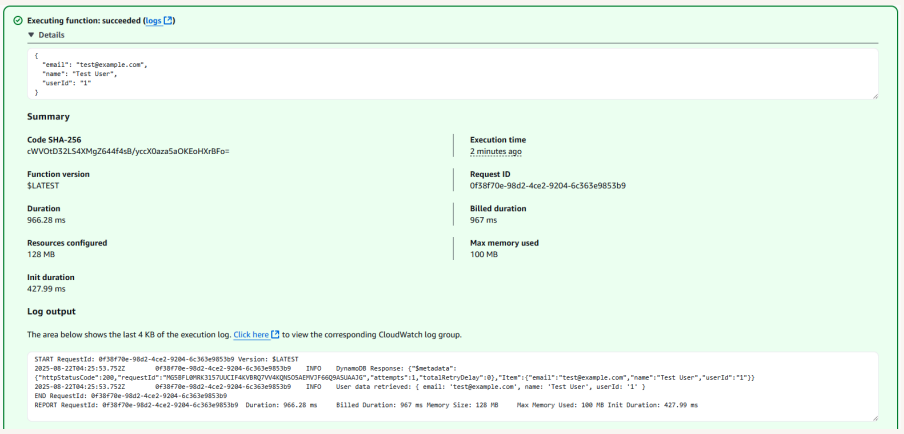
nextwork.org

Fetch Data with AWS Lambda

Louis Moyo



L
Louis Moyo
NextWork Student

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.

L

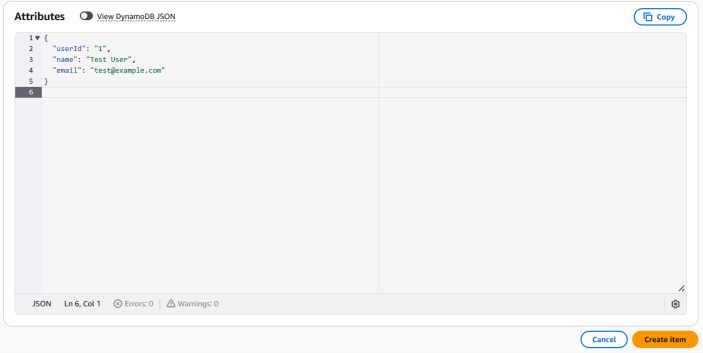
Louis Moyo
NextWork Student

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.



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.

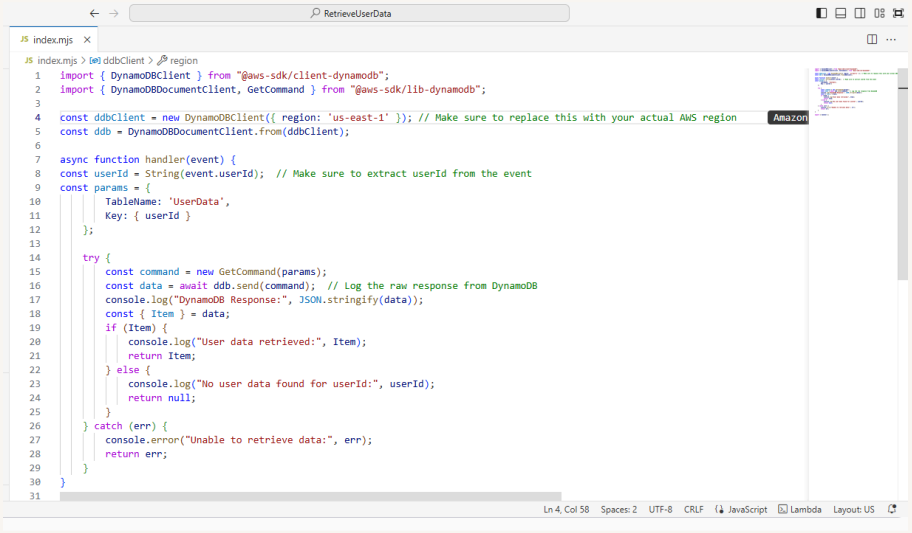
AWS Lambda Function

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.

