Practical Tooling for Serverless Computing

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

Dec 5, 2017 | UCC‘17 | Austin, Texas, USA
Your Tutorial Agenda

Session 1

08.00-09.30 (90’)  Tutorial Basics
   I  Serverless Foundations
   II  Function Developer Tools

Session 2

10.00-12.00 (120’) III  Function Execution Tools
   IV  Research Challenges
    Discussion
Your Tutorial Instructor

Josef Spillner <josef.spillner@zhaw.ch>

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

- sometimes, publishes on FaaS / Serverless topics
  - CARLA‘17 serverless HPC paper
  - four 2017 preprints on arXiv related to tools
  - ongoing work on tracing/debugging as well as serverless developer survey
- co-authored «Architectural Transformations in Network Services and Distributed Systems»
Your Tutorial Equipment

Required Installation

Local accounts → help yourself (git, docker, ...)
Cloud accounts → ask for tutoX account & ssh into 160.85.4.155

Supplementary Material:

Script: https://drive.switch.ch/index.php/s/Upjd0aXCypZjMnZ
Your Tutorial Equipment

Dependencies

- git, python3, ... (probably pre-installed on Ubuntu 16.04)
- python3-flask
- python3-boto3
- openjdk-8-jdk
- maven
- openjfx
- gcc
- awscli
- unzip
Part I - Serverless Foundations
Industry Perspective: JFK 2 Days Ago
Industry Perspective: Cloud Apps

[https://github.com/cncf/lindscipe, Oct’17]
Industry Perspective: FaaS

Function-as-a-Service Landscape

[Image of FaaS landscape with various tools, libraries, frameworks, and platforms]

[info] Astasia Myers, Memory Leak, Oct'17
Academic Perspective: Clouds & FaaS
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“
FaaS Process

monitoring event
sensor data
log entry
git push

... HTTP
XMPP
AMQP

max 1 per hour
triggers/actions
default params

... rules

actions (functions)

results!

... 

[openwhisk.org]

your Python/Java/... functions!

JSON plain text
FaaS in Formal Terms

• programming model
  - functions or methods in diverse programming languages
  - with specific signatures (parameters, return values)
  - sometimes, executable implementations, e.g. containers
• deployment model
  - upload of source files or compiled binaries
  - configuration of entrance handler, memory allocation, etc.
• execution model
  - time limit, e.g. 5 minutes
  - pay-per-use microbilling, e.g. per invocation + 100ms duration
• roughly equal to serverless computing: marketing term
  • for Function-as-a-Service ecosystems
FaaS in Technical Terms

- Function = elementary unit
- App/Bundle = complex unit
- Input
  - application-specific
  - context parameters, direct protocol access (if supported)
- Processing
  - up to the application
- Output
  - application-specific
  - state signalling, direct protocol access (if supported)
FaaS Positioning

Cloud Services

- Hypervisor
  - OS + container engine

- Hypervisor
  - OS + Libs + Runtimes
  - generic runtime

- VM aaS
  - App + Libs + OS

- Container aaS
  - App + Libs

- Package aaS
  - App

- Function aaS
  - Set of Functions

- Unikernel aaS
  - App
  - Library

Application

- pay-per-use
- microservices
- code-level
FaaS Positioning
# FaaS Chronology

<table>
<thead>
<tr>
<th>Industry</th>
<th>2014</th>
<th>2015</th>
<th>2016</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AWS Lambda</td>
<td>Webtask.io</td>
<td>IBM OpenWhisk</td>
<td>Fission</td>
</tr>
<tr>
<td></td>
<td>Hook.io</td>
<td></td>
<td>Azure Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lever OS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Google Cloud Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AWS Step Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Funktion</td>
<td></td>
</tr>
<tr>
<td>Academic</td>
<td></td>
<td>Dripcast</td>
<td>Open Lambda</td>
<td>PyWren</td>
</tr>
<tr>
<td>Community</td>
<td></td>
<td></td>
<td>Chatbot</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Costhat</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lambada</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Wrap-Up Part I
Part II - Function Developer Tools
Faas Synopsis: Python Examples

