Lambda: A first tutorial

Welcome, this walk through guides you to create a couple lambda functions. The first is a simple "Hello World" level function. The second, moves in the direction of using the DynamoDB AWS storage engine to collect data through the lambda function.

We will partially follow some existing AWS tutorials, with a little bit of deviation here and there for using Python 3 instead of Node.js or Python 2.

Step 1: Set Up an AWS Account and the AWS CLI

(Done is prior labs)

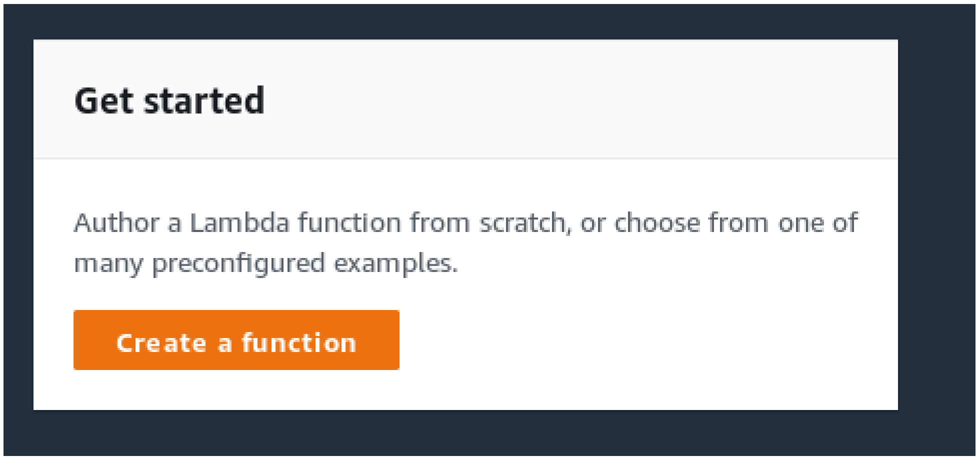
Log into the AWS Console

Step 2: Create a HelloWorld Lambda Function and Explore the Console

http://docs.aws.amazon.com/lambda/latest/dg/getting-started-create-function.html

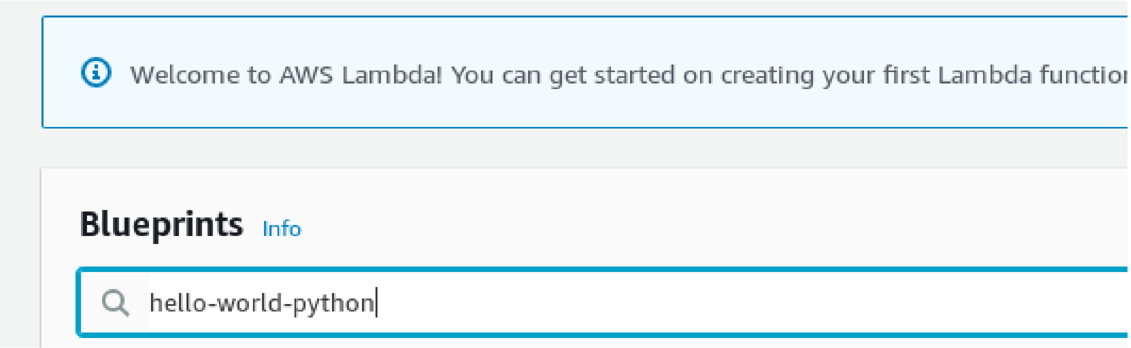
Step 2.1: Create a Hello World Lambda Function

https://console.aws.amazon.com/lambda/home?region=us-east-1#/home



Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

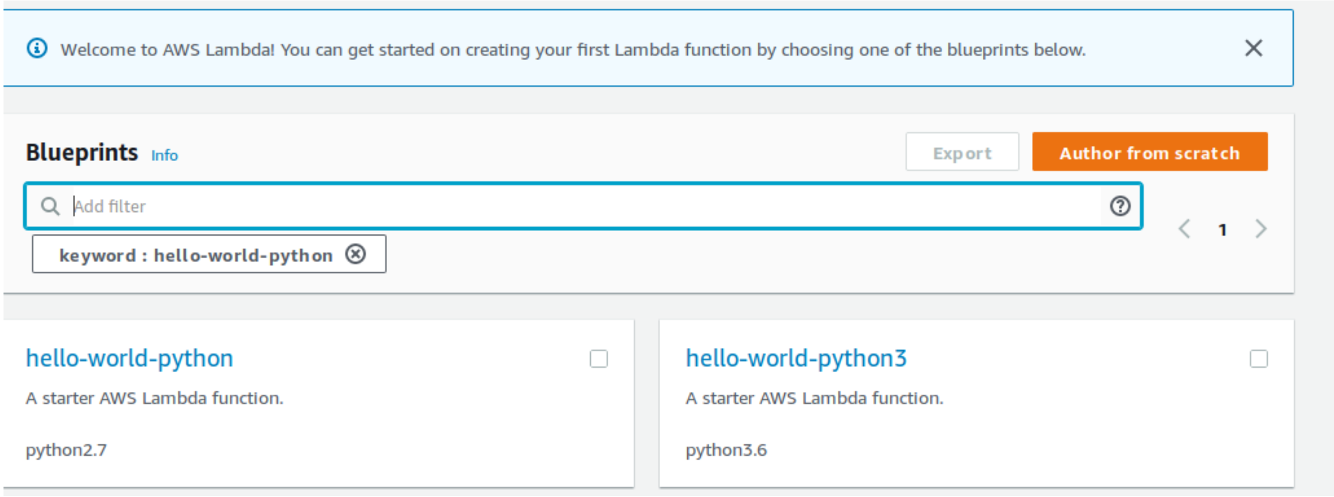


Welcome to AWS Lambda! You can get started on creating your first Lambda functio

Blueprints

hello-world-python

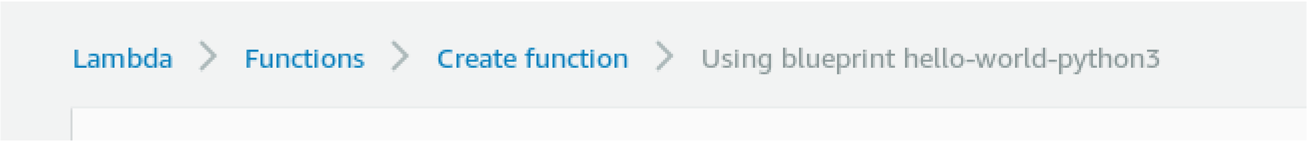
NOTE: We will be using Python 3, not 2.

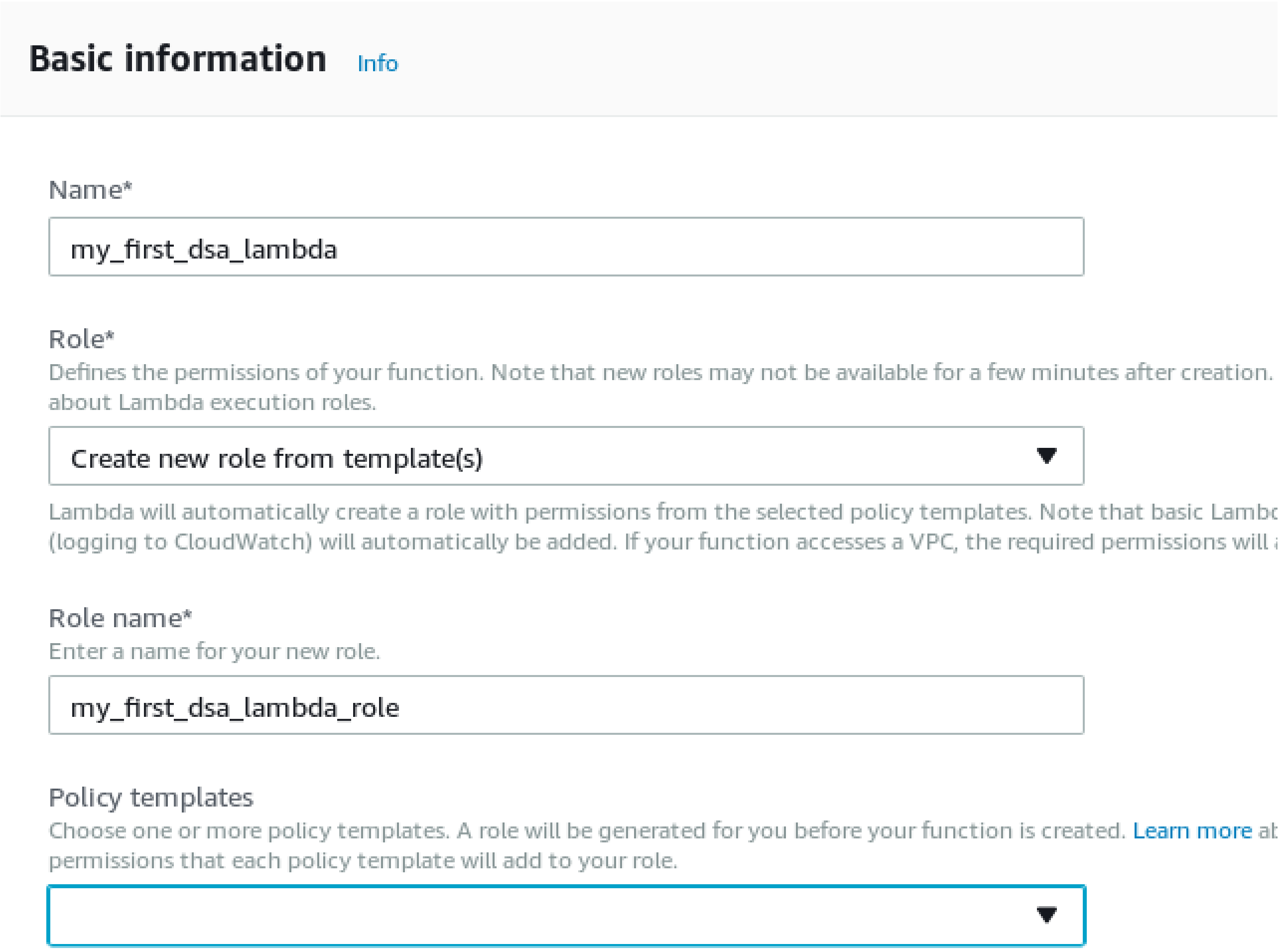

Welcome to AWS Lambda! You can get started on creating your first Lambda function by choosing one of the blueprints below.

hello-world-python
A starter AWS Lambda function.
python2.7

hello-world-python3
A starter AWS Lambda function.
python3.6

SELECT the “hello-world-**python3**” blueprint


Lambda > Functions > Create function > Using blueprint hello-world-python3



Basic information

Name*

my_first_dsa_lambda

Role*

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. about Lambda execution roles.

Create new role from template(s) ▼

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambc (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will a

Role name*

Enter a name for your new role.

