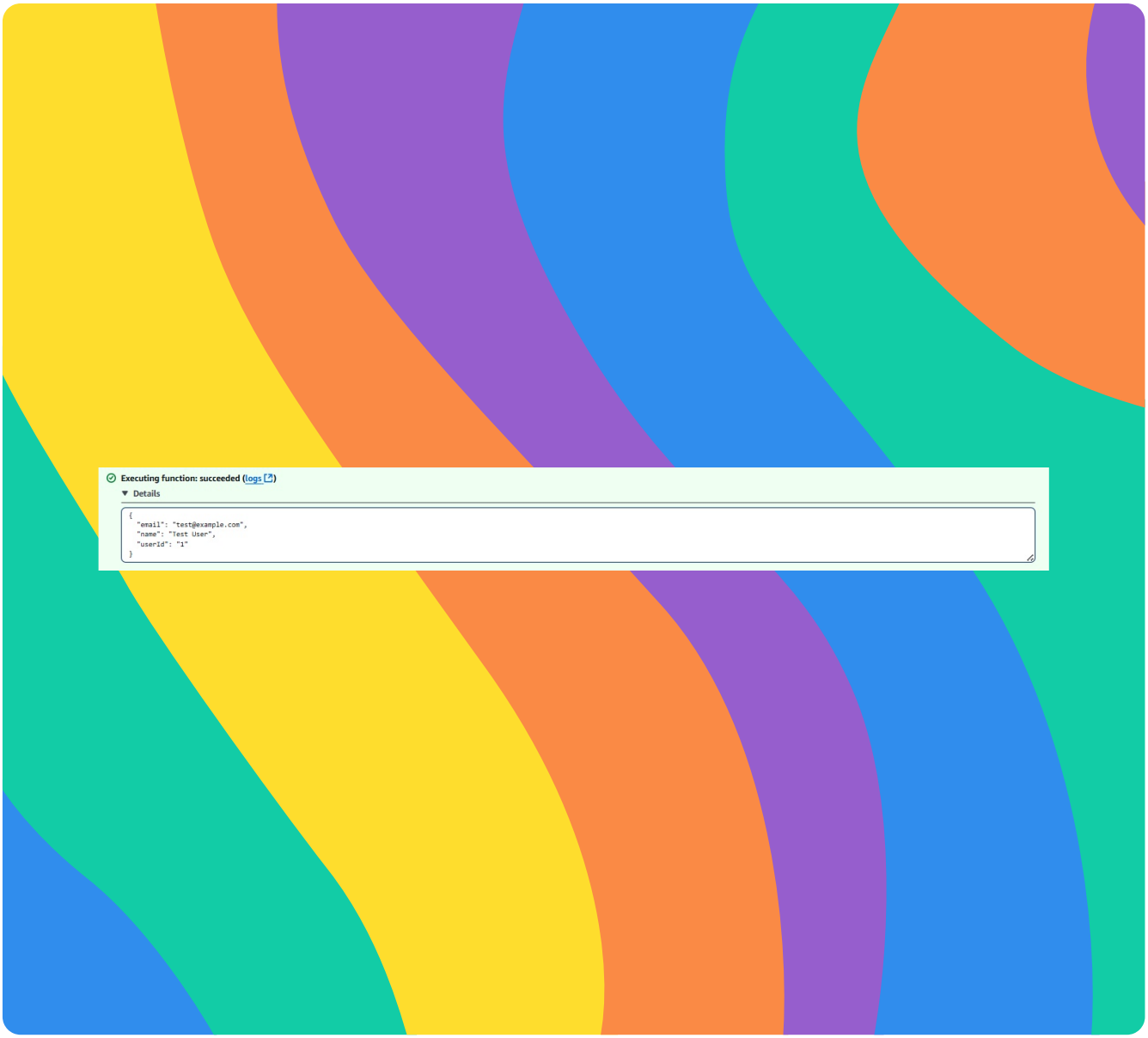
nextwork.org

Fetch Data with AWS Lambda

DI Dineshraj Dhanapathy



DI
Dineshraj Dhanapathy
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to create a DynamoDB table to store user data, build a serverless Lambda function to retrieve that data, and write tests to validate the function's behavior. I'm doing this project to learn how to securely access and manage data using AWS services like Lambda and DynamoDB. I'll also apply IAM policies to control permissions, ensuring my function and database are both secure and follow best practices for serverless architecture.

