PROJECT REPORT

Serverless Sentiment Analysis System using AWS Lambda and API Gateway

COURSE: Cloud Architecture Design

VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI

Submitted By:

SHRUHTI REDDY 23BCE1323

YAZHISAI G 23BCE1826

AKSHAY SM 23BCE1792

FALL SEMESTER

November, 2025

INSTRUCTOR: Priyadharshini R

**2. ABSTRACT**

This project presents the design and implementation of a **serverless sentiment analysis system** using **AWS Cloud Services**, primarily **AWS Lambda**, **Amazon API Gateway**, **Amazon Comprehend**, and **Amazon S3**.  
The system enables users to input textual data through a web interface and receive real-time sentiment analysis results without requiring any backend server infrastructure.

The application leverages **AWS Comprehend**’s natural language processing capabilities to classify user inputs as **Positive, Negative, Neutral,** or **Mixed**, while **AWS Lambda** handles the computation logic in a stateless, cost-effective manner.  
The **serverless architecture** ensures high scalability, reduced maintenance overhead, and pay-per-use efficiency.

This report details the design architecture, implementation steps, deployment configuration, challenges faced, and outcomes achieved.

**3. INTRODUCTION**

**3.1 Background**

With the exponential growth of user-generated content on social media, reviews, and digital platforms, understanding public sentiment has become a critical analytical tool for organizations. Traditional sentiment analysis systems often depend on dedicated servers, leading to scalability and maintenance challenges.

**3.2 Need for a Serverless Solution**

A serverless approach removes the burden of managing infrastructure. Using **AWS Lambda** and **API Gateway**, developers can deploy applications that scale automatically based on demand while paying only for actual compute time.

**3.3 Objective**

The primary objective of this project is to:

* Design and implement a **cloud-native sentiment analysis system**.
* Utilize **AWS Comprehend** for natural language processing.
* Deploy using a **fully serverless stack** with high availability and low operational cost.

**4. LITERATURE REVIEW**

**4.1 Sentiment Analysis Techniques**

Sentiment analysis typically uses natural language processing (NLP) and machine learning techniques to interpret subjective information in text. Earlier approaches relied on rule-based methods and lexicon-based scoring systems. In recent years, deep learning and cloud-based NLP APIs (like AWS Comprehend, Google NLP, and Azure Cognitive Services) have gained popularity for their accuracy and ease of integration.

**4.2 Serverless Computing**

Serverless computing abstracts away server management. AWS Lambda, introduced in 2014, popularized the model where users deploy functions that execute in response to triggers, such as HTTP requests or file uploads, making it ideal for lightweight, event-driven architectures.

**4.3 Related Work**

Previous studies and systems have used AWS Comprehend for text classification, but many were deployed with traditional EC2 instances or containers. This project uniquely integrates **Comprehend with Lambda via API Gateway** and a static front-end hosted on **Amazon S3**, achieving a fully serverless pipeline.

**5. PREVIOUS REPORTS**

Before developing the final version of this AWS project, We completed two preliminary assignments that focused on planning and designing the cloud infrastructure. These assignments involved identifying suitable AWS services, understanding their use cases, and proposing an initial architecture for the project.

It is important to note that some services and design choices mentioned in these earlier reports were later modified or replaced based on practical requirements, performance considerations, and further learning during implementation.

The following sections include the contents of **Digital** **Assignment 1** and **Digital** **Assignment 2** to show the progression of the project — from initial planning to the final deployed solution.

CLOUD ARCHITECTURE DESIGN DA-1

**Cloud-Based Sentiment Analysis of Tweets Using AWS Services**

**Requirement Analysis**

1. **Functional Requirements**

The system should fetch tweets from Twitter/X API based on hashtags, keywords, or usernames.

The system should preprocess tweets (remove hashtags, links, emojis, stopwords).

The system should analyze the sentiment (Positive, Negative, Neutral, Mixed).

The system should store raw and analyzed data for later use.

The system should display results (dashboard or simple console/CSV output).

1. **Non-Functional Requirements**

Scalability → should handle large tweet datasets (AWS S3 + Lambda scaling).

Availability → must be accessible anytime (serverless services).

Performance → response time should be quick for real-time analysis.

Security → Twitter API keys and data stored securely.

1. **Hardware/Software Requirements**

Hardware: None specific (serverless AWS).

Software:

AWS Services: S3, Lambda, Comprehend, API Gateway, QuickSight (optional).

Programming: Python/Node.js for Lambda.

Twitter Developer Account for API access.

