

## **AWS Data Confluence High-Velocity Streaming to Immediate BI**

"AWS Real-Time Data Confluence" is a lightweight, serverless data streaming and analytics system built entirely on Amazon Web Services (AWS). The project demonstrates how real-time events generated from a web interface can be ingested, stored, processed, and analyzed using a fully cloud-native pipeline. It uses Amazon S3 as a scalable data lake, AWS Lambda for serverless compute, Amazon API Gateway for secure ingestion, AWS Glue for automated schema discovery, and Amazon Athena for instant SQL analytics. A simple React interface sends event data (such as user activity, transactions, or application logs) directly into the streaming pipeline, making the system easy to test, extend, and visualize.

The user experience begins from the React application, where any event can be triggered through a button click or form submission. The app sends this data in real-time to API Gateway, which acts as a secure entry point. API Gateway forwards the data to a Lambda function, which processes and stores each event into the S3 data lake using time-partitioned folders. This ensures high scalability, low cost, and immediate availability of data for analytical workloads.

AWS Glue automatically crawls the S3 bucket to detect incoming files and build a table structure in the Glue Data Catalog. This turns raw JSON events into queryable datasets without requiring manual schema creation. Once the tables are created, Amazon Athena can run SQL queries directly on top of the S3 data, enabling instant insights such as counting events, categorizing types, or analyzing attributes. Because Athena is serverless, results are delivered instantly without maintaining any servers or databases.

### **Scenario 1: Live Event Logging**

A developer pushes a button in the React application that generates a "user\_action" event. The API Gateway receives it instantly, triggering the Lambda function, which stores the event in S3 with a timestamp. Minutes later, the developer runs an Athena SQL query to view all actions logged in real time.

### **Scenario 2: Application Monitoring**

A small web application sends system health logs to the same API endpoint. Without modifying infrastructure, Lambda stores each log entry in S3. Using Athena, the team can check error counts, latency patterns, and usage trends directly through SQL.

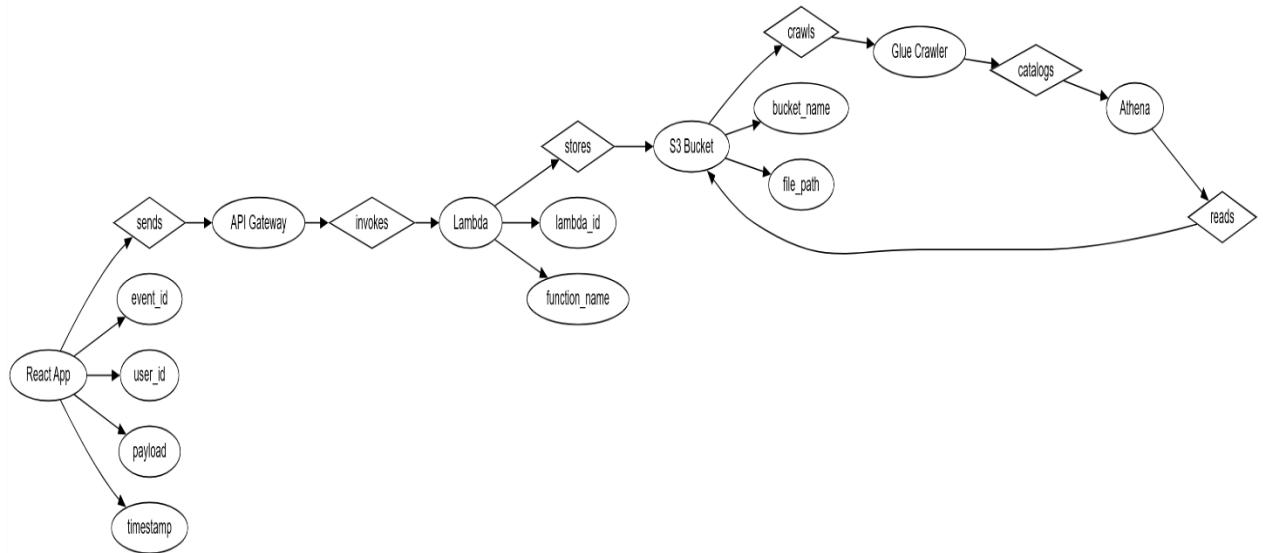
## Scenario 3: Sales Data Stream

An e-commerce demo app sends mock transaction data (amount, product, time). Glue detects the new schema and updates the catalog. Analysts can instantly query revenue, top-selling categories, or hourly sales trends without any storage provisioning.

### Technical Diagram:



## ER Diagram:



## Prior Knowledge:

1. AWS Account Setup:

[https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk\\_M4FfVM-Dh](https://youtu.be/CjKhQoYeR4Q?si=ui8Bvk_M4FfVM-Dh)

2. Web Application Stack : [React](#) || [Javascript fetch API](#)

3. Amazon S3 (Simple Storage Service):

<https://youtu.be/A2N9OIun9dU?si=DguWsoaEqOL8dF1x>

4. AWS Lambda:

[https://www.youtube.com/results?search\\_query=aws+lambda+tutorial](https://www.youtube.com/results?search_query=aws+lambda+tutorial)

5. Amazon S3 (Simple Storage Service):

[https://www.youtube.com/results?search\\_query=aws+s3+tutorial](https://www.youtube.com/results?search_query=aws+s3+tutorial)

6. Amazon API Gateway:

[https://www.youtube.com/results?search\\_query=api+gateway+tutorial](https://www.youtube.com/results?search_query=api+gateway+tutorial)

7. AWS Glue :

[https://www.youtube.com/results?search\\_query=aws+glue+tutorial](https://www.youtube.com/results?search_query=aws+glue+tutorial)

8. Amazon Athena:

[https://www.youtube.com/results?search\\_query=amazon+athena+tutorial](https://www.youtube.com/results?search_query=amazon+athena+tutorial)

9. AWS CloudWatch:

[https://www.youtube.com/results?search\\_query=aws+cloudwatch+tutorial](https://www.youtube.com/results?search_query=aws+cloudwatch+tutorial)

1

10. AWS IAM:

[https://youtu.be/\\_ZCTvmaPgao?si=bydbpAgyRN8x7GhZ](https://youtu.be/_ZCTvmaPgao?si=bydbpAgyRN8x7GhZ)

## **Project Workflow:**

### **1. AWS Account Setup**

- Create an AWS account and configure billing settings.
- Access the AWS Management Console to set up services.

