Serverless Computing: FaaSter, Better, Cheaper and More Pythonic

Josef Spillner <josef.spillner@zhaw.ch>
Service Prototyping Lab (blog.zhaiw.ch/icclab)

February 16, 2018 | Swiss Python Summit
What’s Function-as-a-Service (FaaS)?

- running functions in the cloud (hosted functions)
- real “pay per use“ (per invocation, per load x time unit, e.g. GHz/100ms)
- seemingly “serverless“
Developer’s Vision: Rapid Prototyping

Applied research mission
- technological immersion combined with scientific excellence
- supporting local (Swiss) development & devops companies

Applied to current serverless computing/FaaS environments:
The FaaS Space - in Python

- AWS Lambda
- Google Cloud Platform Functions
- IBM Cloud Functions
- Azure Functions
- spotinst Functions
- weblask Functions
- PyWren [Lambda]
- Zappa [Lambda]
- Chalice [Lambda]
- Dawson [Lambda]
- Apaxes [Lambda]
- Serverless Framework [Lambda, OW, GCF, AF]
- OVH Functions
- IBM Composer [OpenWhisk]
- Whisk-Mocha [OpenWhisk]
- X-Ray [Lambda]
- Step Functions [Lambda]
- MR Refarg [Lambda]
- Fission Workflows [Fission]
- Fission
- Funktion
- Kubeless
- Picasso
- Lever OS
- Docker-LambCI
- Snafu
- Podilizer
- Termite
- Lambada

Faas (Docker)
# Runtime Overview: Providers & Stacks

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Languages</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>Node.js, Java, Python / C#</td>
<td>Service</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>Node.js</td>
<td>Service</td>
</tr>
<tr>
<td>Apache OpenWhisk</td>
<td>Node.js, Swift, Docker* / Python</td>
<td>OSS</td>
</tr>
<tr>
<td>IBM Cloud Functions</td>
<td>&quot;-&quot;</td>
<td>Service</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>Node.js, C# / F# / Python, PHP, ...</td>
<td>Service</td>
</tr>
<tr>
<td>OVH Functions</td>
<td>Node.js, Python, Perl, Go, Bash</td>
<td>Service</td>
</tr>
<tr>
<td>Webtask.io</td>
<td>Node.js</td>
<td>OSS + Service</td>
</tr>
<tr>
<td>Hook.io</td>
<td>Node.js, ECMAScript, CoffeeScript</td>
<td>OSS + Service</td>
</tr>
<tr>
<td>Effe</td>
<td>Go</td>
<td>OSS</td>
</tr>
<tr>
<td>OpenLambda</td>
<td>Python</td>
<td>Academic + OSS</td>
</tr>
<tr>
<td>LambCI Docker-Lambda</td>
<td>Node.js</td>
<td>OSS (re-engineered)</td>
</tr>
<tr>
<td>Lever OS</td>
<td>Node.js, Go</td>
<td>OSS</td>
</tr>
<tr>
<td>Fission</td>
<td>Node.js, Python</td>
<td>OSS</td>
</tr>
<tr>
<td>Funktion</td>
<td>Node.js</td>
<td>OSS</td>
</tr>
<tr>
<td>Kubeless</td>
<td>Python</td>
<td>OSS</td>
</tr>
<tr>
<td>IronFunctions</td>
<td>Node.js, Java, Python, Go, ...</td>
<td>OSS</td>
</tr>
<tr>
<td>→ Fn</td>
<td>&quot;-&quot;</td>
<td>OSS</td>
</tr>
</tbody>
</table>
## Runtime Overview: Python Evolution