**System Design (High-Level Architecture)**

[User / App]

│

▼

|  |
| --- |
|  |

│ Twitter API (X) │

|  |
| --- |
|  |

│

(tweets fetched via Lambda)

│

▼

|  |
| --- |
|  |

│ Amazon S3 Bucket │ ← stores raw tweets

|  |
| --- |
|  |

│

▼

|  |
| --- |
|  |

│ AWS Lambda │ ← preprocess tweets

|  |
| --- |
|  |

│

▼

|  |
| --- |
|  |

│Amazon Comprehend│ ← Sentiment Analysis

|  |
| --- |
|  |

│

▼

|  |
| --- |
|  |

│ DynamoDB / RDS │ ← store results

|  |
| --- |
|  |

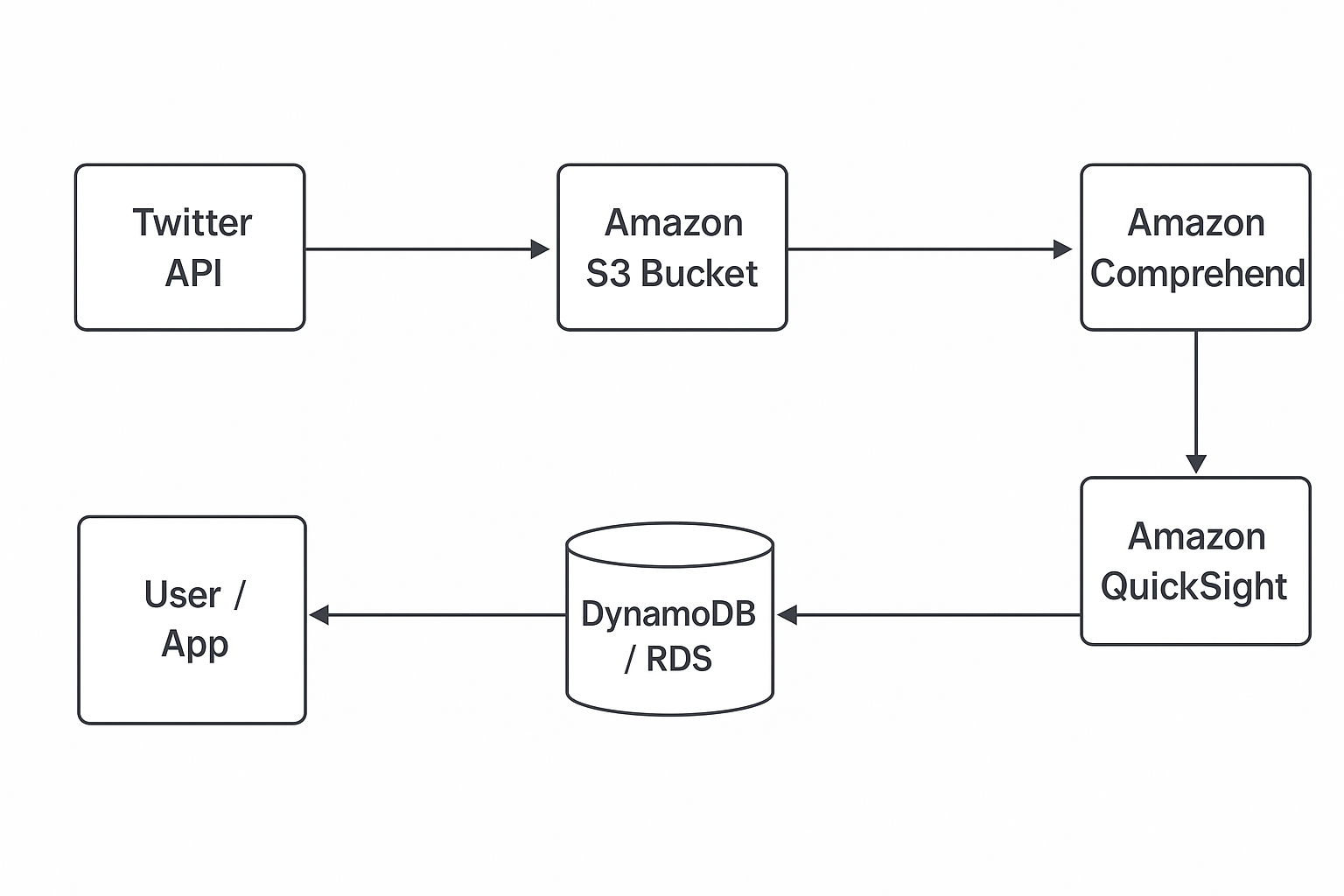
│

▼

|  |
| --- |
|  |

│ Amazon QuickSight │ ← visualize results

|  |
| --- |
|  |



Workflow

1. Data Collection: Lambda fetches tweets via Twitter API → stores in S3.
2. Preprocessing: Lambda cleans data.
3. Sentiment Analysis: AWS Comprehend processes text → classifies sentiment.
4. Storage: Results stored in DynamoDB/RDS.
5. Visualization: QuickSight dashboard shows trends (e.g., % positive/negative over time).

Login AWS

A screenshot of a computer

AI-generated content may be incorrect.

