

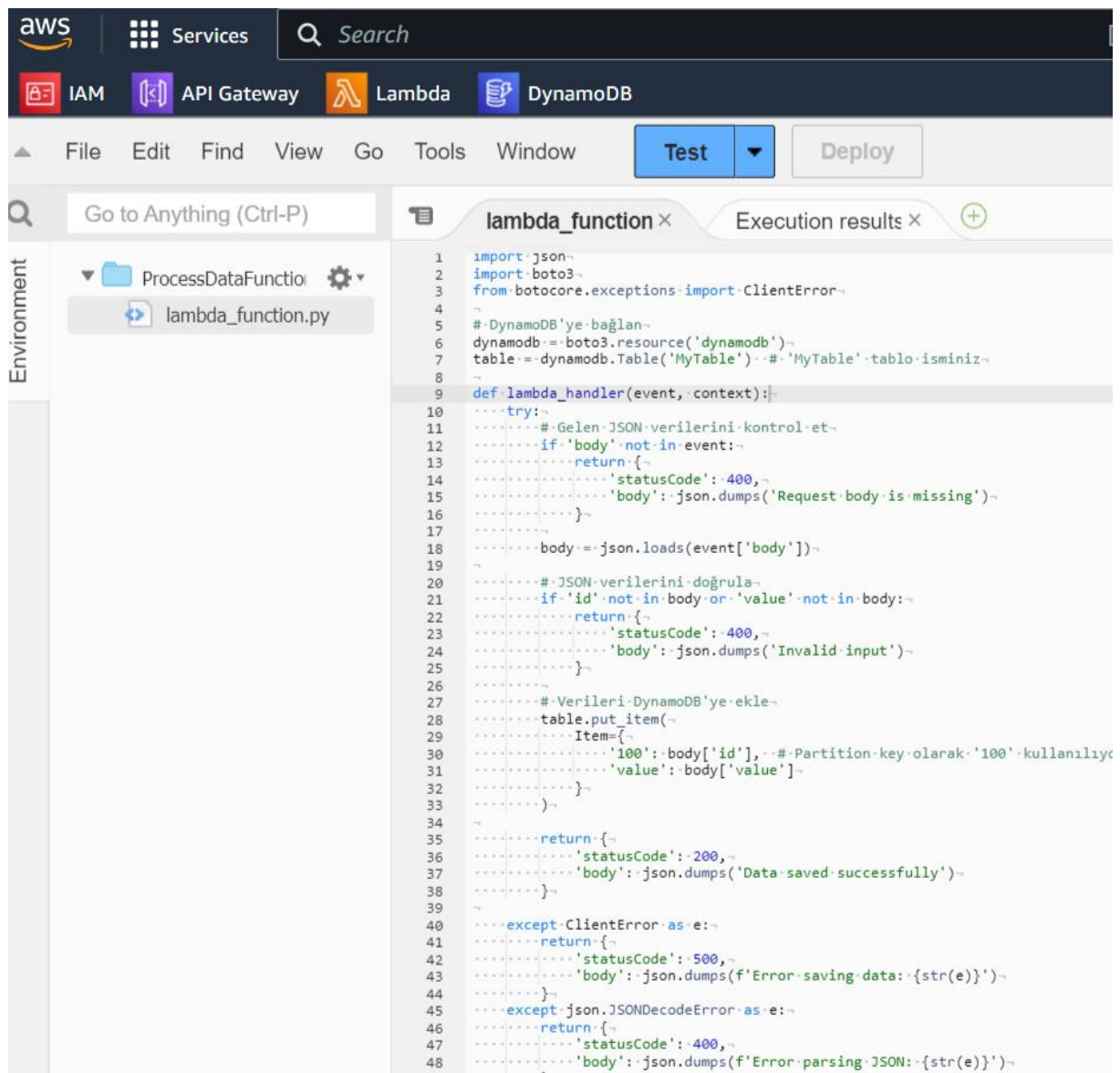
## Objective: Develop and Deploy AWS Lambda Functions

The goal of this task was to create and deploy an AWS Lambda function that processes incoming data and stores it in a DynamoDB table. The Lambda function is triggered by an AWS API Gateway endpoint, and it is responsible for validating and parsing the incoming JSON data, storing valid entries in the DynamoDB table, and handling any errors that occur during the process.