Tools and concepts

Services I used were AWS Lambda and Amazon DynamoDB. Key concepts I learnt include Lambda functions, which allow you to run backend logic without managing servers, and DynamoDB, a fast and flexible NoSQL database service. I also explored IAM roles and policies to securely connect Lambda to DynamoDB, and understood how to fetch data using the AWS SDK. This project helped me grasp serverless architecture, permission management, and real-world use cases like building data-driven web apps.

DI
Dineshraj Dhanapathy
NextWork Student

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was resolving the AccessDenied error by configuring the correct IAM permissions for the Lambda function. It was most rewarding to successfully retrieve data from DynamoDB using a Lambda function and validate everything worked through testing. Seeing the function run smoothly after setting up custom permissions gave me a strong understanding of how AWS services work together securely.

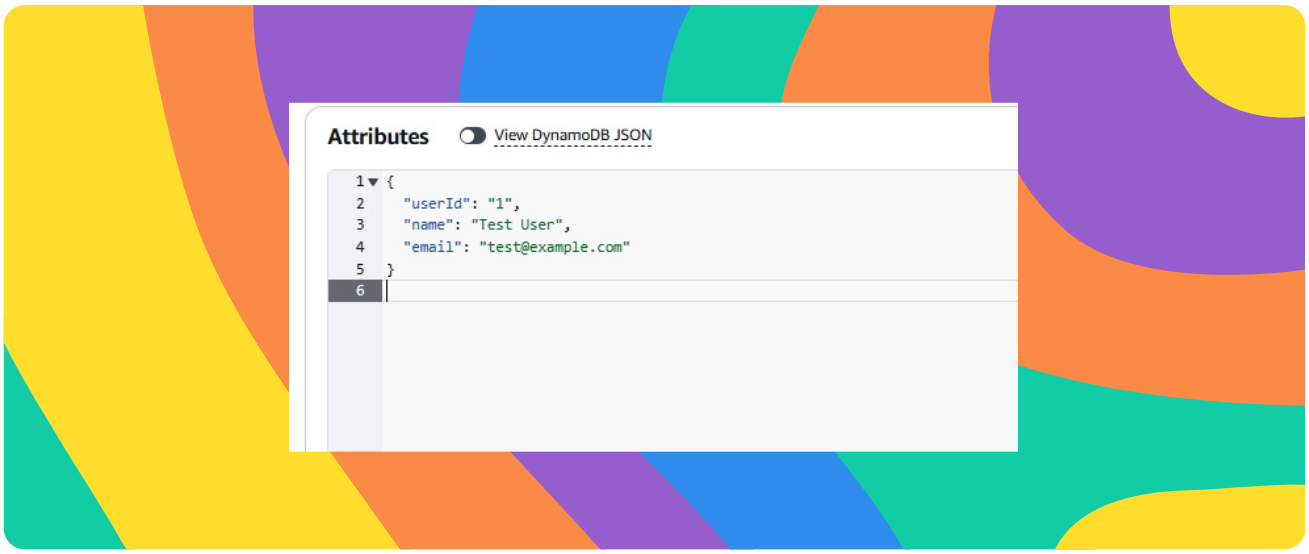
I did this project today to strengthen my understanding of how AWS Lambda and DynamoDB work together in a serverless architecture. I wanted hands-on experience with real-world use cases like data retrieval and permissions management. This project met my goals because I successfully connected a Lambda function to a DynamoDB table, handled permissions securely, and tested everything end-to-end, which deepened my confidence in building serverless apps.

DI
Dineshraj Dhanapathy
NextWork Student

Project Setup

To set up my project, I created a database using DynamoDB. The partition key is userId, which means each user's data in the table is uniquely identified by their userId. This helps DynamoDB organize and retrieve data efficiently by distributing it across partitions. With this setup, I can quickly access specific user data by querying the userId, ensuring fast and scalable performance for my serverless application.