Accessing IAM

A screenshot of a computer

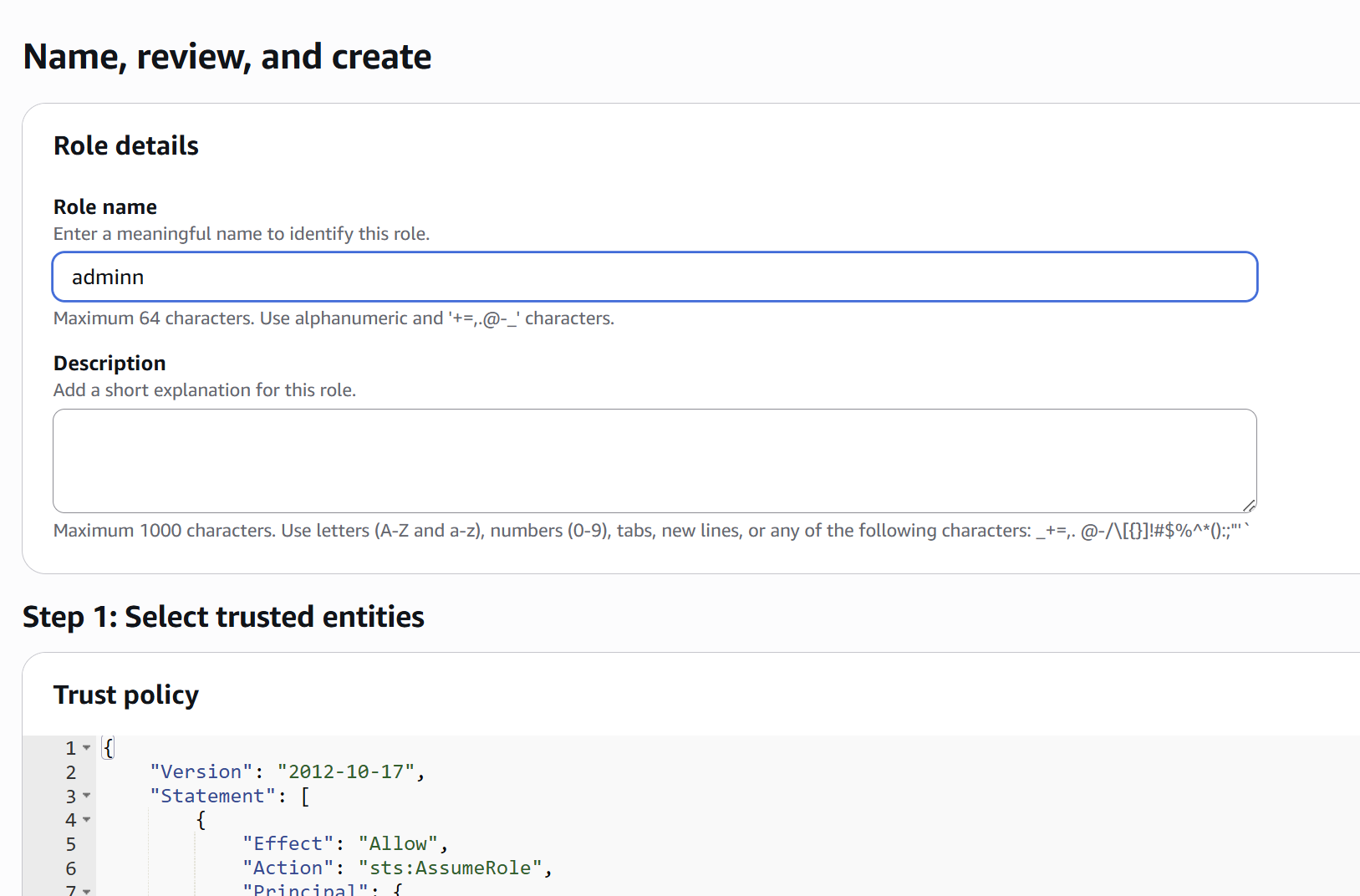
AI-generated content may be incorrect.

Creating Group

A screenshot of a computer

AI-generated content may be incorrect.

