



Fetch Data with AWS Lambda



Louis Moyo

Executing function: succeeded (logs [2])

▼ Details

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

Summary

Code SHA-256 cVVV0D52L54MgZ644f4sBjycxXoaza5aOKEoKXrBFo=	Execution time 2 minutes ago
Function version \$LATEST	Request ID 0F38F70e-98d2-4ce2-9204-6c363e9853b9
Duration 966.28 ms	Billed duration 967 ms
Resources configured 128 MB	Max memory used 100 MB
Init duration 427.99 ms	

Log output

The area below shows the last 4 KB of the execution log. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: 0F38F70e-98d2-4ce2-9204-6c363e9853b9 Version: $LATEST
2023-08-22T04:25:53.752Z          INFO  [Container] Received: {"email": "test@example.com", "name": "Test User", "userId": "1"}
"requestId": "0F38F70e-98d2-4ce2-9204-6c363e9853b9", "region": "us-east-1", "latency": 1, "totalLatency": 967, "delay": 0, "item": {"email": "test@example.com", "name": "Test User", "userId": "1"}
END RequestId: 0F38F70e-98d2-4ce2-9204-6c363e9853b9          INFO  User data retrieved: { email: 'test@example.com', name: 'Test User', userId: '1' }
REPORT RequestId: 0F38F70e-98d2-4ce2-9204-6c363e9853b9 Duration: 966.28 ms    Billed Duration: 967 ms Memory Size: 128 MB Max Memory Used: 100 MB Init Duration: 427.99 ms
```



Introducing Today's Project!

In this project, I will demonstrate how to use AWS Lambda to fetch data from a DynamoDB table. I'm doing this project to learn how serverless functions interact with databases, how to secure access with permissions, and how the data tier fits into a three-tier cloud architecture.

Tools and concepts

Services I used were AWS Lambda and Amazon DynamoDB. Key concepts I learnt include Lambda functions as serverless compute, DynamoDB as a NoSQL database, and the use of execution roles with attached IAM policies to control access. I learnt how to test Lambda with event data and how to tighten security by replacing a broad managed policy with a custom inline policy scoped to just the UserData table.

Project reflection

This project took me approximately 60 minutes. The most challenging part was troubleshooting the AccessDenied error, which taught me how execution roles and their attached policies work together. It was most rewarding to see my Lambda function successfully read data from DynamoDB after applying the correct permissions.



Louis Moyo
NextWork Student

nextwork.org

I did this project today to deepen my understanding of how Lambda functions interact with DynamoDB and how execution roles with the right policies enforce security. Yes, this project met my goals by giving me practical experience in applying least-privilege permissions with a custom inline policy while keeping the Lambda function working.



Project Setup

To set up my project, I created a database using Amazon DynamoDB with a table called UserData. The partition key is userId, which means each record in the table is uniquely identified by a userId, allowing DynamoDB to efficiently organise and retrieve data

In my DynamoDB table, I added a new item with a userId of 1, a name of Test User, and an email of test@example.com. DynamoDB is schemaless, which means I didn't need to predefine the attributes like name or email - I can add different attributes for different items as needed.

The screenshot shows the AWS DynamoDB console interface. A modal window is open for creating a new item in a table named 'UserData'. The 'Attributes' section contains the following JSON code:

```
1 ▾ {  
2   "userId": "1",  
3   "name": "Test User",  
4   "email": "test@example.com"  
5 }  
0
```

Below the code, status information is displayed: 'JSON' (highlighted), 'Ln 6, Col 1', 'Errors: 0', and 'Warnings: 0'. At the bottom right of the modal are 'Cancel' and 'Create Item' buttons, with 'Create Item' being highlighted.

AWS Lambda

AWS Lambda is a serverless compute service that runs code without the need to manage servers. It automatically scales and only runs when triggered, so you only pay for the execution time. I'm using Lambda in this project to fetch user data from my DynamoDB table and return it on demand.



AWS Lambda Function

My Lambda function has an execution role, which is an IAM role that defines what the function is allowed to do in AWS. By default, the role grants basic permissions such as writing logs to CloudWatch, and I can add more permissions (like DynamoDB access) so the function can securely perform its tasks without having unnecessary access to other services.

My Lambda function will take a userId from the event, query the DynamoDB UserData table for that user, and return the user's details if they exist. If no data is found or an error occurs, the function will return either null or an error message, making it easier to troubleshoot issues

The code uses AWS SDK, which is a collection of libraries and tools that make it easier for developers to interact with AWS services. My code uses SDK to connect to DynamoDB, send a query for a userId, and retrieve the matching user data without needing to write low-level API calls.



