



[nextwork.org](http://nextwork.org)

# Fetch Data with AWS Lambda

ME

Melvin J Bonner

Executing function: succeeded ([logs](#))

▼ Details

```
{  
  "email": "test@example.com",  
  "name": "Test User",  
  "userId": "1"  
}
```

# Introducing Today's Project!

In this project, I will demonstrate how to store and manage data relevant to this application. I'm doing this project to learn how to connect DynamoDB to a Lambda Function

## Tools and concepts

The services I used were DynamoDB and Lambda. The key concepts I learned included Lambda functions, IAM permission policies, and DynamoDB tables.

## Project reflection

This project took me approximately 3 hours. The most challenging part was updating the IAM permissions and testing the Lambda function. It was most rewarding to get the positive results on the Lambda function test.

I did this project to complete the four part three-tier architecture series. This project met my goals.

# Project Setup

To set up my project, I created a database using DynamoDB. The partition key is the heart of how DynamoDB organizes data. Think of the partition key as a label that you can use to group similar items.

In my DynamoDB table, I added a new item. DynamoDB is schemaless, which means you can add attributes as you need, and every item in your database can have a different set of attributes.



A screenshot of a JSON editor interface. The title bar says "Attributes" and "View DynamoDB JSON". The main area contains the following JSON code:

```
1 * {
2   "userId": "1",
3   "name": "Test User",
4   "email": "test@example.com"
5 }
6 |
```

The bottom status bar shows "JSON" and "Line 6, Col 1" along with error and warning counts: "Errors: 0" and "Warnings: 0". There is also a "Copy" button in the top right corner.

## AWS Lambda

AWS Lambda is a service that lets you run code without needing to manage any servers. I'm using Lambda in this project to process data and respond to events.

# AWS Lambda Function

My Lambda functions has an execution role, which is create a new new role with basic Lambda permissions. By default, the role grants basic permissions for writing logs to CloudWatch.

My Lambda function will use the AWS SDK for JavaScript to interact with DynamoDB and handle potential errors during the database operation.

The code uses AWS SDK, which is a set of tools that let developers build apps that interact with AWS. My code uses SDK to use prewritten functions for communicating with DynamoDB and getting data from a table.

ME

# Melvin J Bonner

## NextWork Student

nextwork.org

The screenshot shows the AWS Lambda function editor interface. At the top, a green success message says "Successfully updated the function RetrieveUserData." Below it, the "Code source" tab is selected. The code editor displays the following JavaScript function:

```
index.js 1
JS index.js > handler
1 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
2 import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
3 
4 const ddBClient = new DynamoDBClient({ region: 'us-east-1' }); // Make sure to replace this with your actual AWS region
5 const db = DynamoDBDocumentClient.from(ddBClient);
6 
7 async function handler(event) {
8     const userId = String(event.userId); // Make sure to extract userId from the event
9     const params = {
10         TableName: 'userData',
11         Key: { userId }
12     };
13 
14     try {
15         const command = new GetCommand(params);
16         const data = await db.send(command); // Log the raw response from DynamoDB
17         console.log("DynamoDB Response:", JSON.stringify(data));
18         const item = data.Item;
19         if (item) {
20             console.log("User data retrieved:", item);
21             return item;
22         } else {
23             console.log("No user data found for userId:", userId);
24             return null;
25         }
26     } catch (err) {
27         console.error("Unable to retrieve data:", err);
28         return err;
29     }
30 }
```

The left sidebar shows the project structure with a file named "RETRIEVEUSERDATA" containing "index.js". The "TEST EVENTS" section is collapsed. At the bottom, there are "Deploy" and "Test" buttons.

# Function Testing

To test whether my Lambda function works, I will be writing a test that will show me how the function would react if a data request was made. The test is written in Json. If the test is successful, I will see Executing function succeeded.

This test displayed a 'success' because the function itself could run. The function's response was a failed test because Lambda doesn't have permission to access the DynamoDB table.

```
{
  "errorType": "RuntimeUserCodeSyntaxError",
  "errorMessage": "SyntaxError: Unexpected token '{'",
  "trace": [
    "RuntimeUserCodeSyntaxError: SyntaxError: Unexpected token '{'",
    "  at _loadUserApp (file:///var/runtime/index.mjs:1089:17)",
    "  at async UserFunction.js.module.exports.load (file:///var/runtime/index.mjs:1131:21)",
    "  at async start (file:///var/runtime/index.mjs:1300:23)",
    "  at async file:///var/runtime/index.mjs:1306:1"
  ]
}
```

# Function Permissions

To resolve the AccessDenied error I added one permission policy, DynamoDBReadOnlyAccess, it contains the GetItem permission that is needed for Lambda.

There were 6 DynamoDB permission policies I could have chosen from, but I didn't pick FullAccess, FullAccess v2, FullAccess with Data Pipeline, DBExecution, or Invocation access because Lambda needs the GetItem permission.

I also didn't pick FullAccess, FullAccess v2, FullAccess with Data Pipeline, DBExecution, or Invocation because AmazonDynamoDBReadOnlyAccess was the write choice because it contained the GetItem that was needed for Lambda.

ME

Melvin J Bonner  
NextWork Student

[nextwork.org](http://nextwork.org)

Filter by type

X All types ▼ 6 matches

Policy name ▲ | Type

Policy name	Type
<input type="checkbox"/>  AmazonDynamoDBFullAccess	AWS managed
<input type="checkbox"/>  AmazonDynamoDBFullAccess_v2	AWS managed
<input type="checkbox"/>  AmazonDynamoDBFullAccesswithDataPipeline	AWS managed
<input checked="" type="checkbox"/>  AmazonDynamoDBReadOnlyAccess	AWS managed
<input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole	AWS managed
<input type="checkbox"/>  AWSLambdaInvocation-DynamoDB	AWS managed

# Final Testing and Reflection

To validate my permission settings, I reran the test event. The results were successful because my Lambda function now has the necessary permissions to access DynamoDB.

Web apps are a popular use case for using Lambda and DynamoDB. For example, I could use Lambda to help customers find products, get product information or see their order history by fetching data from DynamoDB.



Executing function: succeeded ([Logs](#))  
▼ Details

```
{  
  "email": "test@example.com",  
  "name": "Test User",  
  "userId": "1"  
}
```

# Enhancing Security

For my project extension, I challenged myself to create an inline policy. This will allow for more granular control, I want my Lambda function to only have access to my UserData table.

To create the permission policy, I used JSON because the process is quicker when you have the JSON code.

When updating a Lambda function's permission policies, I could risk granting broader permissions than necessary and allowing unintended access to other AWS services. I validated that my Lambda function still works by testing the function and receiving positive results.

ME

Melvin J Bonner  
NextWork Student

[nextwork.org](http://nextwork.org)

### Policy details

#### Policy name

Enter a meaningful name to identify this policy.

DynamoDB\_UserData\_ReadAccess

Maximum 128 characters. Use alphanumeric and '+,.,@-\_ characters.



[nextwork.org](https://nextwork.org)

# The place to learn & showcase your skills

Check out [nextwork.org](https://nextwork.org) for more projects

