Lesson 5

Using Python

https://www.nimbella.com

Plan

- Creating Nimbella actions
  - using the REPL
  - doctest
- Nimbella support
  - Redis
  - Bucket
- Libraries
  - Virtualenv

Python Actions

- Select the Python runtime: `python:3`
    - with extension `.py` (defaults to `python:3`)
    - with `--kind python:3` or `--kind python:3ai`
        - also `python:2` but forget it, please!
    - with `.python-3.zip` or `.python-3ai.zip`

Single File Python Action

- Entry point:
  `def main(args):`
  - `args` is a dictionary of parameters
  - result of `json.loads(<input>)`
- Logs
  - with `print()`
- Returns:
  - a dictionary too
  - then serialized in json with `json.dumps(<output>)`

Hello in Python

```python
def main(args):
    name = args.get("name", "world")
    print("name:", name)
    return {
        "hello": name
    }
```

Local Testing with Python REPL

```
## Local Testing
# deploy hello
mkdir -p py/packages/default
cp src/hellopy.py py/packages/default/hellopy.py

# Manual test on the repl
PYTHONPATH=py/packages/default PYTHONDONTWRITEBYTECODE=1 python3
from hellopy import *

main({})
# {'hello': 'world'}

main({"name":"Mike"})
# {'hello': 'Mike'}

# exit
```

Python doctest

- embed the test in a python "doc comment"
- prefix input with `>>>`
- leave output as is

```
""" Test main
>>> main({})
{'hello': 'world'}
>>> main({"name":"Mike"})
{'hello': 'Mike'}
"""
```

Self-testing Python Action

- Add this to the end:

```python
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Hello with tests in Python

```python
def main(args):
    """ Test main
    >>> main({})
    name: world
    {'hello': 'world'}
    >>> main({"name":"Mike"})
    name: Mike
    {'hello': 'Mike'}
    """
    name = args.get("name", "world")
    print("name:", name)
    return { "hello": name }
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Test and deploy our hello

```
## Test and deploy our hello
cp src/hellopy2.py py/packages/default/hellopy.py

# testing (WITHOUT __pycache__)
PYTHONDONTWRITEBYTECODE=1 python3 py/packages/default/hellopy.py
# no news is a good news

# deploying
nim project deploy py
nim action invoke hellopy
nim activation logs
nim action invoke hellopy -p name Mike
nim activation logs
```

Using Redis

- Use nimbella sdk for Python:
  - `pip3 install nimbella`
- Local Test tips (see Lesson 4):
  - Start local redis and set environment variables

```
docker run --name redis --rm -p 6379:6379 -d redis --requirepass password
export __NIM_REDIS_IP=127.0.0.1
export __NIM_REDIS_PASSWORD=password
```

Python peculiarities

- `db = nimbella.redis()`
  - there is no async

```
db.set(<key, <value>)
db.get(<key>)
db.del(<key>)
```

- everything is stored as array of bytes
  - hence `decode('utf-8')` to use it as string

Address Book in Python

- From Lesson 2:
  - `get.py` read a key
  - `set.py` set a key
  - `rem.py` delete a key
    - changed name, `del` is a reserved word in python!!!
  - `all.py` returns all the key
    - well, first page actually

`set.py`:

```python
# set.py
import nimbella
import json

def main(args):
    db = nimbella.redis()
    key = "address:"+args.get("name", "")
    value = json.dumps({
            "name": args.get("name", ""),
            "company":  args.get("company", ""),
            "phone": args.get("phone", "")
    })
    return { "body": db.set(key, value) }
```

`get.py`:

```python
import nimbella, json
def main(args):
    db = nimbella.redis()
    key = "address:"+args.get("name", "")
    value = db.get(key).decode('utf-8')
    return { "body": json.loads(value) }
```

`rem.py`:

```python
import nimbella
def main(args):
    db = nimbella.redis()
    key = "address:"+args.get("name", "")
    return { "body": db.delete(key) }
```

`all.py`:

```python
import nimbella
import json

def main(args):
    db = nimbella.redis()
    keys = db.keys("address:*")
    data = db.mget(keys)
    res = [json.loads(i.decode('utf-8')) for i in data]
    return {
        "body": res
    }
```

Local Test with Redis

```
source init.src
mkdir py/packages/addr
cp -v src/{all,get,set,rem}.py py/packages/addr/
PYTHONDONTWRITEBYTECODE=1 PYTHONPATH=py/packages/addr python3
import set
set.main({"name":"Mike","company":"Nimbella","phone":"392"})
import get
get.main({"name":"Mike"})
import all
all.main({})
import rem
rem.main({"name":"Mike"})
all.main({})
```

Frontend: `App.svelte` from Lesson 2

```
# generating front-end
npx degit sveltejs/template py/web
# see Lesson 2 for App.svelte
cp src/App.svelte py/web/src/App.svelte
# configuring web deployment
echo "public" >py/web/.include
echo -e "bucket:\n strip: 1" >py/project.yml
# deploy and test
nim project deploy py
# check in the web browser
```

