

Function-as-a-Service:

A Pythonic Perspective on Serverless Computing

Josef Spillner <josef.spillner@zhaw.ch>
Service Prototyping Lab (blog.zhaw.ch/icclab)
Zurich University of Applied Sciences

Jun 13, 2017 | PyParis

Your Tutorial Agenda

50' FaaS overview and some existing tools

20' Lambada: Decompose your functions

20' Snafu: Run your functions

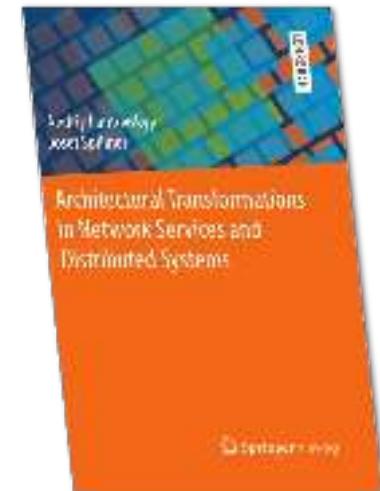
Your Tutorial Instructor

Josef Spillner <josef.spillner@zhaw.ch>

- works at Zurich University of Applied Sciences
- lectures Python programming to undergraduates & masters of advanced studies
- performs research in the Service Prototyping Lab



- co-authored «Architectural Transformations in Network Services and Distributed Systems»
- wrote many rarely used Python things since 2003



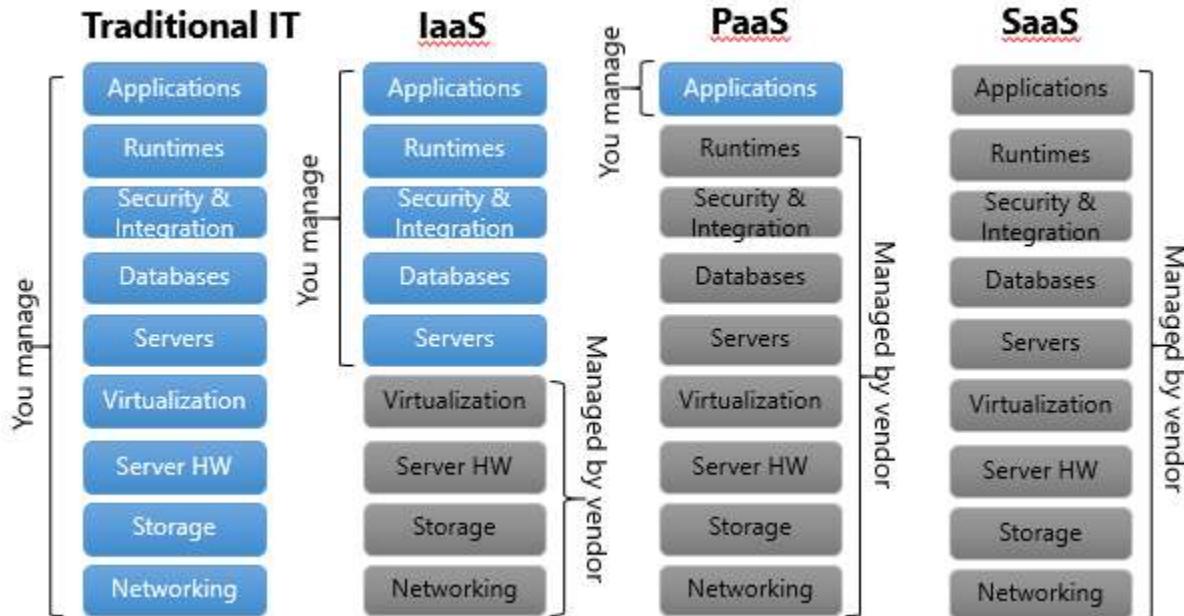
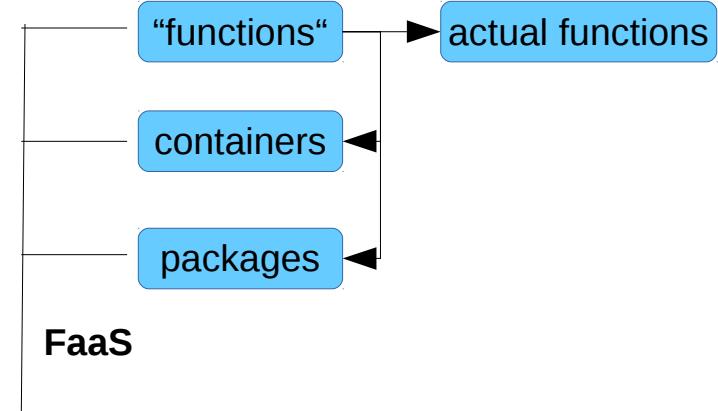
[LS16]