<table>
<thead>
<tr>
<th>Year</th>
<th>Industry Proprietary</th>
<th>Industry Open Source</th>
<th>Academic Community</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td>AWS Lambda (Py2)</td>
<td>Hook.io</td>
<td>Dripcast</td>
</tr>
<tr>
<td>2015</td>
<td>AWS Lambda (Py2)</td>
<td>Webtask.io</td>
<td>Chatbot</td>
</tr>
<tr>
<td>2016</td>
<td>AWS Lambda (Py3)</td>
<td>Google Cloud Functions</td>
<td>Open Lambda (Py2)</td>
</tr>
<tr>
<td>2017</td>
<td>AWS Step Functions</td>
<td>Azure Functions (Py2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IBM OpenWhisk (Py2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kubeless (Py2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fission (Py3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lever OS</td>
<td></td>
</tr>
</tbody>
</table>
Open Source Tools for FaaS

Good News:
• almost all tools in this sphere are open source

Bad News:
• almost none of the large provider runtimes
  • AWS Lambda, MS Azure Functions, Google Cloud Functions, OVH Functions, ...
  • changing now? OpenWhisk, Fn
• first-use barrier
• heterogeneous approaches, no standards → synopsis, deployment, ...
• didactic usefulness
• research/experimentation flexibility, e.g. high-performance execution without isolation or authentication
FaaS Synopsis: Python Examples

AWS Lambda:

```python
def lambda_handler(event, context):
    ""
    event: dict
    context: meta information obj
    returns: dict, string, int, ...
    ""
    # ...
    return "result"
```

OpenWhisk/IBM Functions:

```python
def handler(input):
    ""
    input: dict
    returns: dict
    ""
    # ...
    return {}
```

OVH Functions:

```python
def handler(input):
    ""
    input: dict
    returns: str
    ""
    # ...
    return ""
```

Fission:

```python
def main():
    ""
    input: flask.request.get_data()
    returns: str
    ""
    # ...
    return "result"
```

Azure Functions:

```python
def main():
    from AzureHTTPHelper \ import HTTPHelper
    input = HTTPHelper().post
    # ...
    open(os.environ["res"], "w").\ write(json.dumps({"body": "..."}))
    main()
```

Further differences:
- function naming (mangling on client or service side)
- function granularity (number of entry points)

→ Deployment: provider tools, lambda-uploader, serverless fw, ...

Service Engineering Research Area
ICCLAB SPLAB
Overlay Approach: PyWren

Improved conveyance of “serverless“ paradigm
• no explicit deployment prior to execution
• rather, deploys while executing

```python
def my_function(b):
    x = np.random.normal(0, b, 1024)
    A = np.random.normal(0, b, (1024, 1024))
    return np.dot(A, x)

pwex = pywren.default_executor()
res = pwex.map(my_function, np.linspace(0.1, 100, 1000))
```

How it works:
• tightly bound to AWS
• cloudpickle to AWS S3
• executes Lambda function which reads/writes from/to S3
• parallelisation through map functions
Overlay Approach: Gee’s Lambada

Deployment with dependencies
• requirements.txt file references Lambada framework

```python
def my_function(b):
    x = np.random.normal(0, b, 1024)
    A = np.random.normal(0, b, (1024, 1024))
    return np.dot(A, x)

tune = Lambada(role='...', region='...', memory=128)
@tune.dancer
def my_function_lambda(e, c):
    my_function(e['stddev'])
```

How it works:
• again, tightly bound to AWS
• creation of ZIP packages for manual or automated deployment
## Excursus: “Lambada“ projects

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>First Commit</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>[<a href="https://github.com/Superpedestrian/lambada">https://github.com/Superpedestrian/lambada</a>]</td>
<td>Building multiple Lambdas in one library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Josef Spillner‘s Lambada</td>
<td>Extraction and transformation of Python functions to Lambda</td>
<td>18.04.2016</td>
<td>X</td>
</tr>
<tr>
<td>[<a href="https://gitlab.com/josefspillner/lambada">https://gitlab.com/josefspillner/lambada</a>]</td>
<td>Extraction and transformation of Python functions to Lambda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Çağatay Gürtürk‘s Lambada</td>
<td>JAX-RS API framework for Java Lambdas and API Gateway</td>
<td>31.03.2016</td>
<td></td>
</tr>
<tr>
<td>[<a href="https://github.com/limbidaframework">https://github.com/limbidaframework</a>]</td>
<td>JAX-RS API framework for Java Lambdas and API Gateway</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[<a href="https://github.com/ingenieux/lambada">https://github.com/ingenieux/lambada</a>]</td>
<td>Maven integration for Java Lambdas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uSwitch‘s Lambada</td>
<td>Developing Java Lambdas in Closure</td>
<td>16.06.2015</td>
<td></td>
</tr>
<tr>
<td>[<a href="https://github.com/uswitch/lambada">https://github.com/uswitch/lambada</a>]</td>
<td>Developing Java Lambdas in Closure</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lambada: FaaS Code Transformer