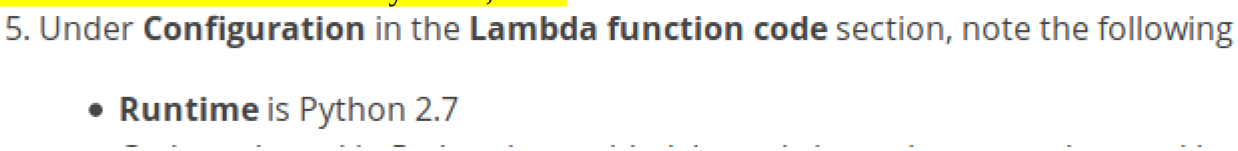
my_first_dsa_lambda_role

Policy templates

Choose one or more policy templates. A role will be generated for you before your function is created. Learn more at permissions that each policy template will add to your role.

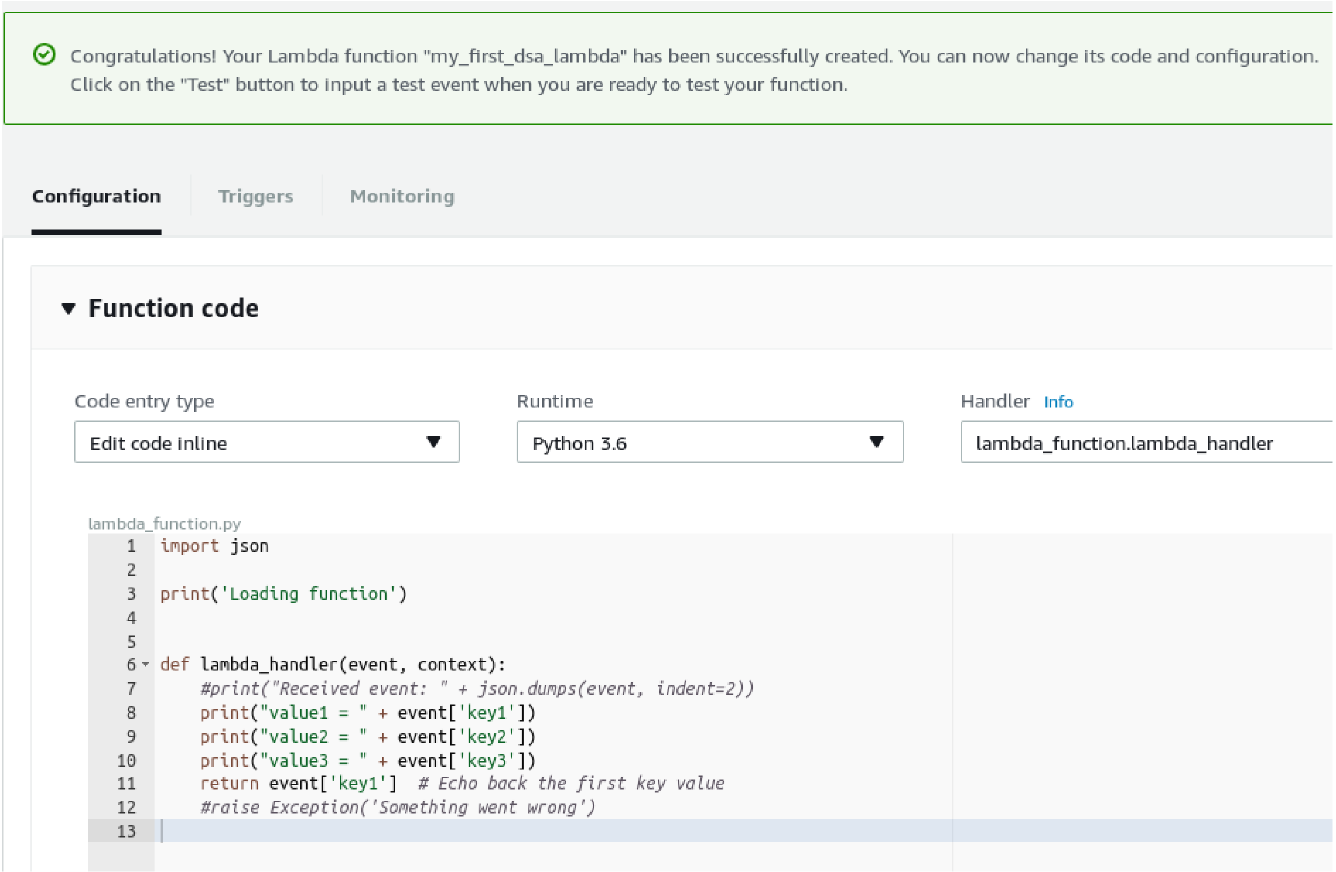
▼

Deviate from Tutorial: Choose Python 3, not 2



5. Under **Configuration** in the **Lambda function code** section, note the following

- **Runtime** is Python 2.7



Congratulations! Your Lambda function "my_first_dsa_lambda" has been successfully created. You can now change its code and configuration. Click on the "Test" button to input a test event when you are ready to test your function.

▼ Function code

Code entry type: Edit code inline ▼

Runtime: Python 3.6 ▼

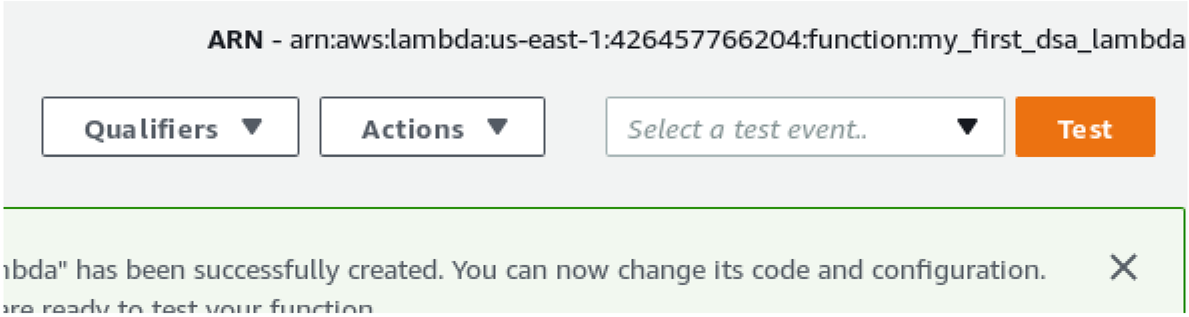
Handler Info: lambda_function.lambda_handler

lambda_function.py

```python
import json

print('Loading function')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    print("value1 = " + event['key1'])
    print("value2 = " + event['key2'])
    print("value3 = " + event['key3'])
    return event['key1']  # Echo back the first key value
    #raise Exception('Something went wrong')

```

Step 2.2: Invoke the Lambda Function Manually and Verify Results, Logs, and Metrics

TESTING:

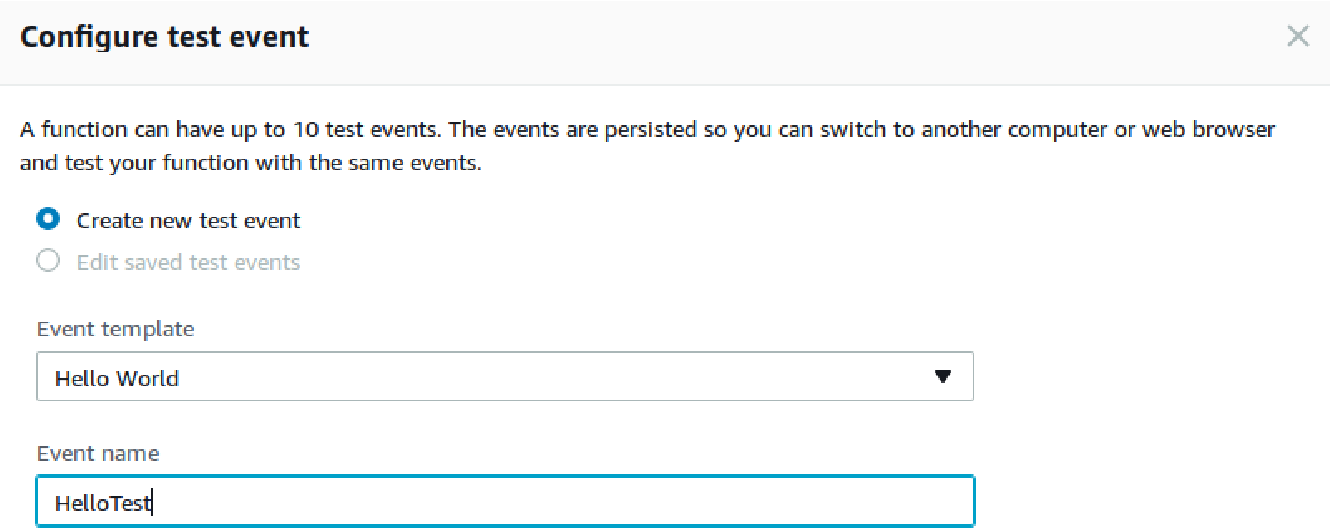


ARN - arn:aws:lambda:us-east-1:426457766204:function:my_first_dsa_lambda

bda" has been successfully created. You can now change its code and configuration. ✕

re ready to test your function

DEVIATE: Specify "Event name"