Service Prototyping Lab + ICCLab



What is 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“



[mazikglobal.com]

Examples of FaaS

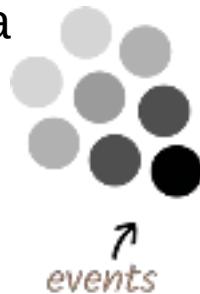
monitoring event

sensor data

log entry

git push

...

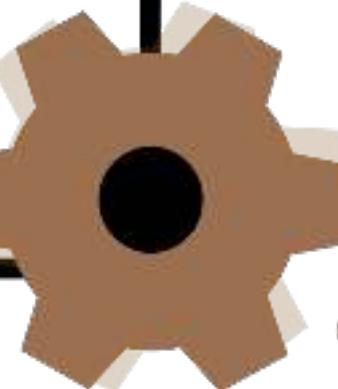


events



triggers

HTTP
XMPP
AMQP
...



actions
(functions)

Your Python
functions!

max 1 per hour

...

[openwhisk.org]

The FaaS Space - in Python





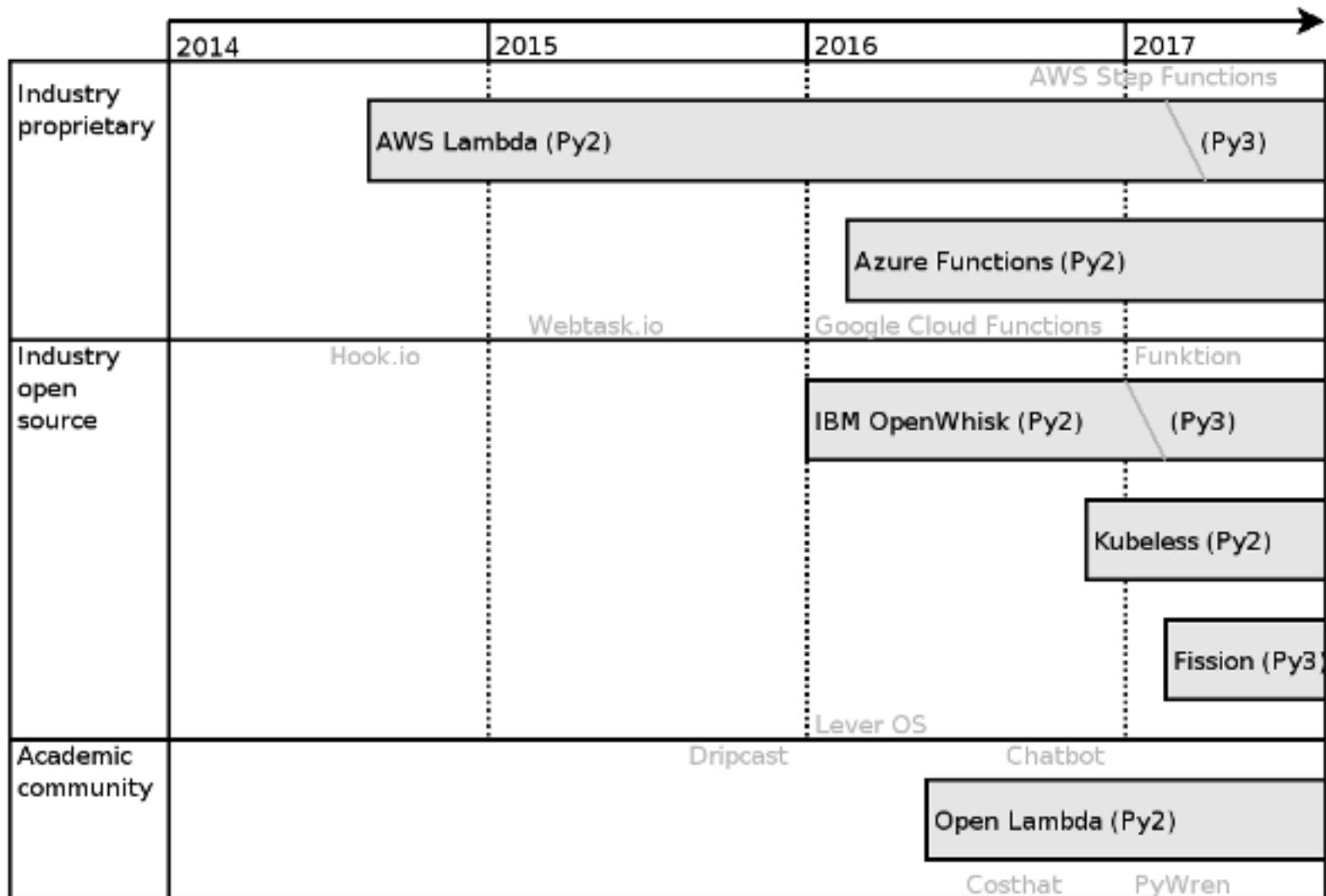
The FaaS Space: Runtimes

Function-as-a-Service offerings in greater detail...

Implementation	Languages	Availability
AWS Lambda	Node.js, Java, Python / C#	Service
Google Cloud Functions	Node.js	Service
IBM/Apache OpenWhisk	Node.js, Swift, Docker* / Python	OSS + Service
Azure Functions	Node.js, C# / F#, Python, PHP, ...	Service
Webtask.io	Node.js	OSS + Service
Hook.io	Node.js, ECMAScript, CoffeeScript	OSS + Service
Effe	Go	OSS
OpenLambda	Python	Academic + OSS
LambCI Docker-Lambda	Node.js	OSS (re-engineered)
Lever OS	Node.js, Go	OSS
Fission	Node.js, Python	OSS
Funktion	Node.js	OSS
Kubeless	Python	OSS

Trend: Sooner or later → gaps will be filled

The FaaS Space: Python runtimes



FaaS Synopsis in Python

AWS Lambda:

```
def lambda_handler(event, context):
    """
    event: dict
    context: meta information object
    returns: dict, string, number, ...
    """
    # ...
    return "result"
```

OpenWhisk:

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

Fission:

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

Azure Functions:

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

Further differences:

- function scoping (e.g. with/without export in JavaScript)
- function naming (mangling on client or service side)

The World's Tools for FaaS Devs



~~\$ wsk~~
does not compile

~~# openlambda~~
\$ bin/admin
(invoke read error)