### Lambda Function Code

I opened a new file to write the AWS Lambda function code. The code was written to process specific input and generate the required output based on the function's logic. The Lambda function was then successfully tested with sample input.

Below is the AWS Lambda function code as displayed in the AWS Lambda console:



```
1 import json
2 import boto3
3 from botocore.exceptions import ClientError
4
5 # DynamoDB'ye bağlan
6 dynamodb = boto3.resource('dynamodb')
7 table = dynamodb.Table('MyTable')
8
9 def lambda_handler(event, context):
10     try:
11         # Gelen JSON verilerini kontrol et
12         if 'body' not in event:
13             return {
14                 'statusCode': 400,
15                 'body': json.dumps('Request body is missing')
16             }
17
18         body = json.loads(event['body'])
19
20         # JSON verilerini doğrula
21         if 'id' not in body or 'value' not in body:
22             return {
23                 'statusCode': 400,
24                 'body': json.dumps('Invalid input')
25             }
26
27         # Verileri DynamoDB'ye ekle
28         table.put_item(
29             Item={
30                 'id': body['id'], # Partition key olarak 'id' kullanılıyor
31                 'value': body['value']
32             }
33         )
34
35         return {
36             'statusCode': 200,
37             'body': json.dumps('Data saved successfully')
38         }
39
40     except ClientError as e:
41         return {
42             'statusCode': 500,
43             'body': json.dumps(f'Error saving data: {str(e)}')
44         }
45
46     except json.JSONDecodeError as e:
47         return {
48             'statusCode': 400,
49             'body': json.dumps(f'Error parsing JSON: {str(e)}')
```

## Testing the Lambda Function

The test was successful, as shown in the screenshot below, where the function processed the data and stored it in DynamoDB without any issues:



The Lambda function was tested using the following URL:

URL: <https://p7nu3fohcuo7tbrwexnqfpgkla0zsapc.lambda-url.us-east-1.on.aws/>

## Sample Input/Output Document

### 1. Sample Input (JSON Format)

This is an example of the data that is sent to the AWS Lambda function via the API Gateway.

```
{ "userId": "12345", "username": "johndoe", "email": "johndoe@example.com", "age": 29 }
```

### Explanation:

- **userId:** A unique identifier for the user.
- **username:** The user's name.
- **email:** The user's email address.
- **age:** The user's age.

### 2. Expected Output

If the input data is valid, the Lambda function will store the data in the DynamoDB table and return a success response.

Json.

```
{  
  
  "statusCode": 200, "body": { "message": "Data successfully saved to DynamoDB" } }
```

### 3. Error Response Example

If the input data is invalid (for example, missing a required field), the Lambda function will return an error message.

```
{ "statusCode": 400, "body": {"error": "Invalid input data. 'email' field is missing." } }
```

### Guide on Setting Up AWS Resources

This guide walks through the steps needed to set up the required AWS resources for the project.

#### Step 1: Create a DynamoDB Table

1. Open the **AWS Management Console**.
2. Go to **DynamoDB**.
3. Click on **Create table**.
4. Provide a **Table name** (e.g., UsersTable).
5. Set the **Partition key** as `userId` (String).
6. Configure read/write capacity (auto-scaling or specific settings).
7. Click **Create** to finalize the table.

#### Step 2: Create the AWS Lambda Function

1. Go to **AWS Lambda** in the AWS Management Console.
2. Click **Create Function**.
3. Select **Author from scratch**.
4. Name your function (e.g., ProcessUserData).
5. In the **Permissions** section, ensure the Lambda function has permissions to write to DynamoDB (by creating an IAM role with DynamoDB write access).
6. Write or upload the Lambda code.
7. Set the environment variables if needed (e.g., the DynamoDB table name).

#### Step 3: Set Up API Gateway

1. Go to **API Gateway** in the AWS Console.
2. Click **Create API** and choose **HTTP API**.
3. Set up a new API and configure the method (e.g., POST) that triggers the Lambda function.

4. Link the API Gateway endpoint to the Lambda function by selecting it in the **Integration Request**.
5. Deploy the API and note the endpoint URL for testing.

#### **Step 4: Test the Lambda Function**

1. Send a POST request to the API Gateway endpoint using a tool like Postman or cURL.
2. Use a sample JSON input (as shown in the **Sample Input/Output Document** section).
3. Verify the data is stored in DynamoDB and review the Lambda function's logs using **CloudWatch**.