Lambda Introduction

With Python and Serverless Framework


WHERE WE'RE GOING
WE WON'T
NEED SERVERS
imgflip.com

Getting Started

Do you have:

- Anaconda Python (Google: anaconda python for mac, download 3.7 version)
- NodeJS (we need npm)
- Serverless (installed by npm)
- awscli
- aws-azure-login
- Docker

Let’s Not Drink The Kool Aid Yet

Lambda is not blazing fast

Lambda does not make DevOps out of job

Lambda can be a very expensive solution

Lambda is good for things that run infrequently

Lambda is good for event driven programming with AWS integration

Just use common sense, Lambda is not a silver bullet for everything

Python Golden Rules

Indentation is everything, there’s no braces {}

Don’t mix spaces with tabs with indentation, use Python friendly editor (e.g. **atom**)

Use 4 spaces for indenting or 2

To name functions, you can use camelcase or use underscore; python people use underscore (except for class name)

Want to know more? Search PEP-8

Python Modules/Libraries

How to install modules?

```
pip install public-module1 public-module2
```

Or, you can create requirements.txt file:

```
requirements.txt:
public-module1
public-module2==version

pip install -r requirements.txt
```

Python Function

```
import os ← this is how you import modules/libs

def my_function(arg1, arg2):
  print(arg1, arg2)
  os.exit(0)

my_function(“foo”, “bar”)
```

Build Your Own Library

```
awesome_lib/
awesome_lib/__init__.py
awesome_lib/class_one.py
awesome_lib/func_one.py

import awesome_lib.class_one.cls_name
import awesome_lib.func_one.real_func
```

Make sure the library is inside the python path

Create Virtual Environment

Use Anaconda to seed python 3 (MacOS comes with python 2.7)

```
conda create -n mylambda python==3.7.6
source activate mylambda OR conda activate mylambda
```

Create working virtual environment for lambda

```
cd project_dir
python -m venv venv
source venv/bin/activate
```

Now you have an isolated virtual environment with python 3

Beginning in Serverless Framework

```
brew install node (if you don't have node)

npm install -g serverless (install serverless)

sls upgrade (if serverless is too old)

aws-azure-login (get sts credential, choose sandbox)

sls create -t aws-python3 -p simple-surname -n full-name

aws-azure-login (get sts credential)
```

Let’s Talk About serverless.yml

Look at the functions section, that’s where the event integration happens

You can also make your serverless.yml more fancy, like:

- Use plugins (e.g. AWS Step Functions)
- Centralise parameters across the serverless.yml
- Provide arguments from sls command line and inject into serverless.yml
- Integrate with Cloudformation to create stack and IAM when you deploy sls
- Integrate with existing Cloudformation via Outputs

Refer to the serverless framework documentation for its full feature

Serverless.yml: The command centre

```
provider:
  stage: dev
  region: ap-southeast-2
```

Save serverless.yml, then deploy:

```
sls deploy
```

Invoke the function:

```
sls invoke -f hello -l
sls invoke local -f hello -l
```

Let’s Modify the Python Code

Put this in your handler hello function (don’t forget the indentation):

```
print(“this is lambda by firstname.lastname”)
```

Deploy it

Invoke it locally (to run this, you don’t need to deploy)

Invoke it remotely

Check cloudwatch log (navigate from lambda monitoring console)

Let’s Talk About Getting Organised

```
cd project_dir
mkdir -p functions/simpleprint
mv handler.py functions/simpleprint/

# modify serverless.yml
functions:
  hello: <- can be different from function name
    handler: functions/simpleprint/handler.hello

sls invoke local -f hello -l
```

FOR SIMPLICITY SAKE, OMIT THE simpleprint dir

Working with External Modules

```
Requirements.txt (at root of project dir):
numpy

handler.py:
import numpy


# look at the size of fullname.zip
ls -lh .serverless/
sls package
ls -lh .serverless/
```

Automatic Packaging of External Modules

Package > exclude makes your zip file leaner from helper artifacts

Functions > exclude and include is a good practice to be specific with what you want to package with your lambda core function

Define serverless-python-requirements plugin and instruct to use docker

```yaml
package:
  exclude:
    - node_modules/**
    - build/**
    - package.json
    - package-lock.json
    - requirements.txt
    - .serverless/**
    - venv/** # if you are using virtual environment for development

functions:
  hello:
    handler: functions/handler.hello
    package:
      exclude:
        - functions/**
      include:
        - functions/handler.py

plugins:
  - serverless-python-requirements

custom:
  pythonRequirements:
    dockerizePip: non-linux
```

Let’s Test External Module Packaging

```
sls package
# woops, it’s broken!
npm init -f
npm install --save serverless-python-requirements
sls package
ls -l .serverless/
```

To test code locally with external module dependencies:

```
pip install -r requirements.txt (within venv)
```

Push Your Fat Zip to the Cloud

Before deploying your code, take a look at your lambda function in AWS console, you can see your python code.

Deploy your fat lambda now

Take a look again at your code, can you see them?

Externalise your bulky modules with Layers.

You can re-use publicly available layers just by referring to its ARN from serverless.yml

Lambda Layers

Used for externalising big modules so your lambda can deploy faster

Checkout our BitBucket:

- project: lambda-fargate-scalfolding
- branch: using-layer

Before you leave, clean up: **`sls remove && shutdown -h`**

Summary

Don’t drink the Kool-Aid!

Learn Python, good language in general, excel in ML and sys admin

Use Python 3.7, not 2.7 unless you have no choice

Organise your lambda codes

Make your code lean, but if it’s fat by design, oh well

If there’s no built-in integration with Lambda, leverage Boto framework