~~\$ az~~
not scriptable

~~\$ kubeless~~
breaks minikube

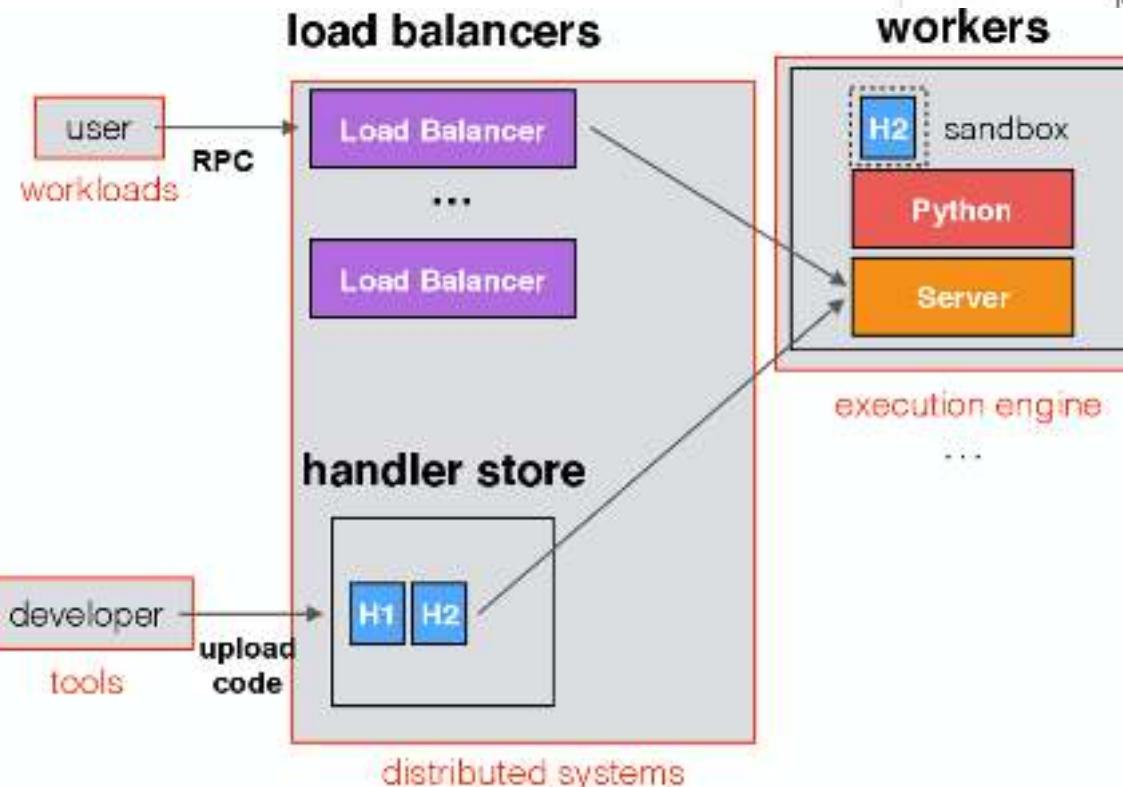


AWS Lambda

~~\$ aws lambda~~
requires account

\$ fission

Open Lambda - Hands-on Time!



Function-as-a-Service: A Pythonic Perspective
Serverless Computing Tutorial @ PyParis
2017

Roel Spuller
di University of Applied Sciences, School of Engineering,
Service Prototyping Lab (http://spla.ch/ical...)
8404 Winterthur, Switzerland
roel.spuller@bluewin.ch

June 9, 2017

Learning objectives
you will practice how about different ways of programming your services.
how to generate and code the code of the tasks by yourself
to build simple function applications

Using Lambda

run your function from code files
or Lambda directly from its own repository. You will furthermore
be able to run recent lambda.

Listing 1: Obtaining Lambda

```
# curl https://gitlab.com/joaoespilane/lambda.git  
by cloning it like temperature.py. It contains one function and one  
file together form your temperature application (Listing 2)
```

Listing 2: Temperature application

```
temperature()  
20 42.22  
AvgTemperature:  
lambda event, context:  
    print("Temperature is", event["temp"]  
    print("Avg temp = ", avgTemp)  
    return avgTemp
```

Please include a function like the one above. You will need which can be used locally but are ready to be deployed in AWS Lambda (Listing 3).

Fission - Hands-on Time!

Fission-as-a-Service: A Python Implementation of Serverless Computing - Tutorial by P. Palusz
2017

Machine Learning Model Deployment with Fission
Serverless Computing with Python Functions
- Machine Learning
- Function Deployment

TensorFlow Example
TensorFlow function for image processing, running
TensorFlow on GPU, and using TensorFlow's
Cloud ML API to predict.

Using Docker

Run a Docker container with the
TensorFlow example function and run it with
TensorFlow's Cloud ML API.

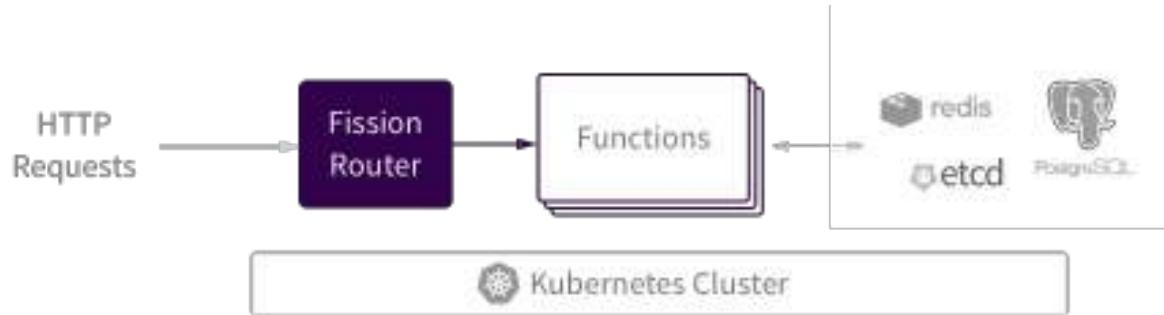
Using Kubernetes

Deploy the TensorFlow function to a Kubernetes
cluster and run it with TensorFlow's Cloud ML API.

Using Fission Functions

Deploy the TensorFlow function to a Fission
functions cluster and run it with TensorFlow's
Cloud ML API.

See Examples for more details on how to use Fission's
functionality to build serverless applications.