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

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

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

**Azure Functions:**
```python
def main():
    from AzureHTTPHelper import\n    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)
# FaaS Synopsis: JavaScript Examples

<table>
<thead>
<tr>
<th>Provider</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>exports.handler (ev, ctx, cb) → {cb(null, {});}</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>exports.fname = function(ev, cb){cb();}</td>
</tr>
<tr>
<td>Bluemix OpenWhisk</td>
<td>function main(&lt;json&gt;){return &lt;json&gt;}</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>module.exports = function(ctx, req){ctx.req = &lt;json&gt;}</td>
</tr>
<tr>
<td>Webtask.io</td>
<td>module.exports = function(cb){cb(err, &lt;json&gt;);}</td>
</tr>
<tr>
<td>- Webtask.io advanced</td>
<td>module.exports = function(ctx, cb){cb(err, &lt;json&gt;);}</td>
</tr>
<tr>
<td>Hook.io</td>
<td>module['exports'] = function fname(hook){hook.res.end(&lt;json&gt;);}</td>
</tr>
</tbody>
</table>
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:

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

```python
code level

def helloworld():
    print("Hello world.")

def main():
    helloworld()

function level

print_monad
helloworld_func

function set level

helloworld_code
main_func
main_code
```
Transformation Rules

- entry points
  - no transformation of main function
- functions definitions
  - adapt to FaaS conventions: parameters, return value
  - scan recursively for function calls
  - export as function unit including dependencies
- function calls
  - if internal, rewire
  - if input/output, replace
  - otherwise, leave unchanged
- monads
  - functional programming with side effects (i.e. input/output as side channel)
Transformation Algorithm (Java Exc.)

1. Create handle method declaration

2. If \( i = n \)
   - Where 'n' is field's number
     - Yes
       - Add to CU all methods from source class
       - Translate added classes into invokers, except method for uploading
     - No
       - If field[j] is accessible
         - Yes
           - Create line in the method declaration which assign to field[i] the certain 'get' method call of the InputType object
         - No
           - If function throws any exception
             - Yes
               - Add the method call declaration to the handler method body
             - No
FaaSification

Definition of “FaaSification“

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

\[ \text{FaaSification} := \text{code analysis} + \text{transformation} + \text{deployment} + \text{on-demand activation} \]

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

Decomposition Categories:
- static code analysis
- dynamic code analysis

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

“Lambdafication“
- targeting AWS Lambda

→ Lambda: FaaSification for Python
→ Podilizer, Termite: FaaSification for Java
  (currently limited to Lambdafication)
Lambada

Local Code

- Local Resource (e.g., File)
  - Class
    - Function/Method B
    - Function/Method/Constructor A
    - Main Function

Cloud Code

- Lambda Function
  - Lambda Function B
  - Lambda Function A

Network Resource (e.g., Shared File)

Network Resource (e.g., DB Server)

analyse functions
rewrite functions
deploy functions
rewire functions

Main Function in VM or Container
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

```python
import time
import math

level = 12
counter = 0
def fib(x):
    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:

```python
return 9
--------------
print("hello")
return 9
--------------
return {"ret": 9}
--------------
return {"ret: 9", "stdout": "hello"}
```

Stubs, via templates:

```python
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:
    def __init__(self):
        self.x = 9

    def test(self):
        return self.x * 2

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

Examples

(not shown: monads, decorators)
Podilizer
Podilizer

Examples