```
JS index.mjs x ⌂ RetrieveUserData
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;
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 }
31
```

Ln 4, Col 58 Spaces: 2 UTF-8 CRLF ↴ JavaScript Lambda Layout: US

Function Testing

To test whether my Lambda function works, I created a test event in the Test tab of the Lambda console and passed in a userId value. The test is written in JSON, which Lambda can easily process. If the test is successful, I'd see the user data for userId 1 returned from the DynamoDB table.

The test displayed a 'success' because the Lambda function's code executed without crashing, but the function's response was actually an access denied error because the execution role didn't have explicit permission to read from the DynamoDB table.

The screenshot shows the AWS Lambda Test Execution interface. The top bar indicates "Executing function: succeeded (logs)" and has a "Details" dropdown. Below this, the function's code is shown:

```
{
  "name": "AccessDeniedException",
  "default": "Client",
  "metadata": {
    "httpStatusCode": 400,
    "awsLambdaEventId": "1016033C681HE3IN5160VM33VYANQNS0AENVJF6609ASUAAJG",
    "attempts": 1,
    "totalRetryDelay": 0
  },
  "_type": "com.amazon.coral.service#AccessDeniedException"
}
```

The "Summary" section provides execution details:

Code SHA-256 q8E7NexfsxuhKT9/d4bGpAVOXJYFAUJjOUDjB0ivK8E=	Execution time 2 minutes ago
Function version \$LATEST	Request ID 437579a8-j922-471f-83e8-688b8204e6b3
Duration 998.93 ms	Billed duration 999 ms
Resources configured 128 MB	Max memory used 99 MB
Init duration 449.12 ms	
Log output	

The log output section contains the following text:

The area below shows the last 4 KB of the execution log. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: 437579a8-e922-471f-83e8-688b8204e6b3 Version: $LATEST
2023-08-22T04:09:13.184Z 437579a8-e922-471f-83e8-688b8204e6b3 ERROR Unable to retrieve data. AccessDeniedException: User: arn:aws:sts::756716632322:assumed-role/RetrieveUserData-role-vpnaud4/RetrievesUserRole is not authorized to perform: dynamodb:GetItem on resource: arn:aws:dynamodb:us-east-1:756716632322:table/UserData because no identity-based policy allows the dynamodb:GetItem action
at threadDefaultError (/var/runtime/node_modules/@aws-sdk/client-dynamodb/lib/index.js:388:29)
```

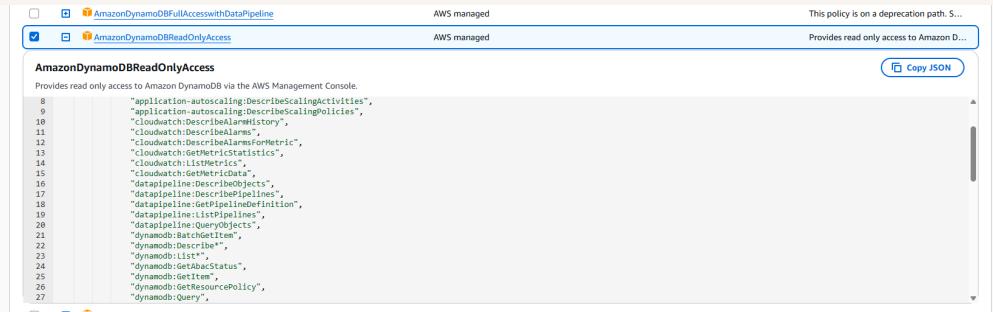


Function Permissions

To resolve the AccessDenied error, I attached the AmazonDynamoDBReadOnlyAccess policy to my Lambda function's execution role because the error showed that the denied action was GetItem, and this policy provides the necessary permission for reading items from DynamoDB.