Overlay Approach: PyWren

Improved conveyance of “serverless” paradigm

- no explicit deployment prior to execution
- rather, deploys while executing

```
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:

- cloudpickle to AWS S3
- executes Lambda function which reads/writes from/to S3
- parallelisation through map functions



Our Tools for FaaS Devs

Podilizer
(Java)

Lambada
(Python)
today

Web2Cloud
(JavaScript)



Lambackup
(file backups)

Lama
(relational data)

Snafu
(FaaS host)
today

Lambada

Definition of “FaaSification”

→ Process of automated decomposition of software application into a set of deployed and readily composed function-level services.

FaaSification := code analysis + transformation + deployment + on-demand activation

Integration Categories:

- generic (code/function unit generation)
- single-provider integration
- multi-provider integration

Depth Categories:

- shallow (file to function)
- medium (function to lines)
- deep (line to many lines)

Decomposition Categories:

- static code analysis
- dynamic code analysis

→ Lambada: FaaSification for Python
(currently limited to Lambdaification)

Lambada

Code Analysis

Dependencies

- imported modules
- global variables
- dependency functions
 - defined in other module
 - defined in same module

Input/Output

- printed lines
- input statements
 - tainting
 - stateful function splitting

```
import time
import math

level = 12
counter = 0

def fib(x):
    global counter
    counter += 1
    for i in range(counter):
        a = math.sin(counter)
    if x in (1, 2):
        return 1
    return fib(x - 1) + fib(x - 2)

if __name__ == "__main__":
    fib(level)
```

Lambada

Code Transformation

Rewrite rules, via AST:

return 9	print("hello")	local_func()
-----	return 9	-----
return {"ret": 9}	-----	local_func_stub()
	return {"ret": 9, "stdout": "hello"}	

Stubs, via templates:

```
def func_stub(x):
    input = json.dumps({"x": x})
    output = boto3.client("lambda").invoke(FN="func", Payload=input)
    y = json.loads(output["Payload"].read().decode("utf-8"))
```

Lambada

Code Transformation

Stateful proxies for Object-Oriented Programming:

```
class Test:           →   class Proxy:  
    def __init__(self):      def __new__(cls, clsname, p=True):  
        self.x = 9            if p: # __new__ must return callable  
        def test(self):       return lambda: Proxy(clsname, False)  
            return self.x * 2  else:  
                           return object.__new__(cls)  
  
                           def __init__(self, clsname, ignoreproxy): ...  
                           def __getattr__(self, name): ...
```

- Test becomes Proxy("Test"), Test() then invokes proxy
- test() becomes remote_test({"x": 9}) through network proxy class
- automatically upon import of class

Lambada

Code Deployment + Activation

```
$ lambada [--local] [--debug] [--endpoint <ep>] <file.py>
$ python3 -m lambada <file.py>
```

