COEN 241 HW 2
Your Own Serverless Infrastructure

Due: Wednesday, November 10th 2021, at **7:00 PM**

Learning Objectives

The aims of the assignment include the following learning objectives:

- Understanding the in-and-out of serverless computing
    - How serverless workloads are written
    - How serverless workloads are executed in terms of OpenFaaS
- Understanding a use case of serverless computing
- How to monitor serverless compute workloads

Prerequisite

You are free to use your personal computer for this assignment. The set of requirements for using your personal computers is as follows:

- CPU with at least 2 cores
- 4 GB memory
- 20 GB free disk space
- OS: Windows, Linux (Ubuntu preferred) or Mac OS X
- Docker or Docker Desktop
- Bash if you are on Linux, Git bash if you're using Windows.
- Brew: If you are using a Mac
- VirtualBox or QEMU: Optional but highly recommended for Windows and Macs

Step 1: Install OpenFaaS CLI

What is OpenFaaS?

OpenFaaS (Functions as a Service) is a framework for building serverless functions with Docker and Kubernetes.. Any process can be packaged as a function and can be monitored via OpenFaaS web UI. Refer to Lecture 9 and 10 for more information.

Installing OpenFaaS CLI

Linux Installation Instructions (Most Recommended)

You can install OpenFa simply by typing the following command in the terminal:

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

Mac OS X (Intel) Installation Instructions

```
$ brew install faas-cli
```

Windows Installation Instructions

Again, installing OpenFaaS directly on Windows is **NOT RECOMMENDED**. If you have a Windows machine, you can use Virtualbox to install it in a Linux VM. If you insist on using bare metal Windows host, you can download the installer from here and install it.

Step 2: Deploy OpenFaaS

As mentioned in the lecture, you can choose to install OpenFaaS in either Kubernetes or faasd. For this assignment, **it is recommended to use faasd**, although if you are more ambitious, you can try out Kubernetes or minikube. The installation instructions are written here and it is also covered in lecture 9 if you need to review it.

In order to install faasd on your Linux VM, you can simply run the following commands.

```
$ git clone https://github.com/openfaas/faasd --depth=1
$ cd faasd

# Install faasd
$ ./hack/install.sh
```

After the installation, you must obtain your username and password by running

```
$ sudo cat /var/lib/faasd/secrets/basic-auth-password
$ sudo cat /var/lib/faasd/secrets/basic-auth-user
```

You can use this information to log in to the OpenFaaS UI or pass the information to the faas-cli by running the following command. You should be able to use faas-cli after this.

```
$ cat /var/lib/faasd/secrets/basic-auth-password | faas-cli login
--username admin --password-stdin
```

Step 3: Verify OpenFaaS Installation

You can check that faasd and OpenFaaS is running by running the following commands:

```
# Check if faasd service is running
$ sudo systemctl status faasd
$ sudo systemctl status faasd-provider
```

Or by looking into the logs by running one or more of the commands below. Note that commands with default:* will print logs for various OpenFaaS components. In order to understand what each component means, please refer to Lecture 9.

```
# Logs for faasd
$ sudo journalctl -u faasd --lines 40
$ sudo journalctl -u faasd-provider --lines 40

# Logs for OpenFaaS services
$ sudo journalctl -t default:gateway --lines 40
$ sudo journalctl -t default:nats --lines 40
$ sudo journalctl -t default:queue-worker --lines 40
$ sudo journalctl -t default:prometheus --lines 40
```

Finally, you can check that the OpenFaaS gateway UI is running correctly on port 8080 in the localhost of the VM you are running. In order to access the UI from the host, you need to forward the port from the host to the guest VM's port 8080. Please refer to the demo on Lecture 9 for the instructions on Virtualbox. If you are on QEMU, please refer to the instructions in the documentation here.

Step 4: Trying out a Function from the Store

Now, given that you have an OpenFaaS cluster running in your local instance, we can try out a function from the OpenFaaS store. You can check what is available by running the following.

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

We will install one of the functions called figlet, which is available in https://github.com/jmkhael/faas-figlet and invoke it using the faas-cli below.

```
# Deploy figlet
$ faas-cli store deploy figlet

# Find the URLs for the function
$ faas-cli store inspect figlet
```

```
# Create some ASCII
$ echo "Hello, FaaS, world" | faas-cli invoke figlet
```

After running the above, you should see something like below.

