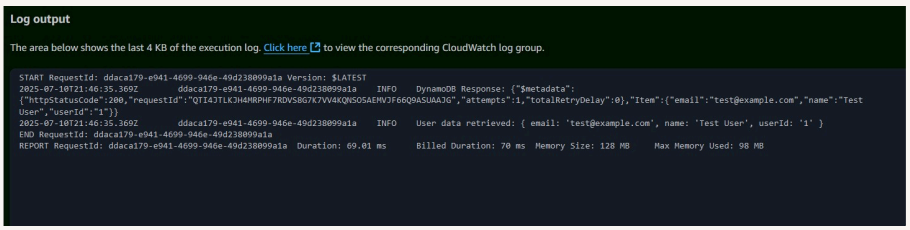
nextwork.org

Fetch Data with AWS Lambda

Chrispinus Jacob



Chrispinus Jacob
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to use AWS Lambda to retrieve data stored in DynamoDB. I'm doing this project to learn how to set up the data tier of an application using DynamoDB, and how to build the logic tier using Lambda functions. This will help me understand how backend components interact in a serverless architecture and how data flows from storage to the user through a Lambda-powered API.

Tools and concepts

Services I used were AWS Lambda, A , and Amazon DynamoDB. Key concepts I learnt include Lambda functions as the logic layer for handling backend tasks, API Gateway for connecting user requests to the function, and DynamoDB as a flexible NoSQL database for storing and retrieving data. I also learnt how to manage IAM roles and permissions to keep my application secure.

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was setting up the correct IAM permissions to allow the Lambda function to access DynamoDB without triggering errors. It was most rewarding to see everything come together—watching the Lambda function successfully retrieve data and return it through the API, showing that the full serverless flow was working correctly.

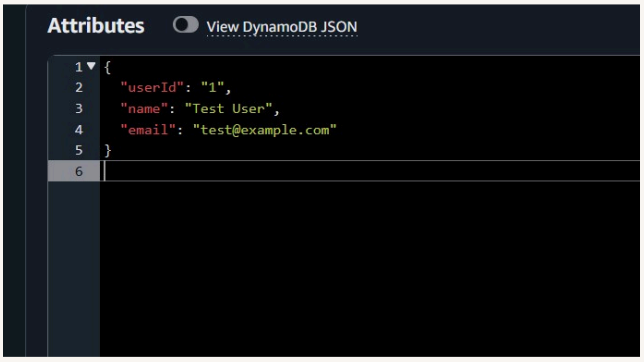
Chrispinus Jacob
NextWork Student

I did this project today to strengthen my understanding of how serverless components like AWS Lambda and DynamoDB work together to build modern web applications. Yes, this project met my goals—it helped me learn how to structure the logic and data tiers, manage permissions securely, and test end-to-end functionality. I now feel more confident in building and deploying basic serverless APIs.

Project Setup

To set up my project, I created a database using DynamoDB. DynamoDB is a NoSQL database that is highly flexible for both data storage and retrieval, making it ideal for serverless applications. The partition key is `userId`, which means it's the key identifier used to uniquely locate each item in the table. By using `userId` as the partition key, I can efficiently fetch specific user data based on the ID submitted in an API request.

In my DynamoDB table, I added a piece of user data to help with testing. DynamoDB is schemaless, which means it doesn't require a fixed structure for all items in the table. Each item can have different fields, and I only need to define the key (like `userId`) when creating the table. This makes it easy to store flexible and varied data.



AWS Lambda

AWS Lambda is a serverless compute service that lets you run code without managing servers. I'm using Lambda in this project to write the logic that retrieves data from DynamoDB.

When a user sends a request, the Lambda function will run automatically, look up the correct item in the database using the user ID, and return that data. This makes Lambda the main part of the logic tier in my app.

