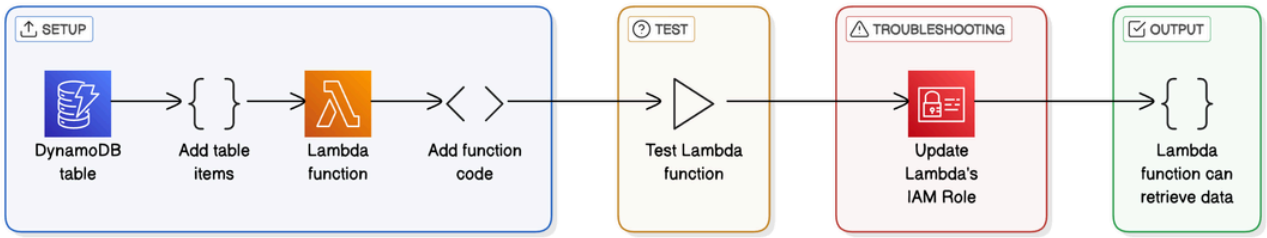
Kareshma
Rajaananthapadmanaban

AWS Three-tier series

Part 3

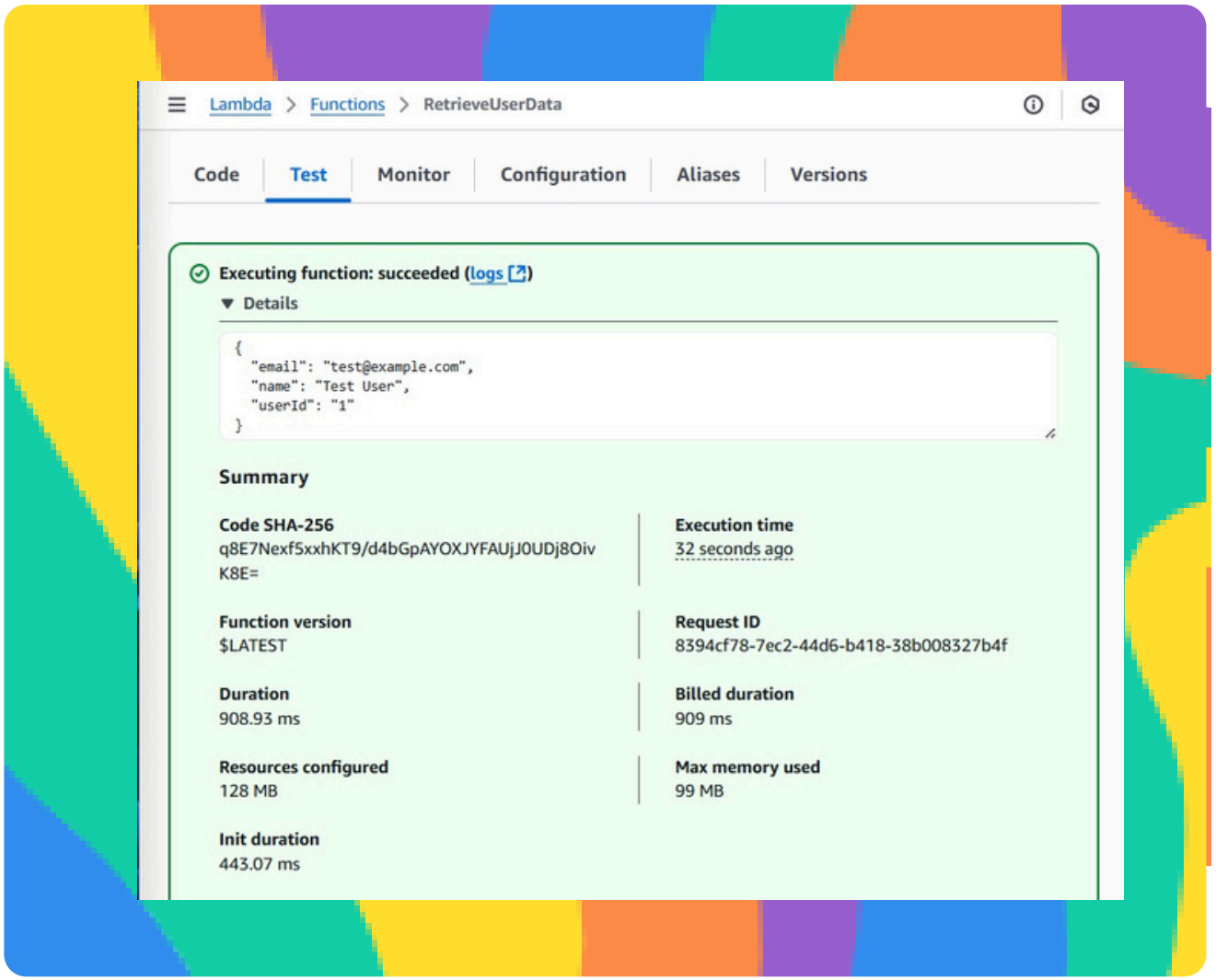
Learn how to build a serverless backend using AWS Lambda and DynamoDB to securely retrieve user data on demand



nextwork.org

Fetch Data with AWS Lambda

Kareshma Rajaananthapadmanaban



Introducing Today's Project!