Configure test event ✕

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

● Create new test event

○ Edit saved test events

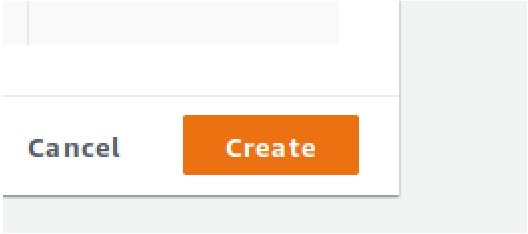
Event template

Hello World ▼

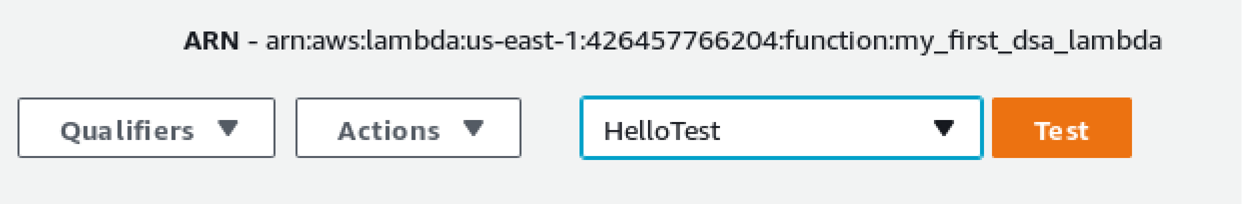
Event name

HelloTest

Then Create

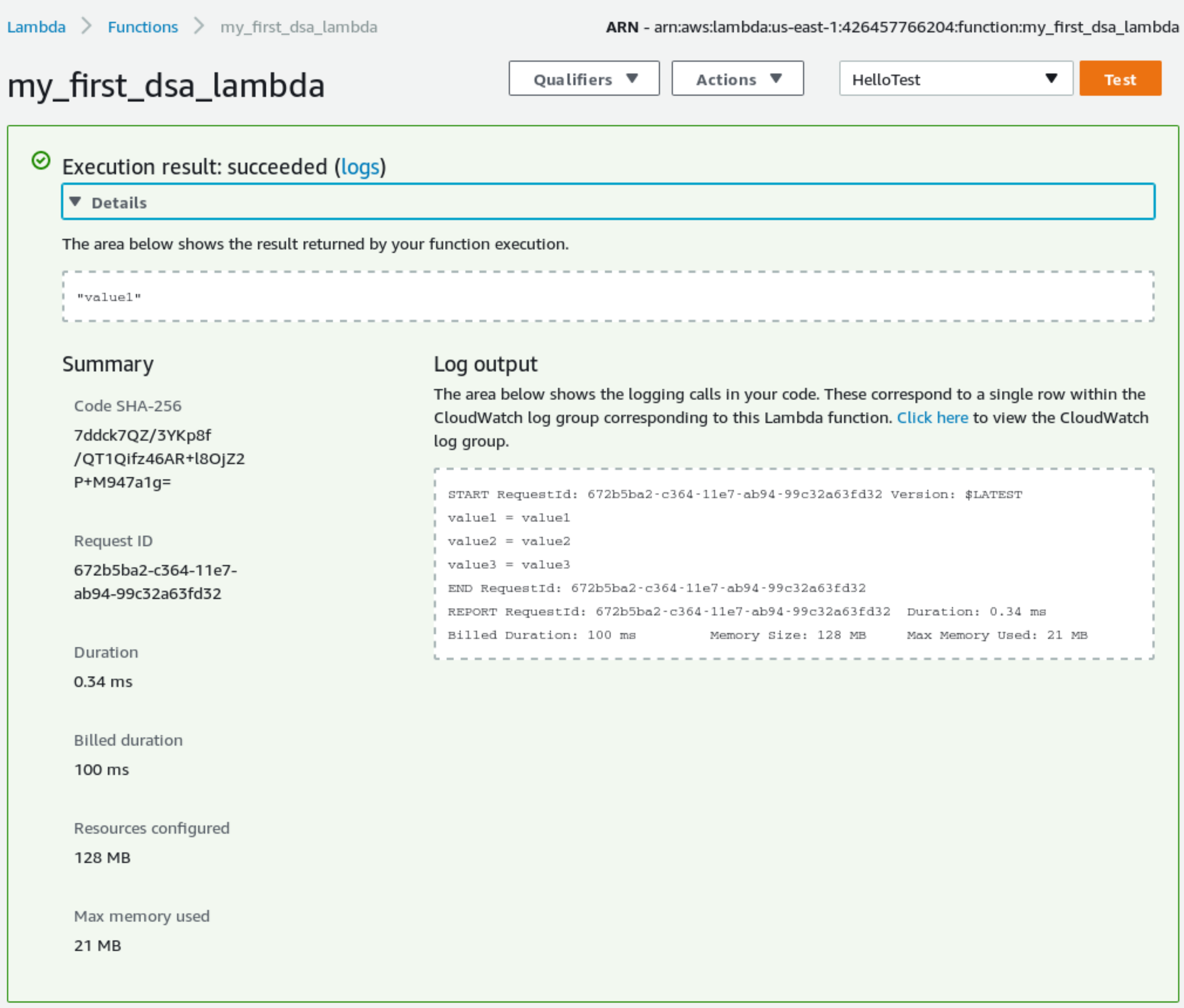


We now have a test event



ARN - arn:aws:lambda:us-east-1:426457766204:function:my_first_dsa_lambda

Click Test Again:



ARN - arn:aws:lambda:us-east-1:426457766204:function:my_first_dsa_lambda

my_first_dsa_lambda

Execution result: succeeded (logs)

▼ Details

The area below shows the result returned by your function execution.

```
"value1"
```

