Maryam Nadhim Edam, EDAM

Practice 2: Implementing face recognition using Functions-as-a-Service

Development and production environment

Windows Subsystem for Linux Windows Subsystem for Linux is a feature of Windows that allows developers to run a Linux environment without the need for a separate virtual machine or dual booting. version:WSL 2

Linux distribution: Ubuntu

DOCKER VERSION 20.10.24

Objectives of the practice:

• Install and deploy a tool for container orchestration: Kubernetes.

• Deploy the functionality of the function catalogue and functions service through OpenFaaS.

• Deploy different available functions for FaaS for face recognition.

• Implement a scalable function that would become a component for biometric identitication of users based on face images.

Installation

For this we need the following in terms of platforms/tools to install:

• Install kubernetes (minikube https://minikube.sigs.k8s.io/docs/start/#what-youll-need).

To install the latest minikube stable release on x86-64 Linux

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/  minikube-linux-amd64

 sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Start your cluster From a terminal with administrator access (but not logged in as root), run:

```
 minikube start
```

• Install a RAS platform: OpenFaaS on top of Kubernetes.

Install OpenFaaS on Kubernetes using arkade.

Arkade is an App installer for Kubernetes. It relies on Helm3 and Kubernetes, and eases and speeds up the installation of over 50 apps. We will use arkade to install OpenFaaS.

To install and run arkade we first need to run minikube:

```
minikube start
```

Then, we install arkade

```
curl -sLS https://get.arkade.dev |sudo sh
```

Then, we install openfaas using arkade:

```
arkade install openfaas
```

After the installation has completed, you will receive the commands you need to run, to log in and access the OpenFaaS Gateway service in Kubernetes.

```
Info for app: openfaas
# Get the faas-cli
curl -SLsf https://cli.openfaas.com | sudo sh

# Forward the gateway to your machine
kubectl rollout status -n openfaas deploy/gateway
kubectl port-forward -n openfaas svc/gateway 8080:8080 &

# If basic auth is enabled, you can now log into your gateway:
PASSWORD=$(kubectl get secret -n openfaas basic-auth -o
jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
echo -n $PASSWORD | faas-cli login --username admin --password-stdin
faas-cli store deploy figlet
faas-cli list
```

You can get this message again at any time with arkade info openfaas.

The kubectl rollout status command checks that all the containers in the core OpenFaaS stack have started and are healthy.

The kubectl port-forward command securely forwards a connection to the OpenFaaS Gateway service within your cluster to your laptop on port 8080. It will remain open for as long as the process is running, so if it appears to be inaccessible later on, just run this command again.

The faas-cli login command and preceding line populate the PASSWORD environment variable. You can use this to get the password to open the UI at any time.

We then have faas-cli store deploy figlet and faas-cli list. The first command deploys an ASCII generator function from the Function Store and the second command lists the deployed functions, you should see figlet listed.

print the admin password for login into the OpenFaaS gateway.

```
 kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo
```

Copy this password to use it to log into the UI.

Now we can open a browser to http://127.0.0.1:8080/ui/ and log in using username admin and the password we just copied.

How to deploy an available function as a service

OpenFaaS has a function template store with some functions already available. This store is implemented with a JSON manifest, which is kept in a public repository on GitHub. Pull Requests (PRs) can be sent to the file to extend and update it, and companies can even have their own stores. The Function Store can be accessed via the CLI using the root command faas-cli store. From there, you can search for a function with faas-cli store list and deploy the one you want with faas-cli store deploy.