### **2. Create an S3 Bucket (Data Lake Storage)**

- Open S3 and create a new bucket to store all application data.
- Enable public access restrictions for security.
- Create folders/prefixes to organize incoming JSON data.
- This bucket becomes the central Data Lake used by Glue and Athena.

### **3. Create IAM Role for Lambda**

- Go to IAM → Create Role.
- Choose trusted entity: **AWS Lambda**.
- Attach only these policies:
  - **AWSLambdaBasicExecutionRole**
  - **AmazonS3FullAccess** (or restricted S3 access for your bucket)
- Name it: **LambdaS3WriterRole**
- This role allows Lambda to write processed data into S3.

## 4. Create Lambda Function (Processing Layer)

- Go to Lambda → Create function → Author from scratch.
- Select Python/Node runtime.
- Assign your IAM role (**LambdaS3WriterRole**).
- Add code to receive input, validate it, and upload JSON objects into S3.
- Test the function using sample payloads in the Lambda test console.

## 5. Create API Gateway (Data Ingestion Endpoint)

- Open API Gateway → Create REST API.
- Create resource: **/stream**.
- Add POST method linked to your Lambda function.
- Enable CORS for your React application.
- Deploy the API to a stage (e.g., **prod**) and copy the Invoke URL.
- This endpoint will act as your real-time streaming input API.

## 6. Build the React Application (Frontend Layer)

- Create a simple UI using ReactJS.
- Add a form that accepts values and sends them to API Gateway using **fetch()**.
- Display submitted values or acknowledgements from the API.
- Host locally or deploy elsewhere (optional).
- This UI becomes the front-facing ingestion dashboard.

## 7. Run AWS Glue Crawler (Schema Discovery)

- Go to Glue → Create Crawler.
- Point the crawler to your S3 bucket where Lambda stores JSON files.
- Choose IAM role: **AWSGlueServiceRole** (auto-created or manual).
- Run the crawler to automatically detect schema.
- A Glue table will be created inside the Glue Data Catalog.

## 8. Create Athena Tables & Execute Queries

- Open Athena → Set query results location in S3.
- Select the database created by the Glue Crawler.
- Query your S3-stored data using SQL.
- Run queries for search, filtering, ordering, and analytics.
- This gives BI-like insights without the need for QuickSight.

## Milestone 1: AWS Account Creation and Login

In this milestone, we will set up an AWS account to access the necessary services for the BloodBridge project.

### Activity 1: Create AWS Account

1. Go to the AWS website (<https://aws.amazon.com/>).
2. Click on the "Create an AWS Account" button.
3. Follow the prompts to enter your email address and choose a password.
4. Provide the required account information, including your name, address, and phone number.
5. Enter your payment information. (Note: While AWS offers a free tier, a credit card or debit card is required for verification.)
6. Complete the identity verification process.
7. Choose a support plan (the basic plan is free and sufficient for starting).
8. Once verified, you can sign in to your new AWS account.

The screenshot shows the AWS homepage. At the top, there's a navigation bar with links for About AWS, Contact Us, Support, English, My Account, Sign In, and a prominent 'Create an AWS Account' button. Below the navigation is a large pink banner with the text 'Start building on AWS today'. Underneath the banner, it says 'Whether you're looking for generative AI, compute power, database storage, content delivery, or other functionality, AWS has the services to help you build sophisticated applications with increased flexibility, scalability, and reliability'. There are two buttons: 'Get started for free' and 'Contact an AWS specialist'. To the right, there's a small message bubble saying 'Hi, I can connect you with an AWS representative or answer questions you have on AWS.' and a yellow 'Ask me' button. On the left, there's a 'Sign in' form for choosing a user type (Root user or IAM user), entering an email address, and a 'Next' button. At the bottom of the sign-in form, there's a link to the AWS Customer Agreement and Privacy Notice. On the right, there's a separate 'AWS Training and Certification' section with a large checkmark icon and a 'Learn more' button.

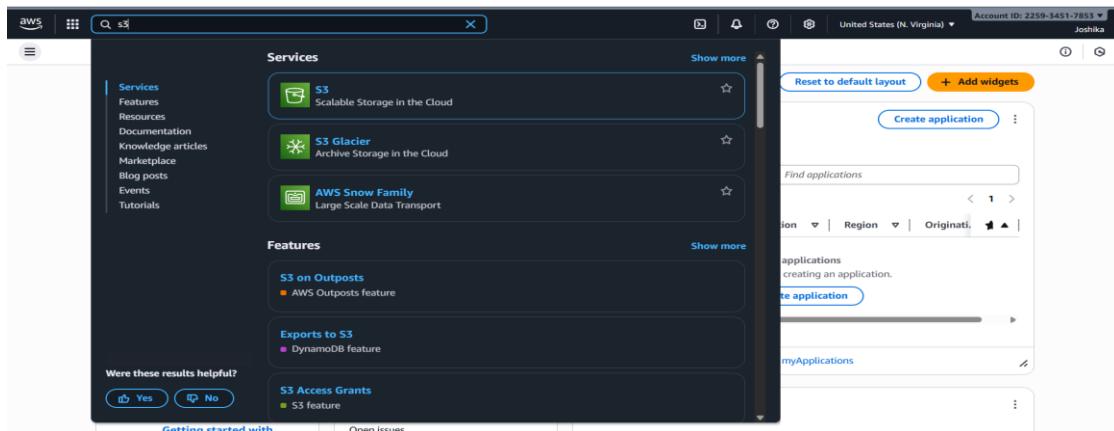
## Milestone 2: Amazon S3 Bucket Creation and Setup

In this milestone, we will create and configure an Amazon S3 bucket that will act as the **central data lake** for our serverless streaming pipeline.

