# **Implementing a Lambda Function for Fetching Cloud Terms**

## **Project Overview**

I set up a Lambda function named FetchTermFromDynamoDB in AWS to fetch cloud-related terms and their definitions from the CloudDefinitions DynamoDB table. The function logic uses code provided in the Tech With Lucy project. This serverless function allows the frontend to query specific terms dynamically, providing a fast, scalable, and fully managed backend solution for the Cloud Dictionary Application.

## **Architecture and Services Involved**

The Lambda function interacts with:

* **AWS DynamoDB**: Stores cloud terms and definitions.
* **IAM Roles**: Provides secure, least-privilege access for Lambda.
* **AWS Lambda**: Handles requests and retrieves data from DynamoDB in real time.

## **Implementation**

**Step 1: Creating the IAM Role for Lambda**

I created an IAM role named LambdaDynamoDBAccessRole with the following policies:

* AmazonDynamoDBReadOnlyAccess – allows querying DynamoDB.
* AWSLambdaBasicExecutionRole – provides standard Lambda execution permissions.

This role guarantees the Lambda function has the required permissions to retrieve data from DynamoDB securely.

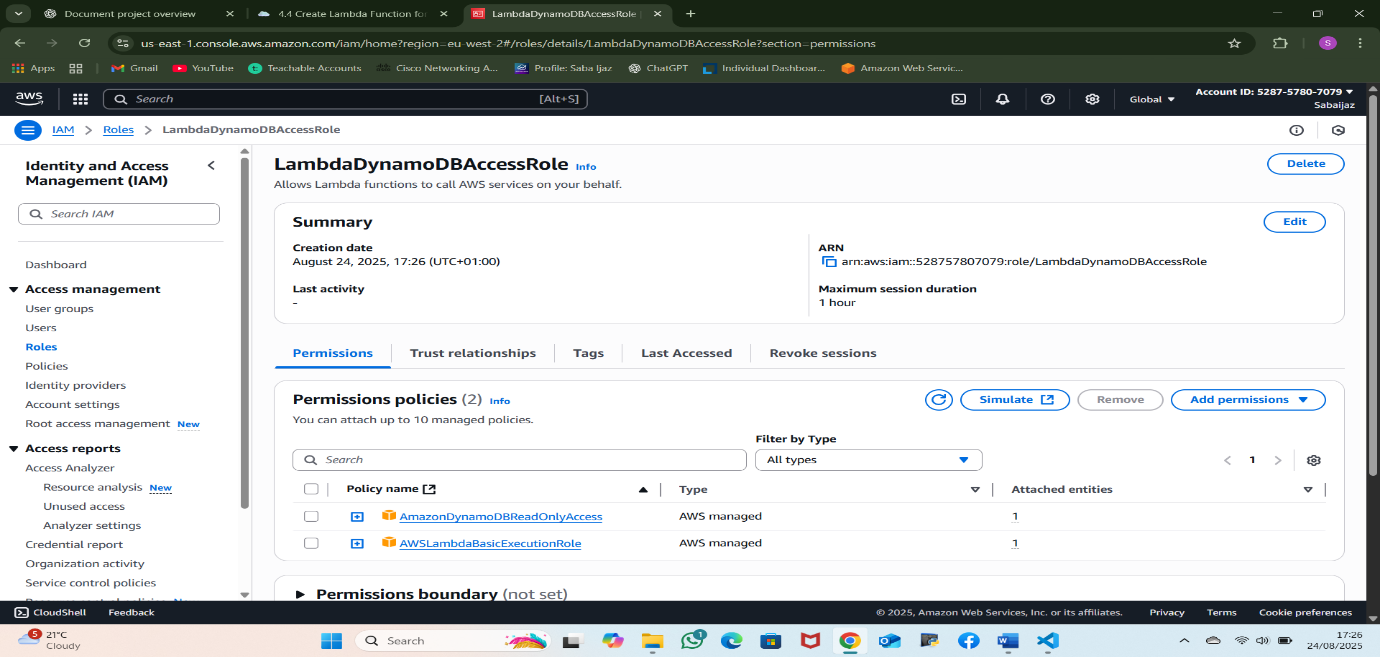


Figure 1: IAM role LambdaDynamoDBAccessRole with attached policies.