```
>>> import lambada
>>> lambada.move(globals() [, endpoint="..."])
```

Local mode: source code modified locally as copy

Remote mode: rewritten source code deployed and invoked

Lambada - Hands-on Time!

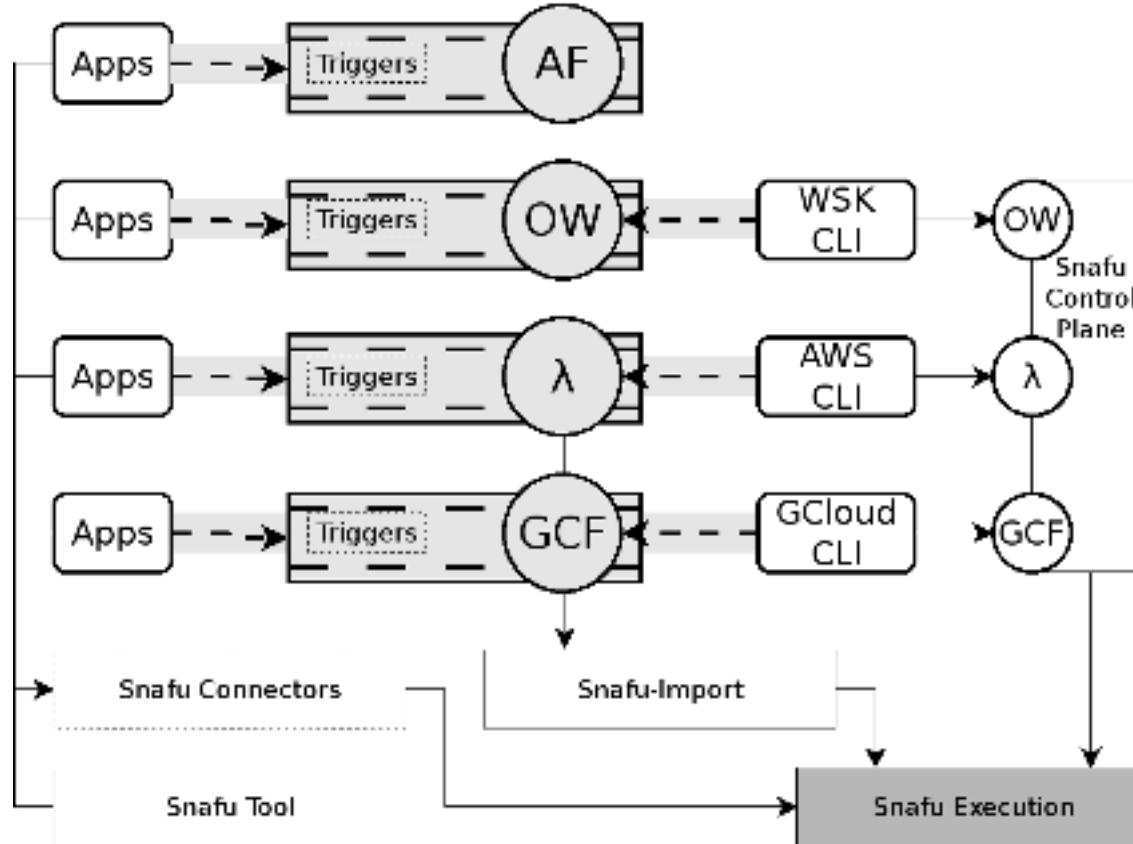


[d0wn.com]



Snafu

The Swiss Army Knife of Serverless Computing