Summary

Code SHA-256

7ddck7QZ/3YKp8f/QT1Qifz46AR+l8OjZ2P+M947a1g=

Request ID

672b5ba2-c364-11e7-ab94-99c32a63fd32

Duration

0.34 ms

Billed duration

100 ms

Resources configured

128 MB

Max memory used

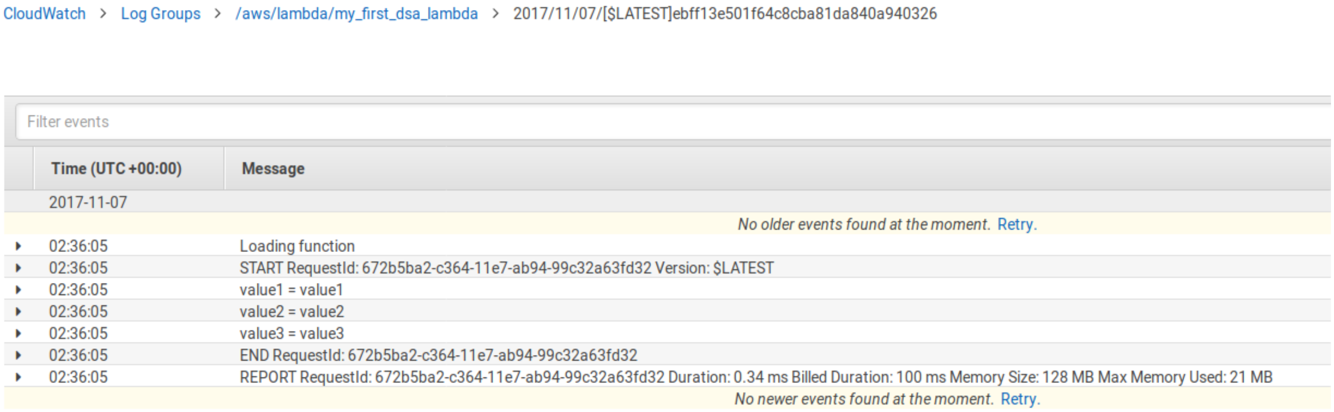
21 MB

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32 Version: $LATEST
value1 = value1
value2 = value2
value3 = value3
END RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32
REPORT RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32  Duration: 0.34 ms
Billed Duration: 100 ms         Memory Size: 128 MB     Max Memory Used: 21 MB
```

Clicking on **(logs)** takes you to CloudWatch:

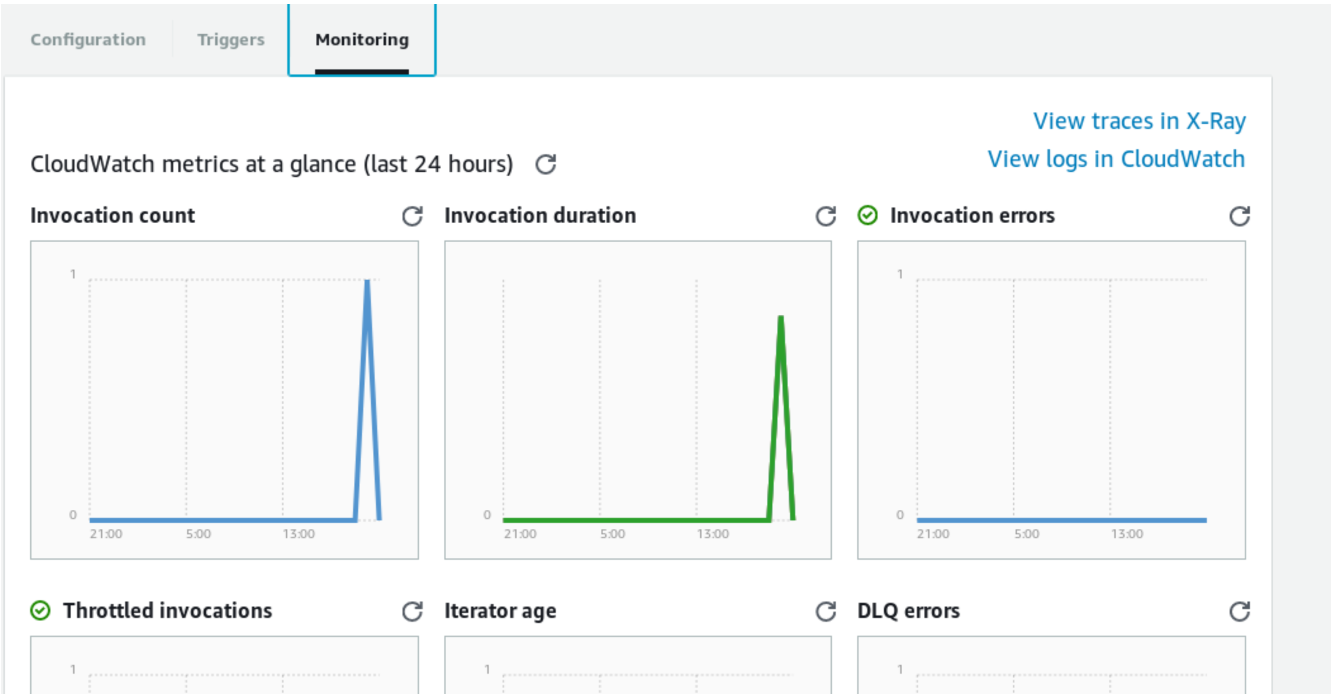


CloudWatch > Log Groups > /aws/lambda/my_first_dsa_lambda > 2017/11/07/[$LATEST]ebff13e501f64c8cba81da840a940326

Filter events

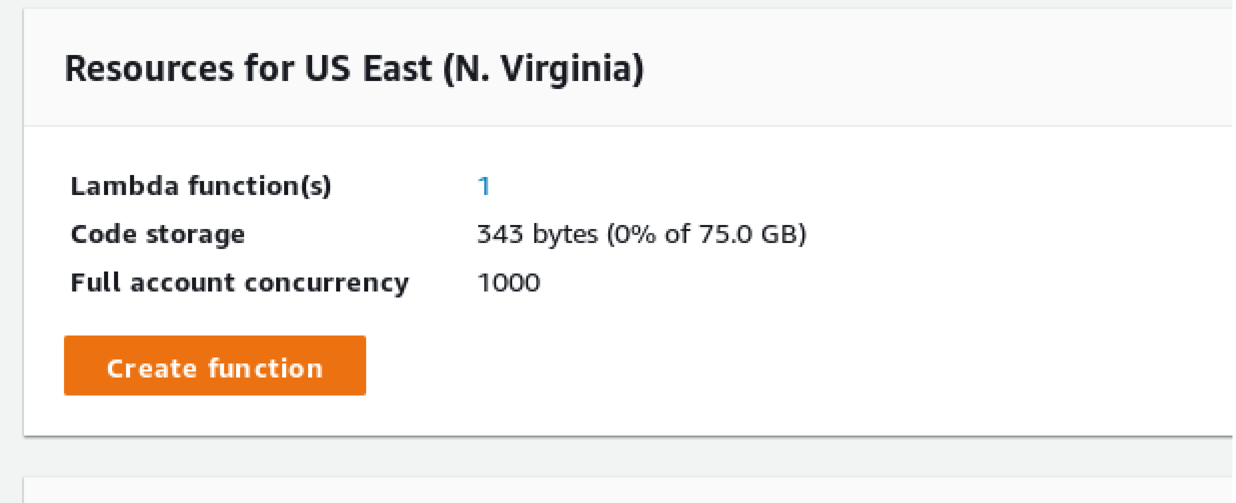
```
Time (UTC +00:00)   Message
2017-11-07
                    No older events found at the moment. Retry.
02:36:05            Loading function
02:36:05            START RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32 Version: $LATEST
02:36:05            value1 = value1
02:36:05            value2 = value2
02:36:05            value3 = value3
02:36:05            END RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32
02:36:05            REPORT RequestId: 672b5ba2-c364-11e7-ab94-99c32a63fd32 Duration: 0.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 21 MB
                    No newer events found at the moment. Retry.
```

Additionally, the Monitoring Tab under the test output gets you plots of usage history.



http://docs.aws.amazon.com/lambda/latest/dg/with-on-demand-https-example-configure-event-source_1.html

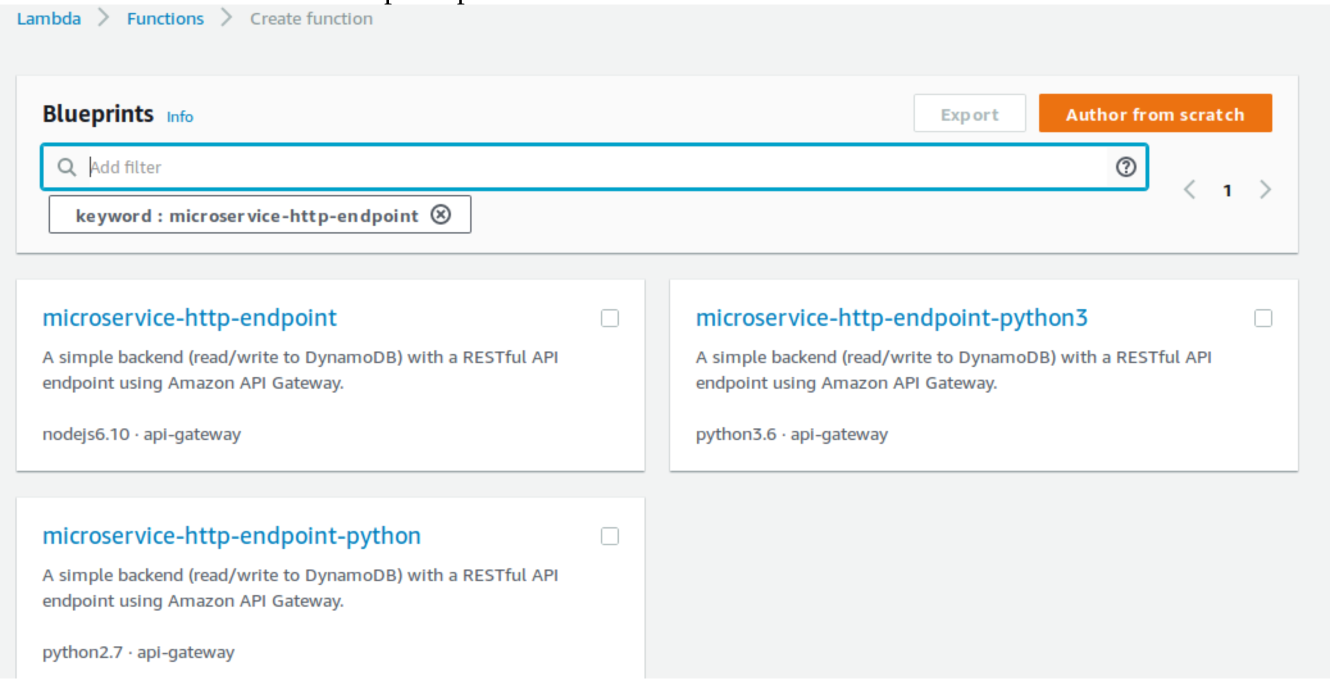
Step 3: Create a Simple Microservice using Lambda and API Gateway



Resources for US East (N. Virginia)

Lambda function(s)	1
Code storage	343 bytes (0% of 75.0 GB)
Full account concurrency	1000

Search with “microservice-http-endpoint”



Lambda > Functions > Create function
Blueprints Info
Export
Author from scratch
Add filter
keyword : microservice-http-endpoint
1

microservice-http-endpoint □
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
nodejs6.10 · api-gateway

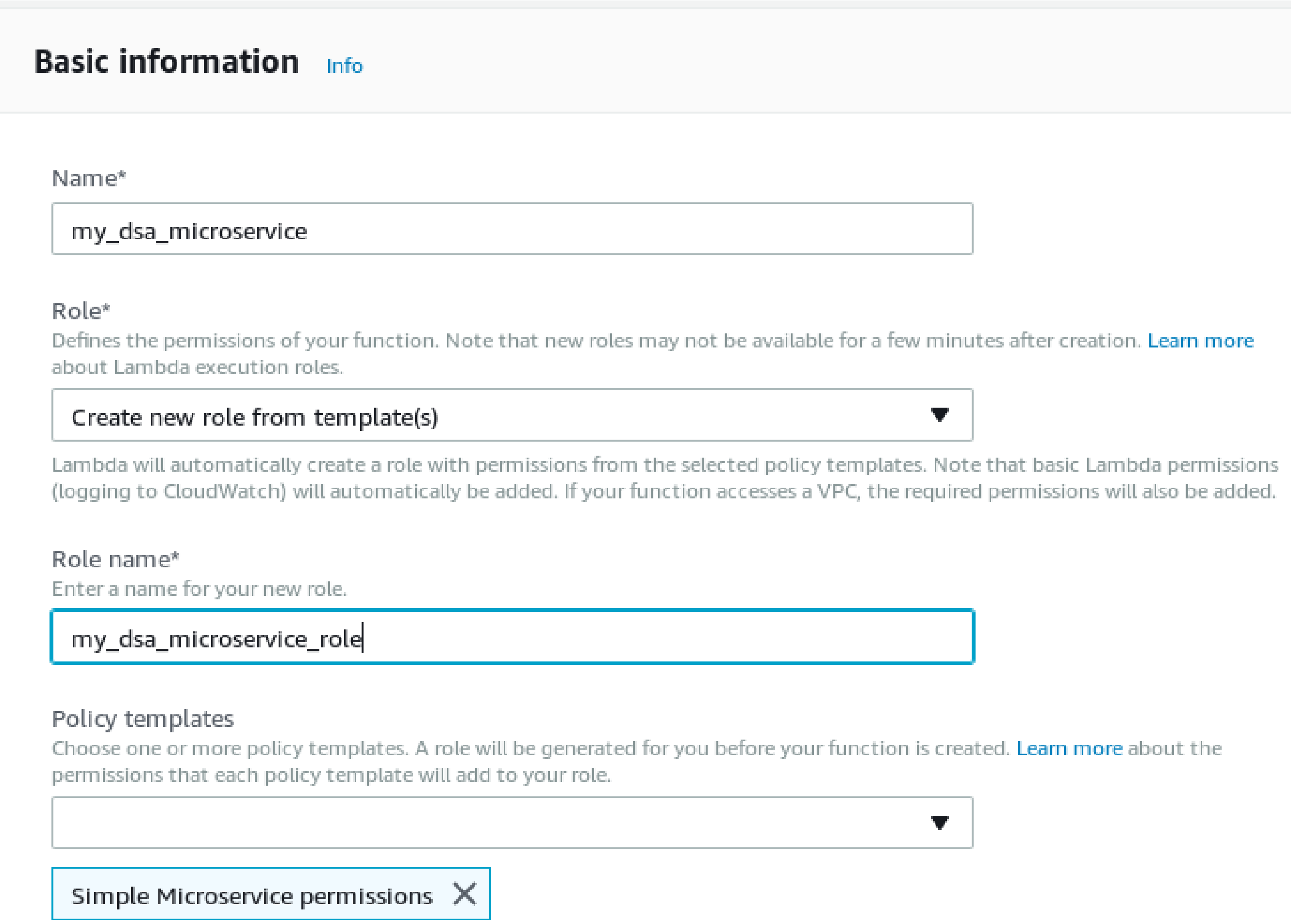
microservice-http-endpoint-python3 □
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
python3.6 · api-gateway

microservice-http-endpoint-python □
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.
python2.7 · api-gateway

DEVIATE: We are going to choose the Python 3 version.

Name your function something unique and create a unique role name that is similar

Example:



Basic information Info

Name*

my_dsa_microservice

Role*

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

Create new role from template(s)

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

Role name*

Enter a name for your new role.

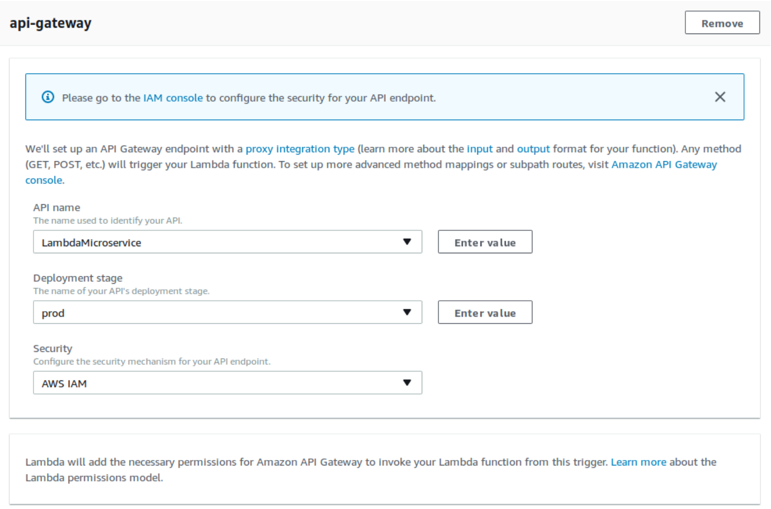
my_dsa_microservice_role

Policy templates

Choose one or more policy templates. A role will be generated for you before your function is created. Learn more about the permissions that each policy template will add to your role.

Simple Microservice permissions

Take note of the API gateway. There is a lot of customization and options here for your later real-world pipeline development.



api-gateway

Please go to the IAM console to configure the security for your API endpoint.

We'll set up an API Gateway endpoint with a proxy integration type (learn more about the input and output format for your function). Any method (GET, POST, etc.) will trigger your Lambda function. To set up more advanced method mappings or subpath routes, visit Amazon API Gateway console.

API name

The name used to identify your API.

LambdaMicroservice

Deployment stage

The name of your API's deployment stage.

prod

Security

Configure the security mechanism for your API endpoint.

AWS IAM

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Note some things in the Python Code:

1. it imports some Python Libraries,
2. instantiates the DynamoDB client object,
3. then defines a function.

```
import boto3   # You have seen this in prior labs
import json    # You have seen this in prior courses

print('Loading function')
dynamo = boto3.client('dynamodb')

# Respond forms up a response dictionary object,
#   which happens to mimic JSON syntax
def respond(err, res=None):
    return {
        'statusCode': '400' if err else '200',
        'body': err.message if err else json.dumps(res),
        'headers': {
            'Content-Type': 'application/json',
        },
    }
```

Then it defines the actual body of the request handler:

… Go to next page

I have added some extra comments into the code below …

```
def lambda_handler(event, context):
    # ...

