| 2022-Fall |
| --- |
| Cloud Computing |

| 학번 | 2015150220 |
| --- | --- |
| 이름 | 이윤수 |

# 실습과제 1 REPORT

**1. Docker Hub** ( ID : dbstn322)

**2-1. app.py 및 Dockerfile**

app.py

(1) Ping-service

```
from flask import Flask, request
import requests

app = Flask(__name__)

@app.route('/', methods=['get'])
def greeting():
    print('say hello to a stranger')
    return '[sender_ping] Hello stranger, I am ping container! nice to meet u!!!!!!'

@app.route('/ping', methods= ['get', 'post'])
def ping_message():
    if request.method == 'GET':
        pong_url = "http://pong_con:5001/pong"
        send_msg = '[sender_ping] --ping--> '
        send_data = {'msg': send_msg}
        print("send a message to pong container.")
        response = requests.post(pong_url, json= send_data)
        print("got a reponse from the pong container.")
    return response.text

if __name__ == '__main__':
    app.run(host= '0.0.0.0', port = 5000, debug = True)
```

(2) Pong-service

```
from flask import Flask, request

app = Flask(__name__)
@app.route('/', methods=['get'])
def greeting():
    if request.method == 'GET':
        print('say hello to a stranger')
        return '[sender_ping] Hello stranger, I am pong container! nice to meet u!!!!!!'

@app.route('/pong', methods=['get','post'])
def pong_message():
    if request.method =='POST':
        print('got a message from ping container.')
        send_data = request.get_json()
        send_msg = send_data['msg']
        receive_msg = '[receiver_pong] Hi Hi Hi'
        print('send a response to the ping conatiner.')
        return send_msg + receive_msg

if __name__ == '__main__':
    app.run(host= '0.0.0.0', port = 5001, debug = True)
```

**Dockerfile** : ping 과 pong 둘 다 동일하게 작성.

```
FROM ubuntu:latest

RUN apt-get update -y && apt-get upgrade -y

RUN apt-get install -y python3 python3-pip nano vim

ADD requirements.txt /

RUN pip install -r requirements.txt

ADD app.py /

ENTRYPOINT ["python3","app.py"]
~
```

**2-2. 설명**

웹서버를 띄우기 위해서 Flask 를 사용했고, HTTP 메서드를 통해 데이터를 주고 받기 위해서 requests 모듈과 Flask 내장 request 모듈을 함께 활용했습니다.

(1) ping 의 app.py

두 가지 경로 ('/' 와 '/ping')에 대해 각각 하나의 함수가 실행되도록 작성했습니다. 먼저 '/'의 greeting 함수는 해당 url 로 들어오는 GET 요청에 대해 항상 'Hello Stranger ~' 라는 인사말을 반환하도록 했습니다.

다음으로 '/ping'의 ping_message 함수의 경우 GET 요청이 들어오면, pong 에게 보낼 메시지 (send_msg) 를 json data 형태로 만들어 pong_url 에 POST 요청을 보내도록 했습니다. 이에 대한 pong 의 반응을 response 에 받아 그 내용을 GET 요청에 대한 응답으로 최종 반환했습니다. 이때 pong_url 의 포트 번호 앞에 'pong_con' 이라는 주소를 사용했는데, 이는 원래 호스트 네임에 해당하는 부분으로 pong_service 를 담을 컨테이너 이름과 일치합니다. 도커 네트워크를 써서 브릿지로 연결하기 때문에 컨테이너 이름과 포트번호를 가지고 특정 서버에 접근하는 것이 가능했습니다. 이렇게 하지 않고 0.0.0.0 으로 남겨둘 경우 오류가 발생해 연결이 되지 않았습니다.

(2) pong 의 app.py

pong의 경우에도 url별로 함수를 나누어 단순한 GET 요청에 대해 인사말을 반환하는 '/'의

greeting과 앞서 ping이 보낸 메시지에 대한 응답을 전송하는 '/pong'의 pong_message 함수를 만들었습니다. Greeting 함수의 경우 ping과 동일하게 작성했습니다. 한편 pong_message에서는 들어오는 ping의 POST 요청에 대해서 request에 담겨 있는 send_msg 와 pong의 응답인 'Hi Hi Hi' 부분을 합쳐서 반환하도록 작성했습니다.

(3) Dockerfile

각 라인 별로 설명하면 다음과 같습니다.

FROM - 우분투 최신 이미지를 베이스 이미지로 불러왔습니다.

RUN - 리눅스 기반 운영체제인 우분투에 대해 기본적인 업데이트와 업그레이드를 진행했습니다.

RUN - 실행할 파일과 패키지, 모듈 등이 파이썬 기반이기 때문에 거기에 필요한 python3, pip3를 설치했습니다. 컨테이너 내부 파일을 확인할 목적으로 nano와 vim도 설치했습니다.