In this project, I will demonstrate how to **retrieve data** from a **DynamoDB** table using a **serverless AWS Lambda** function. I'm doing this project to learn how serverless architectures manage and fetch backend data efficiently, forming the data tier of a scalable three-tier application.

Tools and concepts

Services I used were **AWS Lambda** and **DynamoDB**. Key concepts I learnt include Lambda functions as serverless compute, how to securely interact with DynamoDB using **IAM roles** and **fine-grained permissions,** how to write and test Lambda functions, and the importance of using inline policies for tighter security.

Project reflection

This project took me approximately 1 hour and 15 minutes to complete. The most challenging part was **resolving** the **AccessDenied** error by updating and refining IAM permissions. It was most rewarding to see my Lambda function successfully retrieve data from DynamoDB after securing it with a tightly scoped inline policy.

placeholder

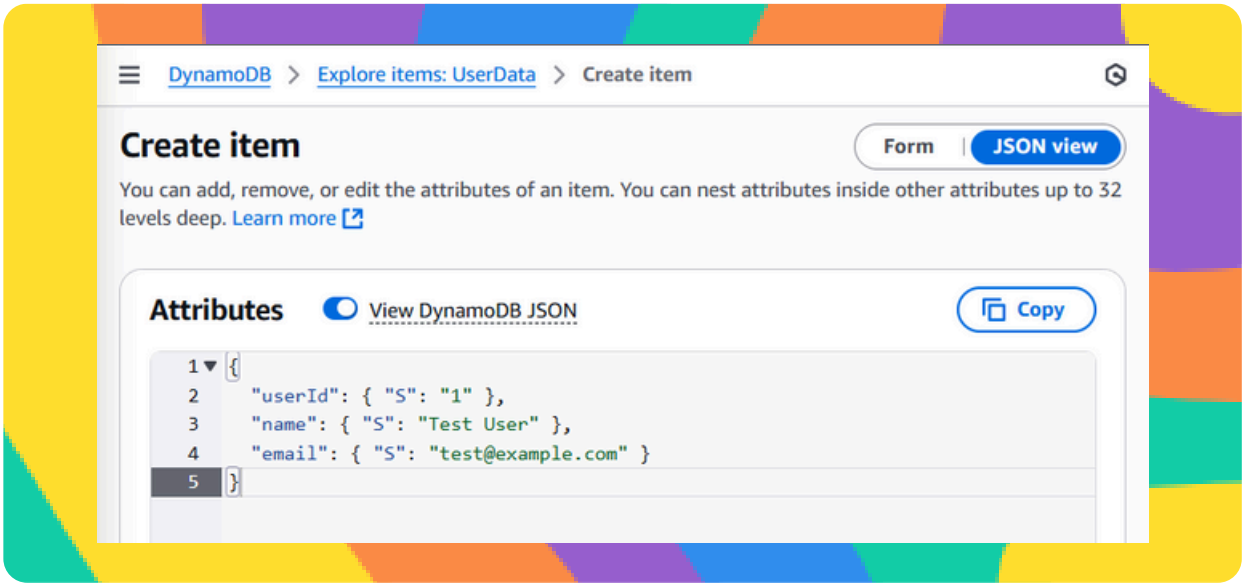
I did this project today to deepen my hands-on understanding of using AWS Lambda with DynamoDB in a real-world, serverless context.

This project met my goals by helping me practice secure permission setup, API integration, and efficient data retrieval all essential for building scalable cloud-native applic ations.

Project Setup

In this step, I will **create a DynamoDB table** because I need a scalable, serverless database to store and retrieve user data efficiently. This forms the foundation of the data tier in our three-tier architecture and allows our Lambda function to **interact** with persistent **backend storage**.

To set up my project, I created a database using DynamoDB called UserData. The **partition key** is userId, which means each item in the table is uniquely identified by a user ID. This key helps DynamoDB efficiently organize, store, and retrieve user- specific data across distributed servers.



Attributes Panel

I will **add** a **sample user item** to my UserData table because my Lambda function will need actual data to retrieve during testing. This ensures that when the function is invoked later, it can successfully query and return a real user record from DynamoDB.

In my DynamoDB table, I added a sample item with userId as **"1"**, along with the **user's name** and **email**. DynamoDB is schemaless, which means I didn't need to define name or email attributes in advance each item can have different attributes, giving me **flexibility** to **store varied data**.

I will create a **Lambda function** to retrieve data from my DynamoDB table because I want to build a serverless backend that can **fetch user data** dynamically based on requests. This allows my application to query the database without managing any servers.