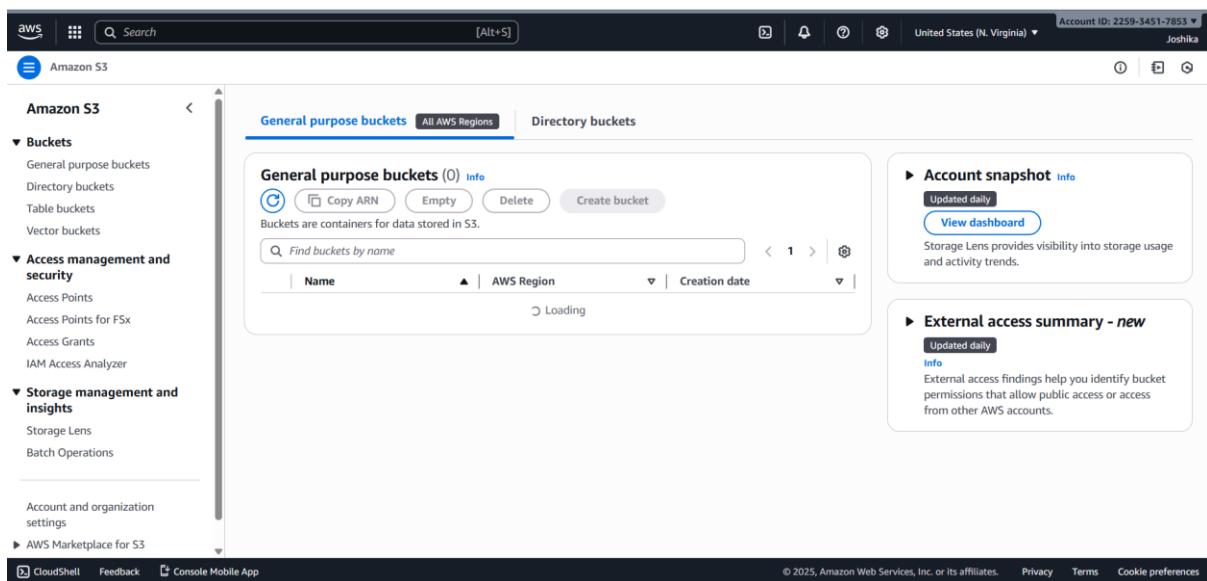
All real-time events from the web application will be stored here before being processed by AWS Glue and analyzed using Amazon Athena.

- **Activity 2.1: Access the S3 Console**

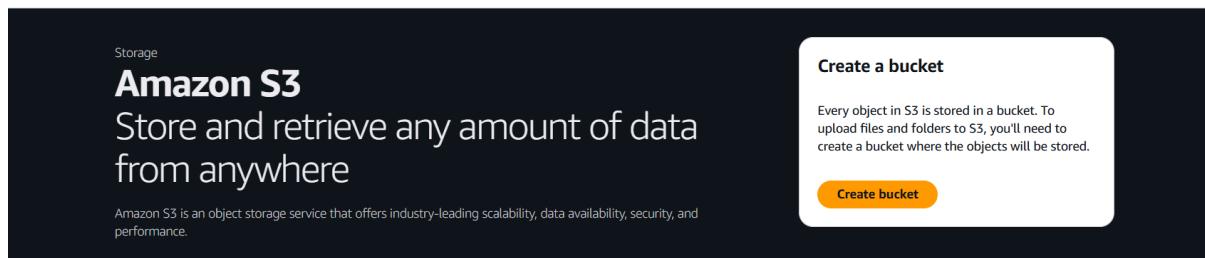
- Log in to the **AWS Management Console**.
- Search for **S3** in the AWS search bar.
- Click on **Amazon S3** to open the S3 dashboard.



- **The S3 interface**



- Click on create bucket



Fill in the following settings:

#### ◆ Bucket Details

- **Bucket name:** my-streaming-data-lake  
(Bucket names must be unique globally.)
- **AWS Region:** Select your region  
(*In your screenshot: Asia Pacific (Sydney) ap-southeast-2*)

#### ◆ Bucket Options

Keep all default settings:

- Block Public Access – **ON (Recommended)**
- Versioning – **Disabled**
- Encryption – **Enabled (Default SSE-S3)**
- Finally , click on the Create Bucket.

The screenshot shows the AWS S3 console. The left sidebar has sections for 'Buckets', 'Access management and security', 'Storage management and insights', and 'Account and organization settings'. The main area shows 'General purpose buckets (1)' with a table. The table has one row for 'my-streaming-data-lake' with columns for Name, AWS Region, and Creation date. The creation date is listed as 'November 29, 2025, 18:55:17 (UTC+05:30)'. There are also 'Copy ARN', 'Empty', and 'Delete' buttons for the bucket. Below the table are sections for 'Account snapshot' and 'External access summary - new'.

Name	AWS Region	Creation date
<a href="#">my-streaming-data-lake</a>	Asia Pacific (Sydney) ap-southeast-2	November 29, 2025, 18:55:17 (UTC+05:30)

- **Activity 2.2: Create Folders in the Bucket**

- Open the bucket → Click **Create folder**.

Inside the newly created bucket:

- Create folder: **raw**
- Create folder: **athena-results**

These folders will later be accessed by:

- AWS Lambda (writing data)
- AWS Glue (crawling data)
- AWS Athena (reading data)

- Create two folders:

## 📁 Folder Structure

raw/  
athena-results/

The screenshot shows the Amazon S3 console interface. The top navigation bar includes the AWS logo, search bar, and account information (Account ID: 5539-3098-1805, Region: Asia Pacific (Sydney), User: Bellarmkonda Thishithasai). The left sidebar has a tree view with 'Amazon S3' expanded, showing 'Buckets' (General purpose buckets, Directory buckets, Table buckets, Vector buckets), 'Access management and security' (Access Points, Access Points for FSx, Access Grants, IAM Access Analyzer), and 'Storage management and insights' (Storage Lens, Batch Operations). The main content area shows the 'my-streaming-data-lake' bucket details. The 'Objects' tab is selected, displaying two objects: 'athena-results/' and 'raw/'. A table below lists these objects with columns for Name, Type, Last modified, Size, and Storage class. Buttons for Actions (including Copy S3 URI, Copy URL, Download, Open, Delete, and Create folder) are at the top of the object list. A search bar at the bottom left allows finding objects by prefix.

Name	Type	Last modified	Size	Storage class
athena-results/	Folder	-	-	-
raw/	Folder	-	-	-

## MILESTONE 3: IAM Role Creation and AWS Lambda Setup

In this milestone, we configure the backend services required for our real-time data streaming pipeline.

We start by creating an IAM role that provides permissions for Lambda to interact with Amazon S3.

Next, we configure the Lambda function **json\_event\_ingestor**, which processes incoming JSON events and stores them inside the S3 Data Lake.

### Activity 3.1: Create IAM Role for Lambda

In this activity, we will create the IAM role that allows the Lambda function to write to S3 and generate CloudWatch logs.

#### Steps:

1. Log in to AWS Console → Go to **IAM**.
2. Select **Access Management** → **Roles** from the left menu.
3. Click on **Create role**.
4. Choose **Trusted Entity: AWS Service**.
5. Select **Lambda** as the use case.
6. Click **Next** to add permissions.