Chrispinus Jacob
NextWork Student

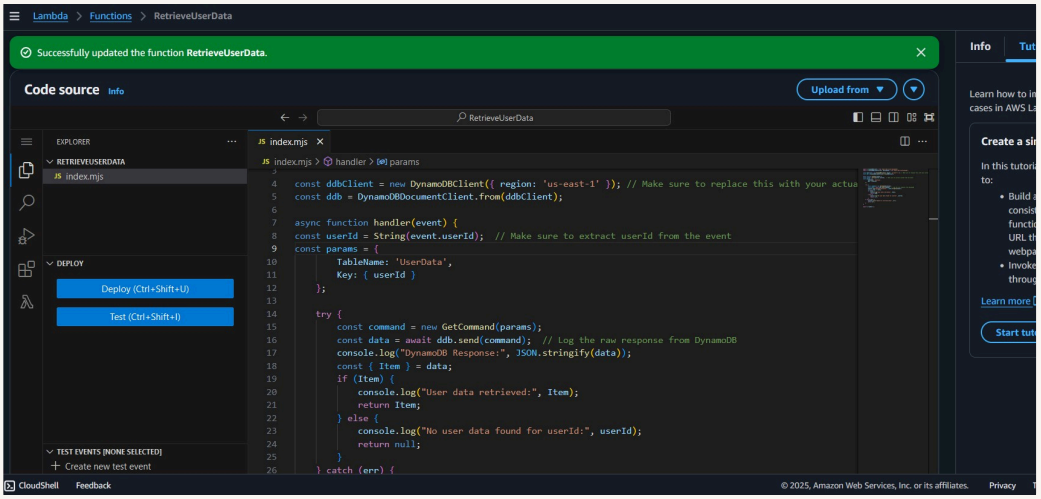
AWS Lambda Function

My Lambda function has an execution role, which is a set of permissions that tells AWS what the function is allowed to do. By default, the role grants basic permissions like writing logs to Amazon CloudWatch. However, to allow my function to read from DynamoDB, I need to update the role and add specific permissions for accessing the DynamoDB table. This ensures the function can securely retrieve the data it needs during execution.

My Lambda function will retrieve data from a DynamoDB table. The first part of the code focuses on getting the data by using the user ID to query the table and return the matching item. The second part of the code handles sending a response back to the user—either returning the found data or an error message if something went wrong.

The code uses AWS SDK, which is a collection of tools that allow developers to interact with AWS services through code. My code uses the SDK to connect to DynamoDB, send a request to get data using the user ID, and receive the result. This makes it easy for the Lambda function to communicate with DynamoDB without needing to write low-level HTTP requests.

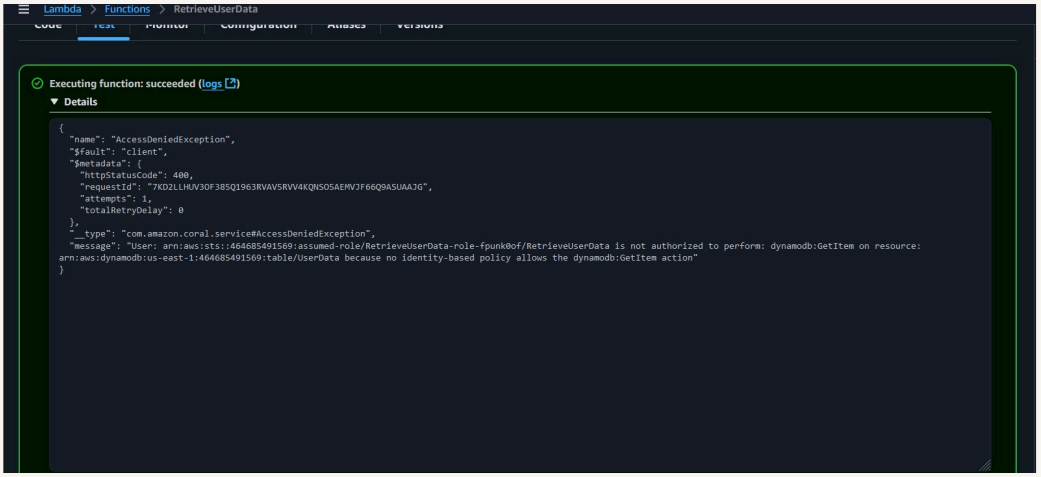
Chrispinus Jacob
NextWork Student



Function Testing

To test whether my Lambda function works, I ran a test using the Test tab in the Lambda console. The test is written in JSON and includes a sample event, such as a user ID that the function should use to look up data in DynamoDB. If the test is successful, I'd see the correct piece of user data returned in JSON format. However, if the function doesn’t have permission to access the DynamoDB table, the test will return an error message indicating a permissions issue.

The test displayed a 'success' because the Lambda function ran without crashing and returned a response. But the function's response was actually an error because the Lambda execution role didn’t have the correct permissions to access the DynamoDB table. This means the function was able to execute, but it couldn't retrieve the data, resulting in an error message in the output instead of the expected user information.

