2-2 Lambda code

Tuesday, August 24, 2021

6:57 AM

1 
3 
4 
5 
6 
7 
8 
9 
in&x.js 
const AWS = require( taws-sdk' ) ; 
const dynamoDB 
= new AWS.DynamoDB({region: 'use-east-2' , 
exports. handler = 
(event, context, callback) => { 
console. log(event) ; 
const age = event. age; 
callback(null, 
age) ; 
apiVersion . 
• '2012-08-10'}); 



Why do we need outside of handler function?

It's an advanced concept but in general, the code running here is not online all the time.

Whenever a trigger occurs running your lambda function, AWS quickly spins up like a server environment, some wrapper containing your lambda function.

This is really fast but it doesn't tear down this wrapper once your function finishes, it keeps it alive for a couple of minutes. Therefore if your function executes a couple of times in a short time span, it will reuse that wrapper.

Let's take advantage of this

If that wrapper is being started, it will execute everything in this file, not just your function handler.

If the wrapper is still up though, it will not re-execute the part outside of your function handler and that of course gives us a little performance edge.

If we put the code which doesn't need to get executed on each triggering event outside of that handler and that's just what we are doing here. Because importing the SDK and setting up the DynamoDB object, that's always going to be the same process, so we don't necessarily need to put that in our function handler.

1 
2 
3 
4 
5 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
Execution results x 
const AWS = require( ' aws-sdk'); 
const dynamodb = new AWS.DynamoDB({region: 'use-east-2' , 
exports. handler = 
context, callback) { 
const params = 
Item: { 
"Userld": { 
apiVersion. 
• '2012-08-101}); 
S: " askdfosjf" 
"Age": { 
"Height" : 
N: "72" 
"Income" : { 
N: "2500" 
TableName: " compare-yourself" 
dynamodb. putltem(params, function(err, 
if (err){ 
console. log(err) ; 
callback(); 
} else { 
console. log(data) 
data) ; 
callback(null, 
data) { 

But we cannot test our lambda function because it gives us authorization error. So, We need to give a permission to test our code successfully.

below is basic execution role(default role)

Permissions 
Policy usage 
Policy summary 
JSON 
Tags 
Policy versions 
Access Advisor 
Edit policy 
"version" : 
"2012-10-17", 
" Statement" : 
" Effect" : 
'"Allow" , 
"'Action": " logs : CreateLogGroup" , 
"Resource : " 
" arm : : logs : Us -east-2 3833265es63e: k" 
"Effect" : 
"Allow" , 
"Action " : 
"logs :CreateLogStream" , 
"10gs : PutLogEvents " 
"Resource" 
"arn: aws : logs : us -east -2: 383326505630 : log-group: / aws/lambda/ cy-store-data 

This allow us two things

* 1. create a log group
  2. create a log stream

So, We are going to attach a new policy which is

'AmazonDynamoDBFullAccess'

1 
2 
3 
4 
5 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
25 
26 
27 
28 
29 
30 
31 
inckx.js 
const AWS require( aws-sdk' 
const dynamodb = new ' us-east-2' , 
exports. handler = (event, 
context, callback) { 
const params = { 
Item: { 
"Userld" 
S: "user_" + Math. random() 
"Age": { 
N: event. age 
"Height" 
N: event. height 
" Income": { 
N: event. income 
apiVersion: 
' 2012-08-10 • ; 
TableName: "compare-yourself" 
dynamodb. putltem(params, function (err, 
if (err) { 
console. log(err) ; 
callback(err) ; 
} else { 
console. log(data); 
callback(null, data); 
data) { 

Now a better way though is to pass the right format by API Gateway then this whole body mapping template will make more sense.

If we visit the API Gateway, we can wrap our extraction of age, height and income here into quotation marks and this will now not pass a string which holds $inputRoot.age but it holds the value of that expression but as a string, you learned this in the API Gateway module.

So this now gives the mapping template a whole new sense. Because now, we not only extract data which would have been there before anyways but we also transformed the type of data and if we receive

a number, we automatically turn it into a string here already.

**Test Run**

Request: /compare-yourself 
Status: 200 
Latency: 460 ms 
Response Body 
"your-age" Request Body 
"age" : 48, 
"height" 
"income" : 1804 

compare-yourself 
Overview Items 
O 
Metrics 
Alarms 
Capacity 
Create item 
Actions v 
Scan: [Table] compare-yourself: Userld 
Indexes 
Height 
72 
72 
69 
Global Tables 
Income 
1800 
More v 
Viewing 1 to 3 items 
Sca n 
[Table] compare-yourself: Userld 
O Add filter 
Start search 
Userld O 
dasf787af8safa 
oanvosdjflj 
user 0.07026859306250022 
Age 
28 
28 
48 

Here, we get exactly the data we sent stored here without an error because we transformed into a string in the body mapping template of API Gateway and then we use this data here in lambda by accessing it on the event object.

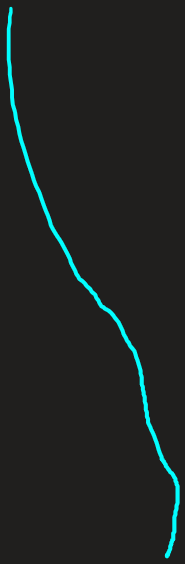
This shows the full chain from sending data with API Gateway over transforming it to passing it to lambda, to using it there to create a new entry on DynamoDB.

Web Testing



codepen.io

1



?

İframeCons01eRunner- 
İframeCons01eRunner- 
X04c013047d 
712.js:1 
auc013047d 
712. is:ı 

chrome

console

3

DynamoDB

Userld O 
dasf787af8safa 
oanvosdjflj 
Age 
28 
28 
48 
26 
Height 
72 
72 
69 
58 
Income 
2500 
2500 
1800 
user 
user 
0.07026859306250022 
0.295392264251503 