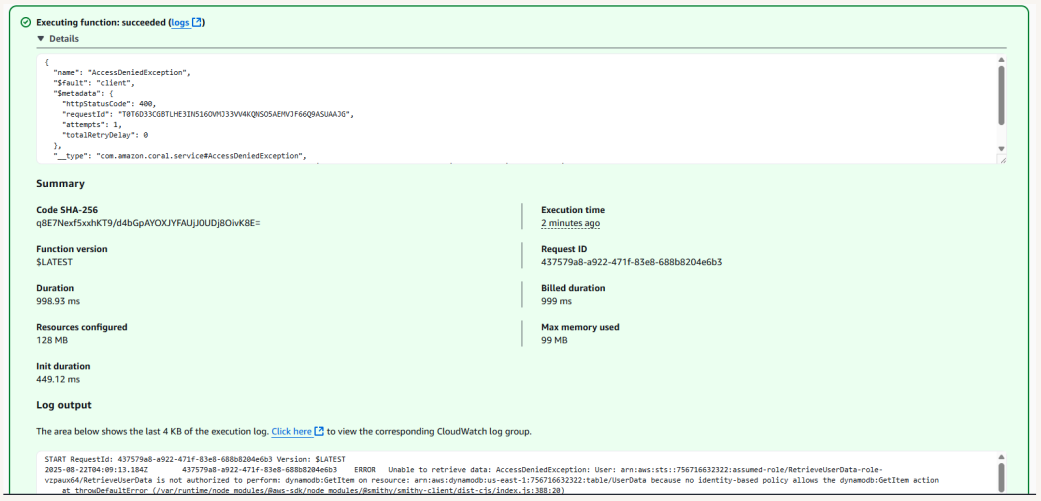
Louis Moyo
NextWork Student



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.

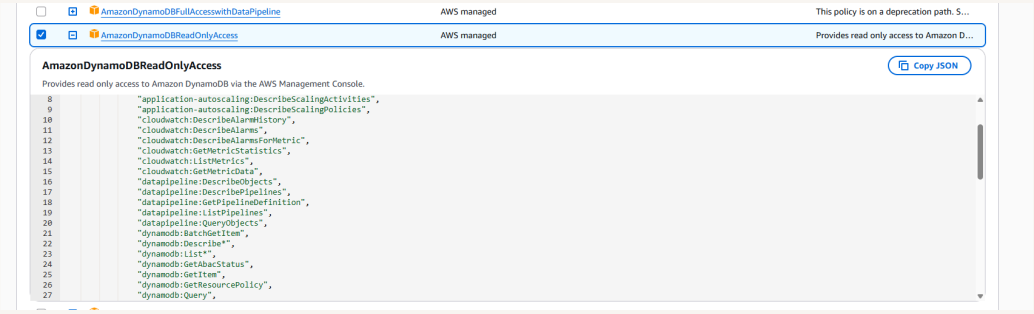


Function Permissions

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.

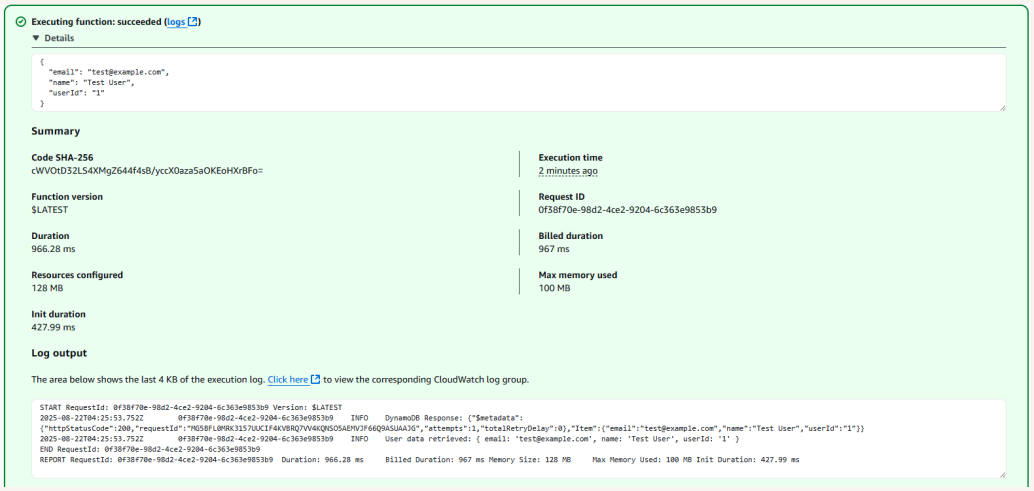
I also didn't pick the full access or stream-related policies because they grant more permissions than my function needs. AmazonDynamoDBReadOnlyAccess was the right choice because my Lambda function only needs to read data with actions like GetItem, and this policy follows the principle of least privilege.