```java
package pkg;

class HelloWorld {
    private String greeting() {
        return "Hello, world."
    }

    public void greetWorld() {
        System.out.println(this.greeting());
    }

    public static final void main(String args[]) {
        new HelloWorld().greetWorld();
    }
}
```
public void greetWorld() {
    String awsAccessKeyId = "# your Access Key Id provided by AWS";
    String awsSecretAccessKey = "# your Secret Access Key provided by AWS";
    String regionName = "# the AWS awsRegion for created Lambda functions. Example: 'awsRegion: eu-west-2'";
    String functionName = "pkg_HelloWorld_greetWorld";
    Region region;
    AWSCredentials credentials;
    AWSLambdaClient lambdaClient;
    credentials = new BasicAWSCredentials(awsAccessKeyId, awsSecretAccessKey);
    lambdaClient = (credentials == null) ? new AWSLambdaClient() : new AWSLambdaClient(credentials);
    region = Region.getAwsRegion(Regions.fromName(regionName));
    lambdaClient.setAwsRegion(region);
    awsl.pkg.HelloWorld.greetWorld.InputType inputType = new awsl.pkg.HelloWorld.greetWorld.InputType();
    ObjectMappe objectMapper = new ObjectMapper();
    String json = "";
    try {
        json = objectMapper.writeValueAsString(inputType);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
    awsl.pkg.HelloWorld.greetWorld.OutputType outputType = null;
    try {
        InvokeRequest invokeRequest = new InvokeRequest();
        invokeRequest.setFunctionName(functionName);
        invokeRequest.setPayload(json);
        outputType = objectMapper.readValue(byteBufferToEndian(outputType),Charset.forName("UTF-8"),awsl.pkg.HelloWorld.greetWorld.OutputType.class);
    } catch (Exception e) {
    }
}
Podilizer

Examples

```java
import java.nio.ByteBuffer;
import java.nio.charset.Charset;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import pkg.*;

public class LambdaFunction implements RequestHandler<InputType, OutputType> {
    public OutputType handleRequest(InputType inputType, Context context) {
        greetworld();
        { 
            OutputType outputType = new OutputType();
            return outputType;
        }
    }

    private String greeting() {
        public static final void main(String[] args) {
            new HelloWorld().greetworld();
        }

        public static String ByteBufferToString(ByteBuffer buffer, Charset charset) {
            byte[] bytes;
            if (buffer.hasArray()) {
                bytes = buffer.array();
            } else { 
                bytes = new byte[buffer.remaining()];
                buffer.get(bytes);
            }
            return new String(bytes, charset);
        }