#### Attach the following policies:

- **AWSLambdaBasicExecutionRole**  
(Allows writing logs to CloudWatch)
- **LambdaS3WritePolicy** (*Your inline policy*)  
This policy allows your Lambda function to upload JSON files into your S3 bucket my-streaming-data-lake.

**Name the Role:** LambdaS3WriterRole

The screenshot shows the AWS IAM Roles list. The left sidebar navigation includes 'Identity and Access Management (IAM)', 'Access management' (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management, Temporary delegation requests), and 'Access reports' (Access Analyzer, Resource analysis, Unused access). The main content area displays a table titled 'Roles (7)'. The table has columns for 'Role name', 'Trusted entities', and 'Last activity'. The newly created role, 'LambdaS3WriterRole', is listed with 'AWS Service: lambda' as the trusted entity and '14 hours ago' as the last activity. Below the table, there's a section titled 'Roles Anywhere' with options for 'Access AWS from your non AWS' (X.509 Standard) and 'Temporary credentials'.

**Figure: IAM roles list displaying the newly created LambdaS3WriterRole.**

The screenshot shows the details page for the LambdaS3WriterRole. The left sidebar navigation is identical to the previous screenshot. The main content area shows the role's summary: 'Allows Lambda functions to call AWS services on your behalf.' It includes fields for 'Creation date' (November 29, 2025, 19:05 (UTC+05:30)), 'Last activity' (6 hours ago), 'ARN' (arn:aws:iam::553930981805:role/LambdaS3WriterRole), and 'Maximum session duration' (1 hour). Below the summary, there are tabs for 'Permissions', 'Trust relationships', 'Tags', 'Last Accessed', and 'Revoke sessions'. The 'Permissions' tab is selected, showing 'Permissions policies (2)'. It lists two policies: 'AWSLambdaBasicExecutionRole' (AWS managed, Type: AWS managed, Attached entities: 1) and 'LambdaS3WritePolicy' (Customer inline, Type: Customer inline, Attached entities: 0). There are buttons for 'Edit', 'Delete', 'Simulate', 'Remove', and 'Add permissions'.

**Figure: Attached permission policies for LambdaS3WriterRole.**

## Activity 3.2: AWS Lambda Function Creation

In this activity, we create the Lambda function that receives incoming event data and stores it in the S3 Data Lake with timestamp-based folder structure.

### Steps:

1. Go to **AWS Console → Lambda → Functions**.
2. Click on **Create Function**.
3. Choose:

Author from scratch  
Function name: json\_event\_ingestor  
Runtime: Python 3.9

4. Under **Execution Role**, select:

Use existing role → LambdaS3WriterRole

The screenshot shows the AWS Lambda Functions list. On the left, there's a sidebar with options like Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main area is titled 'Functions (2)' and lists two functions: 'json\_event\_ingestor' and 'athena\_query\_handler'. Both functions are listed as 'Zip' type with 'Python 3.9' runtime. The 'json\_event\_ingestor' function was last modified 16 hours ago, while the 'athena\_query\_handler' was last modified 14 hours ago. There are 'Actions' and 'Create function' buttons at the top right of the list.

Function name	Description	Package type	Runtime	Last modified
json_event_ingestor	-	Zip	Python 3.9	16 hours ago
athena_query_handler	-	Zip	Python 3.9	14 hours ago

**Figure: Lambda functions list showing json\_event\_ingestor.**

## Activity 3.3: Lambda Function Configuration & Code Upload

Once the Lambda function is created, we configure triggers and upload the event ingestion code.

### Steps:

- Open **json\_event\_ingestor** function.
- In the function overview diagram, verify that **API Gateway** is connected as a trigger.
- Open the **Code** tab.
- Replace the default Python code with your implementation.

```

import json
import uuid
import datetime
import boto3
s3 = boto3.client("s3")
BUCKET = "my-streaming-data-lake"
def lambda_handler(event, context):
    body = json.loads(event["body"])
    now = datetime.datetime.utcnow()
    prefix = now.strftime("raw/%Y/%m/%d/%H/")
    filename = f"{uuid.uuid4()}.json"
    key = prefix + filename
    s3.put_object(
        Bucket=BUCKET,
        Key=key,
        Body=json.dumps(body),
        ContentType="application/json"
    ) return {
        "statusCode": 200,
        "headers": {"Access-Control-Allow-Origin": "*"},
        "body": json.dumps({"message": "Event stored", "file": key})
    }

```

**Description**

Last modified  
23 hours ago

**Function ARN**  
arn:aws:lambda:ap-southeast-2:553930981805:function:json\_event\_ingestor

**Function URL** | Info

**Figure: Configuration details of json\_event\_ingestor Lambda function.**

```

def lambda_handler(event, context):
    body = json.loads(event["body"])
    now = datetime.datetime.utcnow()
    prefix = now.strftime("%Y/%m/%d/%H")
    filename = f"{prefix}-{uuid.uuid4()}.json"
    key = prefix + filename
    s3.put_object(
        Bucket=BUCKET,
        Key=key,
        Body=json.dumps(body),
        ContentType="application/json"
    )
    return {
        "statusCode": 200,
        "headers": {"Access-Control-Allow-Origin": "*"},
        "body": json.dumps({"Message": "Event stored", "File": key})
    }