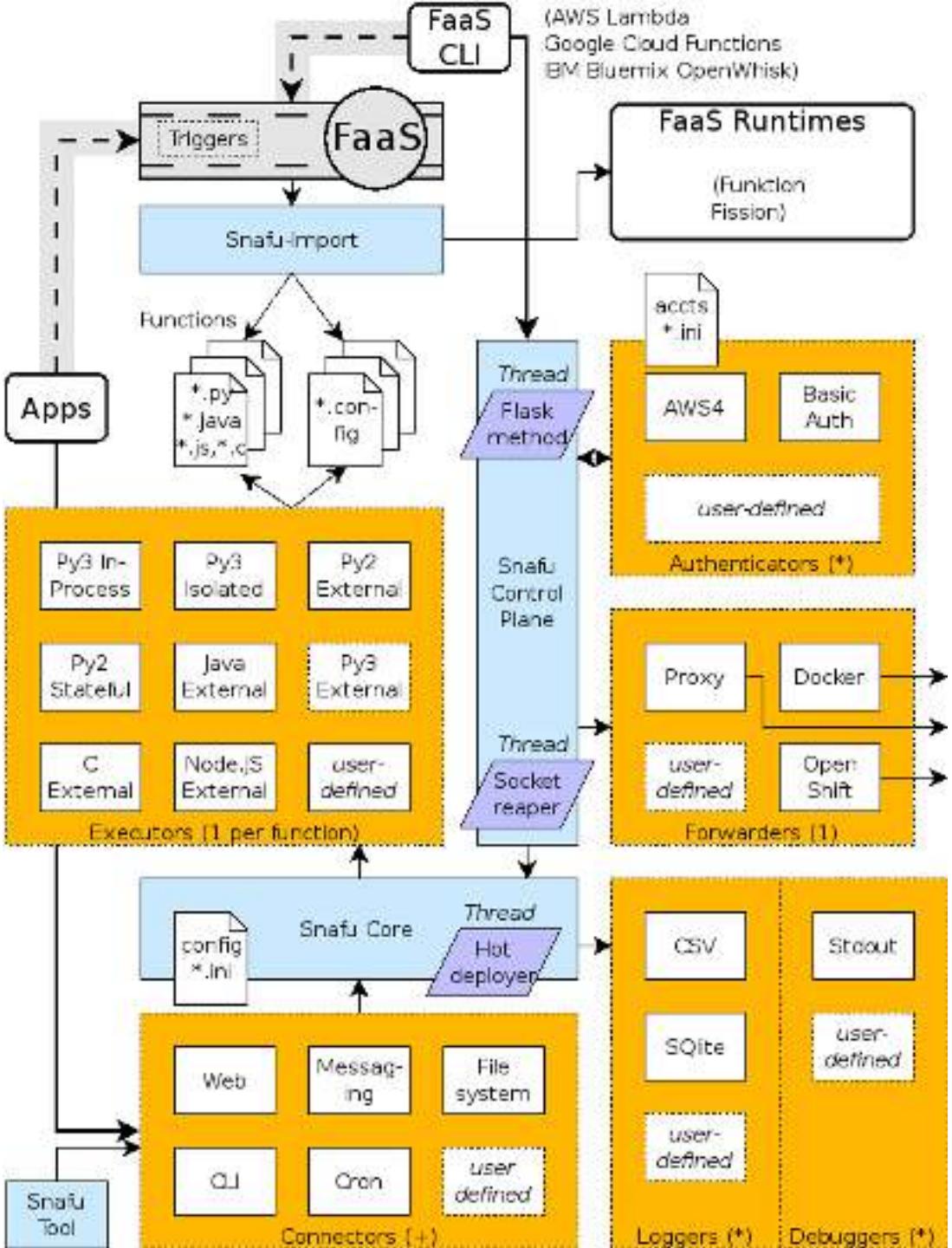
Snafu

Current Implementation

- scalable from developer single instance to multi-tenant deployments
- executes Python 2 & 3, Java, JavaScript, C
- integrates with FaaS ecosystem at-large
- extensible subsystems

SLOC: ~1800
(including subsystems: ~800)

