KNIX MicroFunctions Tutorial

Paarijaat Aditya

KNIX Team

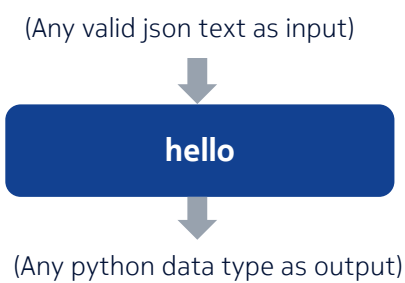
Agenda

- Hello world: deploying a single python function
- Function with non-standard dependency
- Workflows with two functions in a sequence
- Workflow with parallelly executing functions
- Resize image
  - KNIX's built-in key-value store
  - Package custom dependencies as zip

Hello world

Concepts:

1. Functions
2. Event/input
3. Context
4. Code editor
5. Test



"abc" [1,2,3] {"key1": 5}

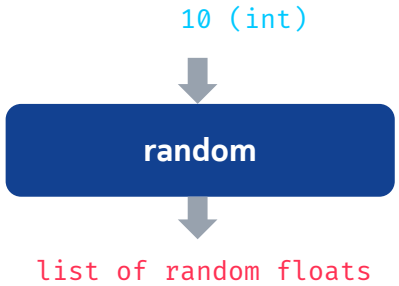
```
def handle(event, context):
  print("Hello world")
  print(str(event))
  print(str(type(event)))
  return event
```

event = function input (a python data type depending on the input json text)

context = KNIX MicroFunctions api object

- get(key)
- put(key, value)
- delete(key)
- add_workflow_next(name, value)
- get_instance_id()

Non standard dependencies




```
import numpy as np
```

Non-standard dependency

```
def handle(event, context):
    a = np.random.normal(size=event)
    b = a.tolist()
    print(str(b))
    return b
```

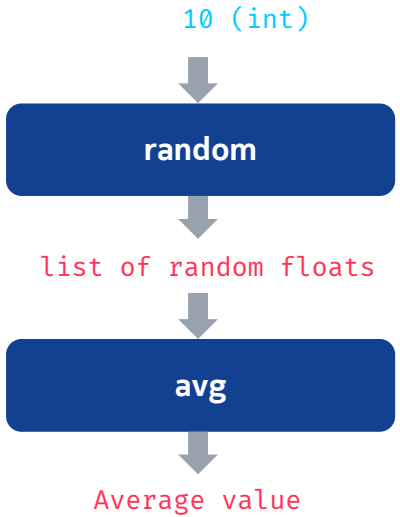
Concepts:

1. requirements.txt
2. Workflows
3. Invocation using url

Workflow with 2 functions

Concepts:

1. Function-to-function communication



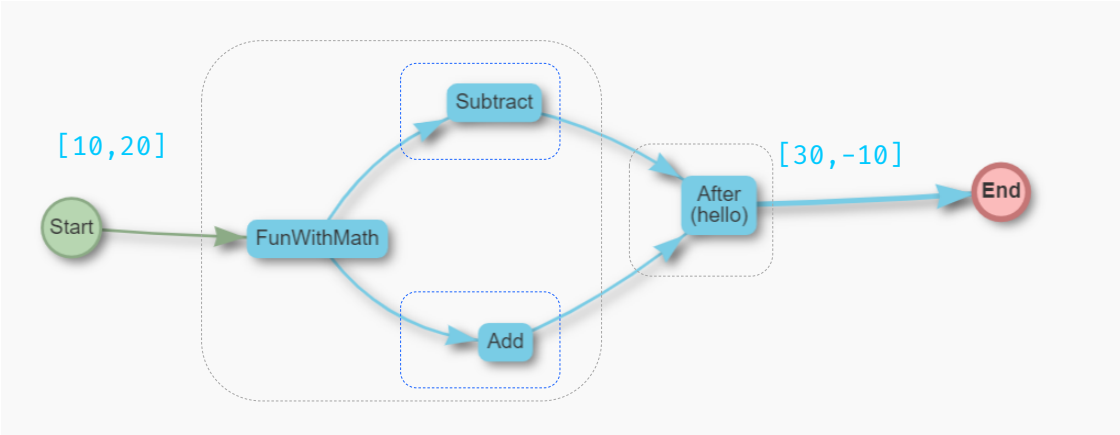

```
def handle(event, context):
    avg = sum(event)/len(event)
    print('avg = ' + str(avg))
    return avg
```

Invoking functions in parallel

Concepts:

1. Amazon States Language Parallel state

```
def handle(event, context):
    time.sleep(8)
    return event[0] - event[1]
```

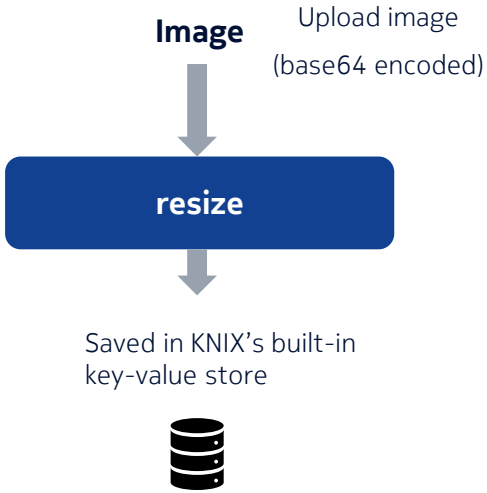



```
def handle(event, context):
    return event[0] + event[1]
```