In my DynamoDB table, I added a sample user item with attributes like userId, name, and email. DynamoDB is schemaless, which means I didn't need to predefine all possible attributes—each item can have a different structure. This flexibility allows me to store only the relevant data for each user and easily expand or modify the schema later without redesigning the entire table, making it perfect for dynamic and evolving applications.



DI
Dineshraj Dhanapathy
NextWork Student

AWS Lambda

AWS Lambda is a serverless computing service that lets you run code without provisioning or managing servers. It automatically handles the infrastructure, so you only need to focus on writing your code. Lambda runs your code in response to events, and you only pay for the compute time you use—no charges for idle time. It scales automatically to handle any number of requests. I'm using Lambda in this project to retrieve user data from DynamoDB, securely and efficiently, without needing to manage backend infrastructure.

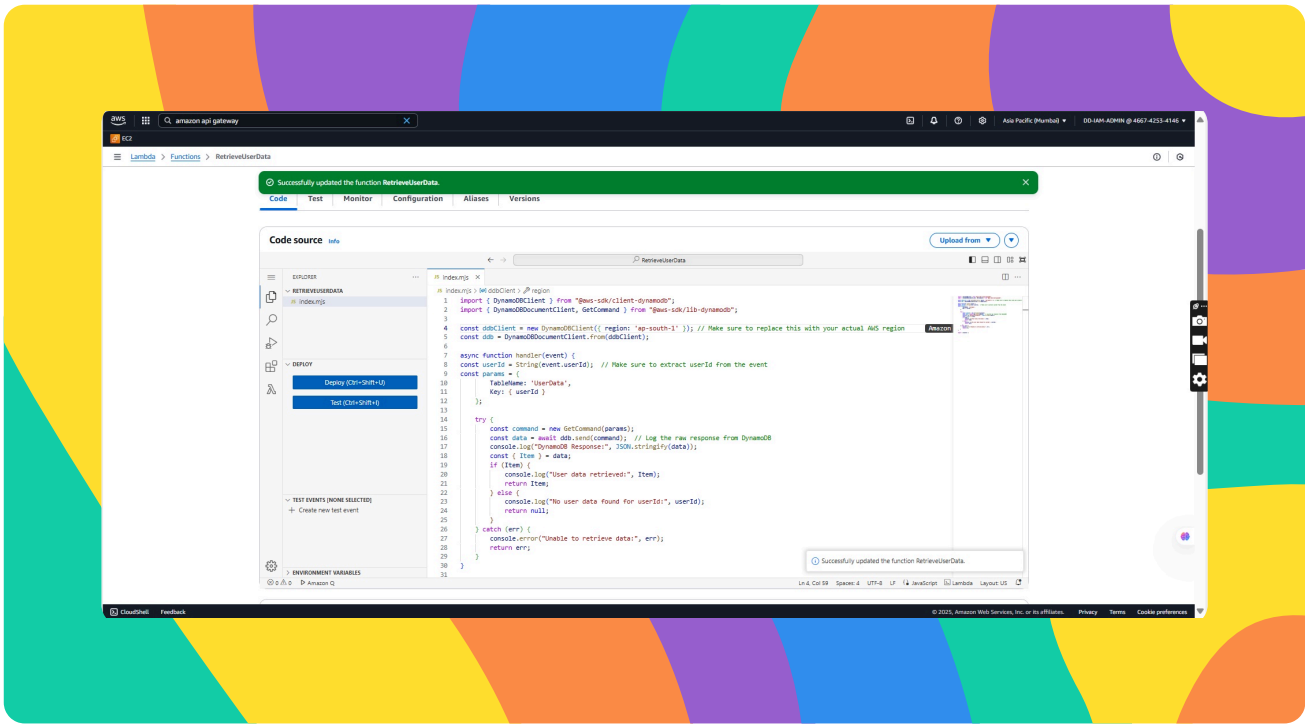
DI
Dineshraj Dhanapathy
NextWork Student

AWS Lambda Function

My Lambda function has an execution role, which is an IAM role that defines what actions my function is allowed to perform on other AWS services. By default, the role grants basic permissions such as writing logs to Amazon CloudWatch. This helps with monitoring and debugging. However, to allow my function to interact with DynamoDB and retrieve user data, I need to attach additional permissions. This execution role ensures the function only accesses what it needs, following the principle of least privilege for better security.