AmazonDynamoDBFullAccesswithDataPipeline
AWS managed
This policy is on a deprecation path. S...
AmazonDynamoDBReadOnlyAccess
AWS managed
Provides read only access to Amazon D...
AmazonDynamoDBReadOnlyAccess
Copy JSON
Provides read only access to Amazon DynamoDB via the AWS Management Console.
"application-autoscaling:DescribeScalingActivities",
"application-autoscaling:DescribeScalingPolicies",
"cloudwatch:DescribeAlarmHistory",
"cloudwatch:DescribeAlarms",
"cloudwatch:DescribeAlarmsForMetric",
"cloudwatch:GetMetricStatistics",
"cloudwatch:ListMetrics",
"cloudwatch:GetMetricData",
"datapipeline:DescribeObjects",
"datapipeline:DescribePipelines",
"datapipeline:GetPipelineDefinition",
"datapipeline:ListPipelines",
"datapipeline:QueryObjects",
"dynamodb:BatchGetItem",
"dynamodb:Describe*",
"dynamodb:List*",
"dynamodb:GetAbacStatus",
"dynamodb:GetItem",
"dynamodb:GetResourcePolicy",
"dynamodb:Query",

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.

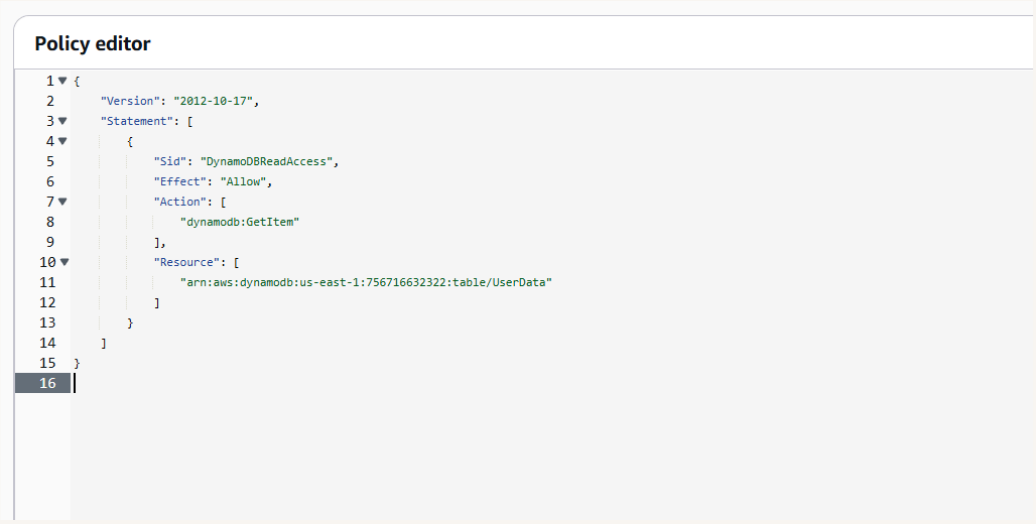


Enahancing Security

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
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "DynamoDBReadAccess",
"Effect": "Allow",
"Action": [
"dynamodb:GetItem"
],
"Resource": [
"arn:aws:dynamodb:us-east-1:756716632322:table/UserData"
]
}
]
}

The place to learn & showcase your skills

Check out nextwork.org for more projects