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.

NOTE: add your pawprint to the function's name, roles, tables, …

Step 1: Set Up an AWS Account and the AWS CLI
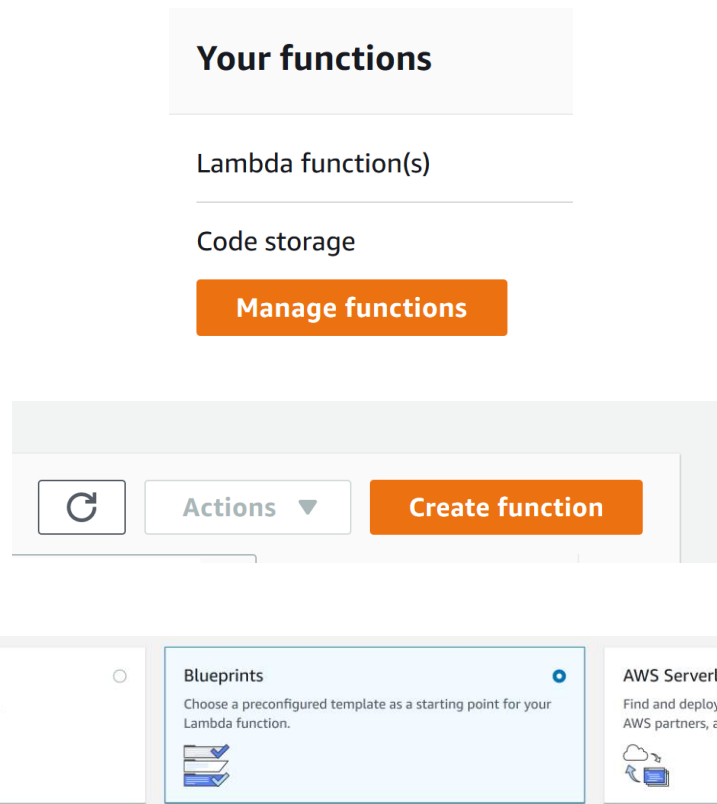
(Done in prior labs)

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

Choose manage functions >> create function>> blueprints

Your functions

Lambda function(s)

Code storage

Manage functions

C   Actions ▼   **Create function**

Author from scratch
Start with a simple "hello world" example.

Blueprints
Choose a preconfigured template as a starting point for your Lambda function.

AWS Serverless Application Repository
Find and deploy serverless applications published by AWS, AWS partners, and other developers.

Blueprints Info

Blueprint name : hello-world-python

hello-world-python

hello-world-python3

Blueprints Info

Add filter

Blueprint name : hello-world-python ⊗

hello-world-python	○	hello-world-python3	●
A starter AWS Lambda function.		A starter AWS Lambda function.	
python2.7		python3.6	

SELECT the "hello-world-**python3**" blueprint >> configure

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

Basic information Info

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. **Learn more** about Lambda execution roles.

> Create a new role from one or more templates. ▼

Lambda automatically creates a role with permissions from the selected policy templates. Basic Lambda permissions (such as logging to Amazon CloudWatch) are automatically added. If your function accesses a VPC, the required permissions are also added.

Role name
Enter a name for your new role.

> my_first_dsa_lambda_role

> ⓘ This new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

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.

> ▼

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.

Configuration   Triggers   Monitoring

▼ Function code

Code entry type	Runtime	Handler Info
Edit code inline ▼	Python 3.6 ▼	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

Qualifiers ▼   Actions ▼   Select a test event.. ▼   **Test**

...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

Cancel   **Create**

We now have a test event

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

| Qualifiers ▼ | Actions ▼ | HelloTest ▼ | Test |

Click Test Again:

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

my_first_dsa_lambda

| Qualifiers ▼ | Actions ▼ | HelloTest ▼ | Test |

⊘ Execution result: succeeded (logs)

▼ Details

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

```
"value1"
```

Summary

Code SHA-256

7ddck7QZ/3YKp8f
/QT1Qifz46AR+l8OjZ2
P+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:

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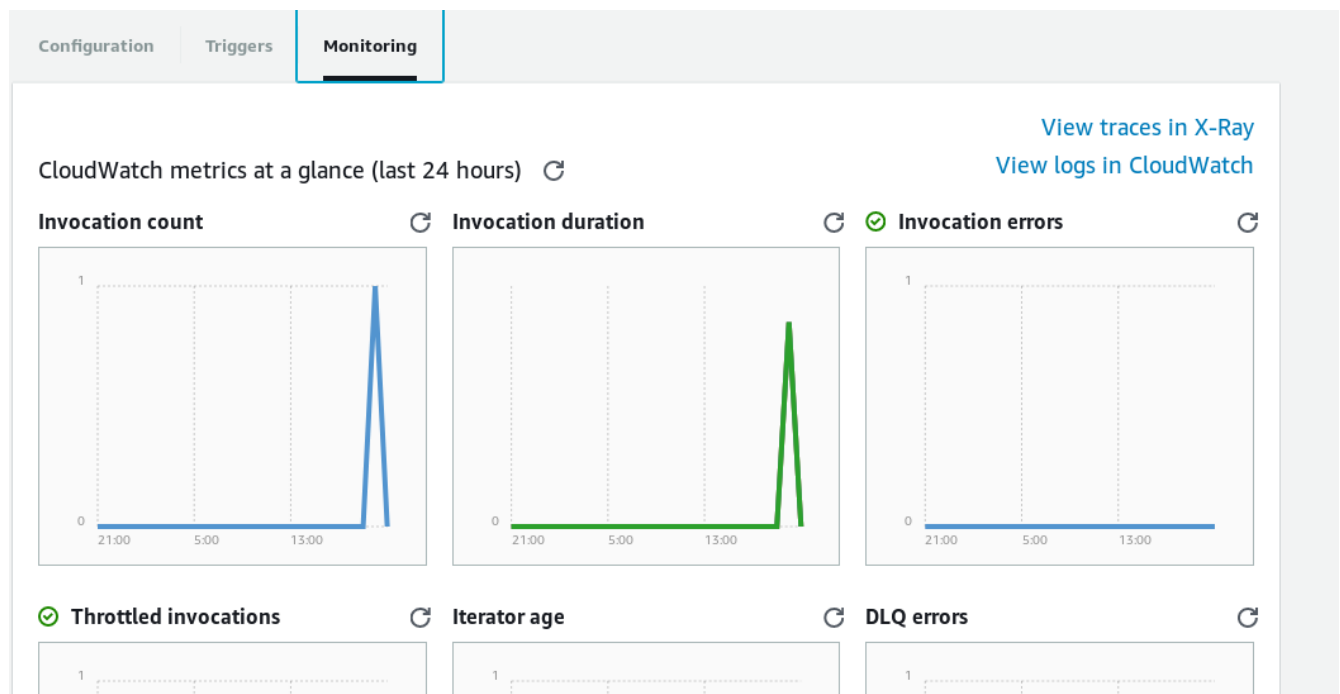
http://docs.aws.amazon.com/lambda/latest/dg/with-on-demand-https-example-configure-event-source_1.html

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

Search with "microservice-http-endpoint"

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 a new role from one or more templates. ▼

Lambda automatically creates a role with permissions from the selected policy templates. Basic Lambda permissions (such as logging to Amazon CloudWatch) are automatically added. If your function accesses a VPC, the required permissions are also added.

Role name

Enter a name for your new role.

my_dsa_microservice_role

ⓘ This new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

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.
For this lab hit remove to remove it.

api-gateway   Remove

ⓘ 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 ▼   **Enter value**

Deployment stage
The name of your API's deployment stage.

prod ▼   **Enter value**

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.

```python
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 …

```python
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.

```python
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),
    }
```

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

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

- **Runtime** is Node.is 6.10

Finally, Create the function

Cancel   Previous   **Create function**
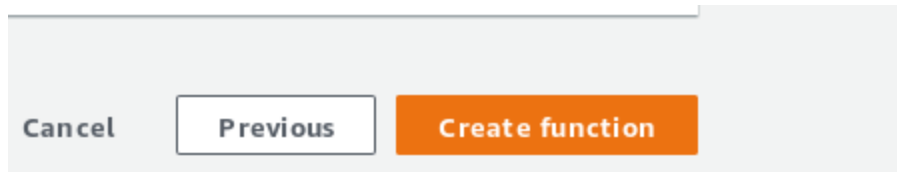
Now we have defined our Microservice function

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

my_dsa_microservice

Qualifiers ▼   Actions ▼   Select a test event.. ▼   **Test**

⊘ 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.

