Microservices and Serverless in Python projects

José Manuel Ortega
Europython 2018
@jmortegac
Agenda

• Microservices in python
• Introducing Serverless and Function as a Service
• Python frameworks for AWS
• AWS Lambda functions with **zappa** and **chalice**
• Deploy AWS lambda functions from aws console
Microservices vs Serverless

- **serverless**: Término de búsqueda
- **microservices**: Término de búsqueda

Interés a lo largo del tiempo

*Graph showing the interest over time for the search terms 'serverless' and 'microservices'. The graph includes a bar chart and a line graph with data points marked with dates such as 30 jun. 2013, 5 abr. 2015, and 8 ene. 2017.*
Microservices

Tornado

Twisted
Asynchronous calls with asyncio and aiohttp

```python
import asyncio
import aiohttp

@asyncio.coroutine
def fetch_page(url):
    response = yield from aiohttp.request('GET', url)
    body = yield from response.read()
    return body

content = asyncio.get_event_loop().run_until_complete(fetch_page('http://python.org'))
print(content)
```
REST API Development

django

REST framework

Flask

web development, one drop at a time
Performance

- 8% less memory
- 6% faster response times

Flask

web development, one drop at a time

django

REST framework
Microservices with GraphQL
GraphQL + Python
Microservices and Serverless in Python projects
Microservices and Serverless in Python projects
**Distributed Messaging**

**ZeroMQ** \zero-em-queue\, \ØMQ\:
- Connect your code in any language, on any platform.
- Carries messages across inproc, IPC, TCP, TIPC, multicast.
- Smart patterns like pub-sub, push-pull, and router-dealer.
- High-speed asynchronous I/O engines, in a tiny library.
- Backed by a large and active open source community.
- Supports every modern language and platform.
- Build any architecture: centralized, distributed, small, or large.
- Free software with full commercial support.
```python
_context = zmq.Context()
_publisher = _context.socket(zmq.PUB)
url = 'tcp://{}:{}'.format(HOST, PORT)

def publish_message(message):
    try:
        _publisher.bind(url)
        time.sleep(1)
        myjson = json.dumps(message)
        _publisher.send(myjson)
    except Exception as e:
        print "error {}".format(e)
    finally:
        _publisher.unbind(url)
```
class ZClient(object):

    def __init__(self, host=HOST, port=PORT):
        """Initialize Worker""
        self.host = host
        self.port = port
        self._context = zmq.Context()
        self._subscriber = self._context.socket(zmq.SUB)
        print "Client Initiated"

    def receive_message(self):
        """Start receiving messages""
        print "receive_message"
        self._subscriber.connect('tcp://{}:{}, format(self.host, self.port))
        self._subscriber.setsockopt(zmq.SUBSCRIBE, b"

        while True:
            print 'listening on tcp://{}:{}, format(self.host, self.port)
            message = self._subscriber.recv()
            print message
            logging.info('{} - {}'.format(message, time.strftime('%Y-%m-%d %H:%M')))}

if __name__ == '__main__':
    zs = ZClient()
    zs.receive_message()
Microservices benefits

• Separation of concerns
• Services are decoupled from each other
• Managing smaller projects
• More scaling and deployment options
Serverless

Since the release of **AWS Lambda** (and others that have followed), all the rage has been about **serverless architectures**. These allow microservices to be deployed in the cloud, in a fully managed environment where one doesn’t have to care about managing any server, but is assigned stateless, ephemeral **computing containers** that are fully managed by a provider. With this paradigm, events (such as a traffic spike) can trigger the execution of more of these **containers** and therefore give the possibility to handle “infinite” horizontal scaling.
Serverless architecture

• **FaaS** - Function as a Service
• Fully managed computing
  • Provisioning
  • Scalability
  • Monitoring
  • Logging
• Deploy your code
• Pay only for actual usage
Serverless architecture

Upload your code to AWS Lambda

Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity

Lambda runs your code only when triggered, using only the compute resources needed

Pay just for the compute time you use
Serverless uses cases

➢ REST API
  • Stateless services and microservices
  • Suitable for Chat bots

➢ Events
  • File processing (S3 event) & Data ingestion
  • Data/Stream processing
  • Incidents handling (CloudWatch event log)
  • IoT

➢ Scheduled tasks
  • Monitoring, load testing
  • Periodical jobs
Microservices and Serverless in Python projects
Serverless benefits

• No server management
• Automatic scaling and load balancing
• Lower infrastructure costs
• Flexibility and high availability
• Infrastructure managed by service provider
Serverless drawbacks

• The tools around the deployment automation of serverless functions are still in development.