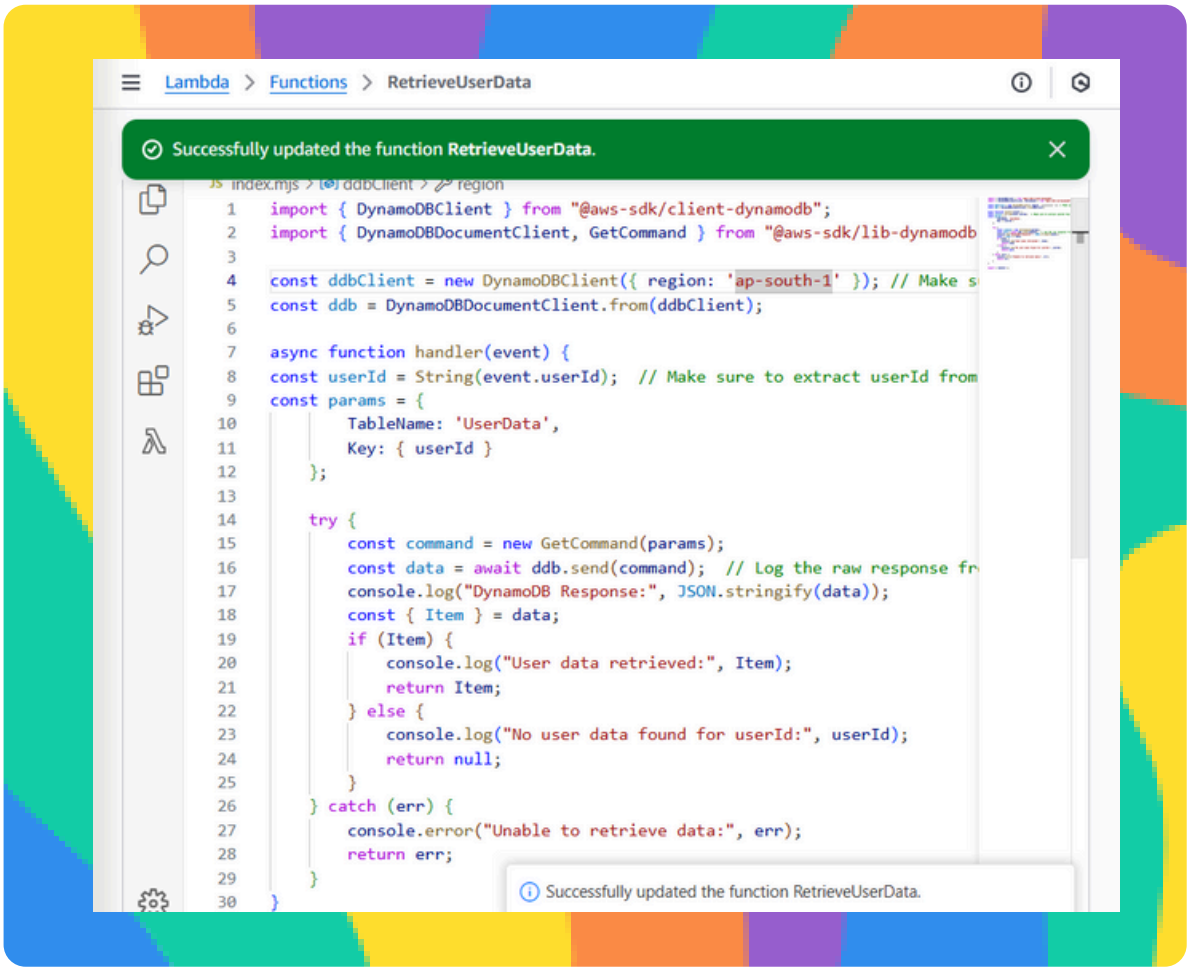
AWS Lambda

AWS Lambda is a **serverless compute service** that lets you **run code** without provisioning or managing servers. I'm using Lambda in this project to retrieve user data from a DynamoDB table efficiently, **without** needing to handle **backend infrastructure** manually.

AWS Lambda Function

My Lambda function has an **execution role,** which is an IAM role that defines what the function is allowed to do within AWS. **By default,** the role grants permissions to **write logs to CloudWatch**, helping me **monitor** and debug my function's execution **securely**.

I will **write** and **deploy** the code for my Lambda function because I need it to fetch user data from the DynamoDB table when triggered. This function logic is essential for enabling dynamic, serverless data retrieval in my backend system.



My Lambda function will retrieve user data from a DynamoDB table based on a provided userId. It uses the **AWS SDK** for JavaScript to **connect to DynamoDB,** send a **GetCommand** to fetch the matching item, and return it. If the item isn't found or an error occurs, it logs the result and returns a null value or error accordingly.