A full list of the functions avilable can be obtained with:

```
$ faas-cli store list
```

In order to search for some topic, you might use grep. For example, for searching for face recognition functions:

```
$ faas-cli store list | grep face
face-detect-pigo      esimov       Face Detection with Pigo
face-detect-opencv    alexellis    face-detect with OpenCV
face-blur             esimov       Face blur by Endre Simo
```

We can see there are two functions for face recognition already available in the OpenFaaS Template Store. We can find more information on them with faas-cli store inspect:

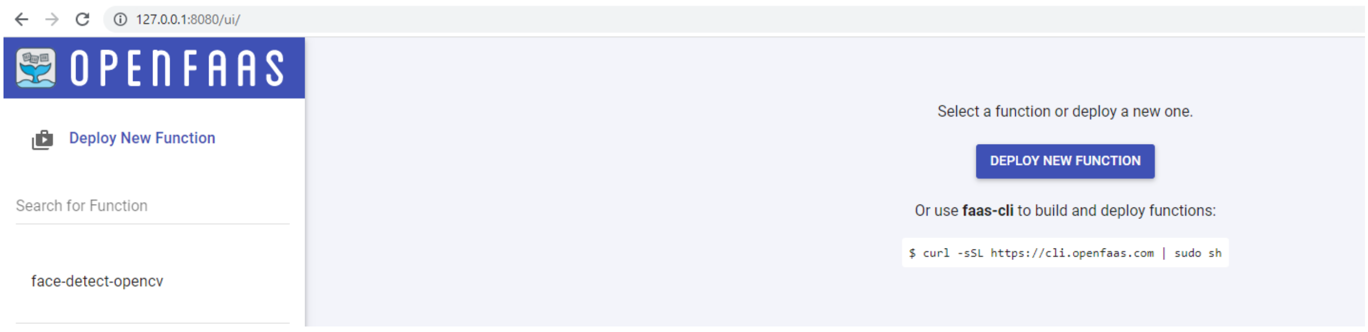
```
$ faas-cli store inspect face-detect-pigo
Title:       Face Detection with Pigo
```

```
Author:      esimov
Description:
Detect faces in images using the Pigo face detection library. You provide an
image URI and the function draws boxes around the detected faces.

Image:    esimov/pigo-openfaas:0.1
Process:
Repo URL: https://github.com/esimov/pigo-openfaas
Environment:
-  output_mode: image
-  input_mode:  url

Labels:
-  com.openfaas.ui.ext: jpg
```

to deploy a function:

```
$ faas-cli store deploy face-detect-opencv

Deployed. 202 Accepted.
URL: http://127.0.0.1:8080/function/face-detect-opencv
```

Here you get the URLs of the function. You can also see it now in the GUI OpenFaaS portal: http://127.0.0.1:8080/ui/.