• There is **no control** over containers when the execution **environments** are created or destroyed

• Debugging, Deploying and monitoring
Cloud providers

• AWS
• Microsoft Azure
• Cloud platform
  • OpenWhisk(OS)
  • Kubeless(OS)
Aws lambda

Amazon S3
Static Content/Chat Web App

Web client
End user

Amazon API Gateway
REST Interface

AWS Lambda
Backend Logic

Amazon DynamoDB
Messages Data Store
Synchronous (push)

Amazon API Gateway

/order

AWS Lambda function

Asynchronous (event)

Amazon SNS
Amazon S3

AWS Lambda function

reqs
Aws lambda functions
def lambda_handler(event, context):
    """Entry point.
    
    event: AWS Lambda uses this parameter to pass in event data to the handler.
    
    context: AWS Lambda uses this parameter to provide runtime information to your handler.
    ""

    return
Create lambda function with awscli

```
$ aws lambda create-function \
  --region eu-west-1 \
  --function-name MyHandler\ 
  --zip-file fileb://handler.zip \ 
  --role arn:aws:iam::XXX:role/MyLambdaRole \ 
  --vpc-config SubnetIds=XXX,SecurityGroupIds=XXX \ 
  --handler handler.handler \ 
  --runtime python3.6 \ 
  --profile personal \ 
  --timeout 10 \ 
  --memory-size 512
```
The deployment package of your Lambda function "helloworld-dev" is too large to enable inline code editing. However, you can still invoke your function right now.

Code entry type
- Upload a .ZIP file

Function package*
- Upload

For files larger than 10 MB, consider uploading via S3.

Environment variables
You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

- Key
- Value
- Remove

Enable encryption helpers
For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.
-
Basic information

Runtime
Python 3.6

Handler
The filename.handler-method value in your function. For example, "main.handler" would call the handler method defined in main.py.

handler.lambda_handler

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.

Choose an existing role

Existing role
You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

helloworld-dev-ZappaLambdaExecutionRole

Description
Zappa Deployment
Frameworks

aws

Zappa

serverless

Lambda
dify

Programmable AWS Lambda for Python

python-λ

Chalice

Microservices and Serverless in Python projects
Amazon API Gateway + Lambda function
The way cloud should be.

Serverless is your toolkit for deploying and operating serverless architectures. Focus on your application, not your infrastructure.

```
# Install serverless globally
$ npm install serverless -g

# Login to your Serverless account
$ serverless login

# Create a serverless function
$ serverless create --template hello-world

# Deploy to cloud provider
$ serverless deploy

# Function deployed! Trigger with live url
$ http://xyz.amazonaws.com/hello-world
```
Zappa architecture
Zappa

Deploy your WSGI apps on AWS Lambda

With Zappa, each request is given its own virtual HTTP "server" by Amazon API Gateway. AWS handles the horizontal scaling automatically, so no requests ever time out. After your app returns, the "server" dies.

- No more tedious web server configuration!
- No more paying for 24/7 server uptime!
- No more worrying about load balancing / scalability!
- No more worrying about web server security!
→ pip install zappa
→ zappa init

Welcome to Zappa!
...
→ zappa deploy
Welcome to Zappa!

Zappa is a system for running server-less Python web applications on AWS Lambda and AWS API Gateway. This `init` command will help you create and configure your new Zappa deployment. Let's get started!

Your Zappa configuration can support multiple production stages, like 'dev', 'staging', and 'production'. What do you want to call this environment (default 'dev'):

AWS Lambda and API Gateway are only available in certain regions. Let's check to make sure you have a profile set up in one that will work.
We found the following profiles: default, adsk forge2?, and adsk forge. Which would you like us to use? (default 'default'):

Your Zappa deployments will need to be uploaded to a private S3 bucket. If you don't have a bucket yet, we'll create one for you too.
What do you want call your bucket? (default 'zappa-68fz81bc0'):

It looks like this is a Flask application. What's the modular path to your app's function? This will likely be something like 'your_module.app'. We discovered: app.app
Where is your app's function? (default 'app.app'):
Does this look okay? (default 'y') [y/n]:

Done! Now you can deploy your Zappa application by executing:

\$ zappa deploy dev

After that, you can update your application code with:

\$ zappa update dev

To learn more, check out our project page on GitHub here: https://github.com/Miserlou/Zappa and stop by our Slack channel here: https://slack.zappa.io

Enjoy!,
~ Team Zappa!
# zappa_settings.json
{
  "dev": {
    "aws_region": "us-east-1",
    "django_settings": "hello.settings",
    "profile_name": "default",
    "project_name": "hello",
    "runtime": "python3.6",
    "s3_bucket": "zappa-huyg6op0s"
  }
}
Zappa deploy

$ zappa deploy <env>

• Zips code and dependencies
• Create AWS Lambda and deploys the zip
• Creates endpoint on API Gateway and links to AWS Lambda
Zappa deploy

```
(py36) → zappa zappa deploy
Calling deploy for stage dev6.
Downloading and installing dependencies..
- sqlite==python36: Using precompiled lambda package
Packaging project as zip.
Uploading zappa-dev6-1523116120.zip (12.8MiB).
100%|   13.5M/13.5M [00:09<00:00, 1.43MB/s]
Scheduling..
Scheduled zappa-dev6-zappa-keep-warm-handler.keep_warm_callback with expression rate(4 minutes)!
Uploading zappa-dev6-template-1523116139.json (1.6KiB).
100%|   1.60K/1.60K [00:00<00:00, 3.09KB/s]
Waiting for stack zappa-dev6 to create (this can take a bit).
75%|   3/4 [00:06<00:02, 2.17s/res]
Deploying API Gateway..
```
Zappa
from flask import Flask
from zappa.async import task
app = Flask(__name__)

task

def make_pie():
    """ This takes a long time! ""
    ingredients = get_ingredients()
    pie = bake(ingredients)
    deliver(pie)

@app.route('/api/order/pie')
def order_pie():
    """ This returns immediately! ""
    make_pie()
    return "Your pie is being made!"
Chalice

• Python Serverless Microframework for AWS
• Each endpoint is a separate function
Chalice

Python Serverless Microframework for AWS

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Merge branch 's3-trigger'</td>
<td>2 months ago</td>
</tr>
<tr>
<td></td>
<td>Make pylint/flake8 happy</td>
<td>13 days ago</td>
</tr>
<tr>
<td></td>
<td>Automatically url decode S3 keys</td>
<td>13 days ago</td>
</tr>
<tr>
<td></td>
<td>Set patch version to 0 on minor version bumps</td>
<td>7 months ago</td>
</tr>
<tr>
<td></td>
<td>Automatically url decode S3 keys</td>
<td>13 days ago</td>
</tr>
<tr>
<td></td>
<td>Ignore not implementederror in coverage</td>
<td>a year ago</td>
</tr>
<tr>
<td></td>
<td>Finish PR for packaging arbitrary directories as wheels</td>
<td>a month ago</td>
</tr>
<tr>
<td></td>
<td>Automatically reload dev server when files change</td>
<td>a month ago</td>
</tr>
<tr>
<td></td>
<td>Print test type</td>
<td>a month ago</td>
</tr>
</tbody>
</table>

Latest commit: 7ac05bc 13 days ago
Chalice

$ pip install chalice
$ chalice new-project helloworld && cd helloworld

$ cat app.py
from chalice import Chalice

app = Chalice(app_name="helloworld")

@app.route("/")
def index():
    return {"hello": "world"}

$ chalice deploy
import requests

URL = 'http://api.apixu.com/v1/current.json?key=51deeb4a20ef476db6b165025181907&q=

@app.route('/weather/{city}')
def weather(city):
    try:
        if city is None:
            return _error("Invalid data (required city)")

        response = requests.get(URL+city).json()
        return Response(body=response,
                        status_code=200,
                        headers={'Content-Type': 'application/json'})
    except Exception as exception:
        raise BadRequestError("Unknown url '%s'" % (URL))
## Chalice methods