Creating Roles

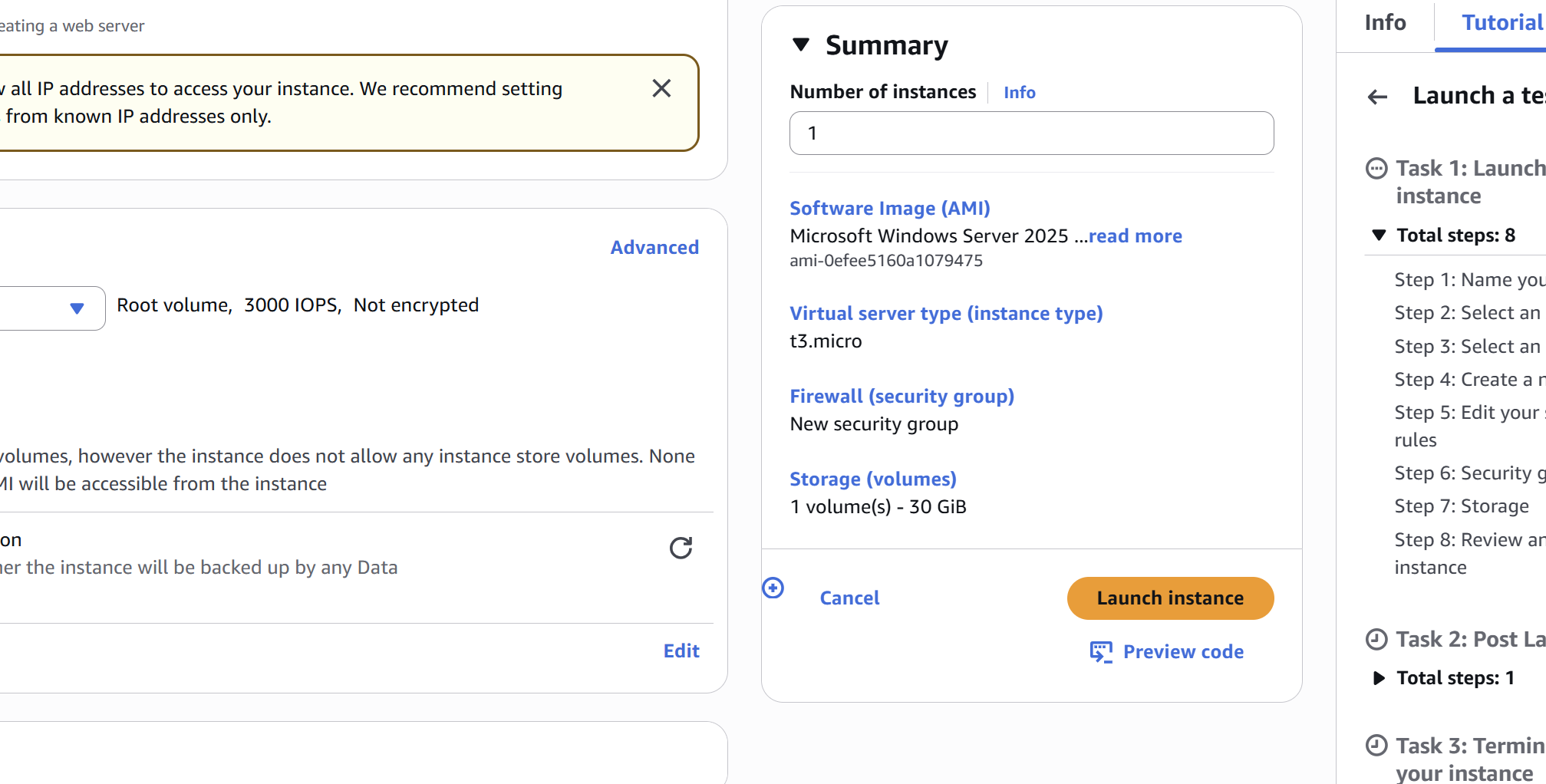


Assigning roles

A screen shot of a computer

AI-generated content may be incorrect.

Launching EC2



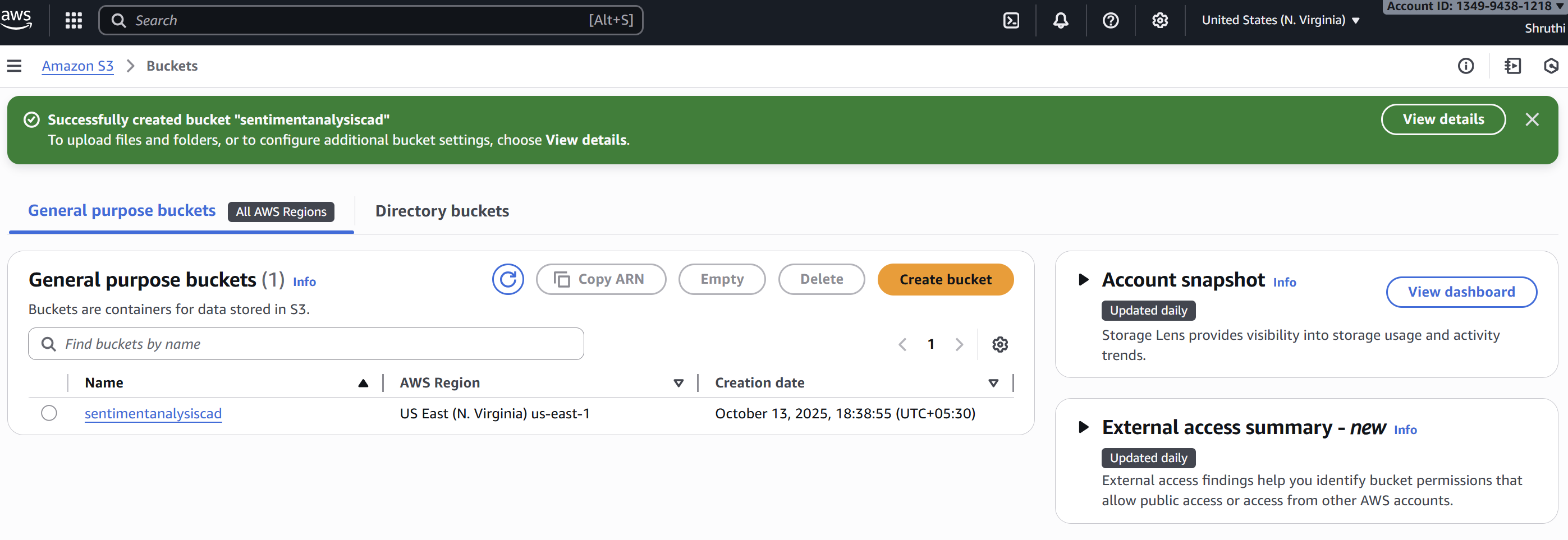
A screenshot of a computer

AI-generated content may be incorrect.