```

**Figure: Python code inside the json\_event\_ingestor Lambda function.**

## MILESTONE 4: API Gateway Setup (Real-Time Data Ingestion Endpoint)

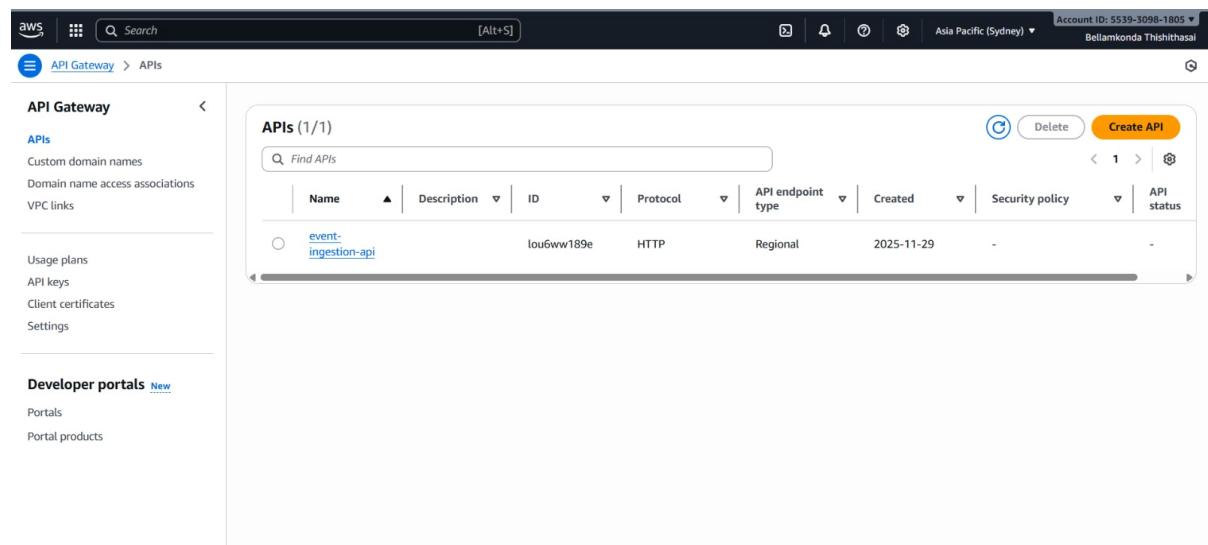
In this milestone, we create the HTTP API that acts as the entry point for the real-time streaming pipeline.

This API receives JSON data from the React frontend and forwards it to the Lambda function **json\_event\_ingestor**.

### Activity 4.1: Access API Gateway Console

#### Steps:

1. Log in to AWS Management Console.
2. Search for **API Gateway**.
3. Open the API Gateway dashboard.



The screenshot shows the AWS API Gateway console. The left sidebar has sections for APIs, Usage plans, API keys, Client certificates, Settings, and Developer portals. The main area is titled 'APIs (1/1)' and shows a single API named 'event-ingestion-api'. The table columns are Name, Description, ID, Protocol, API endpoint type, Created, Security policy, and API status. The API details are: Name: event-ingestion-api, ID: l0u6ww189e, Protocol: HTTP, API endpoint type: Regional, Created: 2025-11-29, Security policy: -, API status: -.

### Activity 4.2: Create a New HTTP API

You created an **HTTP API** (not REST API), as per the steps you followed.

#### Steps:

1. Click **Create API**.
2. Choose **Build → HTTP API**.
3. Select **Add Integration** → choose **Lambda**.
4. Select your Lambda function: **json\_event\_ingestor**

## Activity 4.3: Create Route

You created a single route for accepting events:

**Route:POST /event**

This route ensures that your React app can send events using POST requests.

The screenshot shows the AWS API Gateway Routes page for the 'event-ingestion-api'. On the left sidebar, under 'Develop', 'Routes' is selected. In the main panel, there is a list of routes for the 'event-ingestion-api'. One route is listed: '/event' with a 'POST' method. The 'Route details' section shows the ARN as 'arn:aws:apigateway:ap-southeast-2::apis/lou6ww189e/routes/ymlx1zk'. The 'Authorization' section indicates no authorizer is attached. The 'Integration' section shows the integration ID '9xiqo4n' with a 'Configure' button.

The screenshot shows the AWS API Gateway Routes page for the 'event-ingestion-api'. On the left sidebar, under 'Develop', 'Routes' is selected. In the main panel, there is a list of routes for the 'event-ingestion-api'. One route is listed: '/stats' with a 'GET' method. The 'Route details' section shows the ARN as 'arn:aws:apigateway:ap-southeast-2::apis/lou6ww189e/routes/a151nmk'. The 'Authorization' section indicates no authorizer is attached. The 'Integration' section shows the integration ID 'i2z2l1k' with a 'Configure' button.

#### **Activity 4.4: Enable CORS**

To allow communication between your React app and API Gateway, CORS needs to be enabled.

##### **Settings:**

Access-Control-Allow-Origin: \*  
Access-Control-Allow-Methods: POST  
Access-Control-Allow-Headers: \*

#### **Activity 4.5: Deploy the API**

##### **Steps:**

1. Click **Deploy**.
2. Choose stage: prod
3. Copy the **Invoke URL**.  
This URL will be used in your React App in the fetch() call.

## MILESTONE 5: React Frontend – Real-Time Event Sender

In this milestone, we create a simple React web application that sends JSON events to the AWS API Gateway endpoint.

This frontend behaves like a **streaming producer**, generating test events in real-time.

### Activity 5.1: Create the React Application

Open a terminal in VS Code and run:

```
npx create-react-app frontend
```

Move into the folder:

```
cd frontend
```

Start the development server:

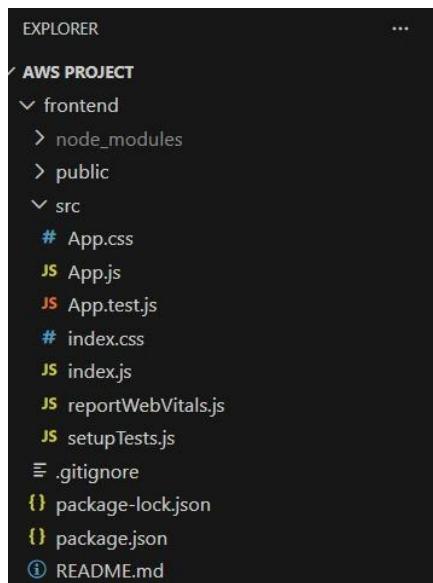
```
npm start
```

This launches the app at:

👉 <http://localhost:3000>

### Activity 5.2: Verify Folder Structure

After successfully creating the React project, the following folder structure is generated:



**Figure: React project folder structure created using Create React App**

## MILESTONE 6: AWS Glue Crawler Setup & Schema Discovery

In this milestone, we configure AWS Glue to automatically detect the schema of the JSON files stored in our S3 Data Lake.

The Glue Crawler scans the raw/ folder inside the bucket, identifies fields in incoming events, and creates a table (raw\_events) inside the AWS Glue Data Catalog.

This allows Athena to query the data using SQL.