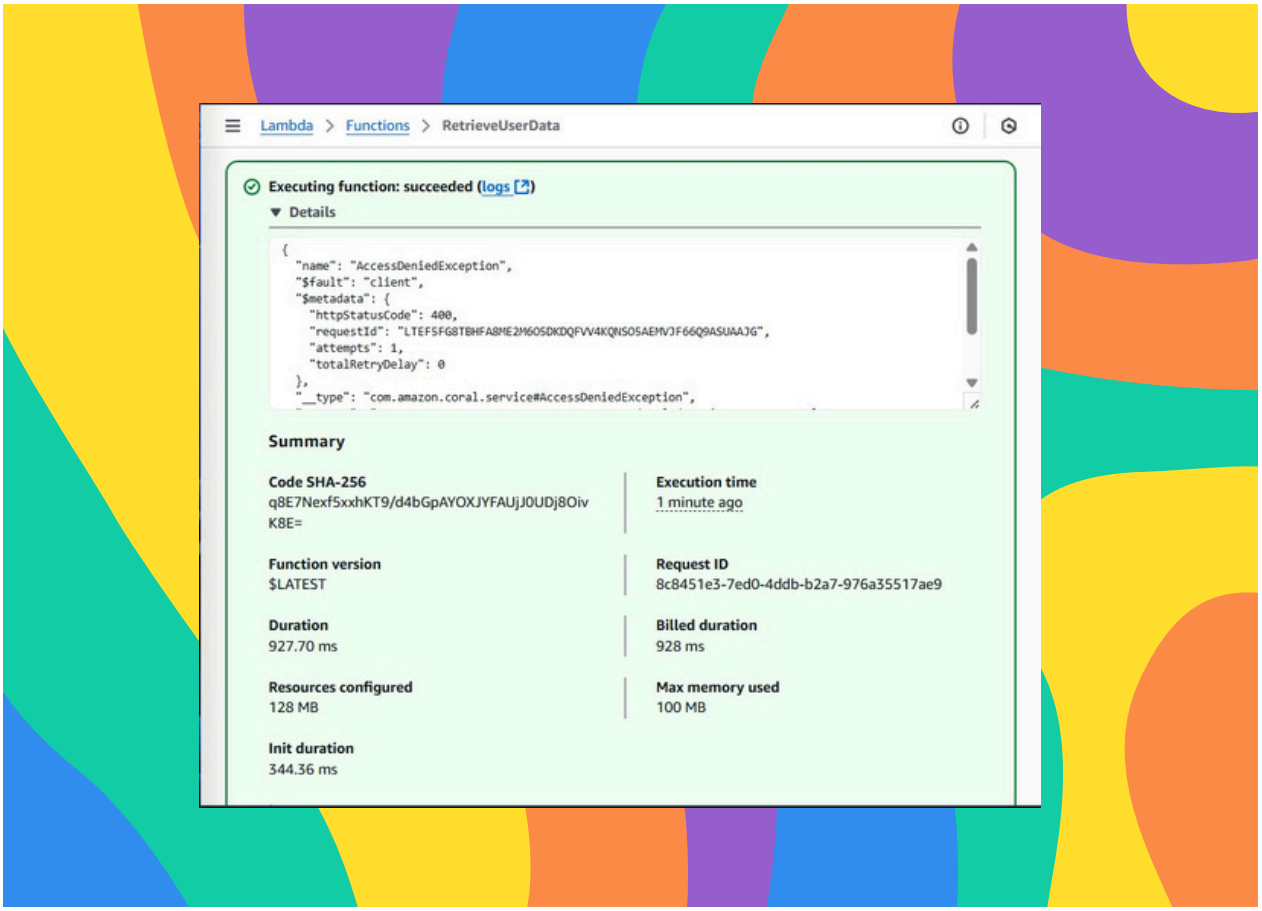
The code uses AWS SDK, which is a **collection of libraries** that make it easier for developers to interact with AWS services using their preferred programming language. My code uses SDK to connect to DynamoDB, send a request to fetch a user's data using their userId, and handle the response without writing low-level API calls.

Function Testing

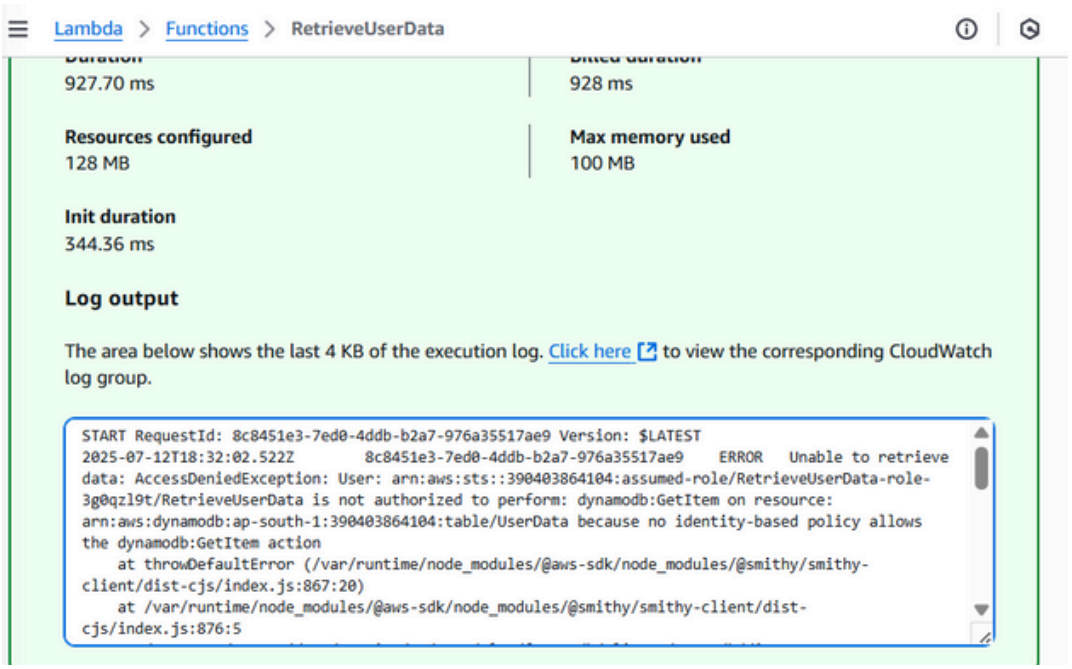
In this step, I will create and **run a test event** for my Lambda function because I want to **validate** whether it can successfully **retrieve data** from the DynamoDB table. Testing helps ensure that the function works as expected before integrating it with other AWS services.

