2. 함수 배포하기

도커 레지스트리 구동하기

OpenFaaS에서 Function을 배포할 때, 생성된 도커 이미지를 레지스트리에 push하고 pull하므로 도커 레지스트리 생성이 필요함

- private registry 생성

```
$ docker run -d -p 5000:5000 --restart always --name registry registry:2
```

- 생성 확인

```
$ docker ps -a
$ netstat -anp | grep 5000
```

- 로컬 레지스트리

```
$ minikube docker-env
$ eval $(minikube -p minikube docker-env)
$ docker ps # minikube 내부 docker와 local이 연결되어 많은 정보가 노출됨
```

첫 파이썬 함수 배포하기

- 공식 문서 : https://docs.openfaas.com/tutorials/first-python-function/

▶️ 함수 생성 및 작성하기

- 폴더 만들기

```
$ mkdir -p ./hello-python && \
  cd ./hello-python
```

- 새 python function 생성하기

```
$ faas-cli new --lang python3 hello-python
```

※ 제공하는 언어들

https://github.com/openfaas/templates

Name	Language	Version	Linux base	Watchdog	Link
dockerfile	Dockerfile	N/A	Alpine Linux	classic	Dockerfile template
go	Go	1.15	Alpine Linux	classic	Go template
node12	NodeJS	12	Alpine Linux	of-watchdog	NodeJS template
node14	NodeJS	14	Alpine Linux	of-watchdog	NodeJS template
node14	NodeJS	16	Alpine Linux	of-watchdog	NodeJS template
node14	NodeJS	17	Alpine Linux	of-watchdog	NodeJS template
node	NodeJS	12	Alpine Linux	classic	NodeJS template
python3	Python	3	Alpine Linux	classic	Python 3 template
python3-debian	Python	3	Debian Linux	classic	Python 3 Debian template
python	Python	2.7	Alpine Linux	classic	Python 2.7 template
java11-vert-x	Java and Vert.x	11	Debian GNU/Linux	of-watchdog	Java LTS template
java11	Java	11	Debian GNU/Linux	of-watchdog	Java LTS template
ruby	Ruby	2.7	Alpine Linux 3.11	classic	Ruby template
php7	PHP	7.4	Alpine Linux	classic	PHP 7 template
csharp	C#	N/A	Debian GNU/Linux 9	classic	C# template

※ Alpine vs Debian

Alpine Linux는 small, simple, secure를 키워드로 하는 리눅스 배포판 중 하나이다. 설치되어 있는 프로그램이 적고 가벼우며, 이때문에 보안 취약점에 노출될 확률도 적은 등의 장점으로 컨테이너 환경에서 선호된다. 그러나 특정 프로그램 설치나 실행환경 구성을 위해서 기초부터 설치를 해야 하거나 해당 리눅스 배포판에서 라이브러리를 사용할 수 없는 경우가 있는 등의 단점이 있다.

- 생성된 디렉터리 구조

```
hello-python/
  |- handler.py # 함수 작성
  |- requirements.txt # 필요 라이브러리 작성
hello-python.yml # build, deploy 등을 위한 yml 파일
template/ # 컨테이너 실행환경을 만들기 위한 템플릿
```

- 함수 작성

```
$ vi hello-python/handler.py
```

```
# handler.py

def handle(req):
    print("Hello! You said: " + req)
```

▶️ 빌드하기

- yml 파일 작성하기

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:9888
functions:
  hello-python:
    lang: python
    handler: ./hello-python
    image: 127.0.0.1:5000/hello-python:latest
```

💥 openfaas가 listen하는 포트로 변경 필요(8080→9888)

💥 “You must provide a username or registry prefix to the Function's image such as user1/function1”에러 발생을 피하기 위해 registry접속정보를 image 이름에 추가

- 컨테이너 이미지 빌드

```
$ faas-cli build -f ./hello-python.yml
```

- 빌드된 도커 이미지 확인

```
$ docker images | grep hello-python
```

- registry에 push

```
$ faas-cli push -f ./hello-python.yml
```

▶️ 배포하기

- 배포

```
$ faas-cli deploy -f hello-python.yml --gateway http://127.0.0.1:9888
```

- 배포 확인하기

```
$ minikube kubectl -- get pods -n openfaas-fn
```

- 테스트해보기

```
$ faas-cli list --gateway http://127.0.0.1:9888
$ echo test | faas-cli invoke hello-python --gateway http://127.0.0.1:9888
```

Function Store를 통한 함수 배포

- openfaas workshop : https://github.com/openfaas/workshop/blob/master/lab2.md

1. openfaas UI 접속
2. Deploy New Function
3. FROM STORE에서 배포할 Function 선택 & Deploy 클릭