My Lambda function will use the AWS SDK for JavaScript to retrieve data from my DynamoDB table. It takes a userId as input, queries the UserData table for that ID, and returns the corresponding user data. The try block handles the database call and returns the result, while the catch block ensures any errors are caught and a helpful error message is returned. This makes the function both functional and user-friendly in handling DynamoDB interactions.

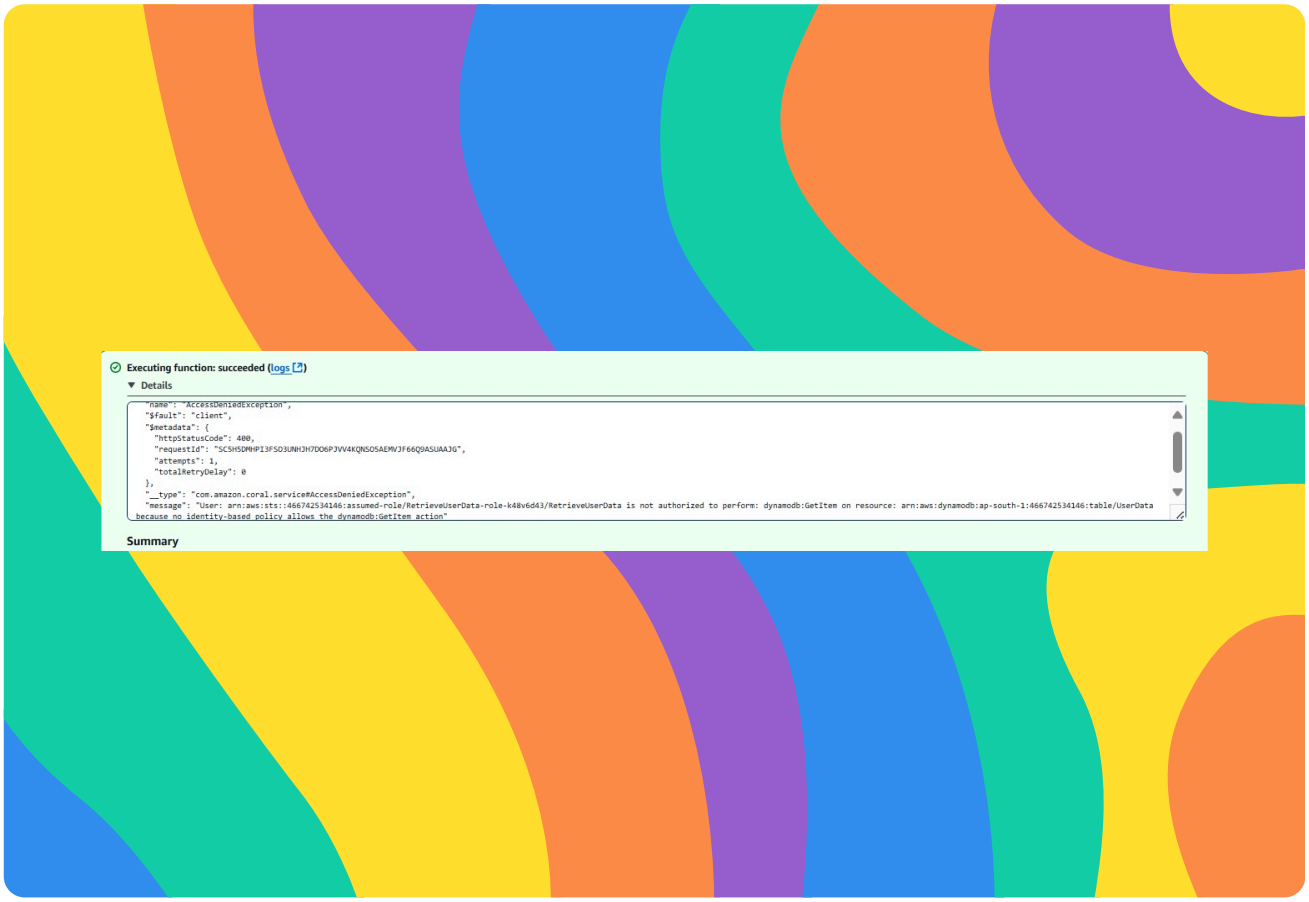
The code uses AWS SDK, which is the Amazon Web Services Software Development Kit—a collection of tools and libraries that makes it easier to interact with AWS services using familiar programming languages. My code uses SDK to connect to DynamoDB, send a query using a userId, and retrieve the matching user data from the UserData table. The SDK handles the low-level communication with AWS, so I don't need to write complex API calls manually. It simplifies my code and makes it more reliable and secure.



