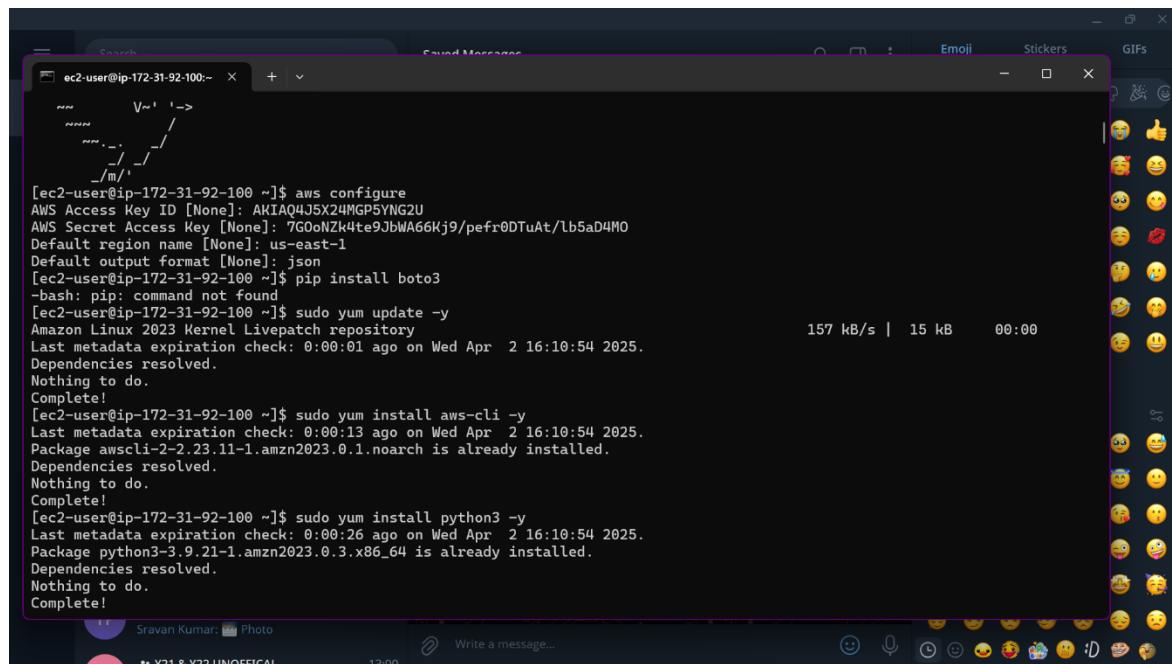
C:\Users\tulas\Downloads>ssh -i "kinesis-KeyPair.pem" ec2-user@ec2-3-84-31-47.compute-1.amazonaws.com
The authenticity of host 'ec2-3-84-31-47.compute-1.amazonaws.com (3.84.31.47)' can't be established.
ED25519 key fingerprint is SHA256:g3/BcOrAzh8Q+8flzktkIExpCEC2B09cz9wIhog8VPo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'ec2-3-84-31-47.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

          _#
  _\_\_ ##_#_      Amazon Linux 2023
  _\_\_ \####\_
  _\_\_ \###|_
  _\_\_ \#/ ___ https://aws.amazon.com/linux/amazon-linux-2023
  _\_\_ V~' '-->
  _\_\_ ._. /
  _\_\_ ._. /-
  _\_\_ /_/
  _\_\_ /_/
  _\_\_ /_/
  _\_\_ /_/
[ec2-user@ip-172-31-92-100 ~]$ aws configure
AWS Access Key ID [None]: AKIAQH4J5XK24MGP5YNG2U
AWS Secret Access Key [None]: 7GOoNZk4te9JbWA66Kj9/pefr0DTuAt/lb5aD4MO
Default region name [None]: us-east-1
Default output format [None]: json

```

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Fig 4.3.1 Configure the Instance with the terminal with the access key, secret key.



```

          V~' '-->
  _\_\_ ._. /
  _\_\_ ._. /-
  _\_\_ /_/
  _\_\_ /_/
  _\_\_ /_/
[ec2-user@ip-172-31-92-100 ~]$ aws configure
AWS Access Key ID [None]: AKIAQH4J5XK24MGP5YNG2U
AWS Secret Access Key [None]: 7GOoNZk4te9JbWA66Kj9/pefr0DTuAt/lb5aD4MO
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-31-92-100 ~]$ pip install boto3
-bash: pip: command not found
[ec2-user@ip-172-31-92-100 ~]$ sudo yum update -y
Amazon Linux 2023 Kernel Livepatch repository
Last metadata expiration check: 0:00:01 ago on Wed Apr  2 16:10:54 2025.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-92-100 ~]$ sudo yum install aws-cli -y
Last metadata expiration check: 0:00:13 ago on Wed Apr  2 16:10:54 2025.
Package awscli-2.2.31-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-92-100 ~]$ sudo yum install python3 -y
Last metadata expiration check: 0:00:26 ago on Wed Apr  2 16:10:54 2025.
Package python3-3.9.21-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!

```

Sravan Kumar: Photo Write a message... 13:00 157 kB/s | 15 kB 00:00

Smiley face icons are visible on the right side of the terminal window.

Fig 4.3.2 Installing all the required libraries and packages

