SPLAB
transcript

type:
tutorial

distribution:
public

status:
final

initiative:
Service Tooling

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

Josef Spillner
Zurich University of Applied Sciences, School of Engineering
Service Prototyping Lab (blog.zhaw.ch/icclab/)
8401 Winterthur, Switzerland
josef.spillner@zhaw.ch

June 9, 2017

Learning Objective

Learn on a practical level about different FaaS programming conventions, runtimes, service providers and tools. Use some of the tools by yourself to FaaSify simple Python applications.

1 Using Lambada

Snafu extracts your functions from code files.

Obtain Lambada directly from its source repository. You will furthermore need Python 3.5 or more recent installed.

Listing 1: Obtaining Lambada

```
git clone https://gitlab.com/josefspillner/lambada.git
```

Start by creating a file `temperature.py`. It contains one function and one class which together form your «temperature application» (Listing 2).

Listing 2: Temperature application

```
def temperature():
    return 42.23
class CityTemperature:
    def temperaturecity(self, city):
        self.city = city
        return 99
```

Now use Lambada to transform the contents into FaaS units which can still be executed locally but are ready to be deployed in AWS Lambda (Listing 3).

Listing 3: FaaSification process

```
./lambada --local temperature.py
```

Then verify the generated Python file and notice its structure. Use any editor of your choice on the newly generated file `temperature_lambdafied.py`.

Afterwards, check the functionality in the Python console. Run the following (Listing 4).

Listing 4: FaaSified temperature application

```
python3
>>> import test_lambdafied
>>> dir(test_lambdafied)

>>> test_lambdafied.temperature()
>>> test_lambdafied.temperature_remote(None, None)
>>> test_lambdafied.netproxy_handler({"d": "{}", "name": "
    temperaturecity", "args": ["Winti"], "classname": "
    CityTemperature"}, None)
```

Finally, it is time to deploy the code to AWS Lambda and execute it there. You can run the command twice to verify that in the second case no redeployment happens (Listing 5). In parallel, you can verify the deployments in the AWS Lambda Dashboard at `https://<orgname>.signin.aws.amazon.com/console`.

Listing 5: FaaSified temperature application

```
./lambada temperature.py
./lambada temperature.py # nothing happens, already deployed
```

2 Using Snafu

Snafu runs your functions in code files.

Start off with obtaining Snafu from its source repository.

Listing 6: Obtaining Snafu

```
git clone https://github.com/serviceprototypinglab/snafu
```

You can directly run Snafu without any configuration. It ships with a number of default functions in various languages. Notice how on first startup, the interpreters for non-Python functions are compiled first. This means that while the first startup is a bit slow, the second and subsequent ones are quote fast (Listing 7).

Listing 7: Running Snafu

```
./snafu
# type Ctrl+D
./snafu # everything compiled
# type Ctrl+D again
```

Now run Snafu in C mode, meaning that the default interpreter is for functions implemented in C. Select the sample C Fibonacci function and execute it with a low parameter n (Listing 8).

Listing 8: Executing a C function

```
./snafu -e c
# type function: fib_so.handler
# type argument: 10
# type Ctrl+D
```

Now remember your previous example functions used with Lambada. Obviously, Snafu can interpret this file as well. Execute Snafu on this file and explore the execution (Listing 9).

Listing 9: Executing a Python function

```
./snafu .../path-to-lambada/temperature.py
```

For testing functions prior to deployment to AWS, it is convenient to use Snafu's control plane mode. In this mode, the default calling convention is the same as in AWS Lambda (Listing 10).

Listing 10: Using the control plane

```
./snafu-control .../path-to-lambada/temperature.py
# notice how no methods are selectable
# type Ctrl+D
# add to source: lambda_handler(event, context):return str(
    temperature())
# & run snafu-control command again
# open a second terminal, and run:
aws --endpoint-url http://localhost:10000 lambda invoke --
    function-name test.lambda_handler --payload '{"event":
    ""}' /tmp/_out
cat /tmp/_out
```

3 Using Fission

Note: You will need to have Kubernetes installed with minikube + kubectl. Ensure that the minikube context is used by kubectl. Furthermore, ensure that your system user is in the group `docker`.

Listing 11: Fission script

```
# prepare
kubectl config use-context minikube
kubectl create -f http://fission.io/fission.yaml
kubectl create -f http://fission.io/fission-nodeport.yaml
export FISSION_URL=http://$(minikube ip):31313
export FISSION_ROUTER=$(minikube ip):31314
```

```
# initiate
fission env create --name python --image fission/python-env
curl https://raw.githubusercontent.com/fission/fission/
    master/examples/python/hello.py > hello.py
# explore & deploy
cat hello.py
fission function create --name hello --env python --code
    hello.py
# output: function 'hello' created
fission route create --method GET --url /hello --function
    hello
# output: trigger '35ad2fae-****-****-****-679d755fde8f'
    created
curl http://$FISSION_ROUTER/hello
# user interface
kubectl create -f https://raw.githubusercontent.com/fission/
    fission-ui/master/docker/fission-ui.yaml
# output: deployment "fission-ui" created
# output: service "fission-ui" created
xdg-open http://$(minikube ip):31319
```

4 Using OpenLambda (tentative)

Listing 12: OpenLambda script

```
git clone https://github.com/open-lambda/open-lambda
cd open-lambda
sudo apt-get install libpython2.7-dev
sudo make
bin/admin new -cluster pyparis
# returns lots of JSON
bin/admin workers -cluster pyparis
# output: Started worker: pid ...
bin/admin status -cluster pyparis
# output: Worker Pings: ...
cat quickstart/handlers/hello/lambda_func.py
# ... minimal Lambda-signature Python function
cp -r quickstart/handlers/hello pyparis/registry
curl -X POST localhost:8080/runLambda/hello -d '{"name": "
    Alice"}'
# -> permission denied!?
```