        public void greetworld() {
            System.out.println(this.greeting());
        }
    }
}
```
Termite

Annotations:
@Lambda(region=..., memory=..., timeout=...)

Nr. 2
Termite

Workflow

(Maven + AspectJ integration)
Wrap-Up Part II
HAVE A BREAK
Part III - Function Execution Tools
## 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>-”-</td>
<td>Service</td>
</tr>
<tr>
<td>Azure Functions</td>
<td>Node.js, C# / F#, Python, PHP, ...</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>-”-</td>
<td>OSS</td>
</tr>
</tbody>
</table>
Runtime Overview: Python Examples
## Runtime Overview: Provider Pricing

<table>
<thead>
<tr>
<th>Service</th>
<th>Monthly Free Tier</th>
<th>Cost per Call</th>
<th>Cost per Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>1e6 calls, 4e5 load</td>
<td>2.00e-7</td>
<td>1.667e-5</td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>unknown, service is in alpha stage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM OpenWhisk</td>
<td>unknown, service is in beta stage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azure Functions</td>
<td>1e6 calls, 4e5 load</td>
<td>2.00e-7</td>
<td>1.600e-5</td>
</tr>
<tr>
<td>Webtask.io</td>
<td>--</td>
<td>1.68e-7 (paid plan)</td>
<td>--</td>
</tr>
<tr>
<td>Hook.io</td>
<td>--</td>
<td>2.00e-3 (small plan)</td>
<td>--</td>
</tr>
</tbody>
</table>

![Graph showing cost-duration product for FaaS providers (Python3 runtime, compute-intensive)](image-url)
## Runtime Overview: Provider Instances

### Amazon EC2

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>CPU Credits/hour</th>
<th>Mem (CIB)</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.micro</td>
<td>1</td>
<td>3</td>
<td>0.5</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>12</td>
<td>2</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.large</td>
<td>4</td>
<td>24</td>
<td>4</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2xlarge</td>
<td>8</td>
<td></td>
<td></td>
<td>EBS-Only</td>
</tr>
</tbody>
</table>

### AWS Lambda

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1.micro</td>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>t1.medium</td>
<td>2</td>
<td>192</td>
</tr>
<tr>
<td>t1.large</td>
<td>4</td>
<td>384</td>
</tr>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>268</td>
</tr>
<tr>
<td>t2.large</td>
<td>4</td>
<td>512</td>
</tr>
</tbody>
</table>

### Lambda @ Edge

(CPU performance proportional to memory allocation)
Runtime Examples: IBM Cloud

Bluemix OpenWhisk & JavaScript/Node.js

Constraints
• code size 48 MB
• payload size 1 MB

Configuration
• runtime environment (from list)
• memory 128-512 MB; default 256
• timeout 0.1-300 s; default 60
• authoring: template or blank document
Runtime Examples: Azure Functions

Azure Functions

Constraints
- undocumented

Configuration
- runtime environment (from list)
- memory 128-1536 MB; default 256
Runtime Examples: Reality Check

OpenWhisk

- $ wsk
  does not compile

AWS Lambda

- $ az
  not scriptable

- $ aws lambda
  requires account

AWS Lambda

- $ kubeless
  breaks minikube

- $ fission
Runtime Examples: OpenLambda
Runtime Examples: Fission
Runtime Examples: Snafu

“Swiss Army Knife“ of Serverless Computing
Snafu in Commercial Clouds

APPUiO
- OpenShift atop IaaS providers or on-premise (Cloudscale, AWS, ...)
- composite application deployment w/ templates

Snafu: OpenShift template
- deploys unprivileged Docker container(s)
- object hierarchy

Template
  ImageStream
  DockerImage
  DeploymentConfig
    ImageStreamTag
  Service
  Route
    Service
    ConfigMap

... containers:
  - args:
    - --authenticator=aws
  command:
    - /opt/snafu-control
    - -s
    - /etc/snafu/snafu.ini
  image: snafu
...
Snafu in Commercial Clouds

Single-tenant mode
- unauthenticated
- single account
- multiple accounts in same instance

Multi-tenant mode
- multiple isolated accounts
- proper scale-out per active account
- selective sharing of functions and accounts from master to tenant instances
Snafu in Research Clouds

Utah, Wisconsin, South Carolina, Massachusetts, ...

- Emulab testbed
- raw access from physical level
- programmable infrastructure via GENI libraries
- >50000 experiments

Snafu: RawPC setup
- Ubuntu 16.04 base
- Snafu repository clone

Experiments:
- boot time → 5-7 min

European Grid Initiative Federated Cloud

Jülich, Fraunhofer SCAI, GRnet, INFN, Cesnet, GWDG, ...

- diverse open stacks and standards
- virtual machines, containers
