

Lambda and Python

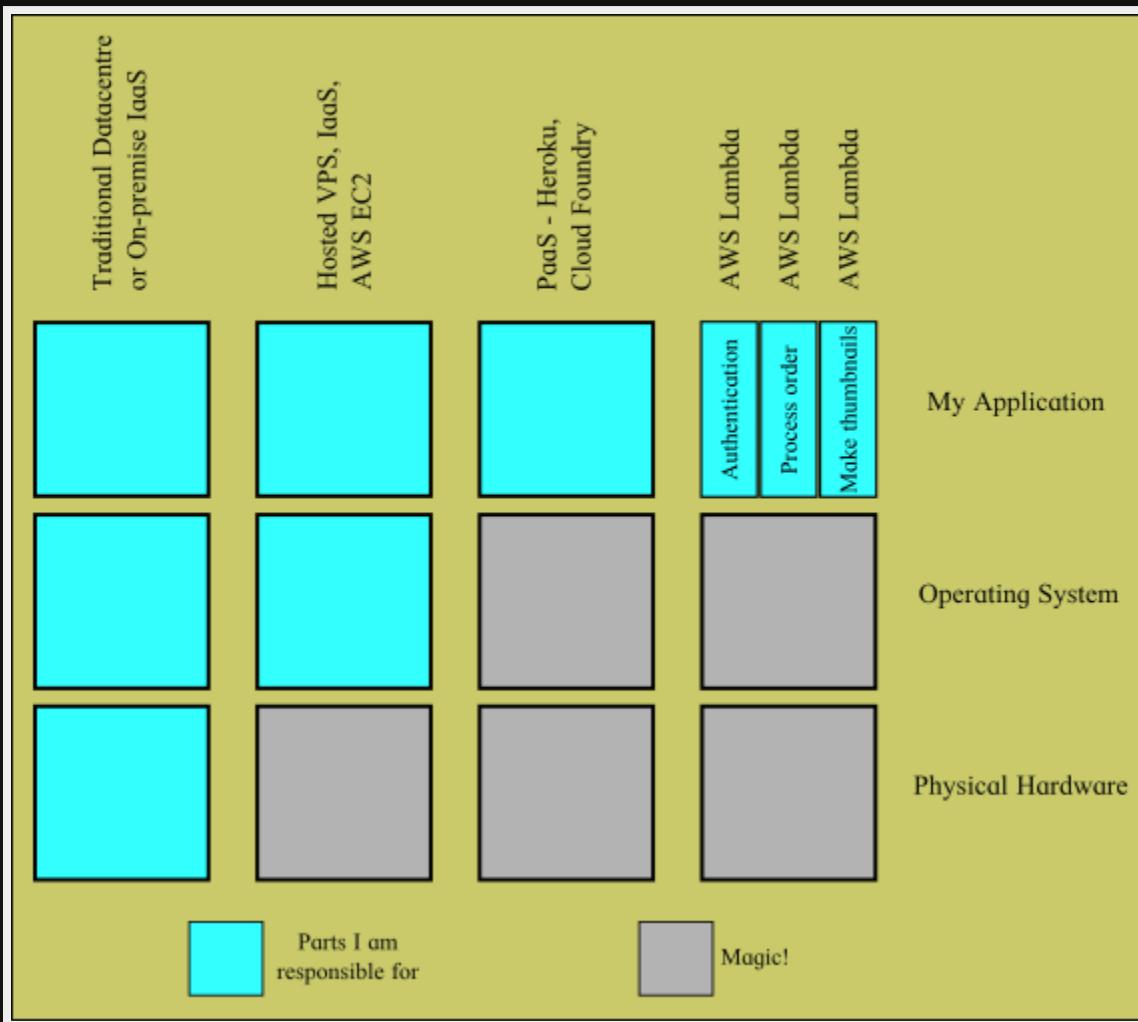
Using AWS Lambda to run Python scripts

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.

Screenshot of the AWS Lambda function configuration page for "helloworld".

The top navigation bar includes "Test", "Actions", "Code", "Configuration", "Event sources", "API endpoints", and the active "Monitoring" tab. The ARN is listed as "arn:aws:lambda:us-east-1:569584872835:function:helloworld".

The "CloudWatch metrics at a glance (last 24 hours)" section displays four charts:

- Invocation count:** Shows two sharp peaks around 10 and 12 at approximately 02:00 and 07:00 UTC.
- Invocation duration:** Shows a green line peaking around 400ms and an orange line peaking around 200ms at approximately 07:00 UTC.
- Invocation errors:** Shows a single peak reaching 10 at approximately 03:00 UTC.
- Throttled invocations:** Shows one invocation at approximately 07:00 UTC.

A link "View logs in CloudWatch" is available in the top right of this section.

The "Execution result: succeeded (logs)" section shows the successful execution output:

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

The "Summary" section provides the following details:

Request ID	7892fb71-458a-11e5-b20e-93b71eaae044
Duration	455.72 ms
Billed duration	500 ms
Resources configured	128 MB

The "Log output" section shows the log entries corresponding to the execution:

```
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 shows the AWS Lambda console interface for configuring a new API. The top navigation bar includes the 'Amazon API Gateway' logo, 'APIs' dropdown, the current API name 'Hello Python API', a 'Resources' dropdown, and a help icon. The main area is titled 'Resources' and shows a tree view of resources under the '/greeting' endpoint. The 'POST' method for this endpoint is selected, indicated by a blue background. The right-hand panel is titled '/greeting - POST - Setup' and contains the following configuration options:

- Choose the integration point for your new method.** (with an info icon)
- Integration type:** **Lambda Function** (selected) **HTTP Proxy**
- Show advanced** (link)
- Lambda Region:** us-east-1 (dropdown menu)
- Lambda Function:** helloworld (text input field)
- Save** (button)

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 shows the Amazon API Gateway Method Test interface. The top navigation bar includes 'Amazon API Gateway', 'APIs', 'Hello Python API', 'Resources', and a help icon. The main area is titled 'Method Execution /greeting - POST - Method Test'. On the left, the 'Resources' sidebar lists '/' and '/greeting', with '/greeting' expanded to show 'POST' and 'GET' methods, where 'POST' is highlighted. The central panel contains instructions to make a test call to the POST method, providing input parameters. It shows a successful test result for the '/greeting' endpoint:

Request: /greeting
Status: 200
Latency: 311 ms

Response Body:

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

Response Headers and Logs sections are also present.

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