**CLOUD ARCHITECTURE DESIGN DA-2**

**Cloud-Based Sentiment Analysis of Tweets using AWS Services**

**Amazon S3 (Simple Storage Service)**



**Purpose**: Centralized cloud storage for tweet data

**Why Used:**

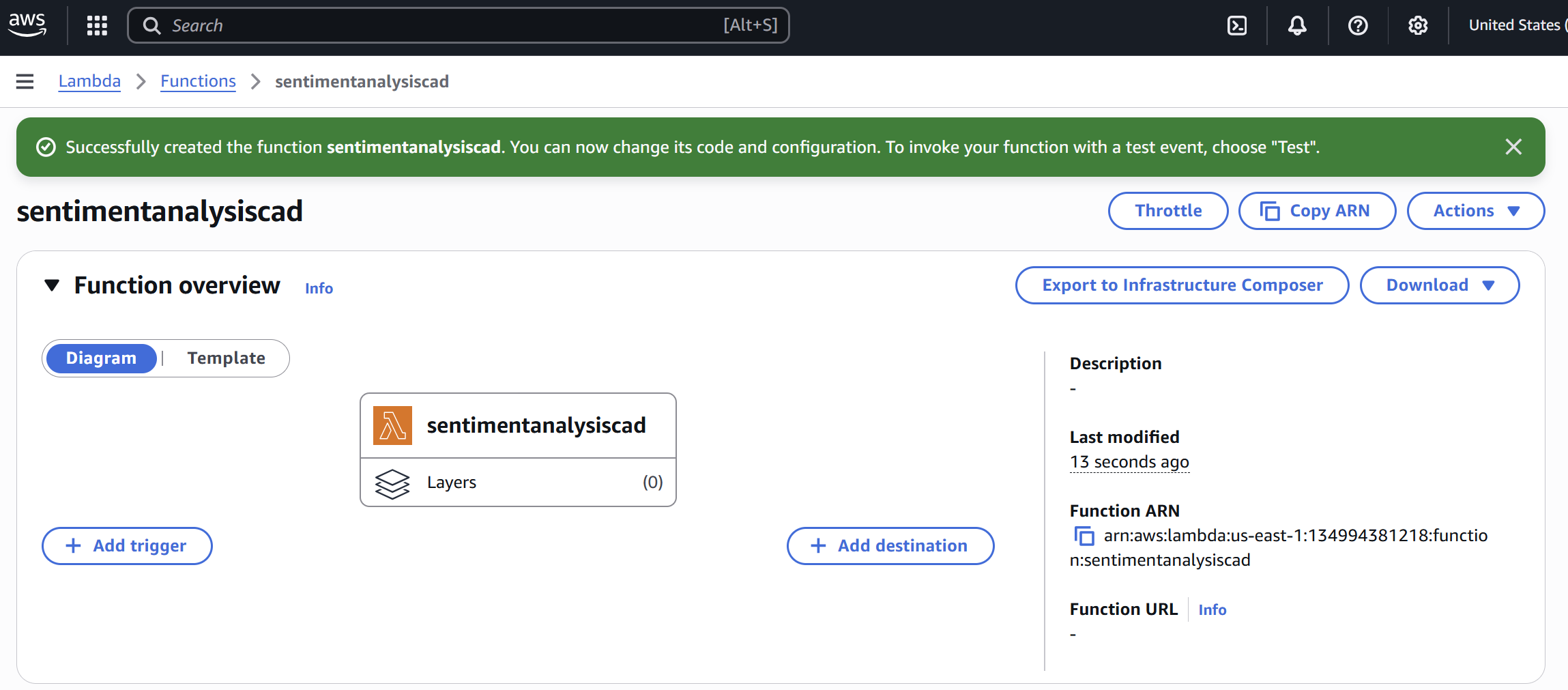
Amazon S3 provides secure, durable, and scalable object storage. It integrates seamlessly with Lambda, making it suitable for storing raw and processed tweet datasets. Its free tier offers 5 GB of standard storage, sufficient for initial data collection and testing.

**How Used:**

The raw tweets retrieved from Twitter API were stored in S3 in JSON or CSV format.

Cleaned and preprocessed tweet data were saved in a separate folder within the same bucket.

S3 event notifications were configured to automatically trigger Lambda functions for further processing whenever new data was uploaded.

**AWS Lambda**

**Purpose**: Serverless compute for automation and processing

**Why Used:**

AWS Lambda allows running backend code without managing any servers. It automatically scales with incoming tweet data, making it ideal for real-time or event-driven workloads. It also reduces costs since users pay only for execution time, and the free tier provides one million requests per month.

**How Used:**

A Lambda function was created to fetch live tweets from the Twitter/X API based on hashtags, usernames, or keywords.

Another Lambda function handled text preprocessing, including cleaning, removing links, emojis, and stopwords.

A final Lambda function performed sentiment analysis by calling Amazon Comprehend, which classifies each tweet as Positive, Negative, Neutral, or Mixed.

Lambda was configured to trigger automatically through Amazon EventBridge (scheduler) or S3 upload events, ensuring a completely serverless and automated workflow.

**Amazon DynamoDB**

A screenshot of a computer

AI-generated content may be incorrect.

**Purpose**: NoSQL database for storing analyzed results

**Why Used:**

DynamoDB is a fully managed NoSQL database that provides fast and predictable performance with seamless scalability. It eliminates the need for server maintenance and integrates efficiently with Lambda. The free tier includes 25 GB of storage and sufficient read/write capacity for small-scale projects.

