API-LAMBDA-DYNAMODB

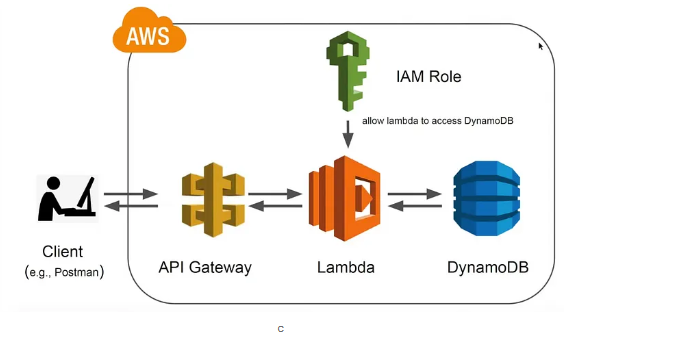
# **AWS API Gateway, Lambda, and DynamoDB Integration**

## **Project Overview**

This project demonstrates how to integrate AWS API Gateway with AWS Lambda and DynamoDB. A contact form hosted on a web page collects user details and submits them to an API Gateway, which triggers a Lambda function to store the data in DynamoDB.

## **AWS Services Used**

* **API Gateway**: Exposes the REST API endpoint to receive HTTP requests.
* **AWS Lambda**: Processes incoming HTTP requests and interacts with DynamoDB.
* **DynamoDB**: Stores the form data in a NoSQL database.
* **Amazon S3** (Optional): Hosts the web page for user interaction.



## **Step-by-Step Implementation**

### **1. Setting Up DynamoDB Table**

1. Navigate to the **AWS Management Console**.
2. Open **DynamoDB** and create a new table named avinashtable.
3. Set the **Primary Key** as email (String type) to uniquely identify each record.
4. Click **Create Table** and wait for the table to be provisioned.

### **2. Creating the Lambda Function**

1. Open **AWS Lambda** and create a new function.
2. Select **Author from scratch**.
3. Name the function storeContactData.
4. Select **Python 3.x** as the runtime.
5. Attach an IAM role with **DynamoDB Full Access** permission.
6. Use the following Python code to handle incoming requests and store data in DynamoDB:

import json

import boto3

def lambda\_handler(event, context):

try:

http\_method = event['httpMethod']

if http\_method == 'POST':

body = json.loads(event['body'])

client = boto3.client('dynamodb')

response = client.put\_item(

TableName='avinashtable',

Item={

'email': {'S': body['email']},

'fname': {'S': body['fname']},

'lname': {'S': body['lname']},

'message': {'S': body['message']}

}

)

return {'statusCode': 200, 'body': json.dumps('Data stored successfully')}

else:

return {'statusCode': 405, 'body': json.dumps('Method Not Allowed')}

except Exception as e:

return {'statusCode': 500, 'body': json.dumps({'error': str(e)})}

### **3. Creating API Gateway**

1. Open **API Gateway** in the AWS Console.
2. Create a new **REST API**.
3. Click on **Resources** and create a new resource named /contact.
4. Select the newly created resource and add a **POST method**.
5. Configure the **POST method** to trigger the storeContactData Lambda function.
6. Deploy the API to a new stage (e.g., dev) and note the **Invoke URL**.

### **4. Creating the Web Form**

Create an HTML contact form to submit data to the API Gateway endpoint:

<form action="https://your-api-gateway-url/contact" method="post">

<label for="fname">First Name:</label>

<input type="text" id="fname" name="fname" required>

<label for="lname">Last Name:</label>

<input type="text" id="lname" name="lname" required>

<label for="email">Email:</label>

<input type="text" id="email" name="email" required>

<label for="message">Message:</label>

<textarea id="message" name="message" required></textarea>

<input type="submit" value="Submit">

</form>

### **5. Testing the Setup**

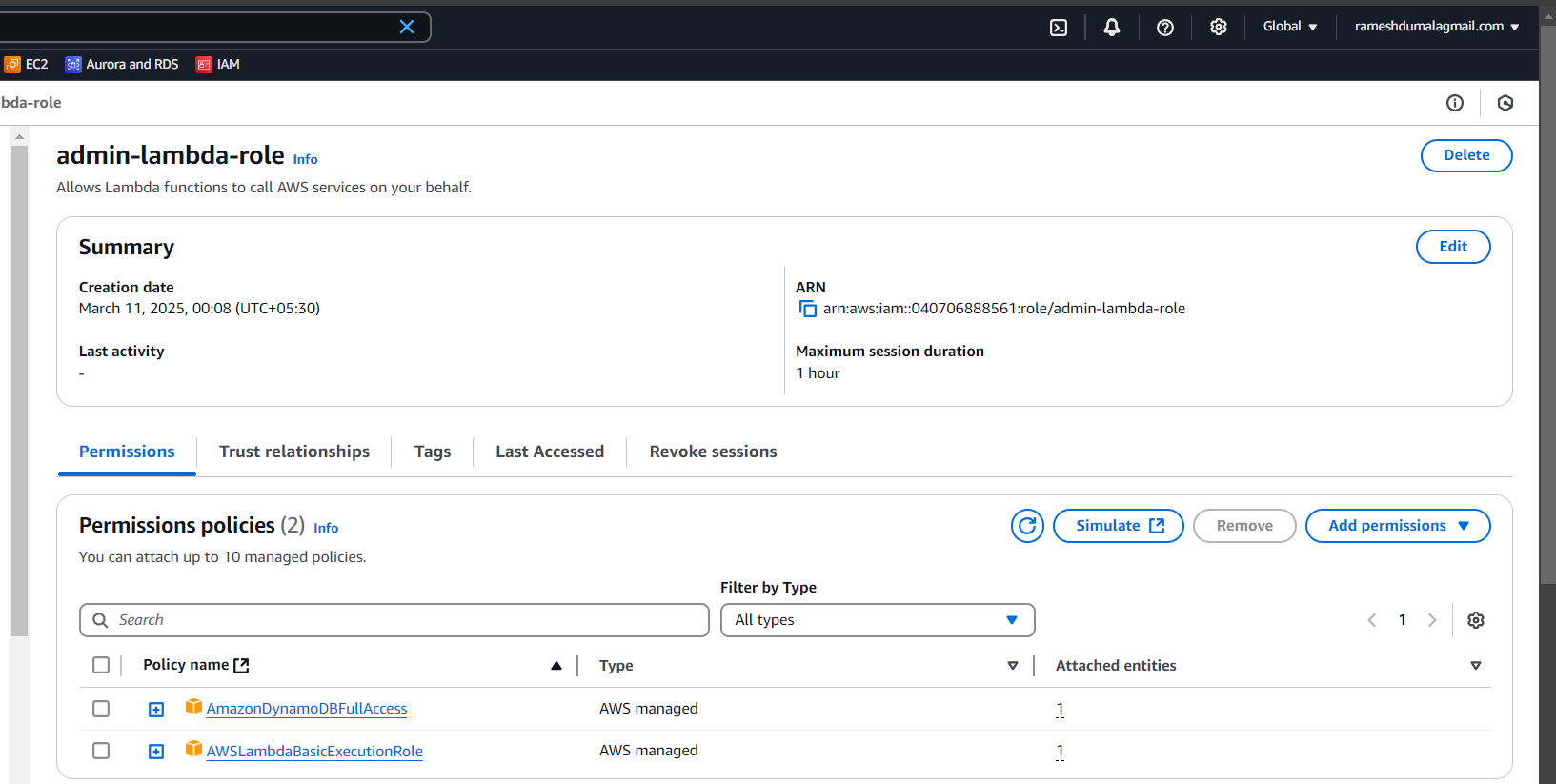
1. Open the contact form in a web browser.
2. Fill in the required details and submit the form.
3. Go to **DynamoDB Console** and verify that the submitted data appears in the avinashtable table.