**Step 2: Developing the Lambda Function**

I deployed a Lambda function named FetchTermFromDynamoDB using Python 3.12 runtime. The Lambda was assigned the LambdaDynamoDBAccessRole IAM role for secure access.

The function logic, sourced from the Tech With Lucy project, queries DynamoDB for a specific term and returns the associated definition.

**Lambda Function Code**

import json

import boto3

dynamodb = boto3.client('dynamodb')

table\_name = 'CloudDefinitions'

def lambda\_handler(event, context):

term = event['queryStringParameters']['term']

response = dynamodb.get\_item(

TableName=table\_name,

Key={'term': {'S': term}}

)

if 'Item' in response:

definition = response['Item']['definition']['S']

return {

'statusCode': 200,

'headers': {

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

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

'Access-Control-Allow-Methods': 'OPTIONS,GET',

'Access-Control-Allow-Headers': 'Content-Type',

},

'body': json.dumps({'term': term, 'definition': definition})

}

else:

return {

'statusCode': 404,

'headers': {

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

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

'Access-Control-Allow-Methods': 'OPTIONS,GET',

'Access-Control-Allow-Headers': 'Content-Type',

},

'body': json.dumps({'message': 'Term not found'})

}

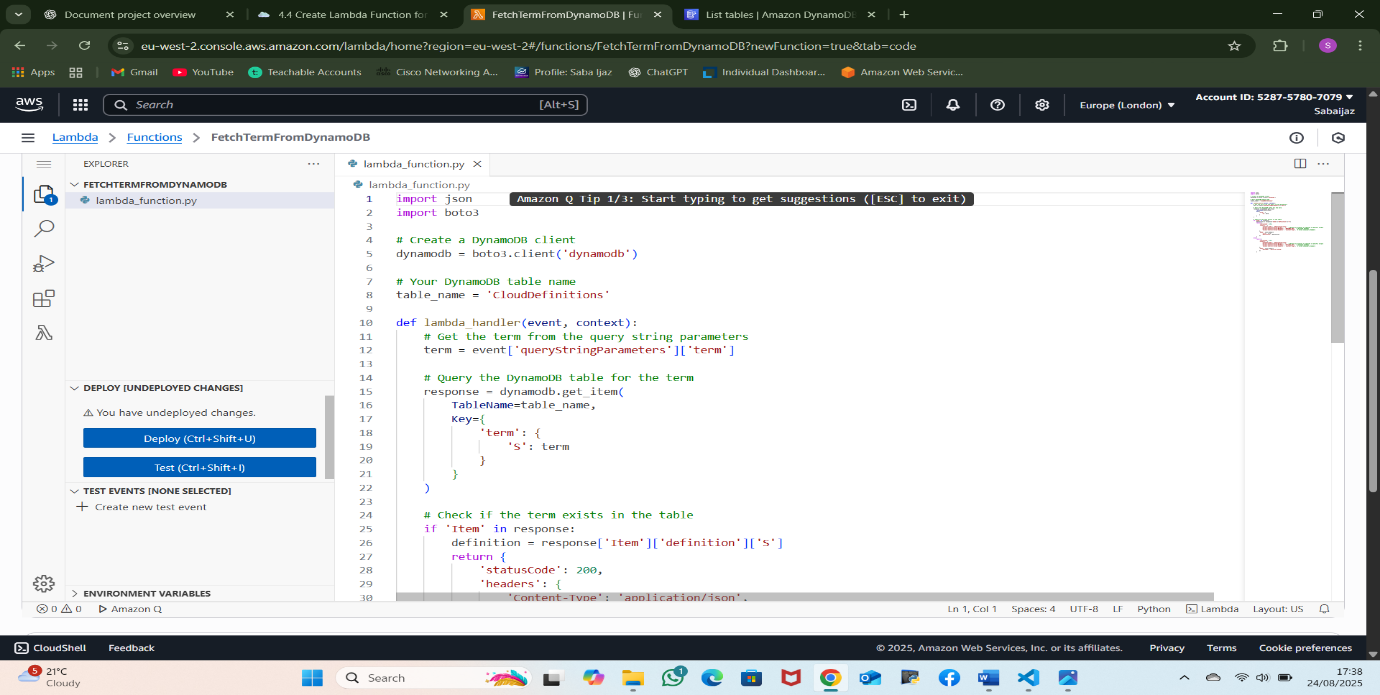


Figure 2: Lambda function code editor in AWS Console.