    # These are the relevant HTTP client actions
    operations = {
        'DELETE': lambda dynamo, x: dynamo.delete_item(**x),
        'GET': lambda dynamo, x: dynamo.scan(**x),
        'POST': lambda dynamo, x: dynamo.put_item(**x),
        'PUT': lambda dynamo, x: dynamo.update_item(**x),
    }

    # Get the operation, typically referred to as “request method”
    operation = event['httpMethod']

    # Test if this is a known event operation
    if operation in operations:
        # if the request has query string parameters,
        #    load them into the payload variable
        payload = event['queryStringParameters'] if operation ==
'GET' else json.loads(event['body'])
        # otherwise, load the body of the request into the payload
        return respond(None, operations[operation](dynamo, payload))
    else:
        # If not an allowed operation, return error message
        return respond(ValueError('Unsupported method
"{}"'.format(operation)))
```

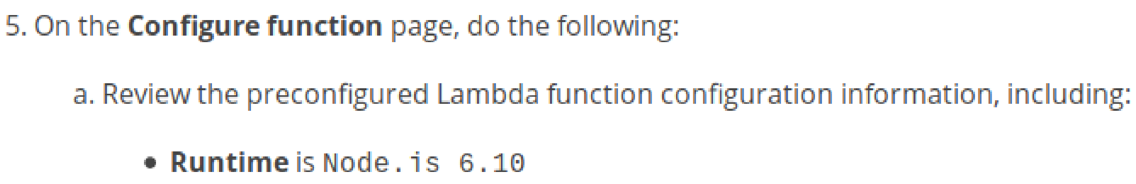
Wait a moment:

What are we doing with the data?

What we have done is created a dictionary where a key maps to a Python lambda function. Here, *lambda* means anonymous, not AWS Lambda.