Rapid prototyping through semi-automated transformation
Support for annotations

PyPI: pip install lambadatransformer

```python
@cloudfunction(memory=128, duration=5)
def my_function(b):
    x = np.random.normal(0, b, 1024)
    A = np.random.normal(0, b, (1024, 1024))
    return np.dot(A, x)
```

$ ./lambada --annotations my_function.py

```python
>>> from lambadalib import lambada
>>> lambada.move(globals(), endpoint=..., local=True)
```

Source level: ast, codegen
Object level: inspect

Target platform: AWS Lambda
Lambada Signature Converter

Going beyond just Lambda: Portable cloud functions

```python
>>> faasconverter: track module: test
>>> faasconverter: convert function foo (x)
>>> faasconverter: converted to module: test_portable.py

def foo(x):
    return 2*x

# FaaS-Converter wrapper for aws
def lambda_handler(event, context):
    return foo(event['x'])

# FaaS-Converter wrapper for ibm
def main(dict):
    return foo(event['x'])
```

Work in progress - helping hands welcome!
Snafu: The “Swiss Army Knife“

Good News:
• developer tooling is improving
• Serverless framework, PyWren, several Lambada’s...

Better News:
• more choice when deploying, executing, testing, migrating, sharing...
Architecture

Subsystems
- parsing functions
- triggering functions (connectors)
- authentication
- forwarding
- executing functions
- logging output and traces

Language support
- Python
- Java, C, JavaScript
- generic (containers)
Snafu Use Cases

Toying/Prototyping/Debugging
- directly from Git: `git clone https://github.com/serviceprototypinglab/snafu`
- or from PyPI: `pip install snafu`

Single-Tenant Operations
`docker run -ti jszhaw/snafu`

Multi-Tenant Operations
Snafu Examples

Integration into the wider FaaS ecosystem

```
$ snafu-import \
  --source <s> \
  --target <t>

$ alias aws="aws \
  --endpoint-url \
  http://localhost:10000"

$ wsk property set \
  --apihost \
  localhost:10000

$ ./tools/patch-gcloud
```
More Snafu Examples

# Zero-configuration interactive mode
$ snafu

# Invocation of a specific function from a known module
$ snafu -q -x helloworld.helloworld functions/helloworld.py

# Combination of various parameters: Java method with Lambda semantics
$ snafu -l sqlite -e java -c lambda -C messaging

# Using the Lambda function/method parser
$ snafu -f lambada -s test.ini

# Run function externally
$ snafu -X gfunctions

# Import/export
$ snafu-import -s gfunctions -t funktion -c myfunction
Even More Snafu Examples

# Zero-configuration FaaS daemon
$ snafu-control

# Lambda compatibility mode
$ snafu-control -a aws -r -d -e docker

# Multi-tenancy account management
$ snafu-accounts --add -k <k> -s <s> -e <ep>

# Safe mode
$ snafu-control -P
Open Function Ecosystems

Towards vibrant decentralised cloud function communities
Open Function Ecosystems

Our “Function Hub“ and “composeless“ prototypes

Work in progress - helping hands welcome (again)!
The Future of FaaS

Converters

Dev Tools

Marketplaces

Optimisers

Transformers

Composers

Runtimes

Deployers

Debuggers

European Symposium on Serverless Computing and Applications (ESSCA) - December 21, 2018, Zurich Toni-Areal - essca2018.servicelaboratory.ch