```
$ pip install snafu
$ docker run -ti jszhaw/snafu
```



Snafu

Standalone mode

- call functions interactively
- batch mode with/without input pipe
- performance, robustness & correctness tests
- development

```
$ snafu
```

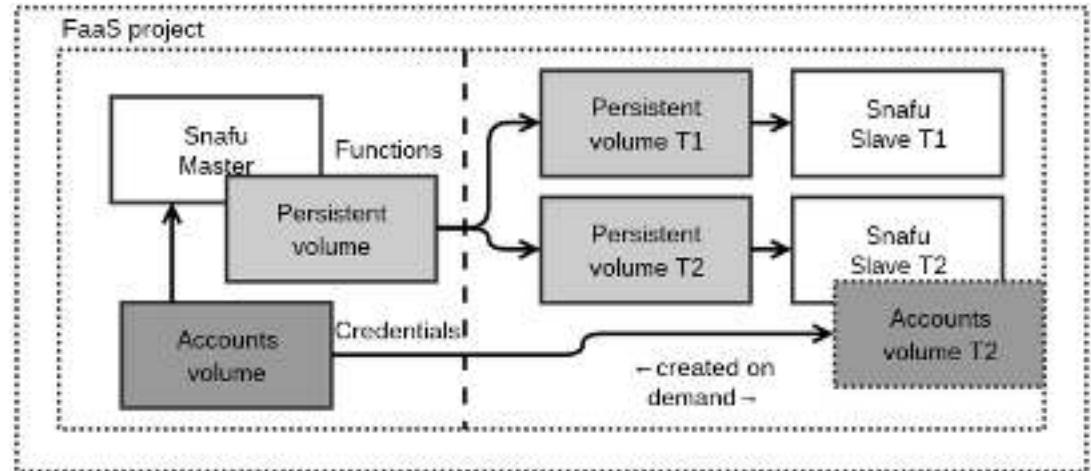
```
$ snafu -x <function> [<file/dir>]
```