**Step 3: Testing the Lambda Function**

I tested the Lambda function using a simulated query event:

{

"queryStringParameters": {

"term": "AWS KMS"

}

}

Results:

* If the term exists, the function returns 200 OK with the term and definition.
* If the term does not exist, it returns 404 Not Found with a descriptive message.

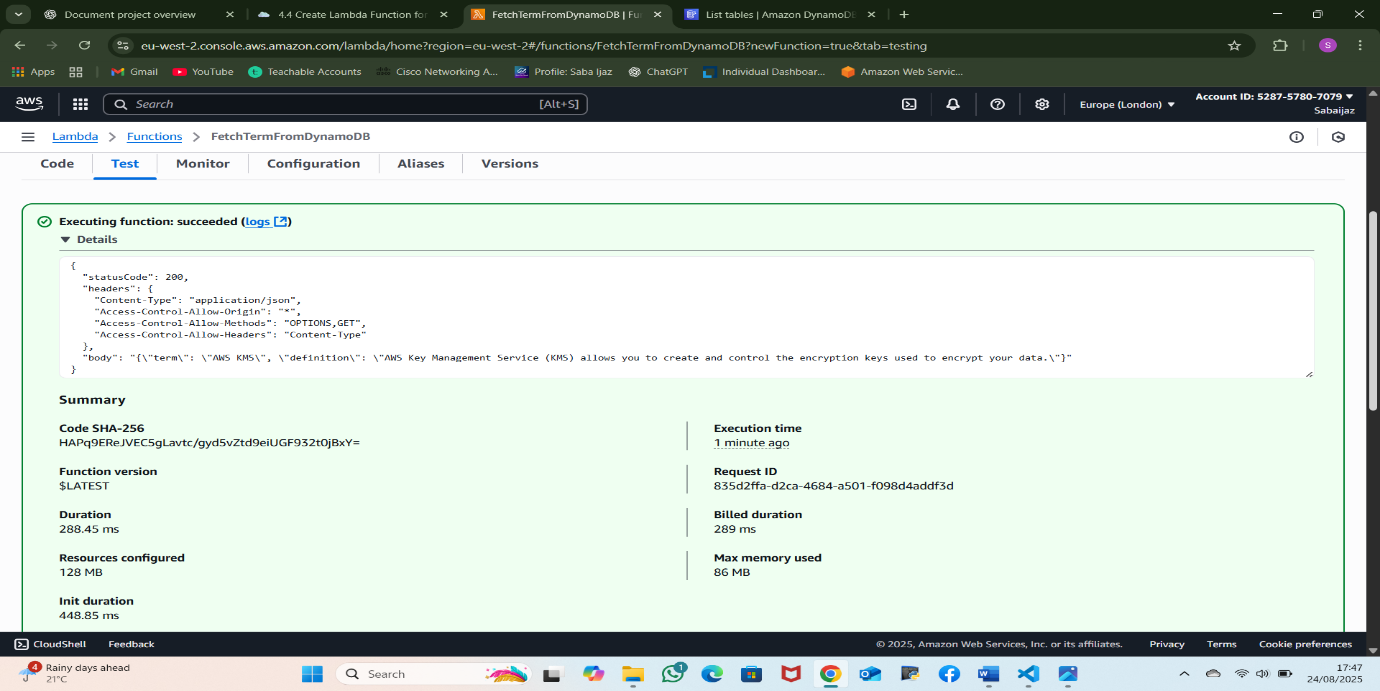


Figure 3: Lambda test execution showing successful retrieval from DynamoDB.

**Outcome**

The Lambda function offers a serverless backend for retrieving cloud terms from DynamoDB, ensuring quick and adaptable term access. It securely connects with API Gateway and supports the React frontend, showcasing hands-on experience with AWS serverless services and efficient backend design.