```

Complete!
[ec2-user@ip-172-31-92-100 ~]$ sudo yum install python3 -y
Last metadata expiration check: 0:00:26 ago on Wed Apr 2 16:10:54 2025.
Package python3-3.9.21-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-92-100 ~]$ sudo yum install python3 python3-pip -y
Last metadata expiration check: 0:00:48 ago on Wed Apr 2 16:10:54 2025.
Package python3-3.9.21-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
=====
Package           Architecture      Version       Repository   Size
=====
Installing:
python3-pip        noarch          21.3.1-2.amzn2023.0.11    amazonlinux  1.8 M
Installing weak dependencies:
libcrypt-compat    x86_64          4.4.33-7.amzn2023    amazonlinux  92 k
=====
Transaction Summary
=====
Install 2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm           2.3 MB/s |  92 kB  00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.11.noarch.rpm           28 MB/s | 1.8 MB  00:00
=====
Total                                         21 MB/s | 1.9 MB  00:00

```

Fig 4.3.3 Checking the version of pip

Step 4 : Creating a Kinesis Data Stream

- Click on create data stream → name the data stream → data stream capacity(provisioned) → click create data stream
- Successfully created data stream.

real_time_data_Stream was successfully deleted.

Create data stream Info

Data stream configuration

Data stream name

real-time-data

Acceptable characters are uppercase and lowercase letters, numbers, underscores, hyphens and periods.

Data stream capacity Info

Capacity mode

On-demand
Use this mode when your data stream's throughput requirements are unpredictable and variable. With on-demand mode, your data stream's capacity scales automatically.

Provisioned
Use provisioned mode when you can reliably estimate your data stream's throughput. With provisioned mode, your data stream's capacity is fixed.

Fig 4.4.1 Create a Amazon Kinesis data stream to analyze the real time data.

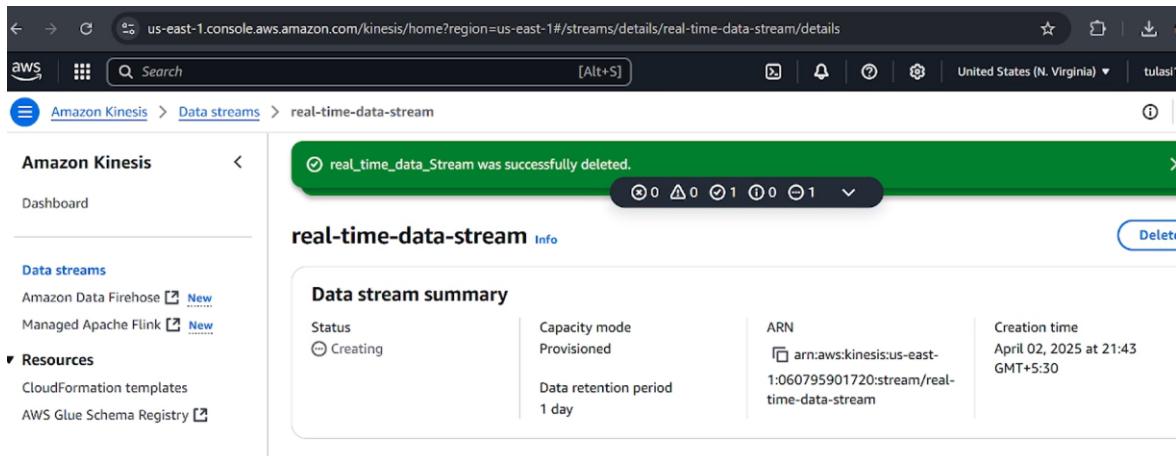


Fig 4.4.2 Wait until you get the active status.

Step 5 : Create a python file

- In cmd write a command (nano producer.py) → write your code which creates the temperature sensor real time data → save it.
- run the code using : Python3 producer.py (command)
- you can see the streaming data.
- You can see the data in the Kinesis data stream.

The terminal window shows two tabs. The top tab shows the output of the command "nano producer.py", which installs collected packages (botocore, s3transfer, boto3) and successfully installs them. The bottom tab shows the contents of the "producer.py" file:

```

ec2-user@ip-172-31-92-100:~$ nano producer.py
GNU nano 8.3
print(f"Sending: {data}")

# Send data to Kinesis stream
kinesis_client.put_record(
    StreamName=stream_name,
    Data=json.dumps(data),
    PartitionKey=data["sensor_id"]
)

time.sleep(2) # Simulating new data every 2 seconds

# Start streaming data
generate_data()

```

Fig 4.5.1 Create a python file(nano)

The screenshot shows a Slack message window titled "ec2-user@ip-172-31-92-100:" containing a series of JSON objects representing sensor data. Each object has a timestamp and temperature values for five different sensors (sensor_1 to sensor_5). The data is being sent at regular intervals. The Slack interface includes a sidebar with emoji and GIFs, and a bottom bar with message input and reaction icons.