There were four DynamoDB permission policies I could choose from, but I didn't pick AWSLambdaDynamoDBExecutionRole or AWSLambdaInvocation-DynamoDB because those are for working with DynamoDB streams and triggering Lambda functions from table changes, not for reading items directly. My function only needs to use GetItem, so those policies wouldn't solve the access denied error.

I also didn't pick the full access or stream-related policies because they grant more permissions than my function needs. AmazonDynamoDBReadOnlyAccess was the right choice because my Lambda function only needs to read data with actions like GetItem, and this policy follows the principle of least privilege.



The screenshot shows the AWS IAM Policy Editor interface. At the top, there are two AWS managed policies listed: "AmazonDynamoDBFullAccessWithDataPipeline" and "AmazonDynamoDBReadOnlyAccess". A tooltip indicates that the first policy is on a deprecation path. Below these, the "AmazonDynamoDBReadOnlyAccess" policy is selected and displayed in a code editor-like area. The policy document is as follows:

```
Provides read only access to Amazon DynamoDB via the AWS Management Console.

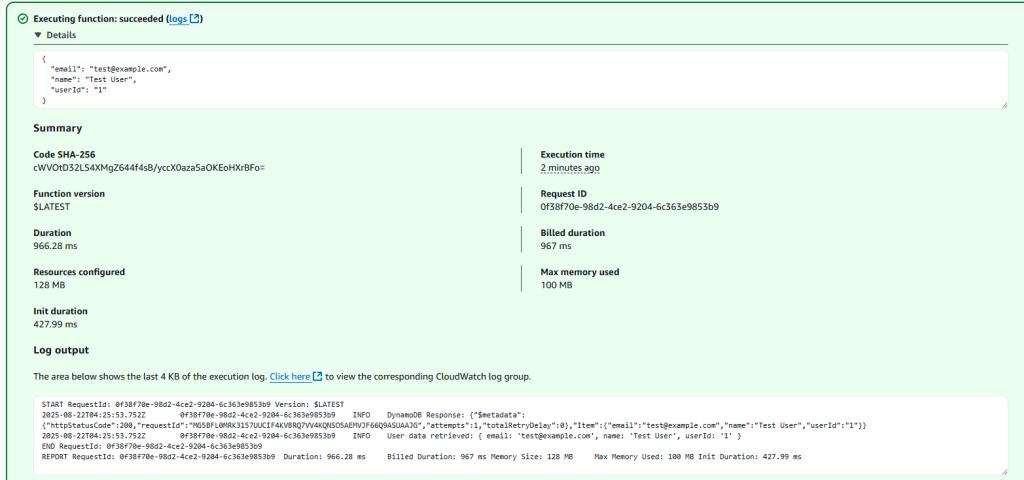
8   "application-autoscaling:DescribeScalingActivities",
9   "application-autoscaling:DescribeScalingPolicies",
10  "cloudwatch:DescribeMetricHistory",
11  "cloudwatch:DescribeAlarms",
12  "cloudwatch:DescribeAlarmsForMetric",
13  "cloudwatch:GetMetricStatistics",
14  "cloudwatch:ListMetrics",
15  "cloudwatch:GetMetricData",
16  "datapipeline:DescribeObjects",
17  "datapipeline:DescribePipeline",
18  "datapipeline:DescribePipelineDefinition",
19  "datapipeline:ListPipelines",
20  "datapipeline:QueryObjects",
21  "dynamodb:BatchGetItem",
22  "dynamodb:DeleteTable",
23  "dynamodb:List",
24  "dynamodb:GetAbacStatus",
25  "dynamodb:GetItem",
26  "dynamodb:GetResourcePolicy",
27  "dynamodb:Query",
```

A "Copy JSON" button is located in the top right corner of the policy editor.