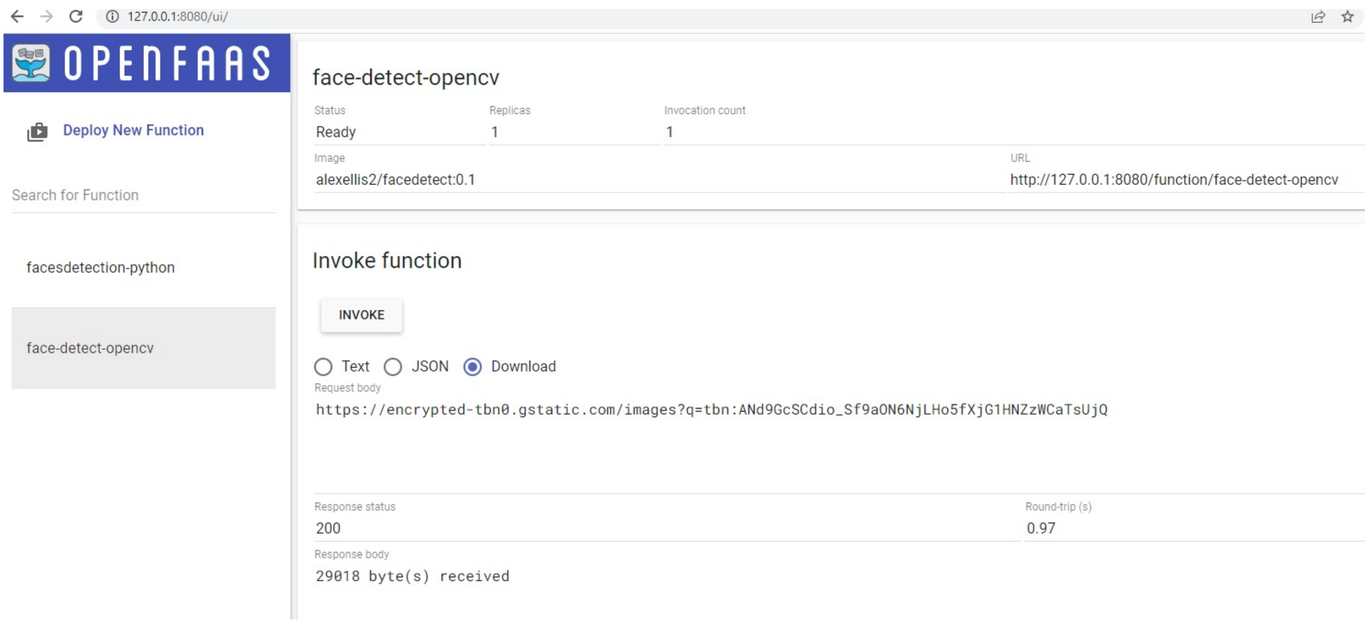


Select a function or deploy a new one.

Or use **faas-cli** to build and deploy functions:

```
$ curl -sSL https://cli.openfaas.com | sudo sh
```

You can run the functions in the UI by entering an URL for an image in the Request body field as shown in the following capture:



face-detect-opencv

Status: Ready
Replicas: 1
Invocation count: 1
Image: alexellis2/facedetect:0.1
URL: http://127.0.0.1:8080/function/face-detect-opencv

Invoke function

Text JSON [✓ Download]

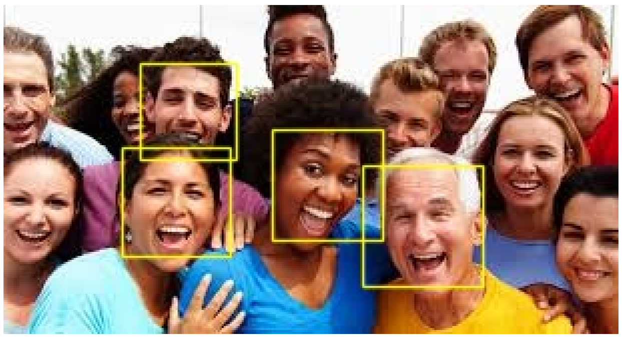
Request body

```
https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSCdio_Sf9aON6NjLHo5fXjG1HNZzWCaTsUjQ
```

Response status: 200
Round-trip (s): 0.97

Response body

```
29018 byte(s) received
```



Developing own functions for FaaS

for the OpenFaaS platform example, the function would be created in this way:

```
$ faas-cli new --lang python3-debian facesdetection-python
```

This creates three files for you:

```
facesdetection-python/handler.py
facesdetection-python/requirements.txt
facesdetection-python.yml
```

The handler.py file contains your code that responds to a function invocation:

```
import cv2
import numpy as np
import urllib.request
import base64
import json
```

```
class Response:
    def __init__(self, faces):
        self.Faces = faces

def detect_faces(image):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    return faces

def handle(req):
    data = req.strip()

# Fetch the image from the URL
try:
    response = urllib.request.urlopen(data)
    img_data = response.read()
except Exception as e:
    return f"Error fetching image from URL: {str(e)}"

# Decode the image
nparr = np.frombuffer(img_data, dtype=np.uint8)
image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

# Detect faces in the image
faces = detect_faces(image)

# Draw rectangles around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Encode the image back to base64
_, encoded_img = cv2.imencode('.jpg', image)
encoded_image = base64.b64encode(encoded_img).decode('utf-8')

# Convert the faces array to a list
faces_list = faces.tolist()

# Create a response object
response = Response(faces_list)
response.ImageBase64 = "data:image/jpeg;base64," + encoded_image  # Add
'data:image/jpeg;base64,' prefix


# Convert the response to JSON
response_json = json.dumps(response.__dict__)

return response_json
```