ADD - 설치할 파이썬 패키지 목록이 담긴 requirements.txt 를 로컬에서 컨테이너로 복사했습니다.

RUN - 해당 목록의 패키지를 실제로 컨테이너 내부에 설치하는 실행 부분입니다.

ADD - 메인 프로그램인 app.py를 컨테이너에 복사했습니다.

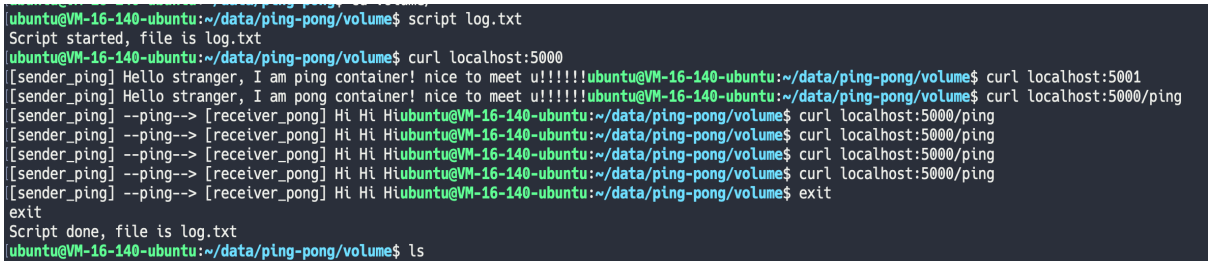
ENTRYPOINT - docker run 과 동시에 서버가 작동하도록 ENTRYPOINT로 app.py를 실행했습니다.

### 2-3 Docker Network

```
ubuntu@VM-16-140-ubuntu:~/data/ping-pong/volume$ sudo docker inspect ping-pong-net
[
    {
        "Name": "ping-pong-net",
        "Id": "d985ec7daeeb803b290bd8d0f766dbff66ad11340fcb1fd39532198033acf1be",
        "Created": "2022-11-03T22:06:51.820910431+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "1d695c5548dec05e6fd2a58066dfe2dab635ec528509eafd2cf2b50e493ac37e": {
                "Name": "ping_con",
                "EndpointID": "cd41f1a3e865402b463a5083bc1a749fca69016ed2c4fc1a896ce14ec1bfa35a",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "6be4e44d4bcee63054b77ea912cdd75e588cc727e43350351026c0252274213d": {
                "Name": "pong_con",
                "EndpointID": "6b660b1b7f62ecb00b9cff7bfa92e3cb478d8320ad226f290e15affb78d64aaf",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Ping-pong-net 이라는 도커 custom network (브릿지)를 생성했습니다. 거기에 docker run 단계에서 '—network' 옵션을 활용해 ping_con 과 pong_con 을 연결했습니다.

### 2-4 ping, pong 메시지 출력




'curl localhost:5000' / 'curl localhost:5001' / 'curl localhost:5000/ping'을 차례로 입력했습니다.

모두 의도한 메시지를 받아볼 수 있었고, 과제의 요구사항대로 ping, pong 이 주고받은 메시지가 온전히 출력된 것을 5 번 이상 실행했습니다.

### 2-5 메시지 양쪽 컨테이너에 저장

위의 캡쳐 이미지에 나타나 있듯이, curl 을 실행하기 이전에 ' script ' 명령어를 사용해서 콘솔에 입력되는 메시지가 log.txt 로 저장되게 했습니다. 이때 log.txt 가 저장된 곳의 경로가 ~/data/ping-pong/volume 이었는데 두 컨테이너를 실행할 당시 아래와 같이 명령어를 입력했습니다.

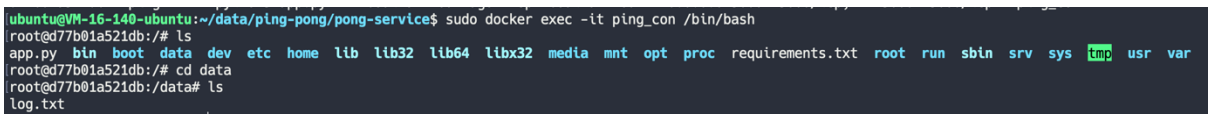
sudo docker run -it --name ping_con --network ping-pong-net -p 5000:5000

**-v ~/data/ping-pong/volume:/data ping:v1**

sudo docker run -it --name pong_con --network ping-pong-net -p 5001:5001

**-v ~/data/ping-pong/volume:/data pong:v1**

**-v** 옵션 즉, 도커 볼륨을 이용해 해당 경로의 파일들이 두 컨테이너의 /data 에 저장되도록 했습니다.




이후 실제로 호스트와 동일한 log.txt 파일이 컨테이너 내부에도 저장돼 있는 것을 확인할 수 있었습니다.