```
Operations = {
          # Delete key returns an anonymous function
          #   which uses the dynamo object above, invokes delete
        'DELETE': lambda dynamo, x: dynamo.delete_item(**x),
          #   etc., etc., etc. ...
        'GET': lambda dynamo, x: dynamo.scan(**x),
        'POST': lambda dynamo, x: dynamo.put_item(**x),
        'PUT': lambda dynamo, x: dynamo.update_item(**x),
    }
```

Ignore the configure function stuff related to Node

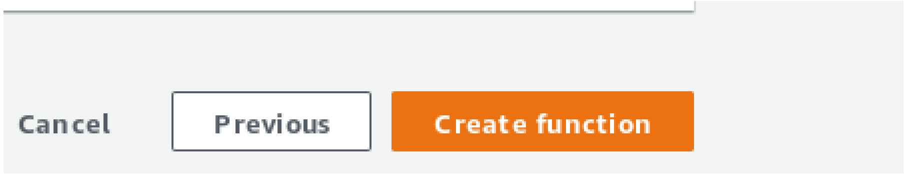


5. On the **Configure function** page, do the following:

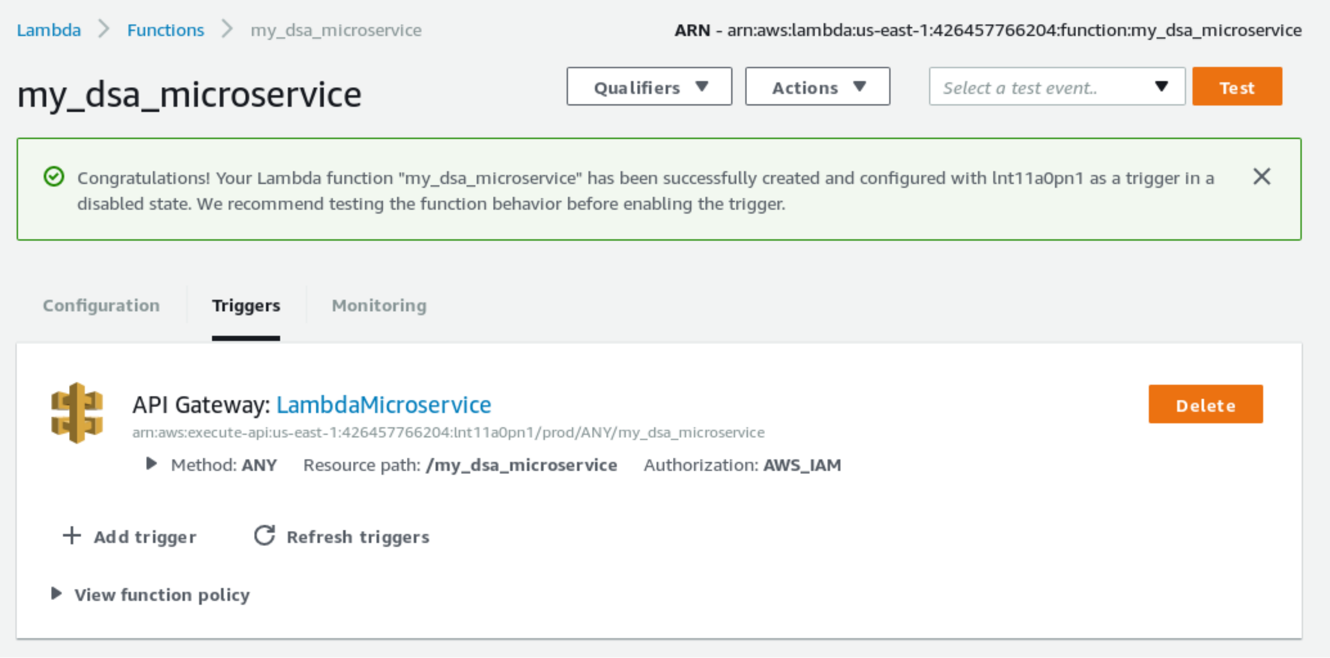
   a. Review the preconfigured Lambda function configuration information, including:

   - **Runtime** is `Node.js 6.10`

Finally, Create the function



Now we have defined our Microservice function



ARN - arn:aws:lambda:us-east-1:426457766204:function:my_dsa_microservice

my_dsa_microservice

Congratulations! Your Lambda function "my_dsa_microservice" has been successfully created and configured with lnt11a0pn1 as a trigger in a disabled state. We recommend testing the function behavior before enabling the trigger.

API Gateway: LambdaMicroservice

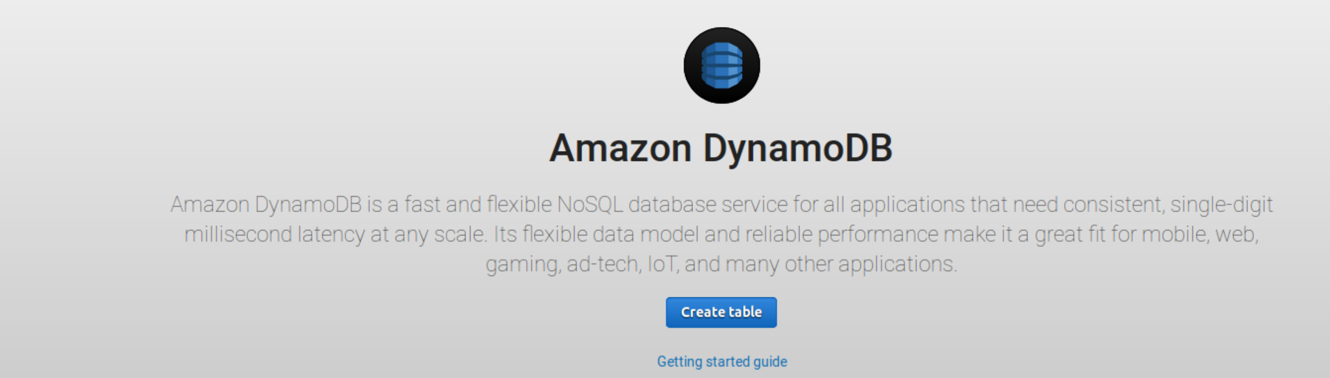
arn:aws:execute-api:us-east-1:426457766204:lnt11a0pn1/prod/ANY/my_dsa_microservice

▶ Method: **ANY** Resource path: **/my_dsa_microservice** Authorization: **AWS_IAM**

IN A NEW BROWSER TAB:

CREATE A DYNAMO DB Table

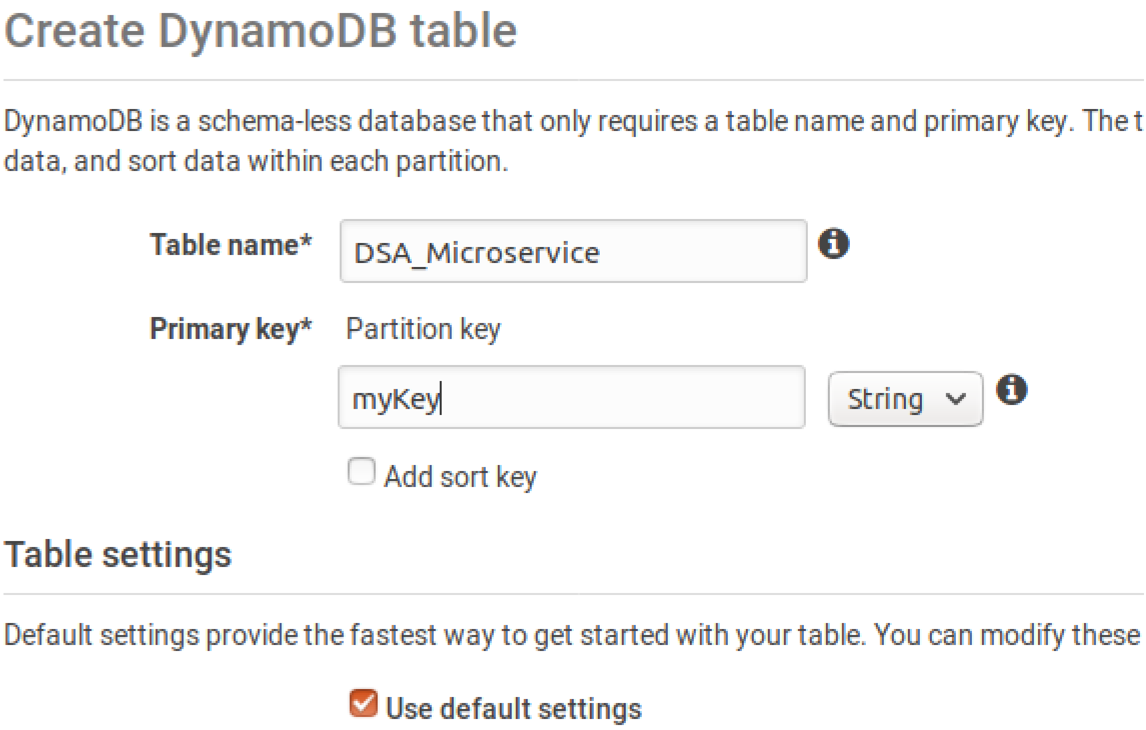
http://docs.aws.amazon.com/lambda/latest/dg/with-ddb-configure-ddb.html#with-ddb-create-buckets



Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, IoT, and many other applications.

We will keep this simple for now:
Table: DSA_Microservice
Key: myKey



Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The t
data, and sort data within each partition.

Table name* DSA_Microservice

Primary key* Partition key

myKey String

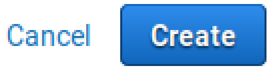
☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these

☑ Use default settings

Then click the Create button.



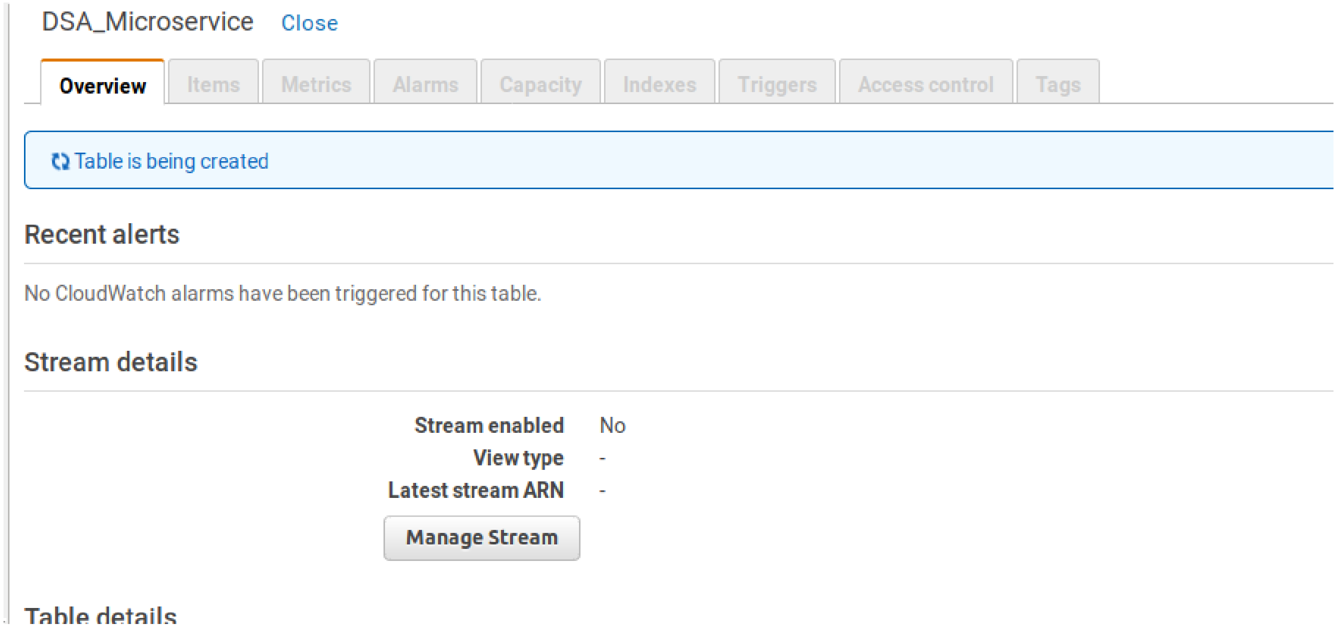

DSA_Microservice Close

Table is being created

Recent alerts

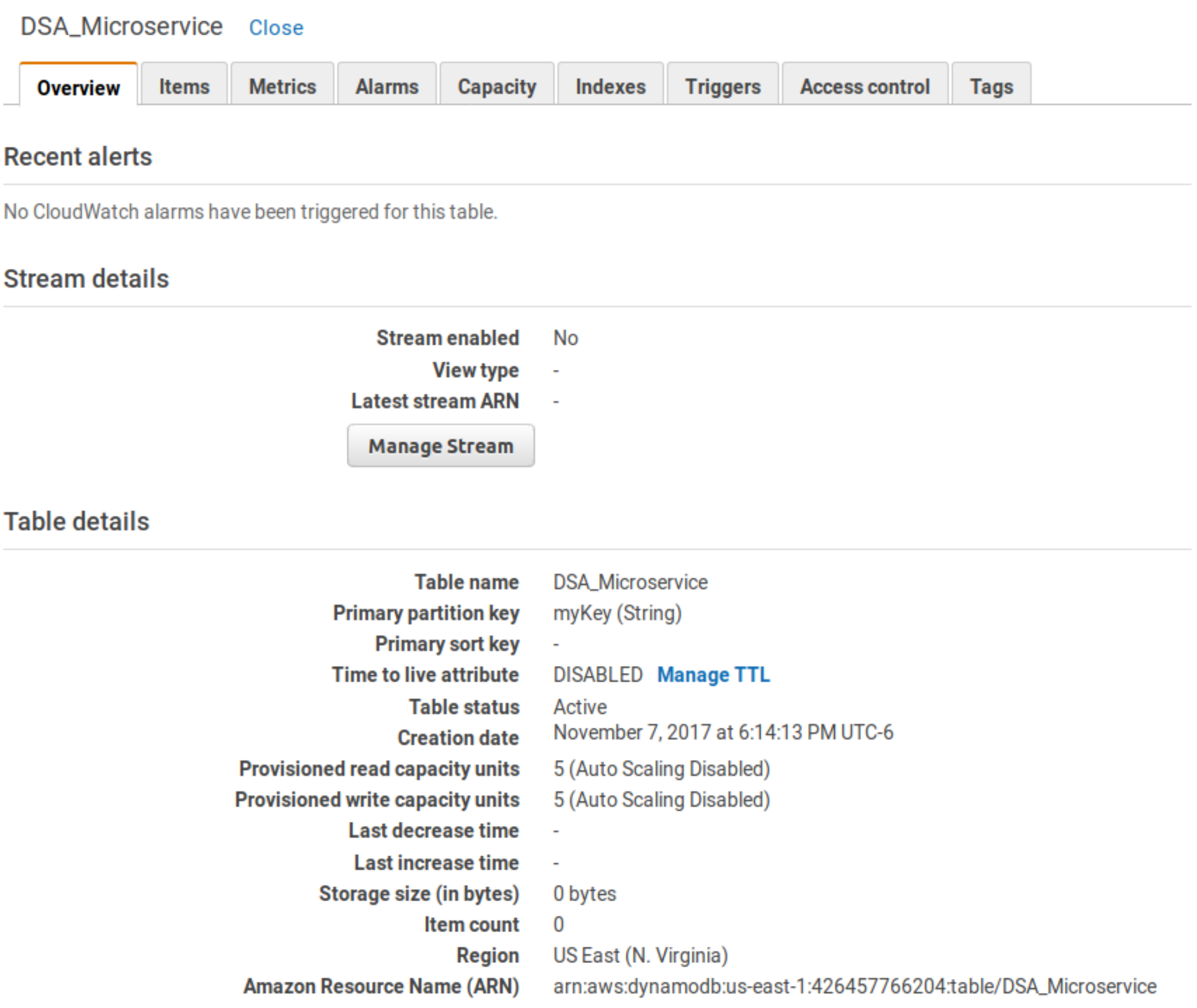
No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

Table details

Once it is done being created:



Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

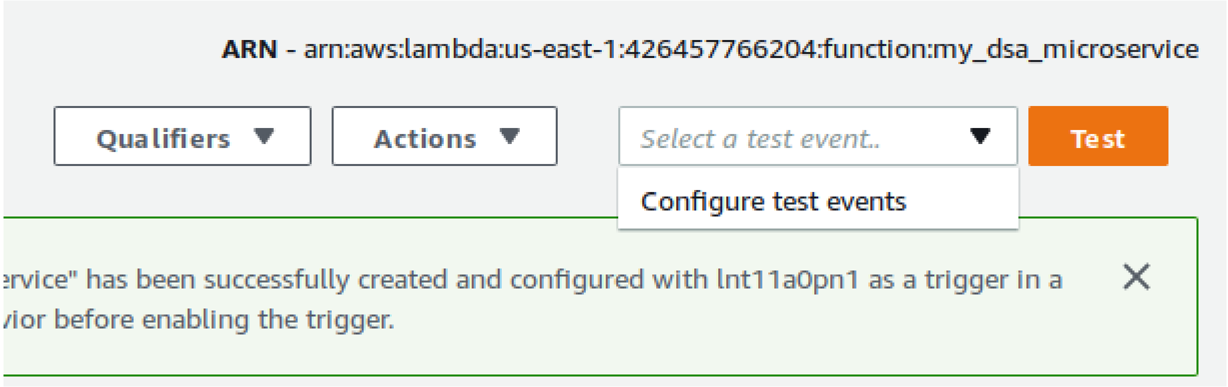
Table details

Table name	DSA_Microservice
Primary partition key	myKey (String)
Primary sort key	-
Time to live attribute	DISABLED Manage TTL
Table status	Active
Creation date	November 7, 2017 at 6:14:13 PM UTC-6
Provisioned read capacity units	5 (Auto Scaling Disabled)
Provisioned write capacity units	5 (Auto Scaling Disabled)
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0
Region	US East (N. Virginia)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-east-1:426457766204:table/DSA_Microservice

http://docs.aws.amazon.com/lambda/latest/dg/with-on-demand-https-example-configure-event-source-test-end-to-end_1.html

Step 3.2: Test Sending an HTTPS Request

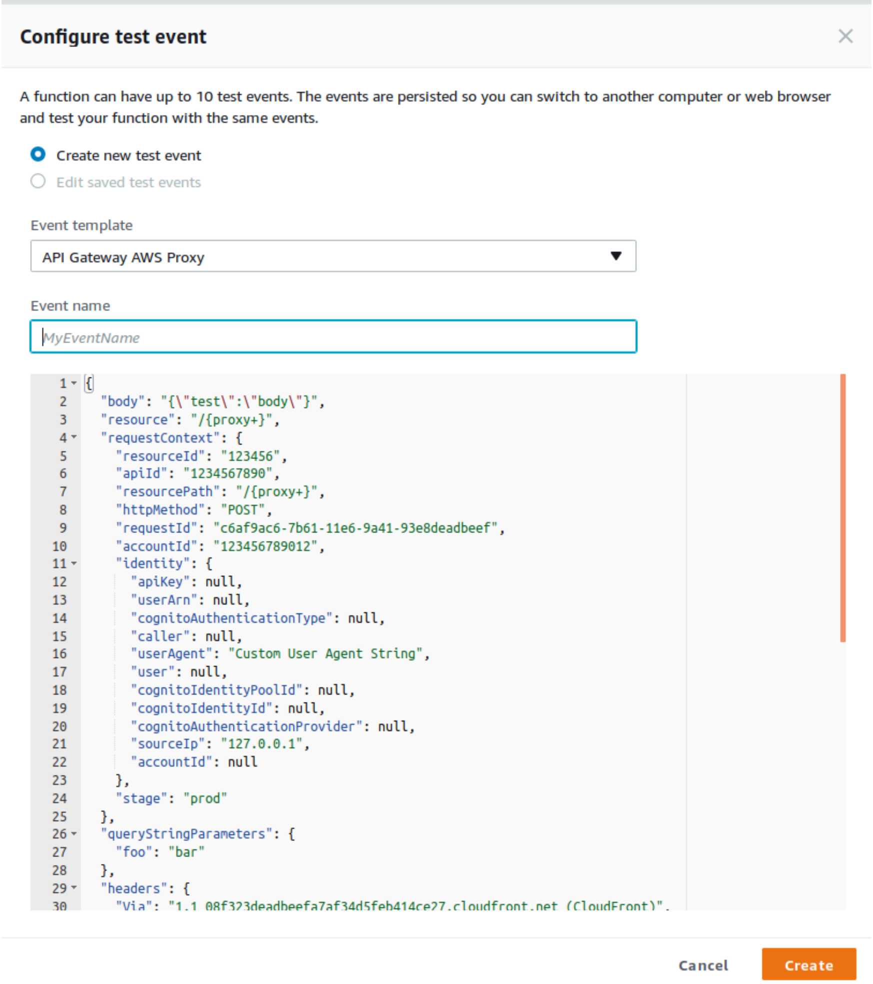
Configure a test



ARN - arn:aws:lambda:us-east-1:426457766204:function:my_dsa_microservice

ervice" has been successfully created and configured with lnt11a0pn1 as a trigger in a
vior before enabling the trigger.

This brings you to the large JSON text



A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

● Create new test event
○ Edit saved test events

Event template: API Gateway AWS Proxy

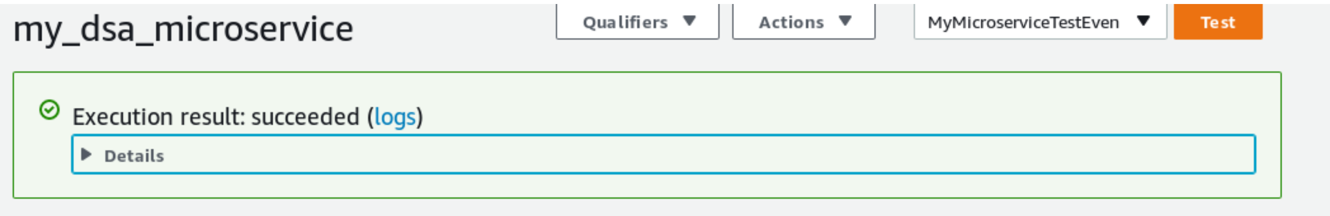
Event name: MyEventName