- >250000 VM instances per year

Snafu: Kubernetes VM
- deployment descriptor
- based off OpenShift templates

Experiments:
- deployment time → 5 s
import geni.portal as portal
import geni.rspec.pg as pg
import geni.rspec.emulab as emulab
pc = portal.Context()

request = pc.makeRequestRSpec()

node = request.RawPC('node')
node.disk_image = 'urn:publicid:IDN+emulab.net+image+emulab-ops: \ UBUNTU16-64-STD'
node.Site('Site 1')

node.addService(pg.Execute(shell="sh", command="sudo git clone \ http://github.com/serviceprototypinglab/snafu /local/snafu"))

pc.printRequestRSpec(request)
EGI Deployment

# on local system
kubectl config set-cluster egi --server=https://HOST.fedcloud-tf.fedcloud.eu
kubectl config set-context egi --cluster=egg --user=josef
kubectl config use-context egi

wget -qc https://raw.githubusercontent.com/serviceprototypinglab/snafu/master/openshift/snafu-control-template.yaml
oc -n zhaw-test1 process -f snafu-control-template.yaml -o yaml > snafu-control-deployment.yaml
scp snafu-control-deployment.yaml YOU@HOST.fedcloud-tf.fedcloud.eu

# remotely on server
kubectl create -f snafu-control-deployment.yaml

# on local system
aws --endpoint-url https://faas.fedcloud-rt.fedcloud.eu:31000 lambda list-functions
# Tracing in Snafu

## Preliminary Results

<table>
<thead>
<tr>
<th>Trace</th>
<th>Method</th>
<th>Root Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking function calls</td>
<td>investigating stackframe</td>
<td>NO</td>
</tr>
<tr>
<td>Function execution time</td>
<td>investigating stackframe</td>
<td>NO</td>
</tr>
<tr>
<td>Network Calls</td>
<td>psutil</td>
<td>YES</td>
</tr>
<tr>
<td>Writes and reads to disk</td>
<td>psutil</td>
<td>NO</td>
</tr>
<tr>
<td>Performance metrics</td>
<td>psutil</td>
<td>NO</td>
</tr>
</tbody>
</table>

---

![Graph](graph.png)

Open Files / Open Network Connections

---

*ICCLAB SPLAB*
Wrap-Up Part III
Part IV - Research Challenges
Research Overview

CostHat

\[ c(s, \zeta) = \sum_{(s_c, n) \in w^\xi} c^\lambda(s_c, n) \]

\[ c^\lambda(e, n) = n \ast \left( \sum_{c_f \in CF^\lambda} c^\lambda_{c_f}(e) \right) + c^\lambda_{c_e}(e) \]

Second International Workshop on Serverless Computing (WoSC) 2017

Part of Middleware 2017.
Challenge: Performance

Empty function response times

- openwhisk-functions-span
- azurefunctions-functions-span
- lambda-functions-span

invocation #

time (ms)
Challenge: Performance

BBP(2000/2500) approximation of pi

- Python2
- PyPy2
- Python2 Lambda
- Python2 Snafu/out-of-process

(time (s))

(# threads)
Challenge: Performance

![Graph showing face detection processing time vs. number of threads]

- Python2
- Python2 Snafu/out-of-process
- Python2 Lambda
- Python2 Lambda parallel
- Lambda timeout barrier
Challenge: FaaSification
Challenge: Deep FaaSification

Precipitation forecast function splitting into $F_1$ and $F_2$

- $F_1$
- $F_2$

Lambda/OpenWhisk/Azure Functions limit
Cloud Functions limit

Time (s)

Lines in $F_1$
Challenge: Debugging
Challenge: Debugging
Challenge: Debugging
Challenge: Ecosystem

IDE
- binding to providers
- binding to transformer tool

transform
publish

FaaS Marketplace
- social connections
- instant execution
- scalable cloud backend
- easy integration

FaaS Transformer Tool
- input adapter: Java
- input adapter: Python
- input adapter: JavaScript
- output adapter: IBM OpenWhisk
- output adapter: AWS Lambda
- output adapter: Azure Functions
- output adapter: Google Cloud Functions

Future role:
FaaS transformation service operator

Cloud (FaaS) provider
hosted function

Application engineer
subscribe

Future role:
FaaS marketplace operator
Challenge: Ecosystem

Top Picks

- **Hello**
  - A bytecode-compiled Java function.

- **fib_lambda**
  - A Python function. This function has been imported from AWS Lambda.

- **sleep**
  - A Python function.

- **fib_so**
  - A compiled C function.

- **helloworldpy2**
  - A Python function. Reliable

- **lambdaenv**
  - A Python function. This function has been imported from AWS Lambda.

- **fib_lambda**
  - A Python function. This function has been imported from AWS Lambda.

- **sleep**
  - A Python function. Reliable

- **Fib**
  - A bytecode-compiled Java function.

- **localfib**
  - A Python function.

- **test_so**
  - A compiled C function.

- **Hello**
  - A bytecode-compiled Java function.
Wrap-Up Part IV

Deep
Performance
Challenge
Debugging
Faasification

ICCLAB
SPLAB
Discussion
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]