**How Used:**

After sentiment analysis, the results (tweet ID, sentiment label, confidence scores, and timestamps) were stored as individual items in DynamoDB.

Each record could be queried later to generate reports or visualizations showing the percentage of positive, negative, and neutral tweets over time.

This structure enabled easy retrieval of recent analyses through APIs or dashboards.

**Amazon RDS (Relational Database Service)**

A screenshot of a computer

AI-generated content may be incorrect.

**Purpose**: Optional relational data storage for reporting and analytics

**Why Used:**

While DynamoDB is used for fast and unstructured storage, Amazon RDS (MySQL/PostgreSQL) is suitable for structured analytical queries, such as tracking sentiment trends for specific hashtags or time periods. The free tier includes 750 instance hours per month, allowing a small database instance to run continuously at no cost.

**How Used:**

Aggregated data such as daily or hashtag-wise sentiment counts were stored in an RDS table (e.g., sentiment\_summary).

SQL queries could be executed to identify top-trending sentiments or compare sentiment distributions.

RDS provided a foundation for connecting visualization tools like Amazon QuickSight for reporting purposes.

**Amazon Comprehend**

A screenshot of a computer

AI-generated content may be incorrect.

**Purpose**: Natural Language Processing (NLP) and Sentiment Analysis

**Why Used**:

Amazon Comprehend is a fully managed Natural Language Processing (NLP) service that uses machine learning to uncover insights from text. It can detect sentiment, extract entities, identify key phrases, and determine language — all without requiring any machine learning expertise or model training.

In this project, Comprehend was used because it provides an accurate and cost-effective way to classify the emotional tone of tweets as Positive, Negative, Neutral, or Mixed. Since the goal was to analyze public sentiment from social media data, Comprehend’s pre-trained sentiment analysis API was ideal. It integrates easily with AWS Lambda and supports real-time or batch processing of text data.

**How Used**:

After tweets were cleaned and preprocessed by AWS Lambda, each tweet text was sent to Amazon Comprehend through the AWS SDK (Python Boto3).

Comprehend’s DetectSentiment API was called, which returned the sentiment label along with confidence scores for each class.

These results were then passed back to the Lambda function, which stored them in DynamoDB or RDS for further use.

The service was used under the AWS Free Tier, which offers 50,000 units of text analysis per month, sufficient for processing thousands of tweets in the prototype stage.

A screenshot of a computer

AI-generated content may be incorrect.

**6. SYSTEM DESIGN**

**6.1 Architecture Overview**

The project uses the following AWS components:

* 1. **Amazon S3** – Hosts the static web front-end.
  2. **Amazon API Gateway** – Acts as the public HTTP interface for client requests.
  3. **AWS Lambda** – Executes the sentiment analysis logic.
  4. **Amazon Comprehend** – Performs text analysis using NLP.
  5. **AWS IAM** – Manages access permissions between services.

**AWS Services Used – What and How**

**1. AWS Lambda (Python Runtime)**

**What:**  
AWS Lambda is a serverless compute service that runs backend code in response to HTTP requests without managing servers.

**How Used:**

* The main **sentiment analysis logic** was written in Python (lambda\_function.py).
* The Lambda function receives text input from the frontend (via API Gateway).
* It calls **Amazon Comprehend** using the AWS SDK (boto3) to analyze the sentiment of the provided text.
* It returns a JSON response containing the detected sentiment (e.g., POSITIVE, NEGATIVE, NEUTRAL, MIXED) and confidence scores.
* Example flow inside Lambda:
* import boto3, json
* comprehend = boto3.client('comprehend')
* def lambda\_handler(event, context):
* body = json.loads(event['body'])
* text = body.get('text', '')
* response = comprehend.detect\_sentiment(Text=text, LanguageCode='en')
* return {
* 'statusCode': 200,
* 'headers': {'Access-Control-Allow-Origin': '\*'},
* 'body': json.dumps(response)
* }

**Why Lambda:**  
It eliminates the need for an EC2 or backend server. It scales automatically and is cost-effective for event-driven workloads.

**2. Amazon API Gateway (HTTP API)**

**What:**  
API Gateway acts as the **entry point** to the backend Lambda function through a public HTTP endpoint.

**How Used:**

* Configured an **HTTP API** that triggers the Lambda function on POST requests.
* Handles **CORS** (Cross-Origin Resource Sharing) to allow the static website (on S3) to call the API.
* The route $default or /analyze forwards requests to the Lambda integration.
* Example endpoint:
* https://<api-id>.execute-api.us-east-1.amazonaws.com/
* You verified its working via curl and used it in index.html with JavaScript fetch() to send text input.

**Why API Gateway:**  
It securely exposes Lambda as an HTTPS endpoint with automatic scaling, logging, and CORS management.

**3. Amazon Comprehend**

**What:**  
Amazon Comprehend is a Natural Language Processing (NLP) service that uses machine learning to analyze text.

**How Used:**

1. Called within the Lambda function using:
2. comprehend.detect\_sentiment(Text=text, LanguageCode='en')
3. Returns both the **sentiment category** (POSITIVE, NEGATIVE, NEUTRAL, MIXED) and **confidence scores**.
4. Example output:
5. {
6. "Sentiment": "POSITIVE",
7. "SentimentScore": {
8. "Positive": 0.97,
9. "Negative": 0.02,
10. "Neutral": 0.01,
11. "Mixed": 0.00
12. }
13. }

**Why Comprehend:**  
It provides pre-trained sentiment analysis without building ML models from scratch, saving time and effort.