Function Testing

To test whether my Lambda function works, I used the Test tab in the Lambda console and created a test event in JSON format with a userId of 1. The test is written in JSON because it's a lightweight, readable format that Lambda understands easily. This input simulates a real request, allowing me to see if the function can successfully retrieve the correct item from the DynamoDB table. If the test is successful, I'd see the user data returned in the response; if not, I'd see an error message indicating what went wrong.

The test displayed a 'success' because the Lambda function executed without any internal code errors—but the function's response was actually an "AccessDeniedException" because it didn't have the required permissions to read from the DynamoDB table. Even though the code was fine, AWS blocked access since the execution role hadn't been explicitly granted DynamoDB read access. So, the function ran but couldn't do what it was supposed to: retrieve user data from the table.


Executing function: succeeded (logs)
Details
"name": "AccessDeniedException",
"$fault": "client",
"$metadata": {
"httpStatusCode": 400,
"requestId": "SC5H5DMHPI3FSO3UNHJH7DO6PJVV4KQNSO5AEMVJF66Q9ASUAAJG",
"attempts": 1,
"totalRetryDelay": 0
},
"__type": "com.amazon.coral.service#AccessDeniedException",
"message": "User: arn:aws:sts::466742534146:assumed-role/RetrieveUserData-role-k48v6d43/RetrieveUserData is not authorized to perform: dynamodb:GetItem on resource: arn:aws:dynamodb:ap-south-1:466742534146:table/UserData because no identity-based policy allows the dynamodb:GetItem action"
Summary