### Activity 6.1: Create AWS Glue Crawler

Steps:

1. Log in to AWS Console → Search for **Glue**
2. In the left menu, click **Crawlers**
3. Click **Create crawler**
4. Enter name:  
**streaming-data-crawler**
5. Choose **Data source** → **S3**
6. Enter S3 path:  
**s3://my-streaming-data-lake/raw/**
7. Choose IAM Role:  
**AWSGlueServiceRole-streaming-crawler**
8. Choose database:  
**streaming\_lake**
9. Click **Create** and then **Run Crawler**

The screenshot shows the AWS Glue interface with the 'Crawlers' section selected. A single crawler named 'streaming-data-crawler' is listed in the table, showing a status of 'Ready'. The table includes columns for Name, State, Last run, Last run time, Log, and Table changes.

Name	State	Last run	Last run time	Log	Table changes
streaming-data-crawler	Ready	Succeeded	November 29, 2...	<a href="#">View log</a>	-

Figure: Showing the crawler name **streaming-data-crawler** with state READY

The screenshot shows the AWS Glue interface with the 'streaming-data-crawler' properties page selected. The crawler properties include the name 'streaming-data-crawler', IAM role 'AWSGlueServiceRole-streaming-crawler', database 'streaming\_lake', and state 'READY'. The 'Crawler runs' section shows one run completed successfully with a duration of 40 seconds.

Name	IAM role	Database	State
streaming-data-crawler	AWSGlueServiceRole-streaming-crawler	streaming_lake	READY

Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table changes
November 29, 2025 at ...	November 29, 2025 at ...	40 s	Completed	0.124	-

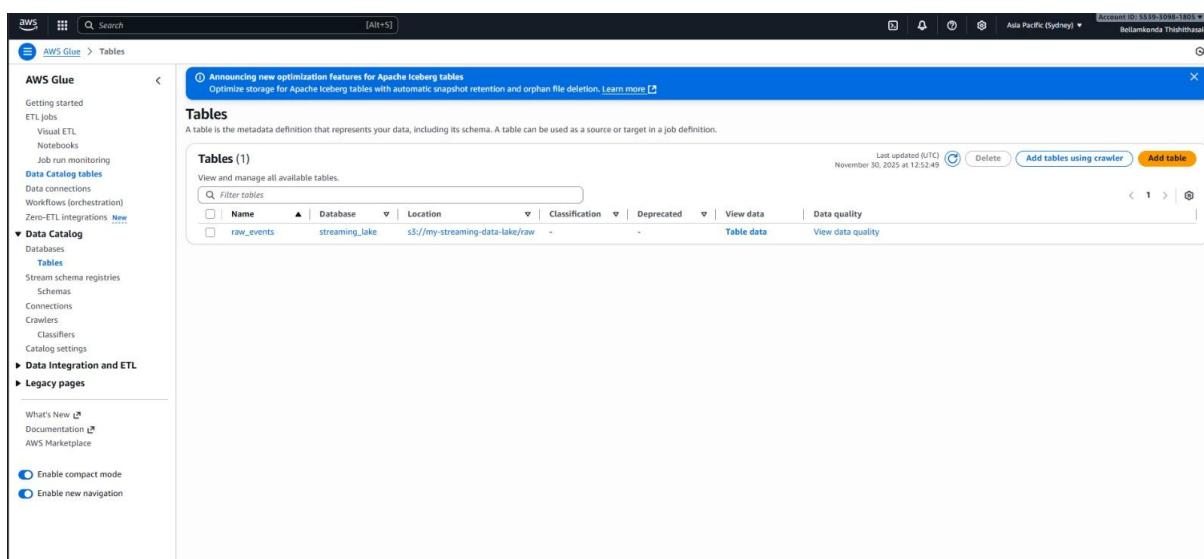
Figure: showing crawler properties, IAM role, database, and last run status “Completed”.

## Activity 6.2: View Tables Generated by Crawler

After the crawler runs successfully, it creates a table inside the Data Catalog.

Steps:

1. Go to **Glue → Data Catalog → Tables**
2. Select database: **streaming\_lake**
3. You should see the table:  
**raw\_events**



The screenshot shows the AWS Glue Data Catalog Tables interface. On the left, there's a navigation sidebar with options like 'AWS Glue', 'Data Catalog tables', and 'Data Catalog'. Under 'Data Catalog', 'Tables' is selected. The main area displays a table titled 'Tables (1)'. The table has one row for 'raw\_events'. The columns shown are Name, Database, Location, Classification, Deprecated, View data, and Data quality. The 'Name' column shows 'raw\_events', 'Database' shows 'streaming\_lake', and 'Location' shows 's3://my-streaming-data-lake/raw'. The 'View data' and 'Data quality' buttons are visible at the bottom of the table row. At the top right of the main area, there are buttons for 'Delete', 'Add tables using crawler', and 'Add table'. A blue banner at the top of the page says 'Announcing new optimization features for Apache Iceberg tables'.

## Activity 6.3: Inspect Table Schema

Click on **raw\_events** to view the schema Glue discovered.

Expected columns:

Column	Type
id	string
type	string
amount	int
currency	string
created	string

Glue extracts this automatically.

The screenshot shows the AWS Glue Table Overview page for a table named 'raw\_events'. The table is located in the 'streaming\_lake' database. Basic details include:

- Name:** raw\_events
- Database:** streaming\_lake
- Description:** -
- Last updated:** November 29, 2025 at 14:27:47
- Classification:** -
- Location:** s3://my-streaming-data-lake/raw
- Connection:** -
- Deprecated:** -
- Column statistics:** No statistics

The Schema section shows 5 columns:

#	Column name	Data type	Partition key	Comment
1	id	string	-	-
2	type	string	-	-
3	amount	int	-	-
4	currency	string	-	-
5	created	string	-	-

## Activity 6.4: Verify Data Availability via Athena

To ensure the Glue table works properly:

1. Go to AWS → Athena
2. Select database: **streaming\_lake**
3. Run sample queries:

### Query 1: List all events

SELECT \* FROM streaming\_lake.raw\_events LIMIT 10;

The screenshot shows the Amazon Athena Query Editor. A single query is present in the editor:

```
1 | SELECT type, COUNT(*) FROM streaming_lake.raw_events GROUP BY type;
```

The sidebar on the left shows the data source (AwsDataCatalog), catalog (None), and database (streaming\_lake). The tables section lists 'raw\_events'.

## Query 2: Group by event type

```
SELECT type, COUNT(*)  
FROM streaming_lake.raw_events  
GROUP BY type;
```

The screenshot shows the AWS Athena Query Editor interface. The left sidebar displays the 'Data' configuration with 'Database' set to 'streaming\_lake' and 'Tables and views' showing 'raw\_events'. The main area contains two tabs: 'Query 1' and 'Query 2'. 'Query 1' is active, containing the SQL command: '1 SELECT \* FROM "streaming\_lake"."raw\_events" limit 10;'. Below the query is a status bar with 'SQL Ln 1, Col 1' and buttons for 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. A note at the bottom right says 'Reuse query results up to 60 minutes ago'.

