

Lambda and Python

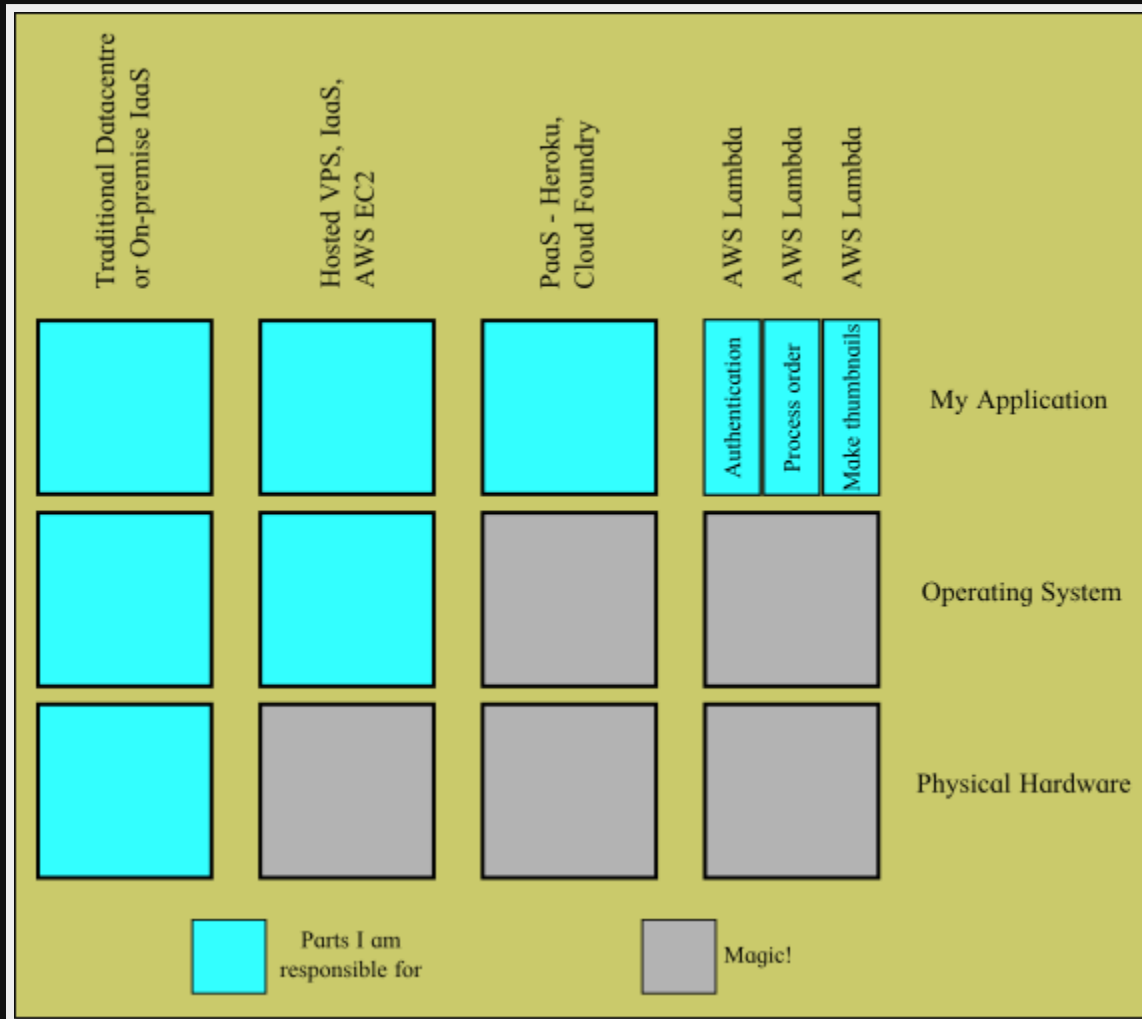
Using AWS Lambda to run Python scripts

James Mitchell



Deploying Your App

What needs to be deployed...



What is AWS Lambda?

" AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you... "

- deploy **functions** not applications
- the code is executed in **response to events** on other AWS services, such as S3, SNS, DynamoDB etc..
- or executed by the Amazon **API Gateway** (⇐ choose this one!)
- only supports Java and Node.js 😞

Adding some Python

We can make it work by getting a node.js script to execute the Python script!

- [WillyG - Python on AWS Lambda](#)
- [Tim Wagner - Using Python in an AWS Lambda Function](#)
- [Eric Hammond provides a wrapper function](#)

1 - Hello World in Python

- accept a JSON object from the wrapper, write JSON to stdout
- use virtualenv to bundle in python and extra libraries

```
import sys
import json

def main(event):
    name = event.get('name', 'Mr. Eastwood')
    response = dict(greeting='Hello', name=name)
    print(json.dumps(response))

if __name__ == '__main__':
    argv = sys.argv[1:]
    event = json.loads(argv[0])
    main(event)
```

[sample code on Github](#)

2.1 - Lambda Wrapper Script

- This script runs our Python script
- Communication is via "stringified" JSON

```
var spawn = require('child_process').spawn;
exports.handler = function(event, context) {
  var response = {};
  var error = null;
  var child = spawn('venv/bin/python', [
    'lambda-function.py',
    JSON.stringify(event, null, 2)
  ]);
  child.stdout.on('data', function (data) {
    console.log("stdout:\n"+data);
    response = JSON.parse(data);
  });
  child.stderr.on('data', function (data) {
    console.log("stderr:\n"+data);
    error = { error: true, message: data.toString('utf8') };
  });
  child.on('close', function (code) {
    if (error !== null ) { context.fail(error); }
    else {context.succeed(response); }
  });
};
```