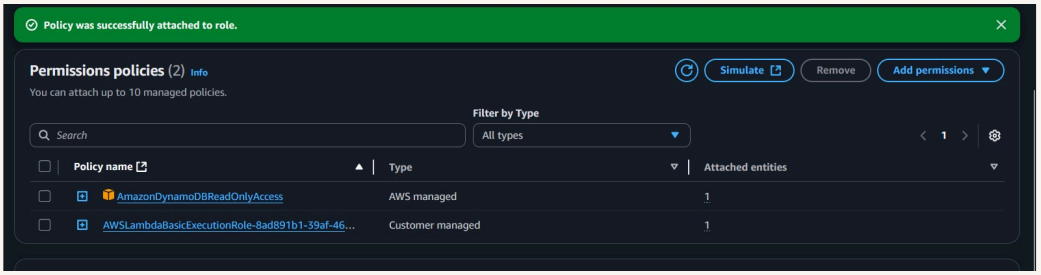

Lambda > Functions > RetrieveUserData
Executing function: succeeded (logs)
Details
{
"name": "AccessDeniedException",
"$fault": "client",
"$metadata": {
"httpStatusCode": 400,
"requestId": "7KD2LLHUV3OF3BSQ1963RVAV5RVV4KQNSO5AEMVJF66Q9ASUAAJG",
"attempts": 1,
"totalRetryDelay": 0
},
"__type": "com.amazon.coral.service#AccessDeniedException",
"message": "User: arn:aws:sts::464685491569:assumed-role/RetrieveUserData-role-fpunk0of/RetrieveUserData is not authorized to perform: dynamodb:GetItem on resource:
arn:aws:dynamodb:us-east-1:464685491569:table/UserData because no identity-based policy allows the dynamodb:GetItem action"
}

Function Permissions

To resolve the AccessDenied error, I updated the Lambda function's execution role to include read-only permissions for DynamoDB, because without these permissions, the function cannot access the table to retrieve data.

There were four DynamoDB permission policies I could choose from, but I didn't pick AmazonDynamoDBFullAccess because it gives too many permissions, including write and delete access, which I don’t need for this project. I only needed read-only access to safely retrieve data without risking accidental changes to the database.

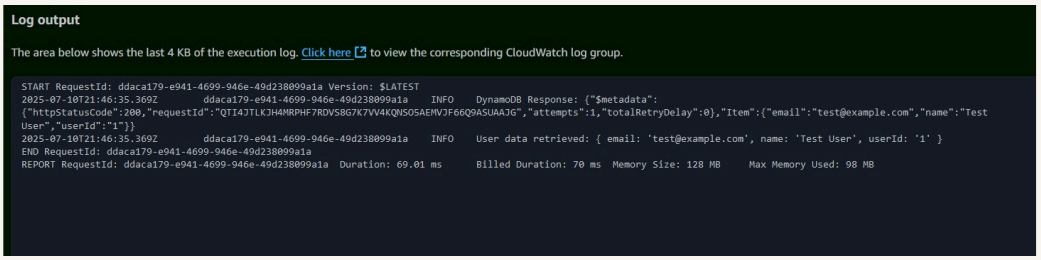
I also didn't pick AmazonDynamoDBFullAccess or AmazonDynamoDBFullAccessWithDataPipeline because they allow full control over all DynamoDB resources, which is more than what my Lambda function needs. AmazonDynamoDBReadOnlyAccess was the right choice because it gives just enough permission for the function to read data from the table—keeping the setup secure and limited to only what's necessary.


Policy was successfully attached to role.
Permissions policies (2) Info
You can attach up to 10 managed policies.
Simulate
Remove
Add permissions
Filter by Type
Search
All types
1
Policy name
Type
Attached entities
AmazonDynamoDBReadOnlyAccess
AWS managed
1
AWSLambdaBasicExecutionRole-8ad891b1-39af-46...
Customer managed
1

Final Testing and Reflection

To validate my new permission settings, I retested the Lambda function using the same test event as before. The results were successful because the updated execution role now has the correct read-only access to DynamoDB, allowing the function to retrieve and return the expected data.

Web apps are a popular use case of using Lambda and DynamoDB. For example, I could build a user dashboard where Lambda functions handle login requests, retrieve user data from DynamoDB, and display it on the page. This setup lets the app run without managing servers, while still delivering fast, dynamic content to users.