```

Sending: {'sensor_id': 'sensor_4', 'temperature': 37.39, 'timestamp': 1743611738}
Sending: {'sensor_id': 'sensor_4', 'temperature': 25.99, 'timestamp': 1743611740}
Sending: {'sensor_id': 'sensor_4', 'temperature': 28.07, 'timestamp': 1743611742}
Sending: {'sensor_id': 'sensor_1', 'temperature': 26.88, 'timestamp': 1743611744}
Sending: {'sensor_id': 'sensor_4', 'temperature': 24.47, 'timestamp': 1743611747}
Sending: {'sensor_id': 'sensor_4', 'temperature': 23.0, 'timestamp': 1743611749}
Sending: {'sensor_id': 'sensor_2', 'temperature': 33.43, 'timestamp': 1743611751}
Sending: {'sensor_id': 'sensor_4', 'temperature': 25.34, 'timestamp': 1743611753}
Sending: {'sensor_id': 'sensor_1', 'temperature': 27.96, 'timestamp': 1743611755}
Sending: {'sensor_id': 'sensor_5', 'temperature': 37.42, 'timestamp': 1743611757}
Sending: {'sensor_id': 'sensor_1', 'temperature': 24.76, 'timestamp': 1743611759}
Sending: {'sensor_id': 'sensor_4', 'temperature': 22.24, 'timestamp': 1743611761}
Sending: {'sensor_id': 'sensor_4', 'temperature': 33.59, 'timestamp': 1743611763}
Sending: {'sensor_id': 'sensor_2', 'temperature': 30.67, 'timestamp': 1743611765}
Sending: {'sensor_id': 'sensor_1', 'temperature': 20.58, 'timestamp': 1743611767}
Sending: {'sensor_id': 'sensor_2', 'temperature': 21.09, 'timestamp': 1743611769}
Sending: {'sensor_id': 'sensor_2', 'temperature': 20.4, 'timestamp': 1743611771}
Sending: {'sensor_id': 'sensor_3', 'temperature': 28.75, 'timestamp': 1743611773}
Sending: {'sensor_id': 'sensor_4', 'temperature': 20.12, 'timestamp': 1743611775}
Sending: {'sensor_id': 'sensor_3', 'temperature': 26.34, 'timestamp': 1743611777}
Sending: {'sensor_id': 'sensor_5', 'temperature': 20.88, 'timestamp': 1743611779}
Sending: {'sensor_id': 'sensor_1', 'temperature': 27.17, 'timestamp': 1743611781}
Sending: {'sensor_id': 'sensor_5', 'temperature': 34.09, 'timestamp': 1743611783}
Sending: {'sensor_id': 'sensor_4', 'temperature': 38.16, 'timestamp': 1743611785}
Sending: {'sensor_id': 'sensor_2', 'temperature': 22.48, 'timestamp': 1743611787}
Sending: {'sensor_id': 'sensor_2', 'temperature': 20.38, 'timestamp': 1743611789}
Sending: {'sensor_id': 'sensor_5', 'temperature': 29.26, 'timestamp': 1743611791}
Sending: {'sensor_id': 'sensor_4', 'temperature': 20.63, 'timestamp': 1743611793}
Sending: {'sensor_id': 'sensor_2', 'temperature': 30.09, 'timestamp': 1743611795}

```

Fig 4.5.2 Here is the real time data

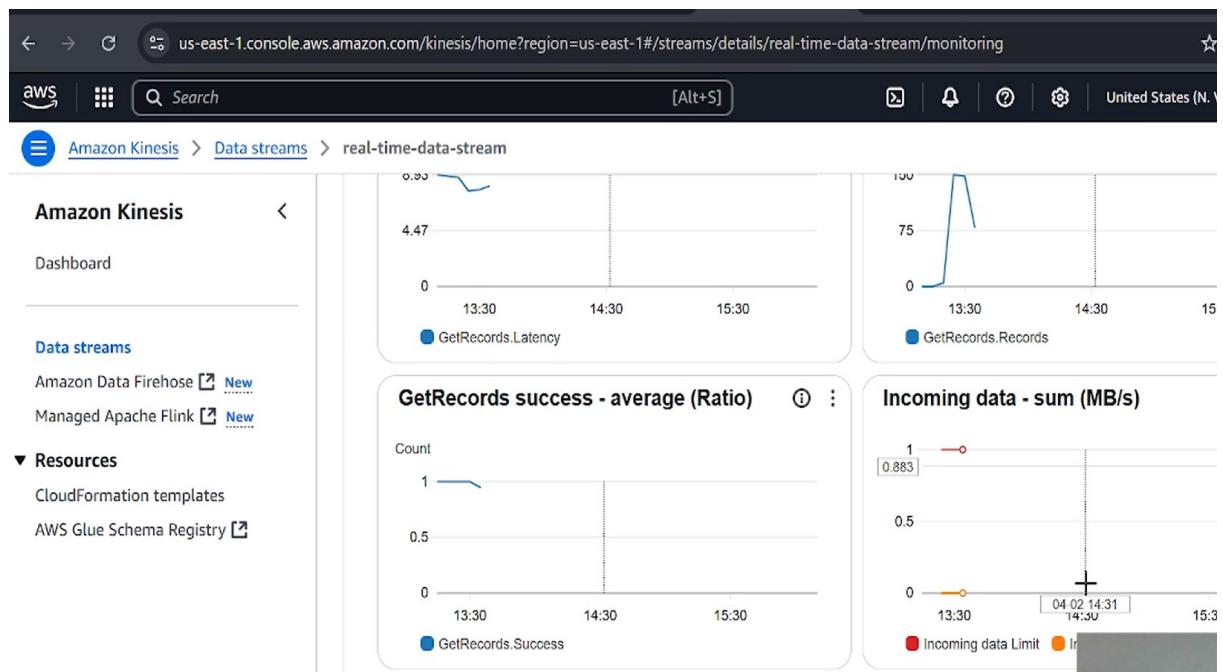


Fig 4.5.3 Here is the incoming data to the data stream .

Step 6 : Creating the S3 Bucket to store the data.

- Click create bucket → General purpose → name the bucket.
- Successfully created bucket.

The screenshot shows the AWS S3 console at the URL us-east-1.console.aws.amazon.com/s3/buckets?region=us-east-1&bucketType=general. The browser address bar and tabs are visible at the top. Below the header, there's a navigation bar with 'Amazon S3' and 'Buckets'. A green success message box displays: 'Successfully created bucket "kinesis-data-storage-2025"' and 'To upload files and folders, or to configure additional bucket settings, choose View details.' Below this, an 'Account snapshot - updated every 24 hours' section is shown, with a note about storage lens visibility. At the bottom, there are two tabs: 'General purpose buckets' (which is selected) and 'Directory buckets'. The main content area shows a table of general purpose buckets:

Name	AWS Region	IAM Access Analyzer
aws-athena-query-results-us-east-1-060795901720	US East (N. Virginia) us-east-1	View analyzer for us
kinesis-data-storage-2025	US East (N. Virginia) us-east-1	View analyzer for us

Fig 4.6 Create a bucket so that you can store the real time data in the AWS S3.

Step 7 : Creating Amazon Data Firehose to deliver the data from data stream to S3 bucket

- Click on create firehose → select the source and destination → name the firehose → add the IAM rule (If not exist create one)
 - Source : Amazon Kinesis Data Streams
 - Destination : Amazon S3
- Policies added :
 - Amazon Kinesis Firehose Full Access
 - Amazon Kinesis Full Access
 - Amazon S3 Full Access.
- Click create firehose.

The screenshot shows the AWS Lambda console interface for creating a new function. The top navigation bar includes the AWS logo, search bar, and various navigation icons. The main content area is titled "Create Lambda function" and displays the second step: "Set runtime and add triggers". A large callout box highlights the "Runtime" dropdown menu, which is currently set to "Node.js 14.x". Below the dropdown, there are sections for "Add trigger" and "Configure environment variables". The bottom of the screen shows the "Next Step" button.

Fig 4.7.1 Creating Kinesis Firehose to deliver the data from data stream to S3 .

The screenshot shows the AWS Lambda console interface for creating a new function. The top navigation bar includes the AWS logo, search bar, and various navigation icons. The main content area is titled "Create Lambda function" and displays the second step: "Set runtime and add triggers". A large callout box highlights the "Runtime" dropdown menu, which is currently set to "Node.js 14.x". Below the dropdown, there are sections for "Add trigger" and "Configure environment variables". The bottom of the screen shows the "Next Step" button.

Fig 4.7.2 Name the firehose.

aws | Search [Alt+S] | X | ?

Amazon Data Firehose > Firehose streams > Create Firehose stream

Server-side encryption (SSE)
Amazon Data Firehose does not support SSE for Firehose stream with Kinesis data stream source as no data is stored at rest. Go to data at rest on the selected Kinesis data stream. If you choose Direct PUT for your Firehose stream, you can enable SSE on the Firehose stream.

Amazon CloudWatch error logging | Info
Choose Enabled if you want Amazon Data Firehose to log record delivery errors to CloudWatch Logs.
 Not enabled
 Enabled

Service access | Info
Amazon Data Firehose uses this IAM role for all the permissions that the Firehose stream needs. To specify different roles for the different permissions, use the IAM role editor.
 Create or update IAM role **KinesisFirehoseServiceRole-kinesis-data--us-east-1-1743610654795**