2.1 - Lambda Function

- upload a ZIP of the directory to S3 bucket
- ...this takes a while to upload 39Mb of Python
- make an IAM role for executing the function
- create the Lambda function

```
$ aws lambda create-function \  
--region us-east-1 \  
--function-name helloworld \  
--code S3Bucket=maungawhau.lambda,S3Key=hello-lambda.zip \  
--role arn:aws:iam::569584872835:role/lambda_basic_execution \  
--handler lambda-function-wrapper.handler \  
--runtime nodejs
```

2.3 - AWS Lambda Console

Once it is running, use the console to check logs and number of calls.

The screenshot shows the AWS Lambda console interface for a function named 'helloworld'. The top navigation bar includes 'Test' and 'Actions' buttons, and the ARN 'arn:aws:lambda:us-east-1:569584872835:function:helloworld'. The 'Monitoring' tab is selected, displaying 'CloudWatch metrics at a glance (last 24 hours)'. This section contains four line graphs: 'Invocation count' (peaking at 12), 'Invocation duration' (peaking at 700ms), 'Invocation errors' (peaking at 10), and 'Throttled invocations' (peaking at 1). Below the metrics is a green checkmark indicating 'Execution result: succeeded (logs)'. The execution result is shown as a JSON object:

```
{ "greeting": "Hello", "name": "Mr. Eastwood" }
```

. The 'Summary' section lists: Request ID 7892fb71-458a-11e5-b20e-93b71eaae044, Duration 455.72 ms, Billed duration 500 ms, and Resources configured 128 MB. The 'Log output' section shows a single log entry:

```
START RequestId: 7892fb71-458a-11e5-b20e-93b71eaae044 2015-08-18T09:21:15.087Z 7892fb71-458a-11e5-b20e-93b71eaae044 stdout: {"greeting": "Hello", "name": "Mr. Eastwood"} END RequestId: 7892fb71-458a-11e5-b20e-93b71eaae044 REPORT RequestId: 7892fb71-458a-11e5-b20e-93b71eaae044 Duration: 455.72 ms Billed Duration: 500
```


5.1 – API Gateway

Configure a new API on AWS, with a resource that executes the Lambda function.

The screenshot displays the Amazon API Gateway console interface. At the top, the breadcrumb navigation shows 'Amazon API Gateway' > 'APIs' > 'Hello Python API' > 'Resources'. A 'Deploy API' button is visible in the top right of the Resources section. The left sidebar shows a tree view of resources: a root resource '/' and a sub-resource '/greeting'. Under '/greeting', there are two methods: 'GET' and 'POST', with 'POST' selected and highlighted in blue. The main content area is titled '/greeting - POST - Setup' and includes a 'Delete Method' button. Below the title, it prompts the user to 'Choose the integration point for your new method.' The 'Integration type' is set to 'Lambda Function' (selected with a radio button), with 'HTTP Proxy' as an alternative. A 'Show advanced' link is present. The 'Lambda Region' is set to 'us-east-1' in a dropdown menu. The 'Lambda Function' name is 'helloworld' in a text input field. A 'Save' button is located at the bottom right of the configuration area.

5.2 - Say “Hello” to my little friend

```
$ curl -X GET https://hostname/test/greeting  
{"name":"Mr. Eastwood","greeting":"Hello"}  
  
$ curl -X POST -d '{"name": "Clint"}' https://hostname/test/greeting  
{"name":"Clint","greeting":"Hello"}
```

5.3 - Again... with Pictures

The screenshot displays the Amazon API Gateway console interface. At the top, the navigation bar includes the Amazon API Gateway logo, the API name 'Hello Python API', and the resource path '/greeting'. The main content area is divided into two panels. The left panel, titled 'Resources', shows a tree view with the root resource '/' expanded to reveal the '/greeting' resource, which has two methods: 'POST' (highlighted) and 'GET'. A 'Deploy API' button is visible above this panel. The right panel, titled 'Method Execution /greeting - POST - Method Test', contains a 'Delete Method' button and a 'Test' button. Below the 'Test' button, the execution results are displayed: 'Request: /greeting', 'Status: 200', and 'Latency: 311 ms'. The 'Response Body' section is expanded, showing a JSON object:

```
{  "name": "Clint",  "greeting": "Hello"}
```

. Other sections like 'Request Body', 'Response Headers', and 'Logs' are collapsed.

Amazon API Gateway | APIs | Hello Python API | Resources

Resources [Deploy API](#)

← Method Execution **/greeting - POST - Method Test** [Delete Method](#)

Make a test call to your POST method. Provide the value of the input parameters for your API call.

POST » /greeting

▶ Request Body

[Test](#)

Request: /greeting
Status: 200
Latency: 311 ms

▼ Response Body

```
{  "name": "Clint",  "greeting": "Hello"}
```

▶ Response Headers

▶ Logs

Conclusions

- it's **cheap**
 - FREE - first million requests per month
 - then \$0.20 per 1 million requests
- minimal deployment hassle
 - no EC2 instance, no server maintenance
- great for a **micro-service**
 - simple endpoint
 - reacting to an event
- terrible for a Django app