**4. Amazon S3 (Static Website Hosting)**

**What:**  
Amazon S3 (Simple Storage Service) hosts the **frontend** as a static website.

**How Used:**

1. Hosted index.html, CSS, and JavaScript files.
2. Configured **Static Website Hosting** and made the bucket public with appropriate policies.
3. The website URL (e.g., http://aws-sentimentt.s3-website-us-east-1.amazonaws.com) calls the API Gateway endpoint.
4. The HTML/JS interface allows users to enter text and see the sentiment results instantly.

**Why S3:**  
It’s low-cost, highly available, and easy to integrate with API Gateway for static web apps.

**5. AWS CLI & AWS SAM (for deployment)**

**What:**

1. **AWS CLI:** Command-line tool to configure, test, and deploy AWS resources.
2. **AWS SAM (Serverless Application Model):** Simplifies packaging and deployment of serverless applications.

**How Used:**

1. Configured credentials with aws configure.
2. Used CLI commands to:
   1. Create API Gateway routes and integrations.
   2. Deploy Lambda function code (sam build, sam deploy).
   3. Set permissions and update configurations.
3. Verified endpoints and tested with curl commands.

**Why CLI & SAM:**  
They provide version-controlled, repeatable deployments and simplify managing multiple AWS services.

**Overall System Workflow**

1. **User Input:** User enters text in the S3-hosted website.
2. **Frontend Call:** JavaScript sends the text to the API Gateway endpoint.
3. **API Gateway:** Routes the request to AWS Lambda.
4. **Lambda Function:** Invokes Amazon Comprehend to detect sentiment.
5. **Comprehend Response:** Returns sentiment data to Lambda.
6. **Lambda Response:** Sends result back through API Gateway to frontend.
7. **Frontend Display:** Shows the sentiment result to the user dynamically.

**7. ARCHITECTURE DIAGRAM**

[User] → [HTML/JS Frontend on S3] → [API Gateway] → [Lambda Function] → [AWS Comprehend] → [Response to User]

**7.1 Data Flow**

1. User enters text into the web application.
2. JavaScript sends a POST request to the API Gateway endpoint.
3. API Gateway triggers the Lambda function.
4. Lambda invokes AWS Comprehend’s detect\_sentiment API.
5. Comprehend returns sentiment classification and confidence scores.
6. Lambda returns the response to the front-end through API Gateway.
7. The result is displayed in the user’s browser.

**8.IMPLEMENTATION**

**8.1 Tools and Services Used**

1. **AWS Lambda (Python Runtime)**
2. **Amazon API Gateway (HTTP API)**
3. **Amazon Comprehend**
4. **Amazon S3 (Static Website Hosting)**
5. **AWS CLI & AWS SAM (for deployment)**

**8.2 Lambda Function Code (Python)**

import json

import boto3

def lambda\_handler(event, context):

try:

body = json.loads(event.get("body", "{}"))

text = body.get("text", "")

if not text:

return {

"statusCode": 400,

"headers": {

"Access-Control-Allow-Origin": "\*",

"Access-Control-Allow-Methods": "OPTIONS,POST,GET",

"Access-Control-Allow-Headers": "\*"

},

"body": json.dumps({"error": "No text provided"})

}

comprehend = boto3.client("comprehend")

response = comprehend.detect\_sentiment(Text=text, LanguageCode="en")

return {

"statusCode": 200,

"headers": {

"Content-Type": "application/json",

"Access-Control-Allow-Origin": "\*",

"Access-Control-Allow-Methods": "OPTIONS,POST,GET",

"Access-Control-Allow-Headers": "\*"

},

"body": json.dumps({

"Sentiment": response["Sentiment"],

"SentimentScore": response["SentimentScore"]

})

}

except Exception as e:

return {

"statusCode": 500,

"headers": {

"Access-Control-Allow-Origin": "\*",

"Access-Control-Allow-Methods": "OPTIONS,POST,GET",

"Access-Control-Allow-Headers": "\*"

},

"body": json.dumps({"error": str(e)})

}

**8.3 Front-End Code (index.html)**

A simple HTML/JS interface hosted on **Amazon S3** for user interaction and API communication.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Sentiment Analysis using AWS Comprehend</title>

<style>

body {

font-family: "Segoe UI", Arial, sans-serif;

background-color: #f5f7fa;

text-align: center;

padding-top: 60px;

color: #333;

}

h2 {

color: #0073bb;

}

textarea {

width: 60%;

height: 120px;

font-size: 16px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 8px;

resize: none;

outline: none;

}

button {

margin-top: 15px;

padding: 10px 25px;

font-size: 16px;

background-color: #0073bb;

color: white;

border: none;

border-radius: 8px;

cursor: pointer;

transition: background-color 0.3s;

}

button:hover {

background-color: #005f99;

}

#result {

margin-top: 25px;

font-size: 18px;

text-align: left;

display: inline-block;

max-width: 600px;

word-wrap: break-word;

}

pre {

background-color: #f0f0f0;

padding: 10px;

border-radius: 6px;

text-align: left;

font-size: 14px;

overflow-x: auto;