Creates a new role or updates an existing one and adds the required policy to it, and enables Amazon Data Firehose to assume it.
 Choose existing IAM role
The role that you choose must have policies that include the permissions that Amazon Data Firehose needs.

Tags | Info
You can add tags to organize your AWS resources, track costs, and control access.
No tags associated with the resource.

Fig 4.7.3 Create a new IAM role for firehose.

aws | Search [Alt+S] | X | ?

IAM > Roles > Create role

Step 1
 Select trusted entity
 Step 2
 Step 3
 Name, review, and create

Select trusted entity | Info

Trusted entity type

- AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation Allows users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy Create a custom trust policy to enable others to perform actions in this account.

Step 2: Add permissions

Policy name	Type	Actions
AmazonKinesisFirehoseFullAccess	AWS managed	Pe
AmazonKinesisFullAccess	AWS managed	Pe
AmazonS3FullAccess	AWS managed	Pe

Step 3: Add tags

Add tags - optional | Info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.
No tags associated with the resource.

Fig 4.7.4 IAM rule

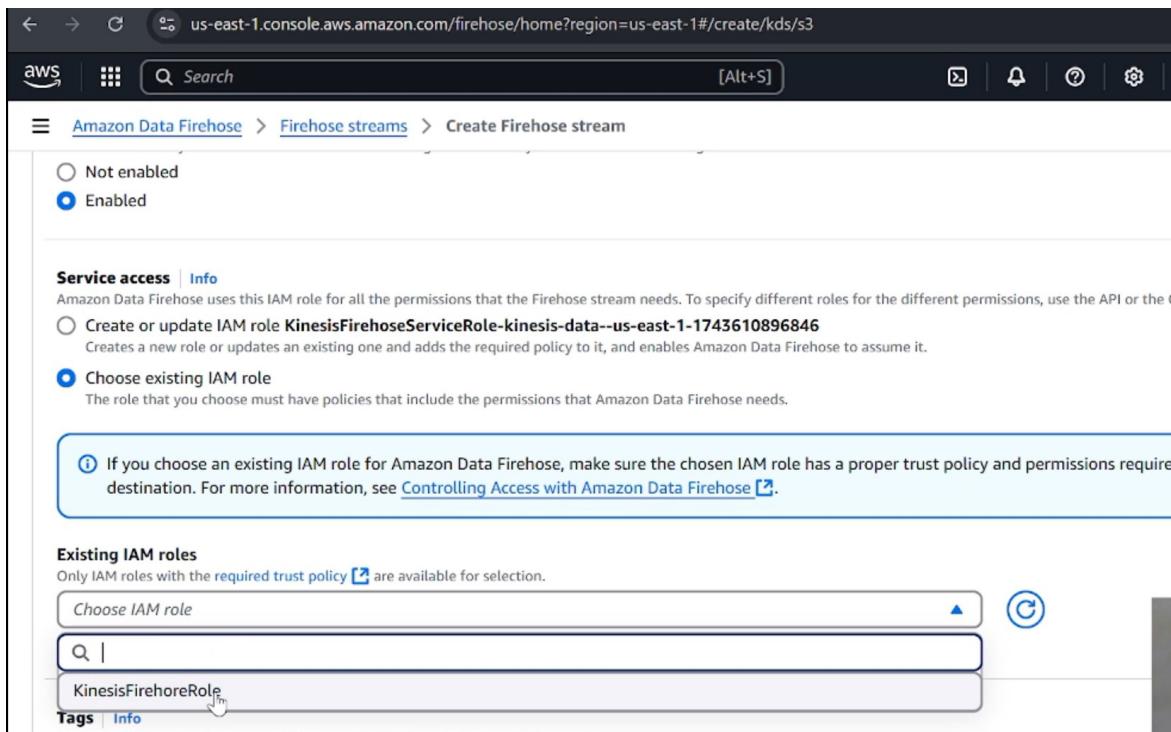


Fig 4.7.5 Select the created role for Kinesis Firehose.

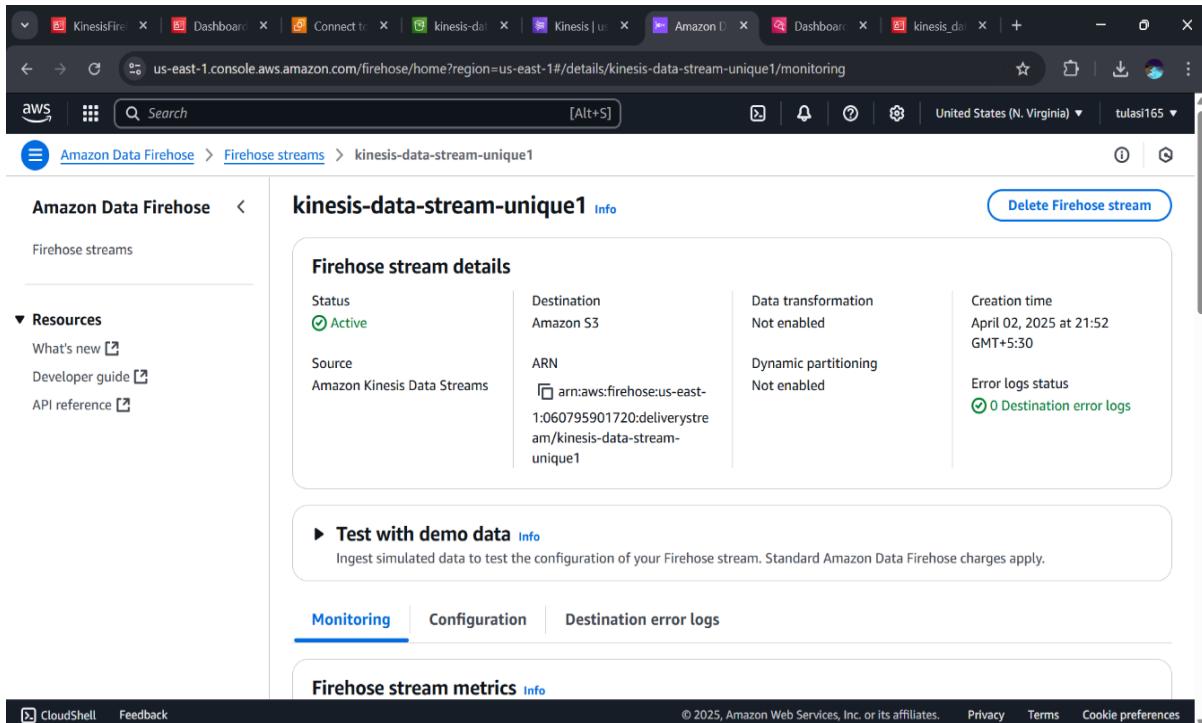


Fig 4.7.6 Kinesis firehose is successfully created.

Step 8: If still there are no objects in the bucket then edit the bucket policy

- Click on permissions → edit bucket policy.
- You can now see the objects in the Bucket.

The screenshot shows the AWS S3 console interface. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/kinesis-data-storage-2025?region=us-east-1&bucketType=general&tab=objects`. The page title is "kinesis-data-storage-2025". Below the title, there are tabs for Objects, Metadata, Properties, Permissions, Metrics, Management, and Access Points. The "Objects" tab is selected. It displays a message: "Objects (0)". There are buttons for Copy S3 URI, Copy URL, Download, Open, Delete, Actions, and Create. A search bar says "Find objects by prefix". Below the search bar, there are filters for Name, Type, Last modified, and Size. A message says "No objects" and "You don't have any objects in this bucket." A blue "Upload" button is visible.

Fig 4 .8.1 Still there is no object in the bucket , you need to edit the bucket policy .

The screenshot shows the "Edit bucket policy" page for the "kinesis-data-storage-2025" bucket. The URL in the address bar is `us-east-1.console.aws.amazon.com/s3/buckets/kinesis-data-storage-2025>Edit bucket policy`. The page title is "kinesis-data-storage-2025". On the left, there is a code editor with the following JSON policy:

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Principal": "*",
7        "Service": "firehose.amazonaws.com",
8      },
9      {
10        "Action": "s3:PutObject",
11        "Resource": "arnaws:s3:::kinesis-data-storage-2025/*",
12        "Condition": {
13          "StringEquals": {
14            "aws:SourceAccount": "06079991728"
15          }
16        }
17      }
18    ]
}
```

