11주차(1)_Docker Compose

Docker Compose

간단히 말해 **컨테이너의 생성을 간략화**해주는 방법입니다. 지난 실습들에서 컨테이너를 생성(run)할 때, 여러 옵션을 입력하여서 실행하는 것을 간략화하기 위해서 docker compose를 사용합니다.

문제는 보통 여러 개의 어플리케이션(모듈)이 하나의 시스템을 구성하고 있다는 것입니다. 기존에 사용했던 Dockerfile의 경우 하나의 이미지를 대상으로 build되는 반면, 각각의 어플리케이션은 하나의 컨테이너고, 즉, 여러 컨테이너가 하나의 큰 시스템을 구축하고 있는 것이죠.

이렇게 **여러 컨테이너를 한 번에 관리할 수 있도록 도와주는 것**이 Docker Compose입니다.

Docker Compose VS Docker File

그러면 저번 시간에 다뤘던 Dockerfile과는 어떤 차이점이 있을까요?

Dockerfile은 **하나의 이미지**를 만들기 위해 사용되는 명령이 포함된 간단한 텍스트 파일이라고 정의할 수 있고,

Docker-Compose는 앱을 구성하는 **여러 모듈(컨테이너, 이미지)를 한 번**의 명령으로 실행할 수 있습니다.

즉, Docker-Compose는 각각의 이미지에 대한 Dockerfile을 build한 뒤에, compose를 사용하여서 여러 이미지를 조합하는 과정으로 보시면 됩니다.

Docker-Compose에서 Dockerfile을 사용할 수 있다는 것입니다.

정리하자면,

- **Dockerfile**: 이미지를 제작(assemble)하기 위해 사용되는 명령이 포함된 간단한 텍스트 파일

  ⇒ 이미지를 build함.

- **Docker Compose**: 앱을 구성하는 모든 서비스를 docker-compose.yml에 정의해서, `docker-compose up`을 사용하여 하나의 명령으로 앱을 실행함.

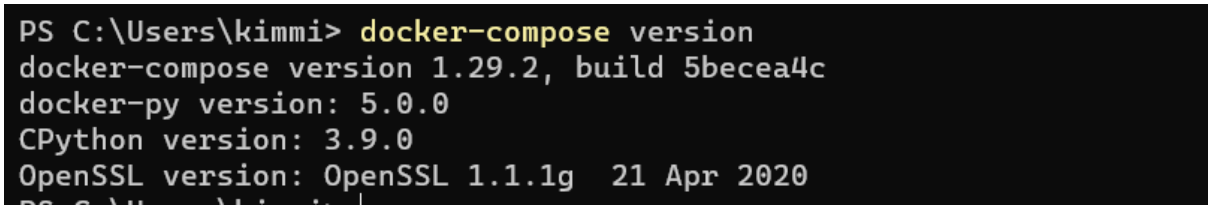
⇒ 앱이 실행되는 동안 컨테이너를 관리함.

설치

- windows/mac은 docker를 설치할 때 자동으로 설치됩니다.
- linux에서는 따로 설치해주세요.
  - [참고] https://docs.docker.com/compose/install/

설치 완료시 다음 명령어를 입력하였을 때, 버전 정보가 떠야합니다.

```
docker-compose version
```



사용 방법

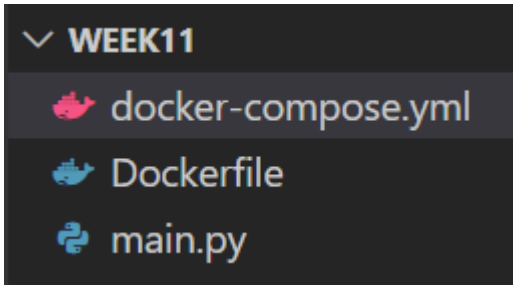
1. 각각의 컨테이너의 Dockerfile을 작성합니다.
2. 이후, `docker-compose.yml`을 작성하는데 각각의 컨테이너를 어떻게 실행할지를 명시합니다.
3. `$ docker-compose up`을 사용하여 yml 파일로 정의한 컨테이너를 실행합니다.

실습

[참고] https://hub.docker.com/_/python

아래의 디렉토리 구조로 파일을 제작해주세요.