```
{
  "body": "{\"test\":\"body\"}",
  "resource": "/{proxy+}",
  "requestContext": {
    "resourceId": "123456",
    "apiId": "1234567890",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "accountId": "123456789012",
    "identity": {
      "apiKey": null,
      "userArn": null,
      "cognitoAuthenticationType": null,
      "caller": null,
      "userAgent": "Custom User Agent String",
      "user": null,
      "cognitoIdentityPoolId": null,
      "cognitoIdentityId": null,
      "cognitoAuthenticationProvider": null,
      "sourceIp": "127.0.0.1",
      "accountId": null
    },
    "stage": "prod"
  },
  "queryStringParameters": {
    "foo": "bar"
  },
  "headers": {
    "Via": "1.1 08f323deadbeefa7af34d5feb414ce27.cloudfront.net (CloudFront)",
```

We must update the test JSON to match our DynamoDB table name (**DSA_Microservice**) and key name (**myKey**). The other parameters are just to get some additional data in the table.

```
{
  "httpMethod": "GET",
  "queryStringParameters": {
    "TableName": "DSA_Microservice"
  }
}
```

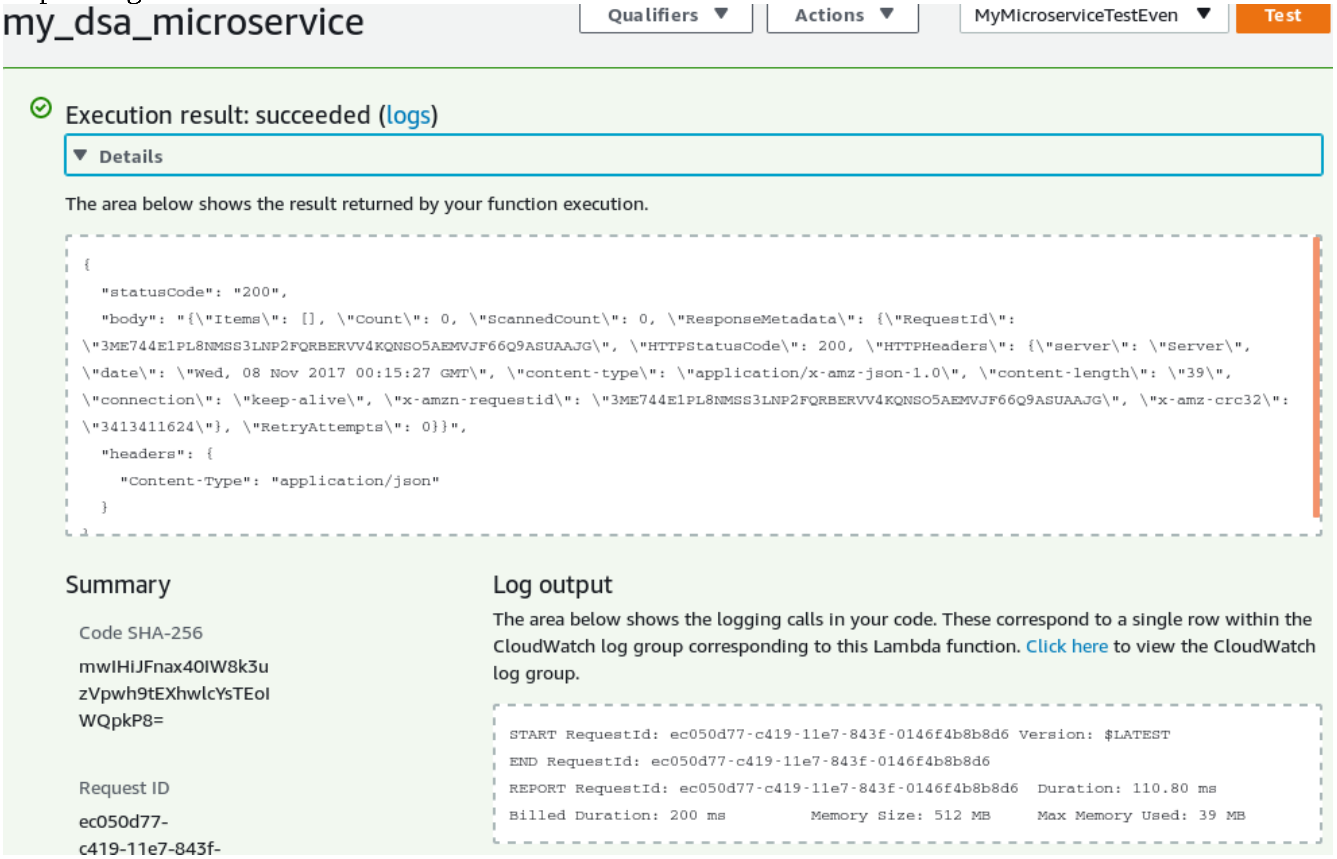
The Save the test, then click Test to execute it.



Execution result: succeeded (logs)

▶ Details

Expanding the details:



Execution result: succeeded (logs)

▼ Details

The area below shows the result returned by your function execution.

```
{
  "statusCode": "200",
  "body": "{\"Items\": [], \"Count\": 0, \"ScannedCount\": 0, \"ResponseMetadata\": {\"RequestId\":
\"3ME744E1PL8NMSS3LNP2FQRBERVV4KQNSO5AEMVJF66Q9ASUAAJG\", \"HTTPStatusCode\": 200, \"HTTPHeaders\": {\"server\": \"Server\",
\"date\": \"Wed, 08 Nov 2017 00:15:27 GMT\", \"content-type\": \"application/x-amz-json-1.0\", \"content-length\": \"39\",
\"connection\": \"keep-alive\", \"x-amzn-requestid\": \"3ME744E1PL8NMSS3LNP2FQRBERVV4KQNSO5AEMVJF66Q9ASUAAJG\", \"x-amz-crc32\":
\"3413411624\"}, \"RetryAttempts\": 0}}",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Summary

Code SHA-256

mwIHiJFnax40IW8k3u
zVpwh9tEXhwlcYsTEoI
WQpkP8=

Request ID

ec050d77-
c419-11e7-843f-

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId: ec050d77-c419-11e7-843f-0146f4b8b8d6 Version: $LATEST
END RequestId: ec050d77-c419-11e7-843f-0146f4b8b8d6
REPORT RequestId: ec050d77-c419-11e7-843f-0146f4b8b8d6  Duration: 110.80 ms
Billed Duration: 200 ms         Memory Size: 512 MB     Max Memory Used: 39 MB
```

Note: Items = [] in the body response! This is an empty list.

Lambda Code revisited!