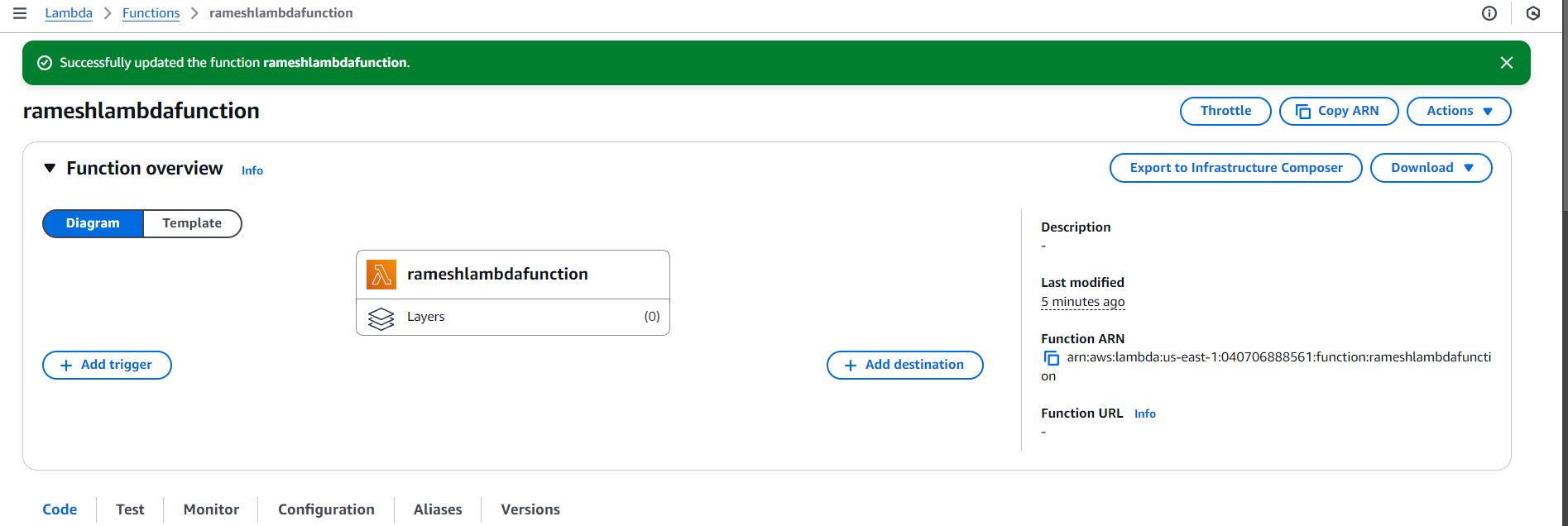
## **Execution Screenshots**

(Screenshots to be added for each step, including DynamoDB table, Lambda function, API Gateway setup, and form submission results.)

## **Conclusion**

This setup successfully integrates API Gateway, Lambda, and DynamoDB to process and store user input from a web form securely and efficiently. This approach allows for serverless, scalable, and cost-effective management of contact form submissions.





<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Contact Form</title>

<style>

body {

font-family: 'Arial', sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

flex-direction: column;

align-items: center;

height: 100vh;

}

form {

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

transition: transform 0.3s ease-in-out;

}

h2 {

text-align: center;

color: #333;

margin-bottom: 20px;

}

label {

display: block;

margin: 10px 0 5px;

color: #555;

}

input,

textarea {

width: 100%;

padding: 10px;

margin-bottom: 10px;

box-sizing: border-box;

border: 1px solid #ccc;

border-radius: 4px;

}

input[type="submit"] {

background-color: #4caf50;

color: #fff;

cursor: pointer;

font-size: 16px; /\* Increased text size for submit button \*/

}

input[type="submit"]:hover {

background-color: #45a049;

}

/\* Additional styles for animation \*/

form:hover {

transform: scale(1.05);

}

/\* Additional styles for footer \*/

footer {

margin-top: 20px;

font-size: 18px;

text-align: center;

}

</style>

</head>

<body>

<h2>Welcome to my AWS Project Task</h2>

<form action="/dev" method="post">

<h2>Contact Us</h2>

<label for="fname">First Name:</label>

<input type="text" id="fname" name="fname" required>

<label for="lname">Last Name:</label>

<input type="text" id="lname" name="lname" required>

<label for="email">Email ID:</label>

<input type="text" id="email" name="email" required>

<label for="message">Message:</label>

<textarea id="message" name="message" rows="4" cols="50" required></textarea>

<input type="submit" value="Submit">

</form>

<footer>

An AWS Project by Avinash Reddy Thipparthi

</footer>

<!-- Simple JavaScript for animation -->

<script>

const form = document.querySelector('form');

form.addEventListener('mouseover', () => {

form.style.transform = 'scale(1.05)';

});

form.addEventListener('mouseout', () => {

form.style.transform = 'scale(1)';

});

</script>

</body>

</html>

import json

import os

import boto3

def lambda\_handler(event, context):

try:

mypage = page\_router(event['httpMethod'], event['queryStringParameters'], event['body'])

return mypage

except Exception as e:

return {

'statusCode': 500,

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

}

def page\_router(httpmethod, querystring, formbody):

if httpmethod == 'GET':

try:

with open('contactus.html', 'r') as htmlFile:

htmlContent = htmlFile.read()

return {

'statusCode': 200,

'headers': {"Content-Type": "text/html"},

'body': htmlContent

}

except Exception as e:

return {

'statusCode': 500,

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

}

elif httpmethod == 'POST':

try:

insert\_record(formbody)

with open('success.html', 'r') as htmlFile:

htmlContent = htmlFile.read()

return {

'statusCode': 200,

'headers': {"Content-Type": "text/html"},

'body': htmlContent

}

except Exception as e:

return {

'statusCode': 500,

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

}

def insert\_record(formbody):

formbody = formbody.replace("=", "' : '")

formbody = formbody.replace("&", "', '")

formbody = "INSERT INTO avinashtable value {'" + formbody + "'}"

client = boto3.client('dynamodb')

response = client.execute\_statement(Statement=formbody)

# Assuming the execute\_statement call returns successfully

return response

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Thank You</title>

<style>

body {

font-family: 'Arial', sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

h2 {

text-align: center;

color: #333;

padding: 20px;

background-color: #fff;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

opacity: 0;

transform: translateY(20px);

transition: opacity 0.5s ease-in-out, transform 0.5s ease-in-out;

}

/\* Additional styles for animation \*/

.visible {

opacity: 1;

transform: translateY(0);

}

</style>

</head>

<body>

<h2 id="thankYouMessage">Thanks for trying this Project. you can verify data in DynamoDB Table.</h2>

<!-- Simple JavaScript for animation -->

<script>

const thankYouMessage = document.getElementById('thankYouMessage');

// Adding a delay before showing the message

setTimeout(() => {

thankYouMessage.classList.add('visible');

}, 500);

</script>

</body>

</html>