To the right of the code editor, there is a sidebar with "Edit statement", "Add actions", "Choose a service", and dropdown menus for "Included" (S3), "Available" (AI Operations, AMP), and "API Gateways". The main content area shows the bucket details again, including the "Objects" tab which lists one object named "2025/".

Fig 4 .8.2 Edit the bucket policy and save changes.

Step 9 : Create the dashboard for the Kinesis data stream.

- Check the Firehose monitoring for data incoming and outgoing.
- Click on add to dashboard → create a dashboard → name the dashboard
- You can see the final dashboard.

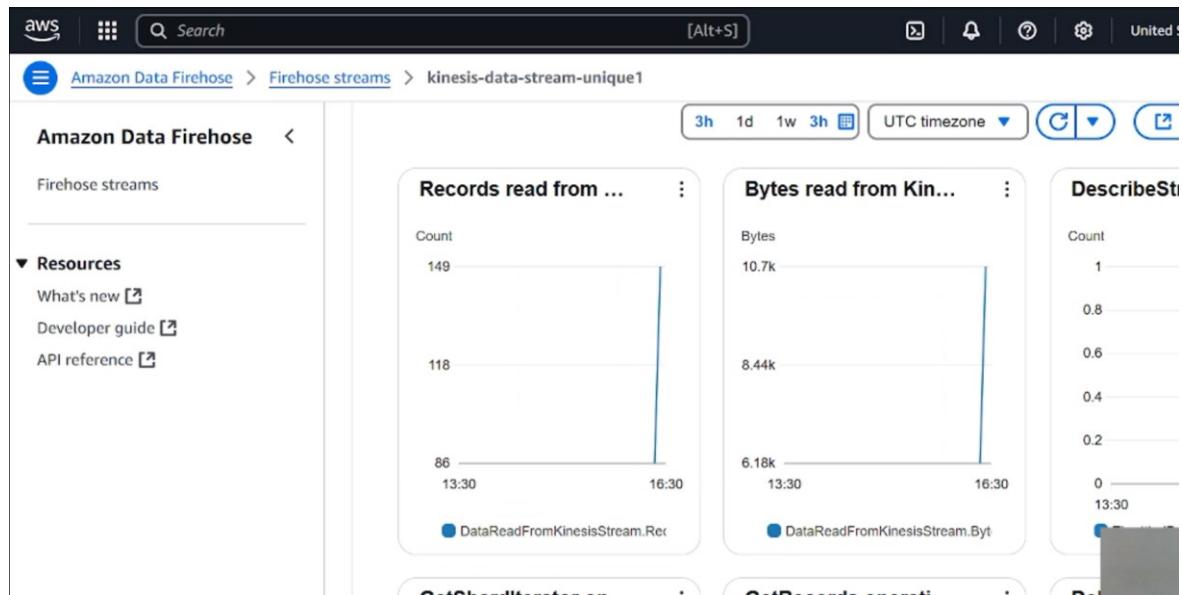


Fig 4 .9.1 Here is the graph with shows records read form kinesis and records delivered to S3.