DI
Dineshraj Dhanapathy
NextWork Student

Function Permissions

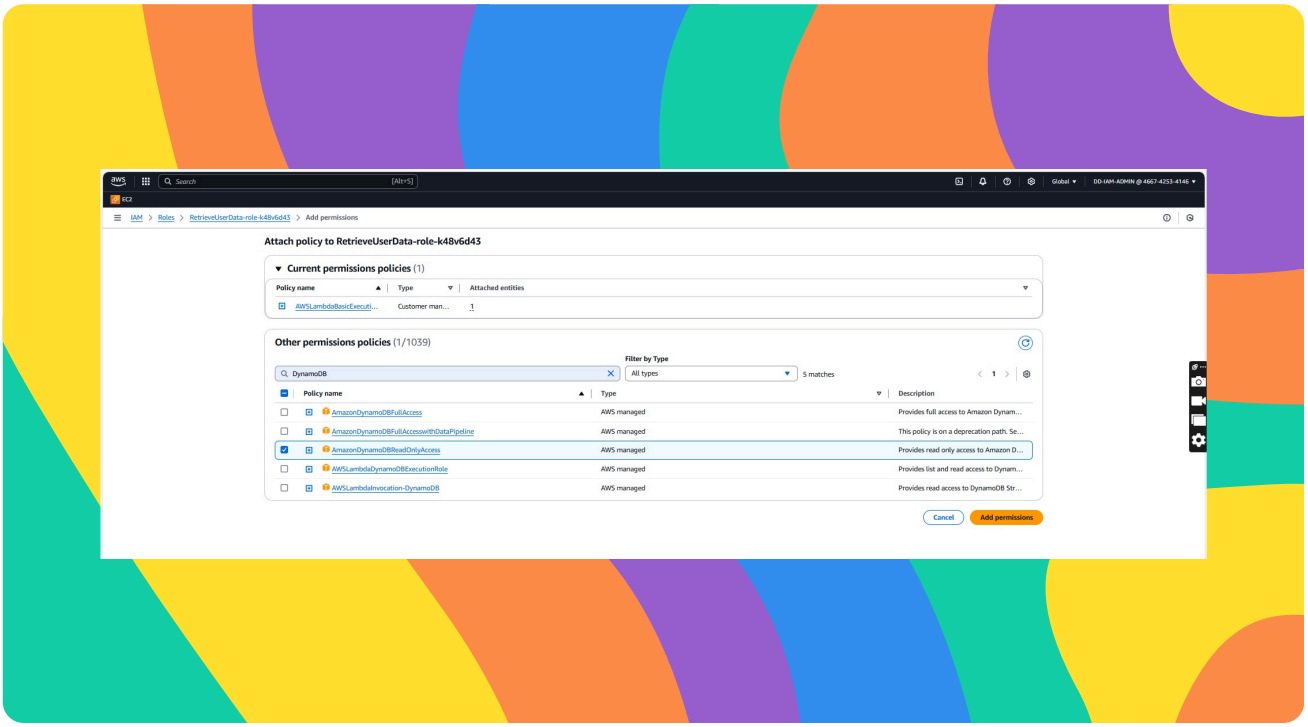
To resolve the AccessDenied error, I will attach the AmazonDynamoDBReadOnlyAccess policy to my Lambda function's execution role, because the error message clearly showed that the function was trying to perform a GetItem operation without having permission. By reviewing the error, I can identify the exact permission needed and apply the correct fix without guessing, ensuring my function can access DynamoDB securely and successfully.

There were four DynamoDB permission policies I could choose from, but I didn't pick AWSLambdaDynamoDBExecutionRole or AWSLambdaInvocation-DynamoDB because those are designed for use with DynamoDB Streams, not for reading data directly from a table. My function just needs to retrieve data using GetItem, not react to real-time table updates. That's why I chose a policy that gives read-only access, which is better suited for this task.

I also didn't pick the other policies like AWSLambdaDynamoDBExecutionRole or AWSLambdaInvocation-DynamoDB because they're meant for handling DynamoDB Streams, not for direct read operations. AmazonDynamoDBReadOnlyAccess was the right choice because it gives my Lambda function read permissions like GetItem, which is exactly what I need to fetch user data from the table without giving unnecessary write or admin privileges.