The requirements.txt file can be used to install pip modules at build time. Pip modules add support for add-ons like MySQL or Numpy (machine learning).

```
numpy
opencv-python-headless
urllib3
```

The .yml contains information on how to build and deploy your function:

```
version: 1.0
provider:
name: openfaas
gateway: http://127.0.0.1:8080
functions:
facesdetection-python:
    lang: python3-debian
    handler: ./facesdetection-python
    image: maryamed14/facesdetection-python:latest
```

lang: The name of the template to build with.

handler: The folder (not the file) where the handler code is to be found.

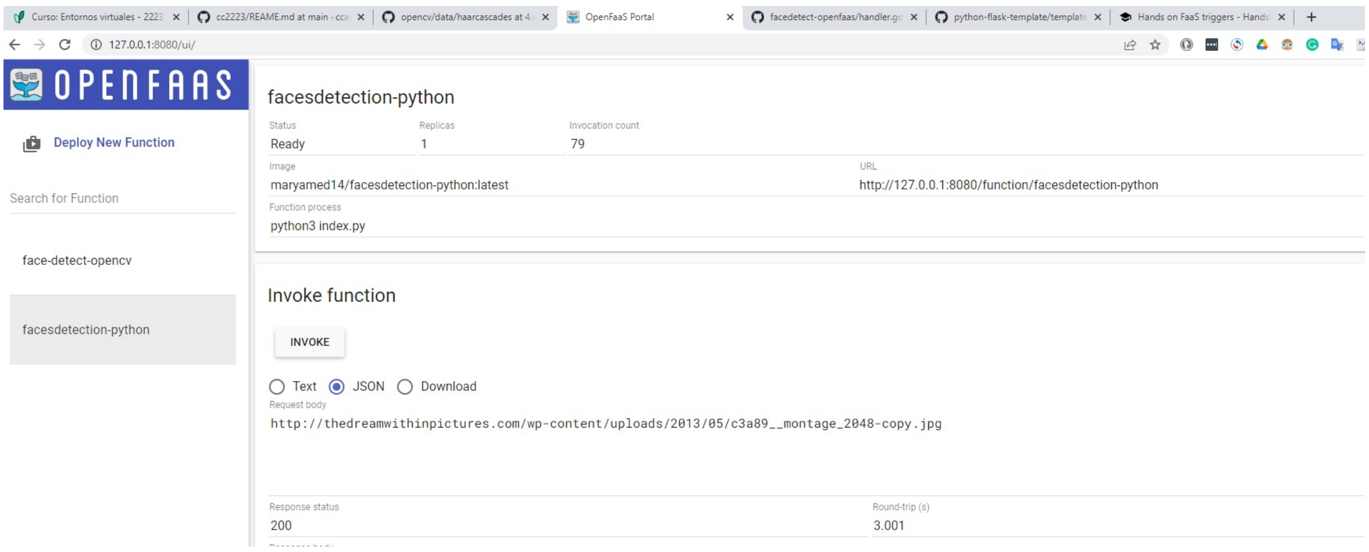
image: The Docker image name to build with its appropriate prefix for use with docker build / push. It is recommended that you change the tag on each change of your code, but you can also leave it as latest.

there are three parts to getting your function up and running both initially and for updating:

faas-cli build: Create a local container image, and install any other files needed, like those in the requirements.txt file. faas-cli push: Transfer the function’s container image from our local Docker library up to the hosted registry. faas-cli deploy: Using the OpenFaaS REST API, create a Deployment inside the Kubernetes cluster and a new Pod to serve traffic. All of those commands can be combined with the faas-cli up command for brevity both initially and for updating.

```
$ faas-cli up -f facesdetection-python.yml
```

a few moments, and then will see a URL printed http://127.0.0.1:8080/function/facesdetection-python



facesdetection-python

Status: Ready
Replicas: 1
Invocation count: 79
Image: maryamed14/facesdetection-python:latest
URL: http://127.0.0.1:8080/function/facesdetection-python
Function process: python3 index.py