To test whether my Lambda function works, I used the Test tab in the Lambda console and entered a JSON event with a userId of "1". The test is written in **JSON** because it's the **standard format Lambda** understands for input events. If the test is successful, I'd see the user's data returned from the DynamoDB table.

The test displayed a "success" because the Lambda function ran without syntax or runtime errors, but the **function's response** was actually an **access denied** error because it didn’t have permission to read from the DynamoDB table. We need to explicitly grant access in the Lambda execution role.



Error while testing a Lambda function



Function Permissions

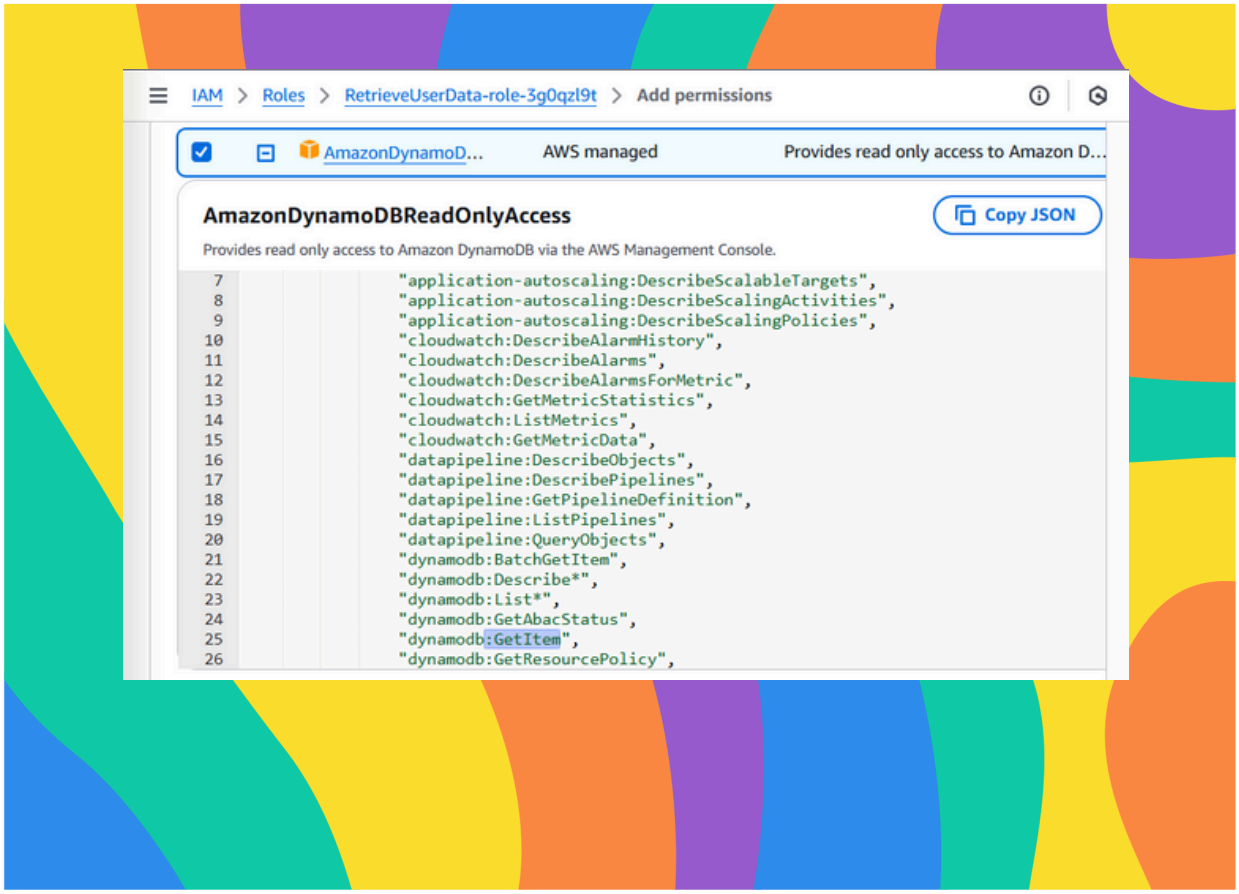
In this step, I will **grant** my Lambda function **read access** to DynamoDB because, without the correct permissions, the function cannot retrieve any items from the database. This step ensures that my Lambda function can s**ecurely interact** with the UserData table.

This policy allows your Lambda function to **read data** (e.g., GetItem, Scan, Query) from DynamoDB, which is exactly what we need to retrieve user info no write or delete access necessary. It's a secure and minimal-permission option, perfect for this use case.

To resolve the AccessDenied error, I reviewed the error message and found that the denied action was dynamodb:GetItem.

This means my Lambda function **lacks read access** to DynamoDB. I'm attaching the AmazonDynamoDBReadOnlyAccess policy because it grants permission to perform read operations like GetItem, resolving the error securely.

There were **four** DynamoDB permission policies I could choose from, but I didn't pick AWSLambdaDynamoDBExecutionRole or AWSLambdaInvocation-DynamoDB because they are meant for working with DynamoDB Streams, not for reading items from a table. My **Lambda function** needs **GetItem access**, which only the AmazonDynamoDBReadOnlyAccess policy provides.