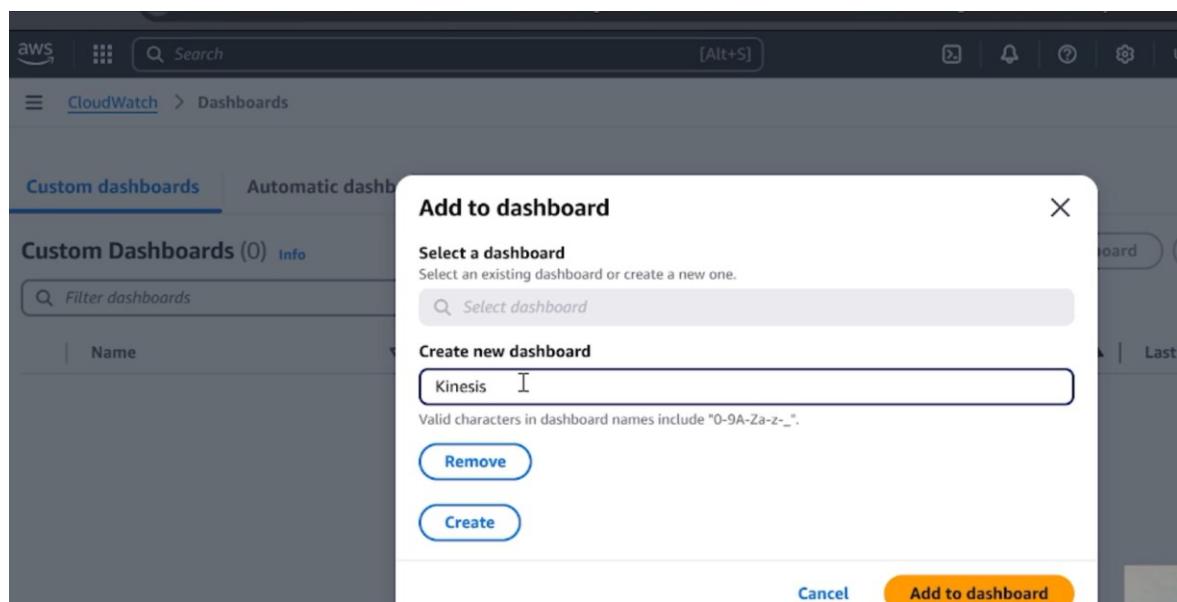


Fig 4.9.2 Dasboard creation

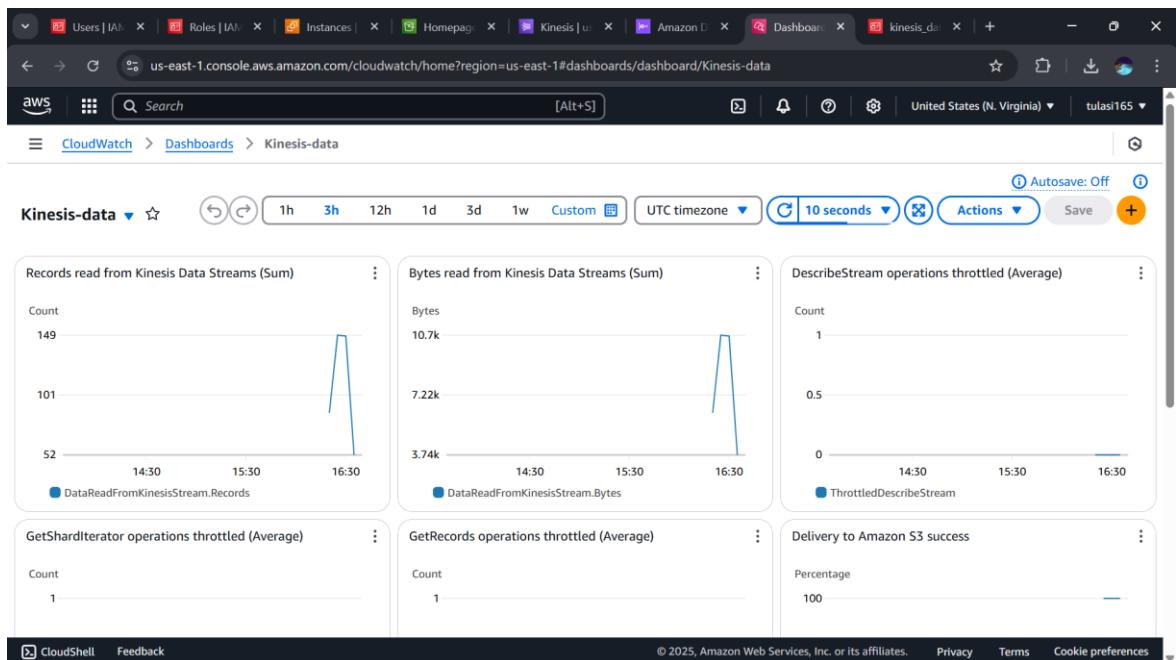


Fig 4.9.3 Here is the dashboard for real time data(temperature sensor data) using Amazon Kinesis.