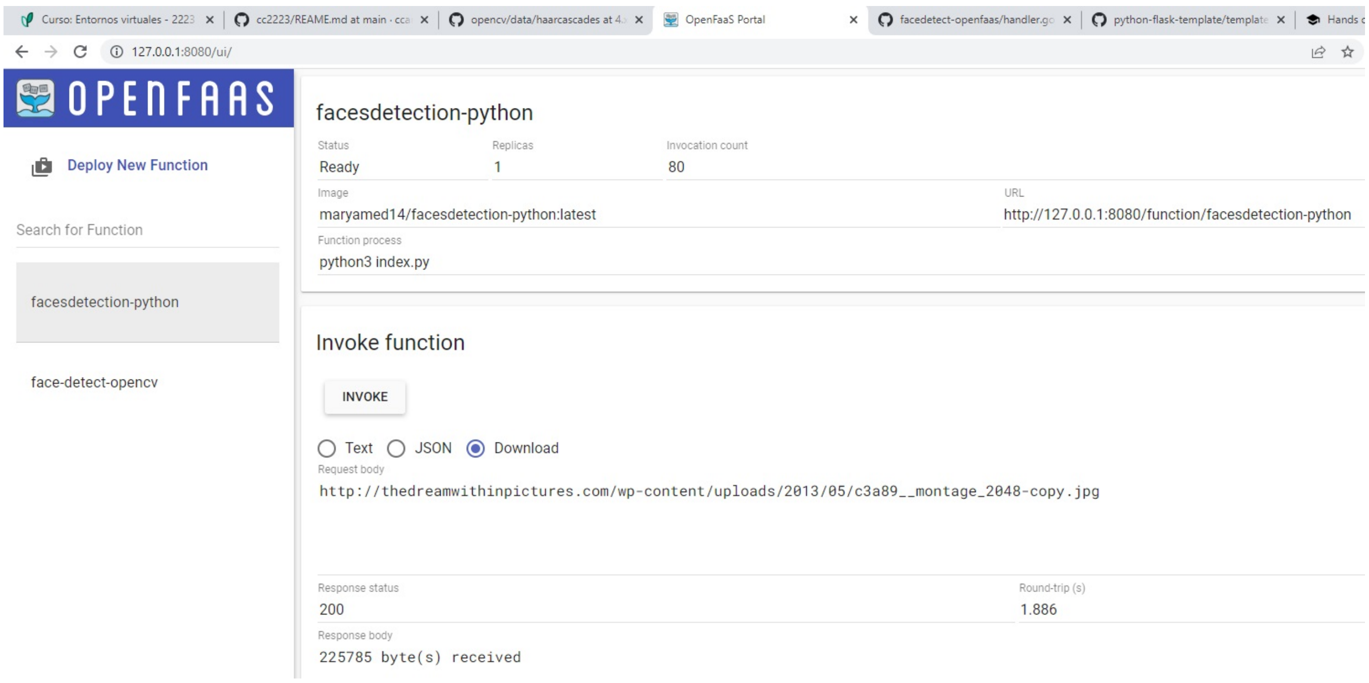
Invoke function

Text [✓ JSON] Download

Request body

```
http://thedreamwithinpictures.com/wp-content/uploads/2013/05/c3a89__montage_2048-copy.jpg
```

Response status: 200
Round-trip (s): 3.001



facesdetection-python

Status: Ready
Replicas: 1
Invocation count: 80
Image: maryamed14/facesdetection-python:latest
URL: http://127.0.0.1:8080/function/facesdetection-python
Function process: python3 index.py

Invoke function

Text JSON [✓ Download]

Request body

```
http://thedreamwithinpictures.com/wp-content/uploads/2013/05/c3a89__montage_2048-copy.jpg
```

Response status: 200
Round-trip (s): 1.886

Response body

```
225785 byte(s) received
```

by using the browser we gain the jpg image from the base64 that we gain from our function