DynamoDB Permission Policies

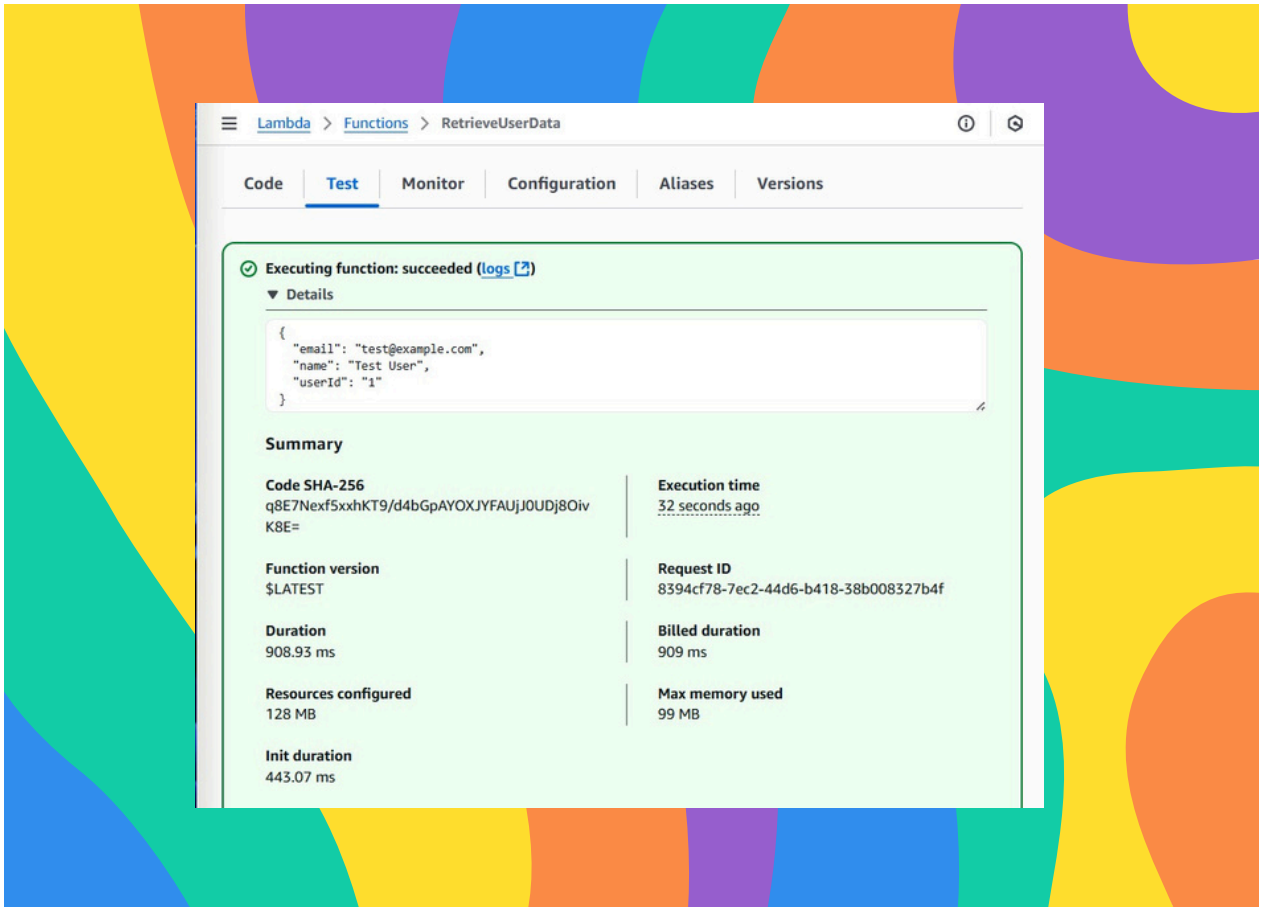
I also **didn't** pick AmazonDynamoDBFullAccess because it **grants unnecessary permissions,** including write and delete access, which could pose a security risk. AmazonDynamoDBReadOnlyAccess was the right choice because it provides just **enough access** like **GetItem** for my Lambda function to securely read data without overexposing the database.