5. LEARNING OUTCOMES

Through the development of the Serverless Real-time Analytics Dashboard using AWS Kinesis, several key technical and conceptual skills were acquired:

Understanding AWS Kinesis for Real-Time Data Streaming

This project provided hands-on experience in ingesting and processing high-velocity data streams using Amazon Kinesis Data Streams, enabling efficient handling of continuous data flows [1].

EC2 for Running Serverless Analytics Dashboard

The use of Amazon EC2 to execute and manage real-time analytics showcased the ability to integrate server-based resources with serverless services, ensuring efficient computational capabilities for data processing [2].

Data Storage and Pipeline Optimization with Amazon S3 and Firehose

Leveraging Amazon S3 as a data lake helped in storing raw and processed data efficiently. Additionally, Amazon Kinesis Firehose was used for automatic data delivery and transformation into storage systems, enhancing the robustness of real-time pipelines [3].

Enhanced Monitoring for Analytics

The project reinforced skills in AWS CloudWatch for real-time system monitoring and logging, helping in performance optimization and failure detection [4].

Scalability and Fault Tolerance in Cloud Architectures

Designing a system capable of handling high-throughput data streams emphasized the principles of fault tolerance, high availability, and scalability in cloud-based architectures [6].

This project has enhanced the understanding of cloud-native architectures and the practical applications of AWS services in real-time analytics.

6. CONCLUSION

The Serverless Real-time Analytics Dashboard using AWS Kinesis demonstrated how AWS services like Kinesis Data Streams, Firehose, S3, EC2, and CloudWatch can efficiently process, store, and analyze streaming data in real time. The project involved creating a temperature sensor data pipeline, which captured and visualized real-time data on an AWS dashboard, enhancing skills in IoT data processing and cloud storage. It also highlighted the design of fault-tolerant and scalable architectures using AWS-managed services, reducing operational complexity while ensuring high performance. This hands-on experience strengthened proficiency in cloud computing, data engineering, and system monitoring, laying a solid foundation for future advancements in cloud-based analytics and IoT applications.

7. FUTURE ENHANCEMENTS

To further improve the functionality and scalability of the serverless real-time analytics dashboard, several enhancements can be considered. Integrating Amazon SNS or EventBridge can enable real-time alerting to notify users of critical thresholds or anomalies. Incorporating Amazon SageMaker or Lookout for Metrics can bring predictive analytics and anomaly detection capabilities, allowing the system to forecast trends and automatically identify unusual patterns. Additionally, a custom front-end dashboard using React, API Gateway, and Lambda can replace QuickSight for greater interactivity and user control. Implementing AWS Cognito and fine-grained IAM policies would enhance security through role-based access management. For long-term storage optimization, S3 lifecycle policies with Glacier archiving can be utilized. Moreover, expanding data ingestion from multiple sources, adding real-time data transformation with Lambda in Firehose, and deploying across multiple regions for high availability can significantly elevate the system's performance, reliability, and user experience.

8. REFERENCES

- [1] Amazon Personalize Customer Outreach on Your E-commerce Platform:
<https://aws.amazon.com/blogs/architecture/amazon-personalize-customer-outreach-on-yourecommerce-platform/>
- [2] E-commerce Personalization with AWS: <https://www.brilworks.com/blog/ecommerce-personalization-with-aws/>
- [3] Research Paper on E-commerce Personalization (arXiv): <https://arxiv.org/abs/2105.06508>
- [4] Real-time Analytics Use Cases and Examples: <https://www.striim.com/blog/real-time-analytics-use-cases-and-examples/#use>
- [5] AWS in Healthcare Industry: <https://aws.amazon.com/blogs/industries/category/industries/healthcare/>
- [6] Detect Game Anomalies with Amazon Lookout for Metrics:
<https://aws.amazon.com/blogs/gametech/detect-game-anomalies-amazon-lookout-for-metricsgame-analytics-pipeline/>
- [7] Real-time Fraud Detection Using AWS Serverless and Machine Learning Services:
<https://aws.amazon.com/blogs/machine-learning/real-time-fraud-detection-using-aws-serverlessand-machine-learning-services/>
- [8] Sentiment Analysis of Social Media – Big Data Analytics Options:
<https://docs.aws.amazon.com/whitepapers/latest/big-data-analytics-options/example-3-sentimentanalysis-of-social-media.html>
- [9] Log Analytics with AWS: <https://aws.amazon.com/what-is/log-analytics/#:~:text=cost%20of%20operations.-,How%20does%20log%20analytics%20with%20AWS%20work?,for%20real%2Dtime%20operational%20intelligence.>
- [10] Amazon Kinesis Video Streams: <https://aws.amazon.com/kinesis/video-streams/?amazon-kinesis-video-streams-resourcesblog.sort-by=item.additionalFields.createdDate&amazon-kinesis-video-streams-resourcesblog.sort-order=desc>