Now you are ready to write your own function!

Step 5: Signing Up for DockerHub

In order for you to be able to push your function to the public, you would need a Docker hub account. Please sign up for one at https://hub.docker.com/ if you don't have one. Make sure you note down your Docker Hub account and the password to be used in the next step.

Step 6: Writing Your Own Function

This is the main part of HW 2.There is a skeleton code to work with in Camino.
First create a base folder (called functions here) for your function like below.
(You may need to run the commands with sudo.)

```
$ mkdir -p ~/functions && cd ~/functions
$ faas-cli new --lang python slack-request
$ faas-cli new --lang python slack-interactive
```

When you run the above commands, you should be seeing two new folders where each folder contains a handler.py file, a yaml file and a requirements.txt file. **Replace the appropriate files with the skeleton code and fill in the blanks in two handler.py and two *.yml files as required**. Once you are done with replacing the fields, run the following command to log into docker first.

```
$ docker login
```

Then run the following functions to deploy your functions.

```
$ faas-cli build -f ./slack-interactive.yml
$ faas-cli push -f ./slack-interactive.yml
```

```
$ faas-cli deploy -f ./slack-interactive.yml
$ faas-cli build -f ./slack-request.yml
$ faas-cli push -f ./slack-request.yml
$ faas-cli deploy -f ./slack-request.yml
```

Once you are done with this, please answer the following questions.

Step 7: Tasks (80 pts)

Here is the list of tasks to do. **All of the code and images must be updated via github.**

- Provide a screenshot of invoking the figlet function (10 pts)
- Complete slack-request/handler.py (20 pt)
- Complete slack-interactive/handler.py (20 pt)
- Provide a screenshot of running the following command (10 pts)
  `sudo journalctl -u faasd --lines 40`
- Provide a screenshot of your OpenFaaS gateway AFTER deploying figlet, slack-handler and slack-interactive functions (10 pts)
- Provide a screenshot of invoking slack-request and slack-interactive functions (10 pts)

Step 6: Questions (20 pts)

After performing the tasks in step 5 answer the following questions.

1. What is the command to invoke the slack-request function (4 pts)?
    a. Via Curl
    b. Via faas-cli
2. What is the output you see when you invoke the slack-request function? (2 pts)
3. What is the command to invoke the slack-interactive function? (4 pts)
    a. Via curl
    b. Via faas-cli
4. What is the output you see when you invoke the slack-interactive function? (2 pts)
5. How would you pass different arguments to the functions? (4 pts)
6. How would you change the slack-interactive function to react to different inputs? (4 pts)

Extra Credit (30 pts)

What you have created for HW 2 is a functional slack application. For the extra credit exercise, you will enable your function to be used as an actual slack app! You can do so by following the instructions below.

- Head over to https://api.slack.com/apps and create a new app
- Create a incoming webhook

- Head over to slash commands and create a new command, e.g., /coen241, set the request url to the public endpoint of your function: http://<YOUR-PUBLIC-IP>:8080/function/slack-request
- Head over to interactive components, set the request url for the interactivity: http://<YOUR-PUBLIC-IP>:8080/function/slack-interactive
- If you do not have a public routable address, have a look at ngrok
- Once you are set, you should be able to see the slash command integration in your slack workspace, head over to slacks documentation if you run into any trouble.

Once this is done, **provide the link to your application and a screenshot of you running your application inside slack for the full credit!** There will be no partial credit for the extra credit.

Submission Instruction

Create a submission for the assignment on the Camino containing the following two pieces of information:

1. a URL such that the command 'git clone URL' would download your repository (assuming that the repository has been shared with the requesting user);
2. a git commit ID within your repository that you want markers to assess that contains the following.
    a. The code of slack-handle & slack-interactive functions **including handler.py, requirements.txt and slack-*.yml**
    b. A txt file containing the answers to the questions
    c. List of screenshots in one of jpeg, png or pdf format..
    d. A txt file explaining each of the screenshots

Note about the Repository

- You can reuse the existing repository to not have to re-invite the teaching staff