Buckets

- Get access to the bucket:

```python
import nimbella
bucket = nimbella.storage()
```

- Access to a "blob" (file) by `name` and upload/download, etc

```python
blob = bucket.blob(name)
blob.upload_from_string(data)
```

Blob Summary:

- `blob.exists()`: test whether a file exists in the store
- `blob.download(options)`: return the contents of a file in the store
- `blob.save(data)`: store contents from memory into the file on the store
- `blob.delete()`: deletes the file from the store
- `blob.copy(destination)`: copies the file within the store
- `blob.move(destination)`: moves the file within the store
- `blob.getSignedUrl(config)`: gets a signed URL

bucket.py:

```python
# bucket.py
import nimbella
def main(args):
    name = args.get("name", "")
    data = args.get("data", "")
    try:
        bucket = nimbella.storage()
        print(bucket)
        blob = bucket.blob(name)
        blob.upload_from_string(data)
        print(blob)
        return { "result": "ok" }
    except Exception as e:
        return { "error": str(e)}
```

Test Bucket Creation in Python

```
# Test Bucket Creation in Python
cp src/bucket.py py/packages/default/bucket.py
nim project deploy py
nim object list
nim action invoke bucket -p name hello -p data world
nim activation logs
nim object list
nim object get hello .
cat hello
```

Python multifile action

- Put in a folder with:
  - `__main__.py` as the main file
    - cannot be changed
  - `def main(args):` as endpoint
    - can be changed with options
- Alternatively, build a `.zip`:
  - name: `.<runtime>.zip`
  - currently, `<runtime>` is `.python-3`, `python-3ai`
  - use `python-2` ONLY AS A LAST RESORT and migrate soon

Render Embeded Image (see Lesson 3)

```python
# Render Embeded Image (see Lesson 3)
import base64, os

def main(args):
    file = "%s/%s" % (os.path.dirname(__file__), "hello.png")
    with open(file, "rb") as f:
        img = f.read()
    return {
        "body": base64.b64encode(img).decode("utf-8"),
        "headers": {
            "Content-Type": "image/png"
        }
    }
```

Deploy Multifile Action

```
## Deploy Multifile Action
mkdir -p py/packages/default/image
cp src/image.py py/packages/default/image/__main__.py
cp src/hello.png py/packages/default/image/hello.png
nim project deploy py
nim action get image --url
```

Try Image Resize on REPL

```
## Image Resize on REPL
# start python3
# load image
from PIL import Image
img = Image.open("src/hello.png")
img.size

# calculate new dimension
w = 150
ratio = img.size[1] / img.size[0]
h = int(ratio * w)

# resize image
img1 = img.resize((w,h), Image.ANTIALIAS)
img1.size
```

resize.py:

```python
import base64, os, io
from PIL import Image
def main(args):
    file = "%s/%s" % (os.path.dirname(__file__), "hello.png")
    img = Image.open(file) # read image
    w = int(args.get("w", "100")) # new width
    ratio = img.size[1] / img.size[0]
    h = int(ratio * w) # new height
    res = img.resize((w,h), Image.ANTIALIAS)
    buf = io.BytesIO()
    res.save(buf, format="PNG") # save image in a buffer
    return { # encode base64 after decoding buffer as a string
        "body": base64.b64encode(buf.getvalue()).decode("utf-8"),
        "headers": { "Content-Type": "image/png" }
    }
```

Test and Deploy resize

```
## Test and Deploy resize
# in a folder outside of packages
mkdir -p py/resize
cp src/resize.py py/resize/__main__.py
cp src/hello.png py/resize/hello.png
# let's zip by ourselves
cd py/resize
zip -r ../packages/default/resize.python-3.zip *
cd ../..
nim project deploy py
nim action get resize --url
# error!
nim activation logs
# missing module
```

We need a virtualenv!

- It is a Python environment with extra libraries.
  - Create and activate with:

```
virtualenv virtualenv
source virtualenv/bin/activate
```

  - Then add your libraries

```
pip3 install Pillow
```

- Must be a folder `virtualenv` within your `.python-3.zip`

But it must be for linux!

- Use the runtime itself to build the virtualenv

```
cd <your-action-folder>
docker run -ti -v $PWD:/mnt --entrypoint=/bin/bash openwhisk/actionloop-python-v3.7
```

- `-ti` interactive terminal
- `--entrypoint=/bin/bash` use the shell
- `-v $PWD:/mnt` make the current directory availabe

Building a virtualenv with Pillow

```
## Building a virtualenv with PIL
cd py/resize
ls
# entering in the image and creating the virtualenv
docker run -ti -v $PWD:/mnt --entrypoint=/bin/bash openwhisk/actionloop-python-v3.7
cd /mnt
ls
virtualenv virtualenv
source virtualenv/bin/activate
pip3 install Pillow
exit
# zipping and deploying
zip -r ../packages/default/resize.python-3.zip *
cd ../..
nim project deploy py
nim action get resize --url
```