DI

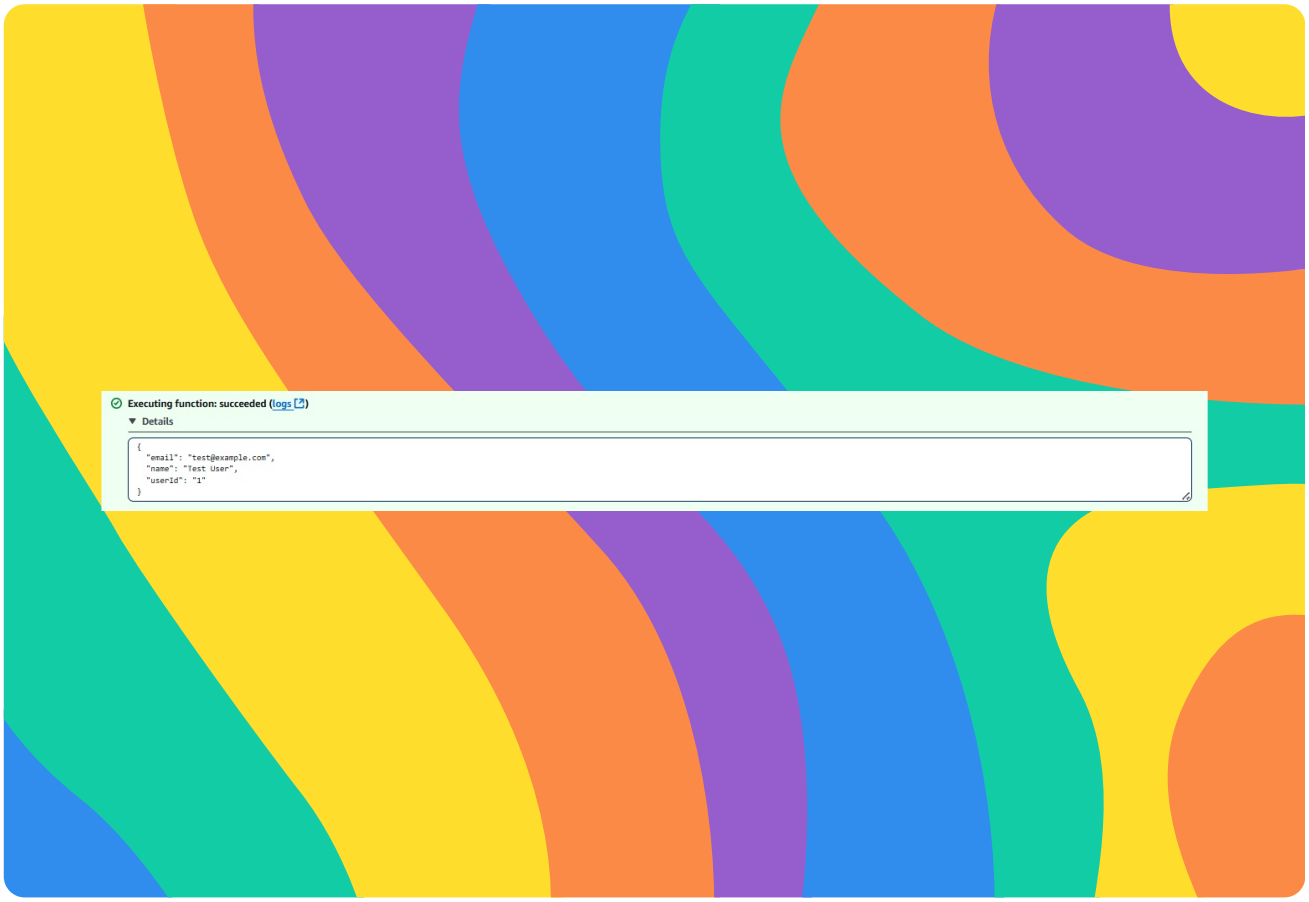
Dineshraj Dhanapathy
NextWork Student



Final Testing and Reflection

To validate my new permission settings, I re-ran the Lambda function test. The results were successful because the function now has the correct permissions to access and read from the DynamoDB table. With the AmazonDynamoDBReadOnlyAccess policy in place, Lambda could fetch the item based on the userId I provided. This confirmed that the access denied issue was resolved and my serverless backend is working as intended.

Web apps are a popular use case of using Lambda and DynamoDB. For example, I could build a user login system that pulls profile data from DynamoDB when someone signs in. I could also create a product catalog app where Lambda fetches item details stored in DynamoDB. This setup is serverless, scalable, and cost-efficient—perfect for dynamic apps that need fast access to data without managing servers.