Configuration   **Triggers**   Monitoring

API Gateway: LambdaMicroservice   **Delete**
arn:aws:execute-api:us-east-1:426457766204:lnt11a0pn1/prod/ANY/my_dsa_microservice
▶ Method: **ANY**   Resource path: **/my_dsa_microservice**   Authorization: **AWS_IAM**

╋ Add trigger   ⟳ Refresh triggers

▶ View function policy

CREATE A DYNAMO DB Table

https://console.aws.amazon.com/dynamodb/home?region=us-east-1#
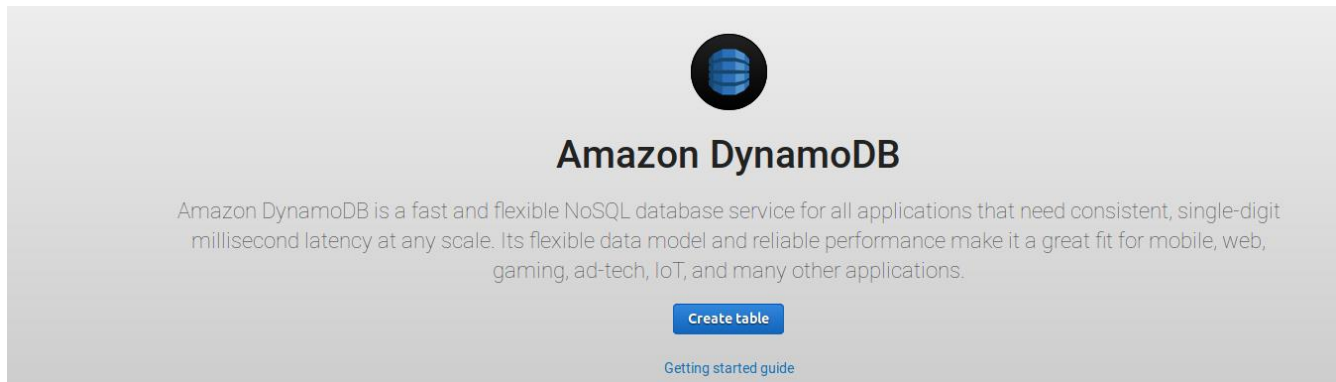
We will keep this simple for now:
Table: DSA_Microservice
Key: myKey

Then click the Create button.

DSA_Microservice   Close

| Overview | Items | Metrics | Alarms | Capacity | Indexes | Triggers | Access control | Tags |

🔄 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	-

Manage Stream

Table details

Once it is done being created:

DSA_Microservice   Close

| Overview | Items | Metrics | Alarms | Capacity | Indexes | Triggers | Access control | Tags |

Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

Manage Stream

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

This brings you to the large JSON text

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.

Expanding the details:

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:

Clicking the Create Item button, we can add some data

Create item

Tree ▾

▼ Item {1}

⊕   myKey String : first_dsa_key

Clicking the Plus-Sign, then append:

Create item

Tree ▾

▼ Item {2}

⊕   myKey String : first_dsa_key
⊕   Fname String : Grant

Then, Save this new record!

DSA_Microservice   Close

| Overview | **Items** | Metrics | Alarms | Capacity | Indexes |

Create item   Actions ▾

Scan: [Table] DSA_Microservice: myKey ⌃

Scan | [Table] DSA_Microservice: myKey

⊕ Add filter

Start search

	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\", \"HTTPStatusCode\": 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:

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

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

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!

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 → ???