## Query 3: Count total events

```
SELECT COUNT(*) FROM streaming_lake.raw_events;
```

This screenshot shows the Amazon Athena Query Editor with three tabs: 'Query 1', 'Query 2', and 'Query 3'. 'Query 3' is active, containing the SQL command: '1 SELECT COUNT(\*) FROM streaming\_lake.raw\_events'. The interface is identical to the previous screenshot, with the 'Data' sidebar, a status bar indicating 'SQL Ln 1, Col 1', and a note about reusing query results.

## Query 4: Sum of all transaction amounts

```
SELECT SUM(amount)  
FROM streaming_lake.raw_events;
```

The screenshot shows the Amazon Athena Query Editor with four tabs: 'Query 1', 'Query 2', 'Query 3', and 'Query 4'. 'Query 4' is active, containing the SQL command: '1 SELECT SUM(amount) FROM streaming\_lake.raw\_events'. The interface is consistent with the previous screenshots, featuring the 'Data' sidebar, a status bar, and a note about reusing query results.

## Outputs:

### Streaming Analytics Dashboard

Real-time Event Data Pipeline

**TOTAL EVENTS**: 0

**TOTAL AMOUNT**: ₹0

**AVERAGE AMOUNT**: ₹0

**Manual Event Form**

Amount (INR)  
Enter amount

Event Type  
payment\_success

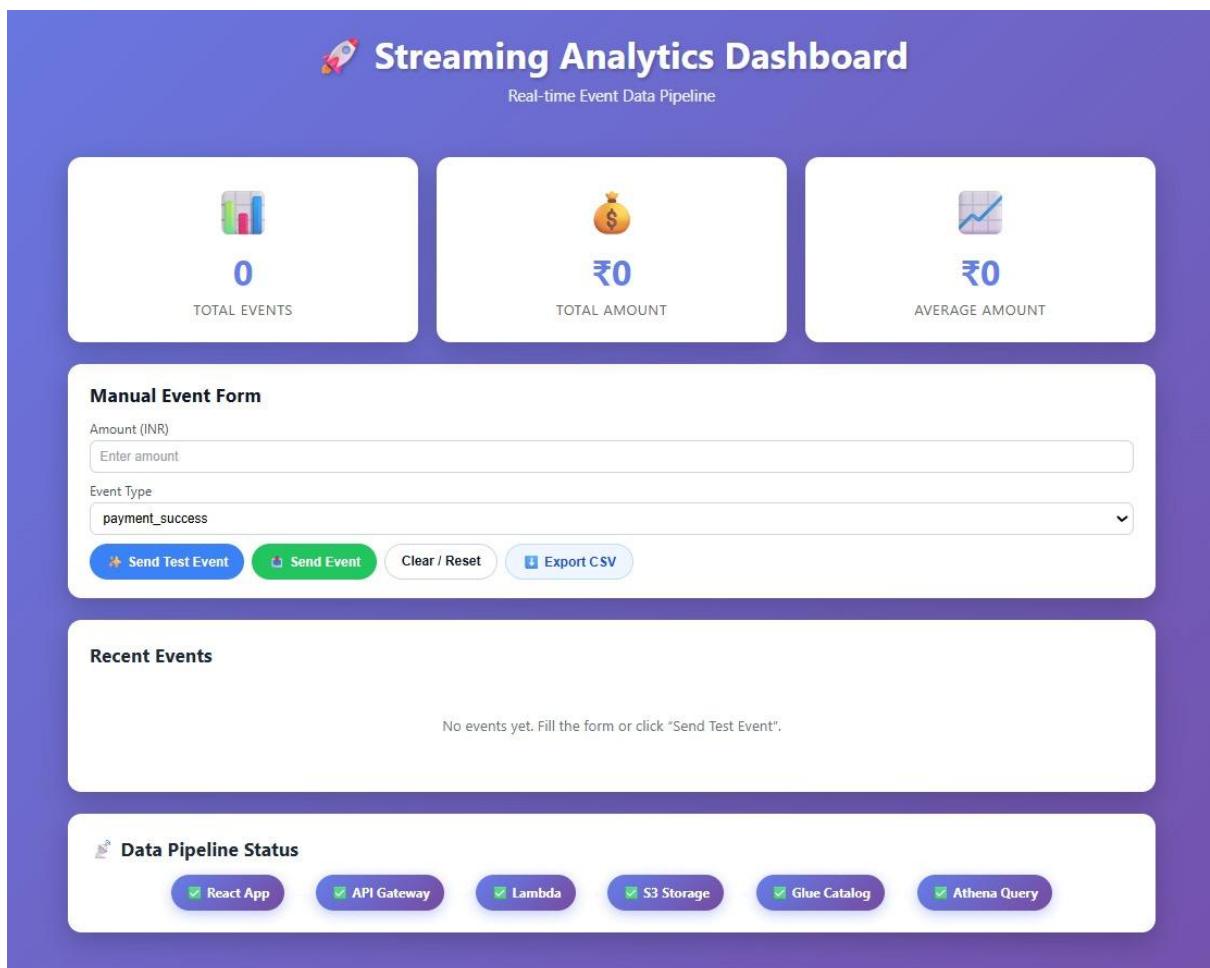
**Send Test Event** **Send Event** **Clear / Reset** **Export CSV**

**Recent Events**

No events yet. Fill the form or click "Send Test Event".