Final Testing and Reflection

To validate my new permission settings, I re-ran the Lambda test with a userId of 1. The results were successful because the function now had the required GetItem permission to read from DynamoDB, so it was able to return the user's data instead of an access denied error.

Web apps are a popular use case of using Lambda and DynamoDB. For example, I could build a serverless backend where users log in to view their profile data, an e-commerce app where product details are fetched on demand, or a mobile app that scales automatically without needing to manage servers.



The screenshot shows the AWS CloudWatch Lambda execution history for a function named "testUser". The execution was successful, indicated by a green checkmark icon. The "Details" section shows the input payload, which contains a single object with fields: "email": "test@example.com", "name": "Test User", and "userId": "1". The "Summary" section provides performance metrics: Duration (966.28 ms), Function version (\$LATEST), and Resources configured (128 MB). The "Log output" section displays the CloudWatch logs for the execution, which show the Lambda function successfully retrieving data from DynamoDB based on the provided userId.

```
Executing function: succeeded (logs [2])
▼ Details
{
  "email": "test@example.com",
  "name": "Test User",
  "userId": "1"
}

Summary
Code SHA-256
cNVV0tD52LS4XMGz644f4sB/yccXoaza5aOKEoHxrbFo=
Function version
$LATEST
Duration
966.28 ms
Resources configured
128 MB
Init duration
427.99 ms
Log output
The area below shows the last 4 KB of the execution log. Click here to view the corresponding CloudWatch log group.

START RequestId: 0f38f70e-98d2-4ce2-9294-6c363e9853b9 Version: $LATEST
2025-08-22T04:25:53.752           0f38f70e-98d2-4ce2-9294-6c363e9853b9  INFO   DynamoDB Response: {"$metadata":
{"RequestId": "0f38f70e-98d2-4ce2-9294-6c363e9853b9", "HTTPMethod": "GET", "HTTPPath": "/2025-08-22T04:25:53.752Z/0f38f70e-98d2-4ce2-9294-6c363e9853b9?Action=GetItem&Key=%7B%22userId%22%3A%221%22%7D", "IsSuccessful": true}, "Item": {"email": "test@example.com", "name": "Test User", "userId": "1"}}
2025-08-22T04:25:53.752           0f38f70e-98d2-4ce2-9294-6c363e9853b9  INFO   User data retrieved: { email: 'test@example.com', name: 'Test User', userId: '1' }
END RequestId: 0f38f70e-98d2-4ce2-9294-6c363e9853b9
REPORT RequestId: 0f38f70e-98d2-4ce2-9294-6c363e9853b9 Duration: 966.28 ms Billed Duration: 967 ms Memory Size: 128 MB Max Memory Used: 100 MB Init Duration: 427.99 ms
```



Enhancing Security

For my project extension, I challenged myself to replace the managed policy with a custom inline policy that only allows access to the specific UserData table. This will tighten security by removing unnecessary permissions and ensuring the Lambda function follows the principle of least privilege.

To create the permission policy, I used an inline policy because it let me restrict my Lambda function to only access the UserData table, instead of granting read access to all DynamoDB tables with a managed policy.

When updating a Lambda function's permission policies, you could risk breaking its access to required resources. I validated that my Lambda function still works by re-running the test event and confirming that it successfully retrieved the user data from the DynamoDB UserData table with the new inline policy applied.



Policy editor

```
1▼ {
2    "Version": "2012-10-17",
3▼   "Statement": [
4▼     {
5       "Sid": "DynamoDBReadAccess",
6       "Effect": "Allow",
7▼         "Action": [
8           "dynamodb:GetItem"
9         ],
10      "Resource": [
11        "arn:aws:dynamodb:us-east-1:756716632322:table/UserData"
12      ]
13    }
14  ]
15 }
16 |
```



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects