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: ddaca179-e941-4699-946e-49d238099a1a Version: $LATEST
2025-07-10T21:46:35.369Z ddaca179-e941-4699-946e-49d238099a1a INFO DynamoDB Response: {"$metadata":
{"httpStatusCode":200,"requestId":"QTI4JTLKJH4MRPHF7RDVS8G7K7VV4KQNSO5AEMVJF66Q9ASUAAJG","attempts":1,"totalRetryDelay":0},"Item":{"email":"test@example.com","name":"Test
User","userId":"1"}}
2025-07-10T21:46:35.369Z ddaca179-e941-4699-946e-49d238099a1a INFO User data retrieved: { email: 'test@example.com', name: 'Test User', userId: '1' }
END RequestId: ddaca179-e941-4699-946e-49d238099a1a
REPORT RequestId: ddaca179-e941-4699-946e-49d238099a1a Duration: 69.01 ms Billed Duration: 70 ms Memory Size: 128 MB Max Memory Used: 98 MB

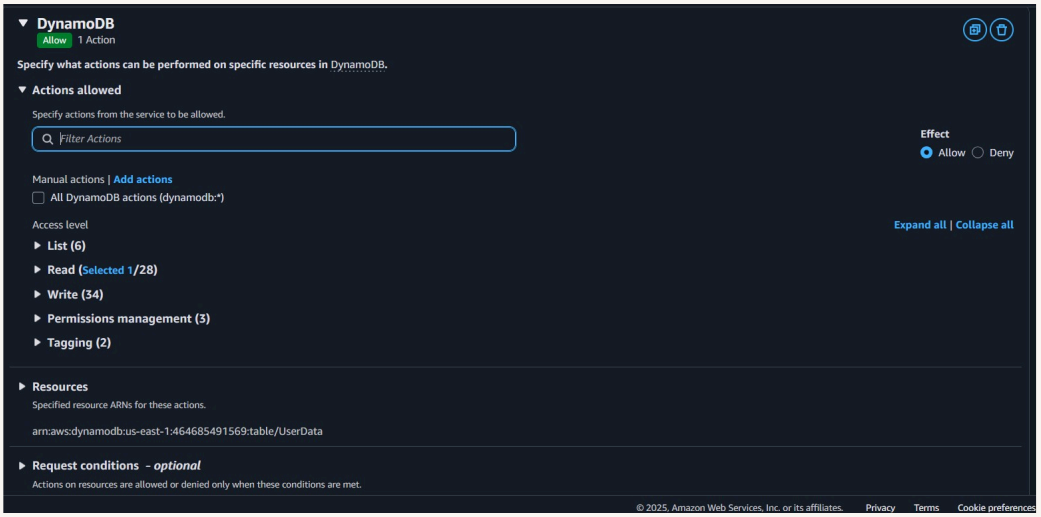
Chrispinus Jacob
NextWork Student

Enahancing Security

For my project extension, I challenged myself to replace the permission policy we granted our Lambda function recently. Instead of using the AWS user-managed policy, I created a custom policy that ensures the function can only access specific user data in DynamoDB. This will enhance DynamoDB security by following the principle of least privilege—giving the function only the exact access it needs and nothing more.

To create the permission policy, I used the visual editor in IAM because it makes it easier to select specific actions and resources without writing JSON manually. This way, I could clearly choose only the read permissions needed for my DynamoDB table, helping to keep the setup secure and easy to manage.

When updating a Lambda function's permission policies, you could risk removing access the function needs to run properly—like losing permission to read from DynamoDB. I validated that my Lambda function still works by running a test in the Lambda console and checking that it returned the correct user data without any access errors.


DynamoDB
Allow 1 Action
Specify what actions can be performed on specific resources in DynamoDB.
Actions allowed
Specify actions from the service to be allowed.
Filter Actions
Effect
Allow
Deny
Manual actions | Add actions
All DynamoDB actions (dynamodb:*)
Access level
Expand all | Collapse all
List (6)
Read (Selected 1/28)
Write (34)
Permissions management (3)
Tagging (2)
Resources
Specified resource ARNs for these actions.
arn:aws:dynamodb:us-east-1:464685491569:table/UserData
Request conditions - optional
Actions on resources are allowed or denied only when these conditions are met.
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

The place to learn & showcase your skills

Check out nextwork.org for more projects