}

.positive { color: green; font-weight: bold; }

.negative { color: red; font-weight: bold; }

.neutral { color: gray; font-weight: bold; }

.mixed { color: orange; font-weight: bold; }

.error { color: red; font-weight: bold; }

</style>

</head>

<body>

<h2>Sentiment Analysis using AWS Comprehend</h2>

<p>Enter your text below and click <b>Analyze Sentiment</b> to see how AWS Comprehend interprets it!</p>

<textarea id="textInput" placeholder="Type something like 'I love AWS!'..."></textarea><br>

<button onclick="analyzeSentiment()">Analyze Sentiment</button>

<div id="result"></div>

<script>

// 👇 Replace with your API Gateway endpoint

const API\_URL = "https://jy5tnaqd5j.execute-api.us-east-1.amazonaws.com";

async function analyzeSentiment() {

const text = document.getElementById("textInput").value.trim();

const resultDiv = document.getElementById("result");

if (!text) {

resultDiv.innerHTML = "⚠️ Please enter some text first.";

return;

}

resultDiv.textContent = "⏳ Analyzing sentiment...";

try {

const response = await fetch(API\_URL, {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ text }),

});

if (!response.ok) {

throw new Error(`API error: ${response.status}`);

}

const data = await response.json();

if (data.Sentiment) {

const sentiment = data.Sentiment.toLowerCase();

const sentimentScore = JSON.stringify(data.SentimentScore, null, 2);

resultDiv.innerHTML = `

✅ <b>Sentiment:</b> <span class="${sentiment}">${data.Sentiment}</span><br><br>

<b>Confidence Scores:</b>

<pre>${sentimentScore}</pre>

`;

} else {

resultDiv.innerHTML = "❌ Unexpected API response.";

}

} catch (err) {

console.error("Fetch error:", err);

resultDiv.innerHTML = "❌ <span class='error'>Error calling API. Check CORS or endpoint.</span>";

}

}

</script>

</body>

</html>

**8.4 AWS Configuration Steps**

1. Create an S3 bucket and enable static website hosting.
2. Upload index.html.
3. Create a Lambda function using the above Python code.
4. Attach an IAM role with permissions for Comprehend access.
5. Create an HTTP API in API Gateway and integrate it with Lambda.
6. Enable CORS configuration for all origins, headers, and methods.
7. Deploy the API and note the endpoint URL.
8. Update the front-end script to call this endpoint.

**9. TESTING AND RESULTS**

**9.1 Test Cases**

| **Input Text** | **Expected Sentiment** | **Result** |
| --- | --- | --- |
| I love AWS! | Positive | ✅ Positive |
| This is terrible. | Negative | ✅ Negative |
| It’s fine. | Neutral | ✅ Neutral |
| The product is okay but expensive. | Mixed | ✅ Mixed |

**9.2 API Response Example**

{

"Sentiment": "POSITIVE",

"SentimentScore": {

"Positive": 0.9982,

"Negative": 0.0002,

"Neutral": 0.0012,

"Mixed": 0.0003

}

}

**9.3 Performance**

1. **Response time:** < 1 second per request
2. **Scalability:** Automatically scales to concurrent users
3. **Cost efficiency:** Zero cost when idle

**10.SCREENSHOTS OF THE PROJECT**

**AWS TERMINAL**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**FRONTEND**

A screenshot of a computer

AI-generated content may be incorrect.

**11.SECURITY AND IAM ROLES**

1. Lambda execution role includes permission to use comprehend:DetectSentiment.
2. API Gateway configured with CORS for controlled cross-origin access.
3. S3 bucket set to public-read only for static web assets.

**12.CHALLENGES AND SOLUTIONS**

| **Challenge** | **Description** | **Solution** |
| --- | --- | --- |
| CORS Errors | Browser blocked API responses | Configured Access-Control-Allow-Origin headers and updated CORS in API Gateway |
| API Gateway 400 Error | Lambda received no payload | Verified POST body and JSON parsing logic |
| Bucket Policy Errors | S3 denied access to website files | Replaced ACLs with proper bucket policy and static hosting configuration |

**13.ADVANTAGES OF SERVERLESS DESIGNS**

1. No infrastructure management
2. Auto-scaling and high availability
3. Pay-per-use pricing
4. Seamless integration with other AWS services
5. Reduced development complexity

**14.FUTURE ENHANCEMENTS**

1. Add **language detection** using detect\_dominant\_language.
2. Extend support for **batch sentiment analysis** of multiple texts.
3. Include **user authentication** using AWS Cognito.
4. Visualize results using Amazon QuickSight or AWS Amplify dashboards.

**15.CONCLUSION**

The project successfully demonstrates a **cloud-native sentiment analysis system** using **AWS Comprehend**, **Lambda**, and **API Gateway**.  
It achieves the goals of scalability, efficiency, and cost optimization through serverless architecture.  
This implementation can serve as a foundational model for advanced NLP-based cloud solutions in various real-world applications like customer feedback analysis, social media monitoring, and business intelligence.

**16.REFERENCES**

* AWS Documentation: <https://docs.aws.amazon.com>
* AWS Comprehend Developer Guide
* AWS Lambda Developer Guide
* Amazon API Gateway User Guide