Final Testing and Reflection

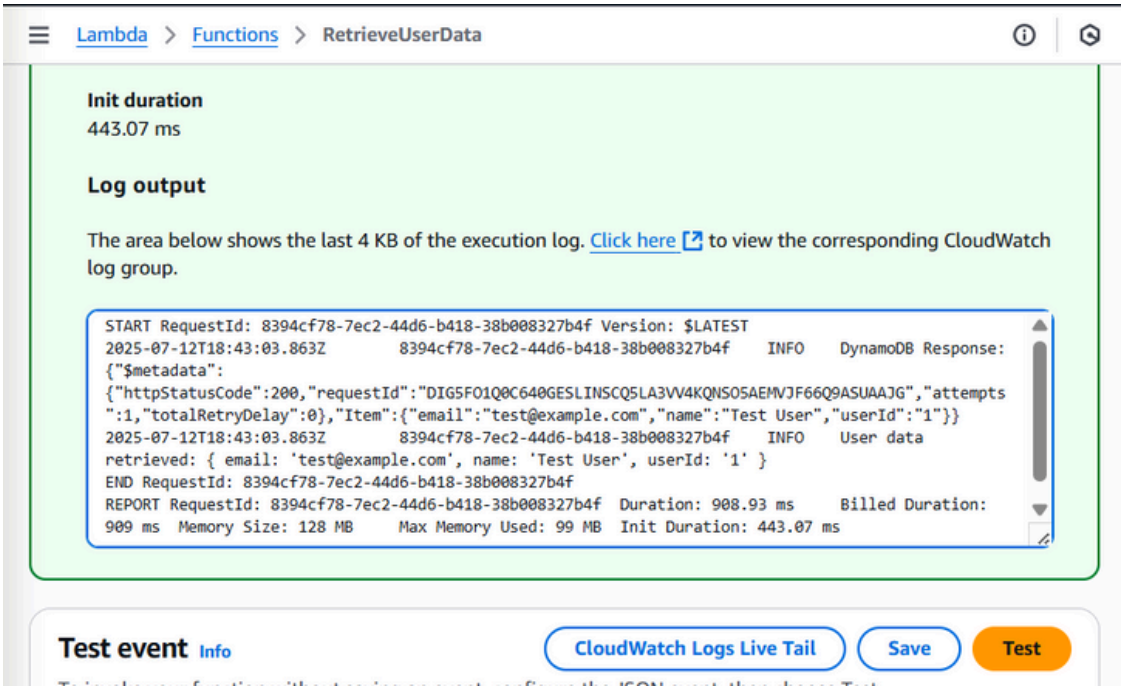
In this step, I will **re-run** my Lambda function test because I've now added the **required** DynamoDB **read permissions**. This will confirm whether the function can successfully retrieve data from the UserData table and verify that the access issue has been resolved.

To **validate** my **new permission settings**, I re-ran the Lambda function test using a sample event with userId: "1". The results were successful and returned the correct user data from DynamoDB because the function now has AmazonDynamoDBReadOnlyAccess, which includes permission to perform the GetItem operation.

Web apps are a popular use case of using **Lambda** and **DynamoDB.** For example, I could build a user profile system that fetches data when someone logs in, create a product catalog that updates dynamically, or design a blog that retrieves posts based on search queries all without managing any servers.