```
operations = {

      'DELETE': lambda dynamo, x: dynamo.delete_item(**x),

      'GET': lambda dynamo, x: dynamo.scan(**x),

      'POST': lambda dynamo, x: dynamo.put_item(**x),

      'PUT': lambda dynamo, x: dynamo.update_item(**x),

  }
```

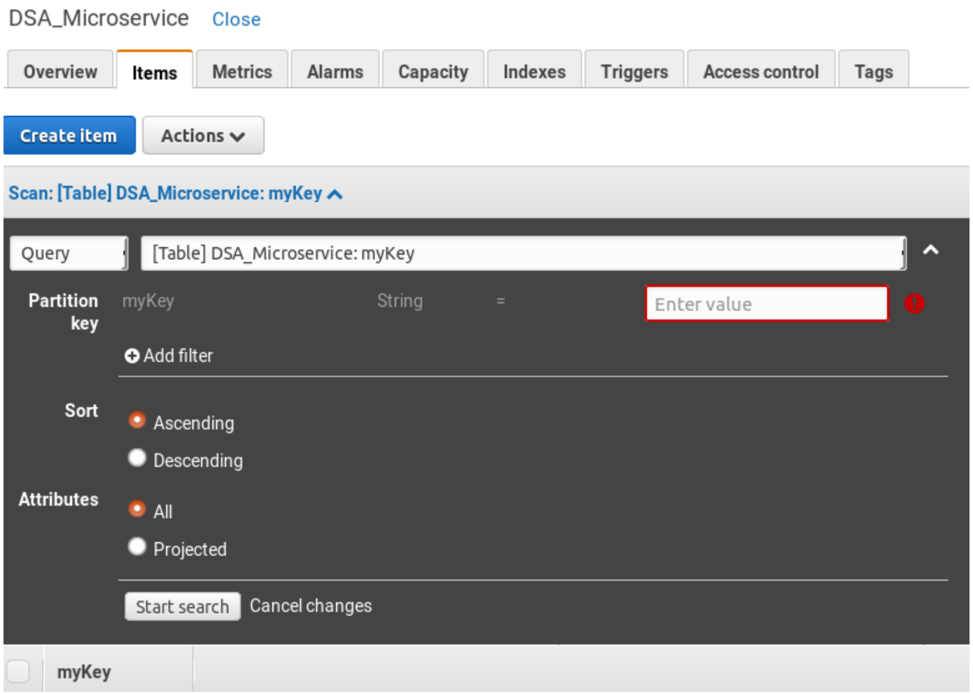
Review this link for actions on the DynamoDB: http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Operations_Amazon_DynamoDB.html

Notice that we are doing one of four actions:

1. delete_item (x)
2. scan (x)
3. put_item (x)
4. update_item (x)

So, the GET request (test code: `"httpMethod": "GET") says to scan for ( x )`

Add some data into the table:



Scan: [Table] DSA_Microservice: myKey

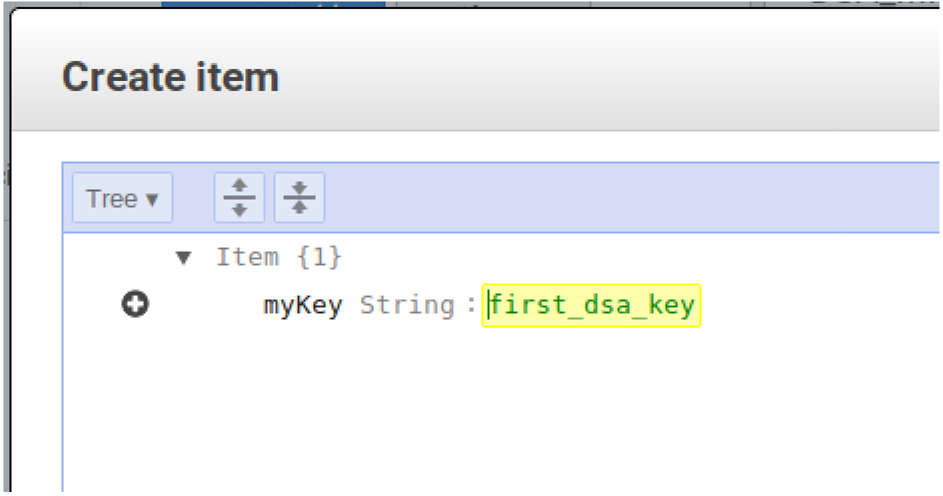
Query | [Table] DSA_Microservice: myKey

Partition key: myKey String = Enter value

Sort: ● Ascending ○ Descending

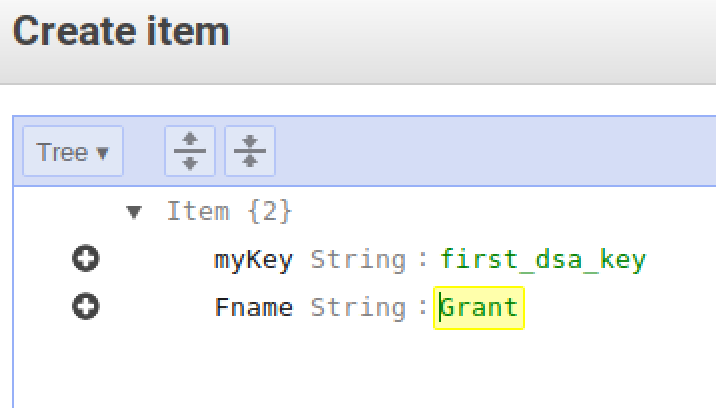
Attributes: ● All ○ Projected

Clicking the Create Item button, we can add some data



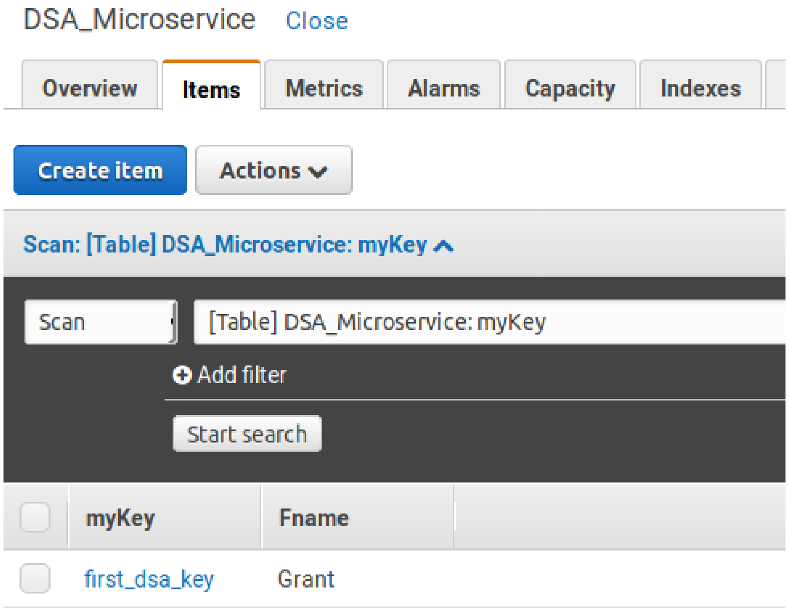
```
▼ Item {1}
    myKey String : first_dsa_key
```

Clicking the Plus-Sign, then append:



```
▼ Item {2}
    myKey String : first_dsa_key
    Fname String : Grant
```

Then, Save this new record!



Scan [Table] DSA_Microservice: myKey

□	myKey	Fname
□	first_dsa_key	Grant

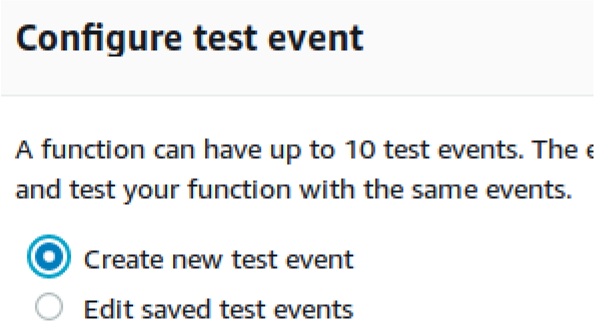
Now we can re-run our Lambda function test and we get the new row back!!!

```
{
  "statusCode": "200",
  "body": "{\"Items\": [{\"Fname\": {\"S\": \"Grant\"}, \"myKey\":
{\"S\": \"first_dsa_key\"}}], \"Count\": 1, \"ScannedCount\":
1, \"ResponseMetadata\":
{\"RequestId\": \"O2JRAOUCUPBTEHSKNEGRV6OR0FVV4KQNSO5AEMVJF66Q9ASUAAJG\", \"HTTPSt
atusCode\": 200, \"HTTPHeaders\": {\"server\": \"Server\", \"date\": \"Wed, 08 Nov
2017 00:41:27 GMT\", \"content-type\": \"application/x-amz-json-1.0\", \"content-
length\": \"92\", \"connection\": \"keep-alive\", \"x-amzn-
requestid\": \"O2JRAOUCUPBTEHSKNEGRV6OR0FVV4KQNSO5AEMVJF66Q9ASUAAJG\", \"x-amz-
crc32\": \"3103740141\"}, \"RetryAttempts\": 0}}",
  "headers": {
    "Content-Type": "application/json"
  }
```

So what does it take to add data to our table?

- 'POST': lambda dynamo, x: dynamo.put_item(**x),

Now we will create a new test event:



Configure test event

A function can have up to 10 test events. The e
and test your function with the same events.

● Create new test event

○ Edit saved test events

Event Name: MyDSAMicroserviceAddData
Body: (Bold change, Blue added)

```
{
  "httpMethod": "POST",
  "body": "{\"TableName\":\"DSA_Microservice\",\"Item\":{\"myKey\":
{\"S\":\"my_second_dsa\"}}}"
}
```

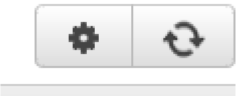
Then Save, and test!!!

NOTE: In the body element, we have written a double quoted escaped version of

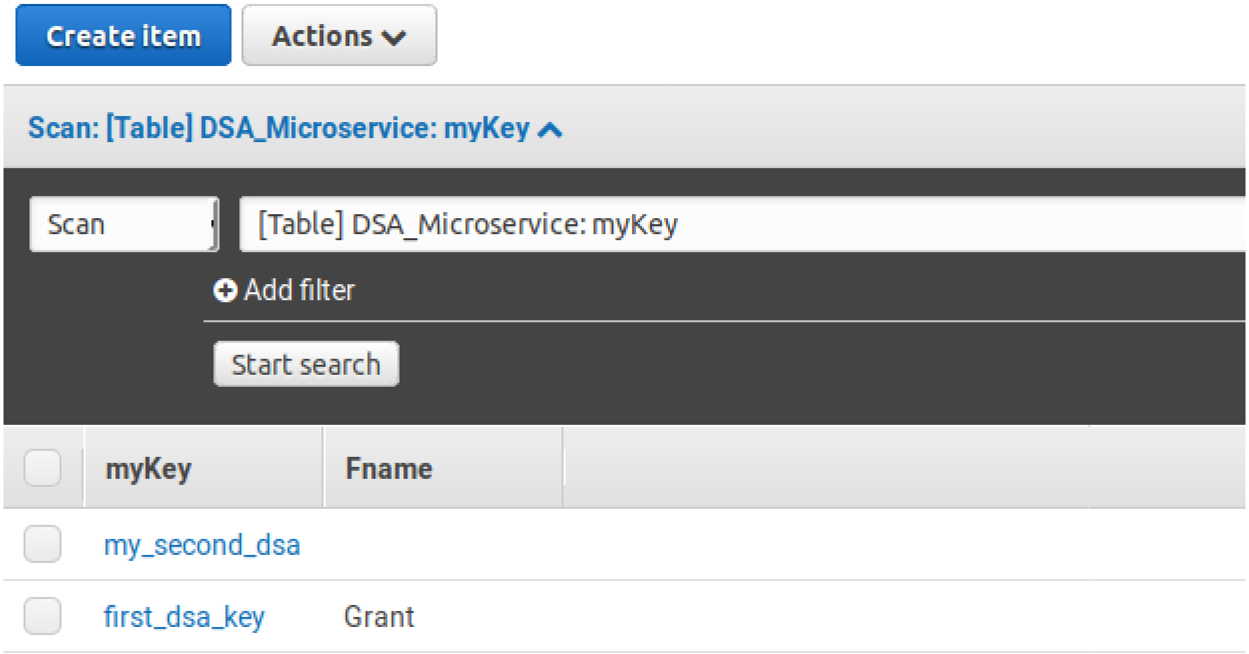
```
{
  "TableName":"DSA_Microservice",
  "Item":{
       "myKey":{"S":"my_second_dsa"}
  }
}
```

The {“S”:”my_second_dsa”} is the JSON format of saying that myKey is a (S)tring, “my_second_dsa”

If we go back to our DynamoDB table and refresh!



We have the new data row!


Create item
Actions
Scan: [Table] DSA_Microservice: myKey
Scan
[Table] DSA_Microservice: myKey
Add filter
Start search

	myKey	Fname
□	my_second_dsa	
□	first_dsa_key	Grant

That is it!

In this tutorial, you created two different lambda functions using Python 3. The latter of the two is an example of processing a request to load data into the NoSQL AWS, DynamoDB.

Image how this can be incorporated into data processing pipelines and integrated with S3 bucket events, DynamoDB or Redshift or in a model S3 → Lambda → S3 → ???