이때 사용한 Dockerfile의 내용은 docker hub의 python 이미지의 Dockerfile 자료를 참고하여 제작하였습니다.




```
# main.py

print("hello")
```

```
# Dockerfile

# 이미지명
FROM python:3

# python 컨테이너에서 사용되는 경로
# 실제로 본 이미지를 실행해보면 /usr/src/app 경로에서 python이 실행되는 것을 확인 가능함
WORKDIR /usr/src/app

# 현재 local 경로에 있는 폴더(week11)를 컨테이너 내부로 copy함
COPY . .

# python ./main.py가 실행됨. 이때, main.py는 본인의 python 파일 이름으로 변경 가능
CMD [ "python", "./main.py" ]
```

```
# docker-compose.yml

version: "2.0"      # 버전

services:         # 아래는 서비스 정의
  python-app:     # services명
    build: .      # Dockerfile의 경로
```

아래의 명령어로 이미지 build 및 실행해주세요.

```
$ docker build -t w11 .
$ docker run -it --name w11_c w11
```

```
PS C:\Users\kimmi\Desktop\week11> docker build -t w11 .
[+] Building 10.6s (5/8)
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 119B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3
 => [auth] library/python:pull token for registry-1.docker.io
 => [1/3] FROM docker.io/library/python:3@sha256:b941b836b18734f4992a168b579b7c16ff4c3b544782953eeab3a590a7338765
 => => resolve docker.io/library/python:3@sha256:b941b836b18734f4992a168b579b7c16ff4c3b544782953eeab3a590a7338765
```

아래처럼 실행되고 종료되는 것을 확인할 수 있습니다.

```
PS C:\Users\kimmi\Desktop\week11> docker run -it --name w11_c w11
hello
PS C:\Users\kimmi\Desktop\week11>
```

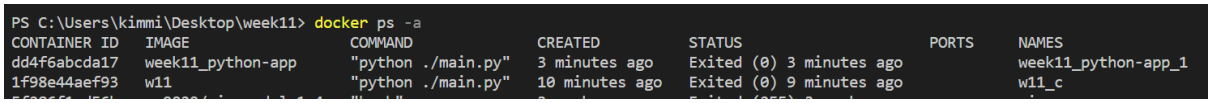
이번엔 docker-compose.yml파일을 사용해서 dockerfile에 정의한 파일을 build해보겠습니다.

```
docker-compose up
```

```
PS C:\Users\kimmi\Desktop\week11> docker-compose up
Creating network "week11_default" with the default driver
Building python-app
[+] Building 3.2s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.1s
 => => transferring dockerfile: 119B                                                                                0.0s
 => [internal] load .dockerignore                                                                                   0.1s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3                                                         2.5s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => [internal] load build context                                                                                   0.1s
 => => transferring context: 394B                                                                                   0.0s
 => [1/3] FROM docker.io/library/python:3@sha256:b941b836b18734f4992a168b579b7c16ff4c3b544782953eeab3a590a7338765   0.0s
 => CACHED [2/3] WORKDIR /usr/src/app                                                                               0.0s
 => [3/3] COPY . .                                                                                                  0.1s
 => exporting to image                                                                                              0.2s
 => => exporting layers                                                                                             0.1s
 => => writing image sha256:6beb79a3ed049de063e8ded703e0e5397c87671b74edf7bddde853673a3c438e                        0.0s
 => => naming to docker.io/library/week11_python-app                                                                0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
WARNING: Image for service python-app was built because it did not already exist. To rebuild this image you must use `docker-compose build`
 or `docker-compose up --build`.
Creating week11_python-app_1 ... done
Attaching to week11_python-app_1
python-app_1  | hello
week11_python-app_1 exited with code 0
```

hello가 정상적으로 출력되고 종료가 되는 것을 확인할 수 있고, `docker ps -a`를 했을 때, dockerfile로 만들었던 **w11_c**와 이번 docker-compose로 만들었던 **week11_python_app_1**이 모두 잘 생성된 것을 확인할 수 있습니다.



docker-compose.yml

각각의 컨테이너를 실행할 때 필요한 옵션을 미리 적어둔 문서입니다. 프로젝트의 root(프로젝트 디렉토리 바로 밑)에 docker-compose.yml 파일을 두고, 아래의 설정들에 맞게 실행하고자하는 컨테이너의 실행 설정을 적어줍니다.

그러면 docker는 compose의 파일을 보고 컨테이너를 실행하게 되는 것입니다.

물론 컨테이너가 1개만 존재한다면 비슷하겠지만, 여러 컨테이너를 한 번에 관리하는 경우에는 해당 기능을 사용하면 훨씬 시간을 단축할 수 있습니다.

대표적인 구성 요소는 다음과 같고, 이외에도 실행 옵션으로 사용하는 모든 것을 정의할 수 있습니다.

구분	내용
version	버전
services	서비스 정의 ⇒ 아래에 서비스 이름 및 옵션들 정의함.
image	사용할 이미지
build	Dockerfile 위치
command	container 안에서 작동하는 명령어 작성
container_name	컨테이너 이름 설정
ports	컨테이너가 host에 오픈하는 포트 번호

기본 명령어

- 실행

  docker-compose 파일을 사용하여 dockerfile에 정의된 컨테이너를 실행합니다.

```
docker-compose up [옵션] [서비스명]

// 예시(docker-copmose 파일이 위치한 경로에서 실행할 때)
docker-compose up
```

- 상태 확인

```
docker-compose ps
```

- 서비스 관련 명령어

```
# 시작
docker-compose start [서비스명]

# 정지
docker-compose stop [서비스명]
```

- 삭제

```
docker-compose rm
```