**Data Pipeline Status**

**React App** **API Gateway** **Lambda** **S3 Storage** **Glue Catalog** **Athena Query**

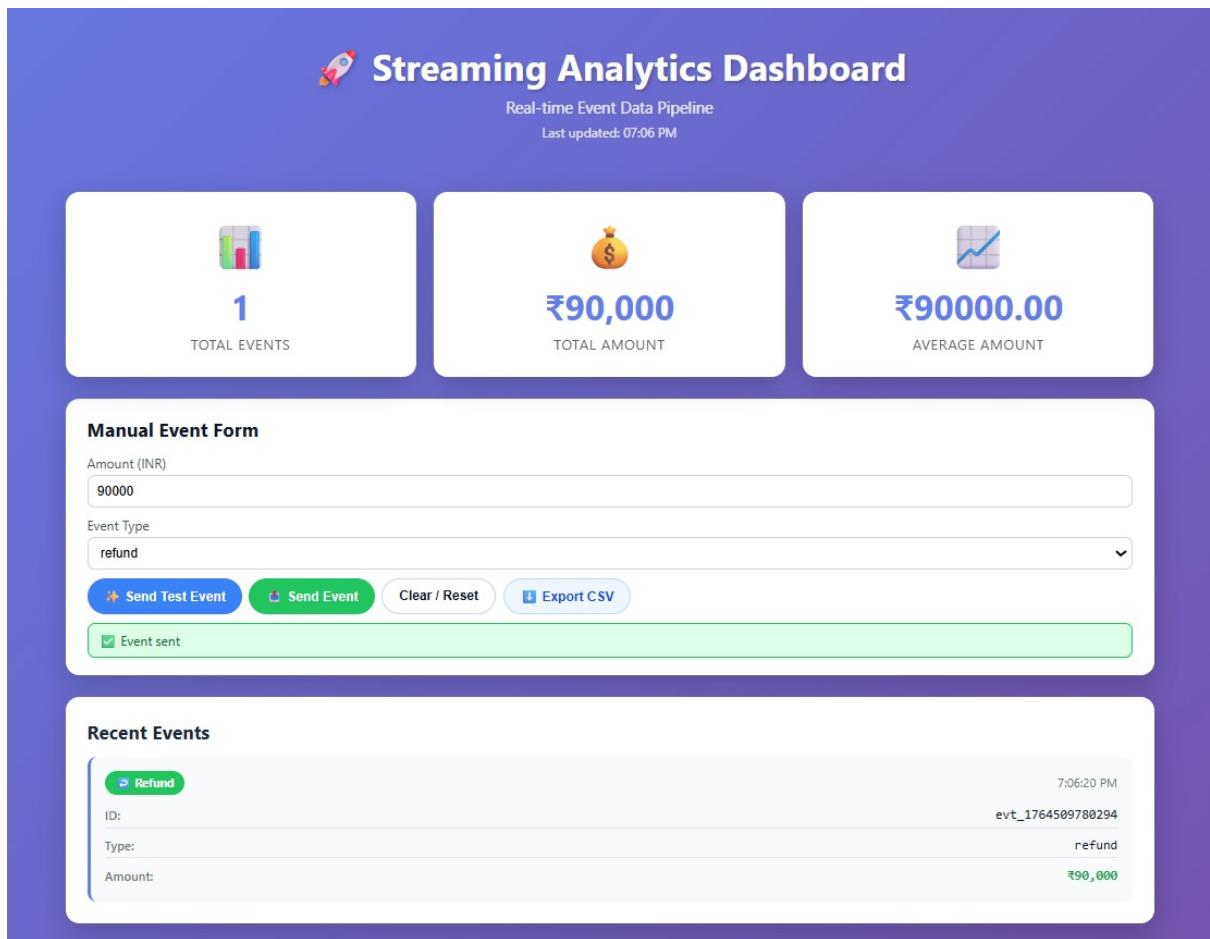


**Manual Event Form**

Amount (INR)  
90000

Event Type  
payment\_success  
payment\_success  
payment\_failed  
refund





## SQL Queries for Real-Time Data Analysis:

```
aws [Alt+S] Search
Amazon Athena > Query editor
Data source: AwsDataCatalog
Catalog: None
Database: streaming_lake
Tables and views: raw_events
Tables (1)
Views (0)

Query 1 : x | Query 2 : x
1 | SELECT type, COUNT(*) FROM streaming_lake.raw_events GROUP BY type;

SQL Ln 1, Col 1
Run again Explain Cancel Clear Create
Reuse query results up to 60 minutes ago

Results (1)
Completed Time in queue: 98 ms Run time: 2.328 sec Data scanned: 2.63 KB
Copy Download results CSV
Search rows
# | type
1 | payment_success
8 | _col1
```

The screenshot shows the Amazon Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (streaming\_lake). Under Tables and views, there is one table named 'raw\_events'. The main area shows a query editor with the following SQL:

```
1 SELECT * FROM "streaming_lake"."raw_events" limit 10;
```

The results section shows 10 rows of data from the 'raw\_events' table:

#	id	type	amount	currency	created
1	evt_test123	payment_success	1000	INR	2025-11-29T20:10:04.940479+05:30
2	evt_1764429591034	payment_success	4669	INR	2025-11-29T15:19:51.034Z
3	evt_1764431014306	payment_success	2091	INR	2025-11-29T15:43:34.306Z
4	evt_1764429100843	payment_success	2290	INR	2025-11-29T15:11:40.843Z
5	evt_1764427682858	payment_success	2421	INR	2025-11-29T14:48:02.858Z
6	evt_1764427692877	payment_success	1905	INR	2025-11-29T14:48:12.877Z
7	evt_1764427593567	payment_success	1976	INR	2025-11-29T14:46:53.567Z

The screenshot shows the Amazon Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (streaming\_lake). Under Tables and views, there is one table named 'raw\_events'. The main area shows a query editor with the following SQL:

```
1 SELECT COUNT(*) FROM streaming_lake.raw_events
```

The results section shows the count of rows in the 'raw\_events' table:

#	_col0
1	8

The screenshot shows the Amazon Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (streaming\_lake). Under Tables and views, there is one table named 'raw\_events'. The main area shows a query editor with the following SQL:

```
1 SELECT SUM(amount) FROM streaming_lake.raw_events
```

The results section shows the total sum of amounts in the 'raw\_events' table:

#	_col0
1	18203

## **Conclusion:**

The AWS Data Confluence project successfully demonstrates a fully serverless, scalable, and real-time data ingestion and analytics pipeline. By integrating **React**, **API Gateway**, **Lambda**, **S3**, **Glue**, **Athena**, and **SNS**, the system efficiently captures data from the frontend, processes it through Lambda, stores structured records in S3, and enables instant querying through Athena. Glue provides automated schema detection, while SNS enhances the solution with immediate alerting through email notifications.

This architecture highlights the power of cloud-native, event-driven design — requiring no servers, automatically scaling with demand, and operating at very low cost. The project proves how real-time data streams can be collected, stored, analyzed, and notified with minimal configuration and maximum reliability. Overall, this system forms a strong foundation for modern analytics workflows and can be extended further with dashboards, AI models, or additional AWS integrations.