```
$ snafu -l sqlite -e java -c lambda -C messaging
```

Snafu

Daemon mode (control plane)

- hosted functions
- multi-tenant provisioning
- per-tenant isolation
- compatibility with existing client tools



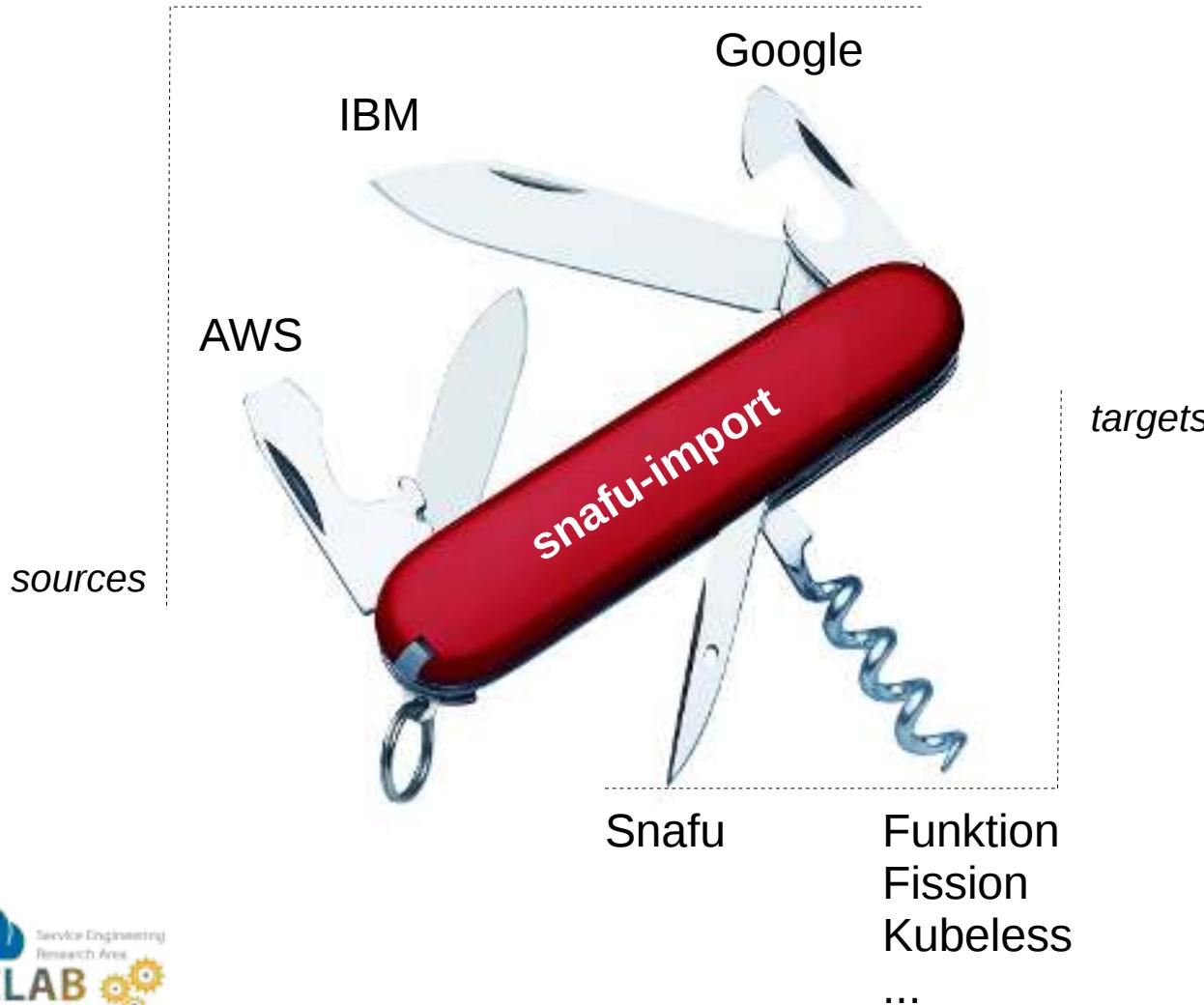
```
$ snafu-control
```

```
$ snafu-control -a aws -r -d -e docker
```

```
# snafu-accounts --add -k <k> -s <s> -e <ep>
```

Snafu

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
```

Snafu - Hands-on Time!



[pinterest.com]

Q&A / Live help session



[dribbble.com]

Further Reading and FaaS Fun

Lama, Lambackup:

- <https://arxiv.org/abs/1701.05945>

Podilizer:

- <https://arxiv.org/abs/1702.05510>

Snafu:

- <https://arxiv.org/abs/1703.07562>

Lambada

- <https://arxiv.org/abs/1705.08169>

On arXiv Analytics:



On GitHub:



[github.com/
serviceprototypinglab]

The image shows two side-by-side screenshots of arXiv preprint pages. Both pages have a light blue header with the arXiv logo and the title 'Anchored Code Analysis and Transformation for FaaS Lambda Functions' by Andrei Popescu. The left screenshot shows the page with a URL of <https://arxiv.org/pdf/1701.05945.pdf>. The right screenshot shows the page with a URL of <https://arxiv.org/pdf/1701.05945v1.pdf>. Both pages contain abstracts, author information, and sections like 'Introduction', 'Background', 'Proposed System', 'Implementation', 'Evaluation', and 'Conclusion'.