<table>
<thead>
<tr>
<th>Resource</th>
<th>HTTP Verb</th>
<th>AWS Lambda</th>
</tr>
</thead>
<tbody>
<tr>
<td>/talks</td>
<td>GET</td>
<td>get_talks</td>
</tr>
<tr>
<td>/talk</td>
<td>POST</td>
<td>add_new_talk</td>
</tr>
<tr>
<td>/talks/{ID}</td>
<td>PUT</td>
<td>update_talk</td>
</tr>
<tr>
<td>/talks/{ID}</td>
<td>DELETE</td>
<td>delete_talk</td>
</tr>
</tbody>
</table>
Chalice methods

```python
@app.route('/talks', methods=['GET'])
def get_talks():
    return get_app_db().list_items()

@app.route('/talks', methods=['POST'])
def add_new_talk():
    body = app.current_request.json_body
    return get_app_db().add_item(
        id=body['id'],
        description=body['description']
    )

@app.route('/talks/{id}', methods=['DELETE'])
def delete_talk(id):
    return get_app_db().delete_item(id)

@app.route('/talks/{id}', methods=['PUT'])
def update_talk(id):
    body = app.current_request.json_body
    get_app_db().update_item(id, description=body.get('description'), state=body.get('state'))
```
Chalice options

Usage: chalice [OPTIONS] COMMAND [ARGS]...

Options:
--version                          Show the version and exit.
--project-dir TEXT                The project directory. Defaults to CWD
--debug / --no-debug              Print debug logs to stderr.
--help                            Show this message and exit.

Commands:
delete
deploy
gen-policy
generate-pipeline Generate a cloudformation template for a...
generate-sdk
local
logs
new-project
package
url
Chalice deploy

Updating IAM policy.
Updating lambda function...
Regen deployment package...
Sending changes to lambda.
API Gateway rest API already found.
Deploying to: dev
Microservices and Serverless in Python projects
A sandboxed local environment that replicates the live AWS Lambda environment almost identically – including installed software and libraries, file structure and permissions, environment variables, context objects and behaviors – even the user and running process are the same.
Calculating cost for AWS Lambda, Azure Functions, Google Cloud Functions, and IBM OpenWhisk

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Request Cost</th>
<th>Compute Cost</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>$3.70</td>
<td>$1.67</td>
<td>$5.37</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>$0.20</td>
<td>$1.60</td>
<td>$1.80</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>$0.40</td>
<td>$1.65</td>
<td>$2.05</td>
</tr>
<tr>
<td>Project Name</td>
<td>Description</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>aws-python-alexa-skill</td>
<td>Add front matter to the examples readme for publishing to site.</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-auth0-custom-authorizer</td>
<td>Added Python AWS Lambda Authorizer</td>
<td>9 days ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-dynamodb-s3-sigurl</td>
<td>Fix issues from reviewer suggestions.</td>
<td>8 months ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-rest-api-with-dynamodb</td>
<td>Fixing SETUP typo in README.md</td>
<td>7 months ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-rest-api-with-faunadb</td>
<td>Add front matter to the examples readme for publishing to site.</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-rest-api-with-pymongo</td>
<td>Update to docs</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-scheduled-cron</td>
<td>Add front matter to the examples readme for publishing to site.</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-simple-http-endpoint</td>
<td>Add front matter to the examples readme for publishing to site.</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>aws-python-telegram-bot</td>
<td>Add aws-python-telegram-bot</td>
<td>9 months ago</td>
<td></td>
</tr>
<tr>
<td>azure-node-simple-http-endpoint</td>
<td>Update the Azure example to match recent updates.</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>google-node-simple-http-endpoint</td>
<td>Merge pull request #159 from serverless/update-gcf-npm-package-version</td>
<td>a year ago</td>
<td></td>
</tr>
<tr>
<td>kubeless-python-schedule</td>
<td>Adapt examples to kubeless 0.5</td>
<td>4 months ago</td>
<td></td>
</tr>
<tr>
<td>kubeless-python-simple</td>
<td>Adapt examples to kubeless 0.5</td>
<td>4 months ago</td>
<td></td>
</tr>
</tbody>
</table>
References


• [https://github.com/Miserlou/Zappa](https://github.com/Miserlou/Zappa)
• [https://github.com/pmuens/awesome-serverless](https://github.com/pmuens/awesome-serverless)

• [https://github.com/aws/chalice](https://github.com/aws/chalice)
Serverless architecture is the next generation of cloud evolution
Thank you!

José Manuel Ortega
jmortega.github.io