Resize image example:

Concepts:

1. Non standard dependency added as a zip file
2. KNIX's built in key-value store



Thank You!

https://github.com/knix-microfunctions/knix

knix.slack.com

Useful commands

Install Pillow in the current folder:

```
pip3 install pillow –t .
```

Or

```
docker run -it --rm -u $(id -u):$(id -g) -v $(pwd):/temp -w /temp python:3.6 pip3 install pillow -t .
```

Zip the contents of the current folder

```
zip –r ../resize.zip .
```

```
import json, base64, io
from PIL import Image

def handle(event, context):
  filename = event['Filename']
  print('resize ' + filename)
  img = io.BytesIO(base64.b64decode(event['EncodedFile']))

  with Image.open(img) as image:
    image.thumbnail(tuple(x/2 for x in image.size))
    buf = io.BytesIO()
    image.save(buf, format=image.format)

    resized_name = filename+'_resize.jpg'
    if context != None:
      context.put(resized_name, base64.b64encode(buf.getvalue()).decode())
      print(resized_name + ' written to datalayer')

  event['Resized'] = filename+'_resize.jpg'
  event['EncodedFile'] = ''
  return event
```

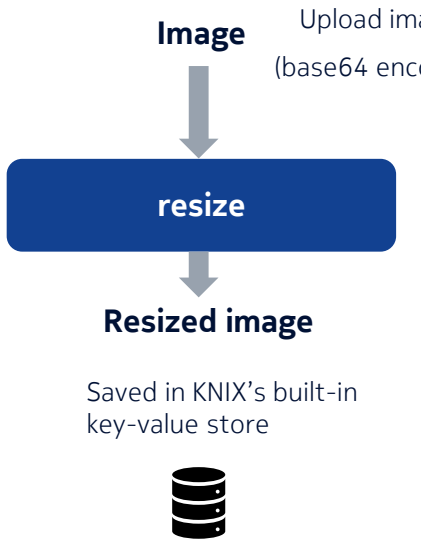
Non standard dependency to be added as a zip file

Decode file

Resize file

Put in key-value store

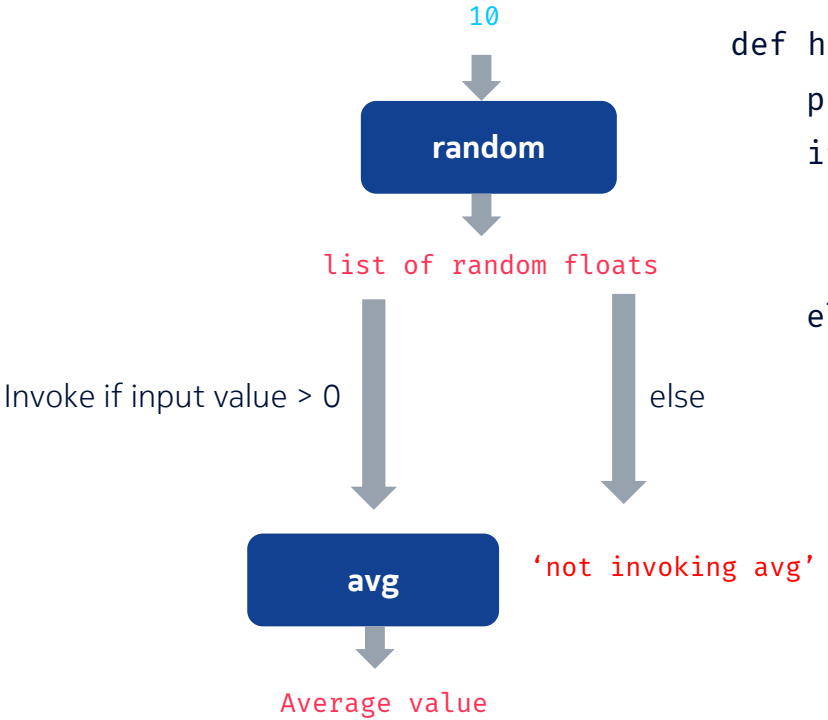
Return the name of resized file



```
#!/usr/bin/env python
import base64, sys, json, requests, time

filename = sys.argv[1]
print('sending ' + filename)
with open(filename, "rb") as image_file:
    encoded_file = base64.b64encode(image_file.read()).decode()

input_dict = {'Filename': filename, 'EncodedFile': encoded_file}
with open('request.json', 'w') as f:
    json.dump(input_dict, f)

urlstr = 'https://wf-mfn1-7c48bdba03c5d21d089ecad68bfe279f.mfn.knix.io:443'
t1=time.time()
r = requests.post(urlstr, json=input_dict, verify=False)  # invoke workflow
t2=time.time()
diff=(t2-t1)
print(diff, r.url, r.status_code, r.reason, r.text)
```

Client-side code to send an image

Conditional invocations




```
def handle(event, context):
    print 'random'
    if event > 0:
        a = np.random.normal(size=event)
        context.add_workflow_next("avg", a.tolist())
    else:
        print 'not invoking avg'
```

Concepts:

1. Amazon States Language Choice state
2. KNIX's dynamic next