Successful test results



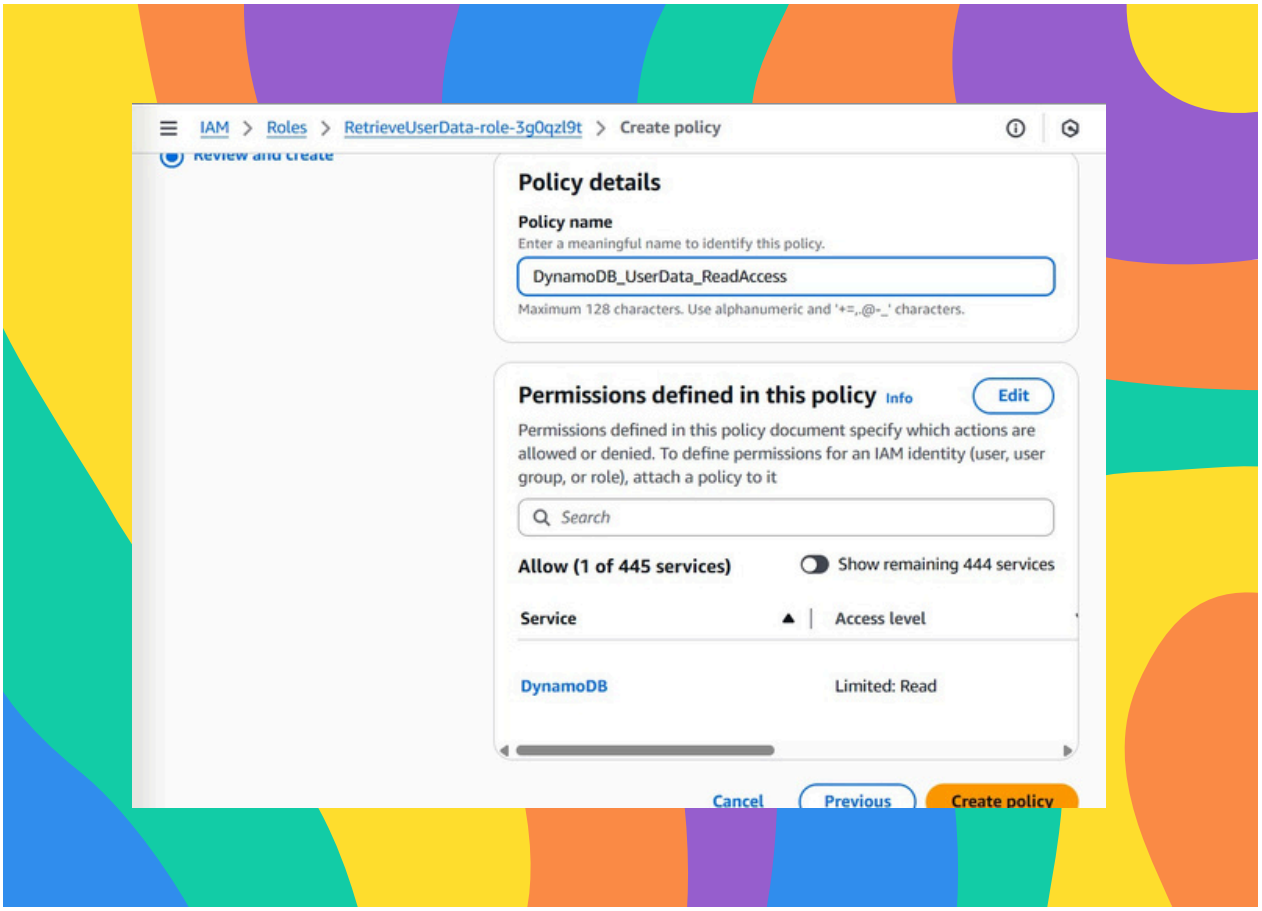
Enahancing Security

In this secret mission, I will **update** my **Lambda function's permission policies** to follow the principle of **least privilege** because it's important to give only the exact access needed. This improves security by reducing the risk of unauthorized actions on my DynamoDB table.

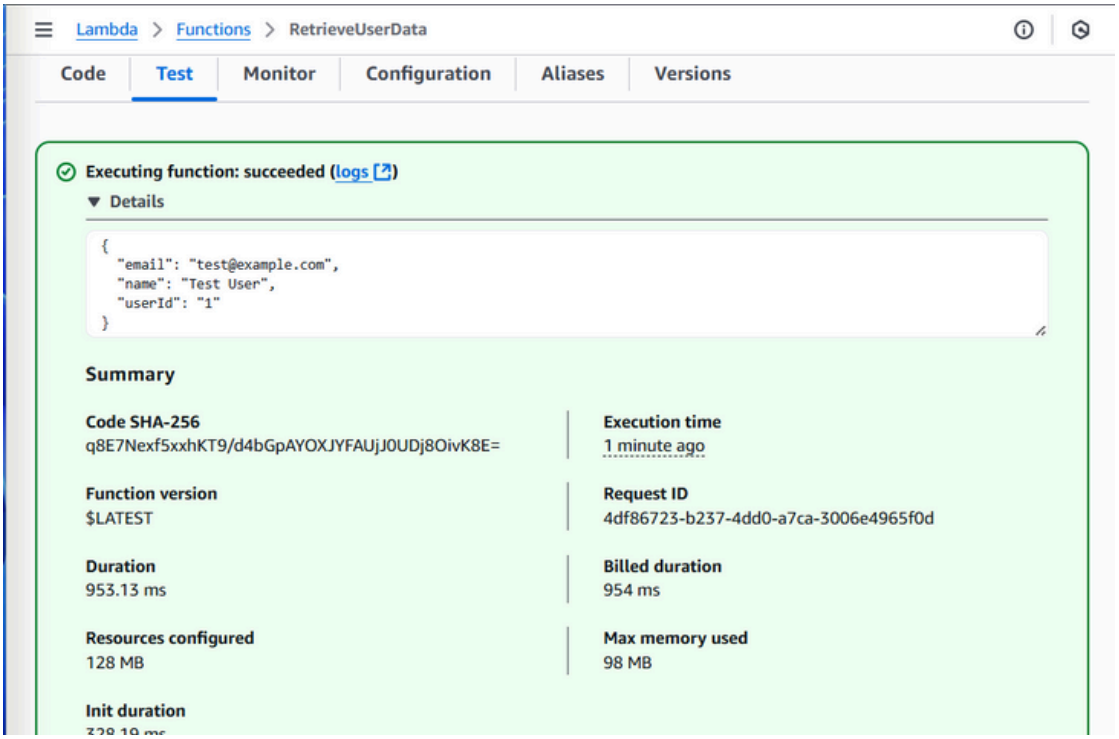
For my project extension, I challenged myself to replace the broad AmazonDynamoDBReadOnlyAccess policy with a **custom inline policy** that **only** allows **access** to the **specific UserData table**. This will tighten security by limiting the Lambda function to exactly what it needs no more, no less.

To create the permission policy, I used the **JSON method** because it gave me more control and was quicker to implement. I'm comfortable writing IAM policies in JSON, and it allowed me to directly **specify** only the **exact GetItem** action for my UserData table, **minimizing excess permissions**.

When updating a Lambda function's permission policies, you could risk accidentally removing access it needs to function properly. I validated that my Lambda function still works by re-running the test in the Test tab and confirming that it **successfully retrieved the item** from my DynamoDB table.



Permission policy setup



Delete the Resources

To avoid incurring unnecessary AWS charges, clean up the resources created during this project:

▼ Delete the Lambda Function

1. Go to the Lambda service in the AWS Console.
2. Select the RetrieveUserData function.
3. Click Actions → Delete.
4. Type confirm to confirm deletion.
5. Click Delete.

▼ Delete the DynamoDB Table

1. Go to the DynamoDB service → Tables from the left panel.
2. Select the UserData table.
3. Click Delete.
4. In the prompt:
5. ✅ Keep “Delete all CloudWatch alarms” checked.
6. ❌ Leave “Create an on-demand backup” unchecked.
7. Type confirm to confirm deletion.
8. Click Delete.

Kareshma
Rajaananthapadmanaban

Follow up for more interesting projects !!!

Check out nextwork.org / my profile for more projects