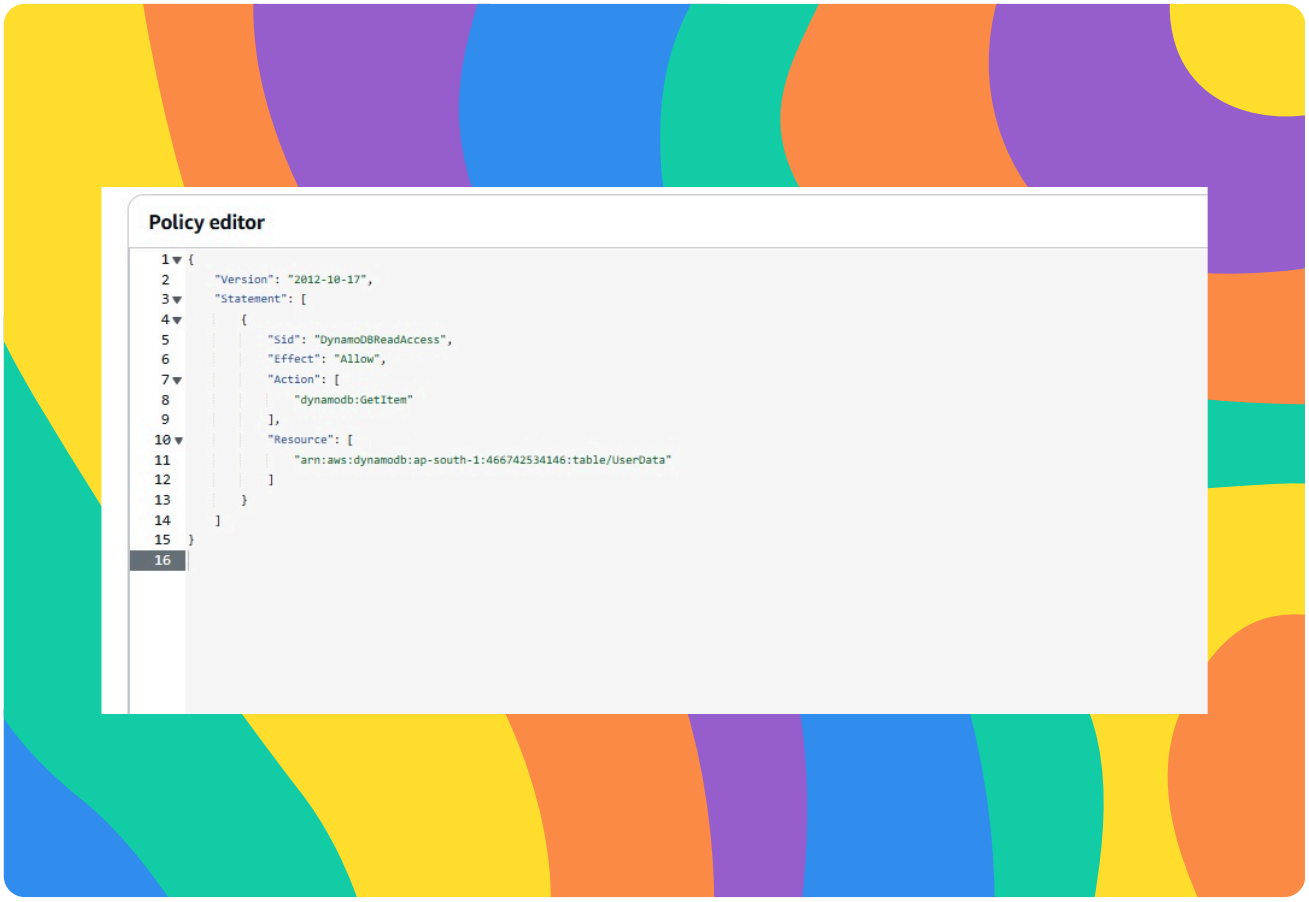

Executing function: succeeded (logs)
Details
{
"email": "test@example.com",
"name": "Test User",
"userId": "1"
}

Enahancing Security

For my project extension, I challenged myself to replace the managed policy with an inline policy to practice fine-tuning IAM permissions. This will make my Lambda function more secure by granting it access only to the specific DynamoDB table it needs—UserData—instead of all DynamoDB tables. By narrowing down the permissions, I reduce the risk of unauthorized access or unintended operations on other resources. This approach follows the principle of least privilege and strengthens the overall security posture of my application.

To create the permission policy, I used an inline policy directly attached to my Lambda function's execution role because it gave me more precise control. Instead of using the broad AmazonDynamoDBReadOnlyAccess managed policy, I created a custom policy that allows only GetItem access on the UserData table. This helps enforce the principle of least privilege, keeping my function secure while still giving it the access it needs to read from DynamoDB.

When updating a Lambda function's permission policies, you could risk breaking its access to required services. I validated that my Lambda function still works by running the test again in the Test tab. The function successfully retrieved data from my DynamoDB UserData table, confirming that the inline policy I created had the correct permissions. This shows that I've locked down access securely without affecting functionality — a big win for both security and performance.



Everyone should be in a job they love.

Check out nextwork.org for more projects