7/10

도커 요약 지금까지의

dockerfile → image는 docker build or docker buildx build

image → container docker run

dockerfile

from

env

run

cmd

Entrypoint

이 때 유의점은 두개 같이 쓸 경우 Entrypoint가 명령어 cmd가 인자

image를 만들 때 image는 base image부터 시작해서

layer가 read only형태로 변경불가로 차곡차곡 쌓이는 것

cache

아래에 변경이 없더라도 위에 변경 즉 패런츠에 변경이 일어나면 아래에는 cache가 적용되지 않고 다 새로 빌드된다.

캐시를 충분히 활용하기 위해선 자주 변경되는 내용은 아래 변경이 안되는건 위에가 기본적인 스킬이다..

명령어는 같지만 시점에 따라 결과가 달라지는 apt-get update같은 경우 유의해서 사용해야 - -no-cache로 빌드하면 해결가능

&& 로 명령어를 묶어주면 layer를 하나로 묶어주기에 cache로 인한 문제 방지 가능

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

image

레포 이름은 깃과 같이 레포 이름

tag는 커밋 메시지

push할 때는 pull네임으로 HOST:PORT/TAG

HOST의 Default는 dockerhub이다.

하나의 이미지에 태그가 여러 개 있을 수 있다.
여러 개의 태그는 하나의 이미지를 가리킨다.

base 이미지로 latest는 사용하면 오류가 많이 생길 수밖에 없다 변경이 일어나니까
digest를 이용하면 가장 안전하게 사용 가능
웬만하면 특정 version으로 한다.

실무에서도 latest는 test 용도로만 배포 절대 금지
가급적 digest 사용해라 지침 내려온다.

Exited ← 이미지 정상 종료
Exited(숫자) ← 이미지에 오류가 있다.

DOCKER RUN

docker run 할 때 -e 여러 개 작용해서 가능하다.
환경변수가 엄청 많으면 .env 파일을 만들고 - - env-file을 통해 전달 가능하다.
- -env-file PATH를 사용해야 한다.
-p는 port publish 내 네트워크가 아닌 다른 곳에 공개를 할 때 사용

-p 내 포트: 컨테이너 포트

-v
volume
컨테이너 내의 데이터를 외부에 저장하고 싶을 때 사용
컨테이너와 volume을 마운트한다.

-v a/b

a가 절대경로나 상대경로라면 host로 인식을 하고
없으면 volume로 인식을 한다.
따라서 volume은 통째로 넣던가 해야지 volume내에 어떠한 경로를 넣는건 힘들다.

network

컨테이너를 여러 개 만들어도 각자의 network를 가지고 있으니 통신이 안된다.
서브넷이 다르면 당연히 라우터가 없으면 통신 x

실습 3

[실습-3] 실습용 이미지 제작하고 푸시하기

- 다음 조건을 만족하는 `Dockerfile`을 작성하세요.
  - `Digest`를 사용하여 `code-server`의 버전을 지정하세요.
    - `sha256:086c9625a89bec4ea651cea26ee5ec7c242fe3d318fdc9142fd89b091bae04e6`
  - 다음과 같이 환경 변수를 설정하세요
    - TZ="Asia/Seoul"
    - PUID=1000
    - PGID=1000
  - 다음 스크립트를 참고하여 패키지들을 설치하세요.
    - Ubuntu Update

```
apt-get update && apt-get -y upgrade
apt install -y ca-certificates curl gnupg software-properties-common
wget unzip apt-transport-https
```

    - Docker CLI

```
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
apt-get install -y docker-ce docker-ce-cli
```

    - AWS CLI

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

    - Terraform

```
apt-get update && apt-get install -y gnupg software-properties-
common

wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null

gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-
keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
tee /etc/apt/sources.list.d/hashicorp.list

apt update && apt-get install -y terraform=1.6.6-1
```

    - Kubectl

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key |
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | tee
/etc/apt/sources.list.d/kubernetes.list

apt-get update && apt-get install -y kubectl
```

    - Kubens & Kubectx

```
git clone https://github.com/ahmetb/kubectx /opt/kubectx  \
    && ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx  \
    && ln -s /opt/kubectx/kubens /usr/local/bin/kubens
```

- `Dockerfile`을 이용하여 이미지를 빌드하세요
  - 이름은 `cloudwave`로 설정해주세요
  - Tag는 `practice.v1`으로 설정해주세요
- `Docker Hub`에 이미지 푸시하세요

```
FROM linuxserver/code-server@sha256:086c9625a89bec4ea651cea26ee5ec7c242fe3d318fdc9
ENV TZ="Asia/Seoul"
ENV PUID=1000
ENV PGID=1000

#ubuntu update
RUN apt-get update && apt-get -y upgrade
RUN apt install -y ca-certificates curl gnupg software-properties-common wget unzi

# Docker CLI
RUN install -m 0755 -d /etc/apt/keyrings  \
    && curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyring
    && chmod a+r /etc/apt/keyrings/docker.asc

RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docke
      $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
      tee /etc/apt/sources.list.d/docker.list > /dev/null

RUN apt-get update && apt-get install -y docker-ce docker-ce-cli

# Kubens & Kubectx

RUN git clone https://github.com/ahmetb/kubectx /opt/kubectx \
    && ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx \
    && ln -s /opt/kubectx/kubens /usr/local/bin/kubens

# Kubectl

RUN curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key  | \
    gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

RUN echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
        https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | \
        tee /etc/apt/sources.list.d/kubernetes.list

RUN apt-get update && apt-get install -y kubectl

# Aws CLI
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.z
RUN ./aws/install

#Terraform
#RUN apt-get update && apt-get install -y gnupg software-properties-common wget -O

# RUN apt-get update && apt-get install -y gnupg software-properties-common wget -
# gpg --dearmor | \
# tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null gpg --no-defau
# --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
# --fingerprint
```

```
# RUN echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \htt
# tee /etc/apt/sources.list.d/hashicorp.list

# RUN apt update && apt-get install -y terraform=1.6.6-1

RUN wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | tee /usr/sha
    && gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-ke
    && echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] htt
RUN apt update && apt-get install -y terraform=1.6.6-1
```

docker buildx build --push -t dlwpdnr213/cloudwave:practice.v1 .

extra

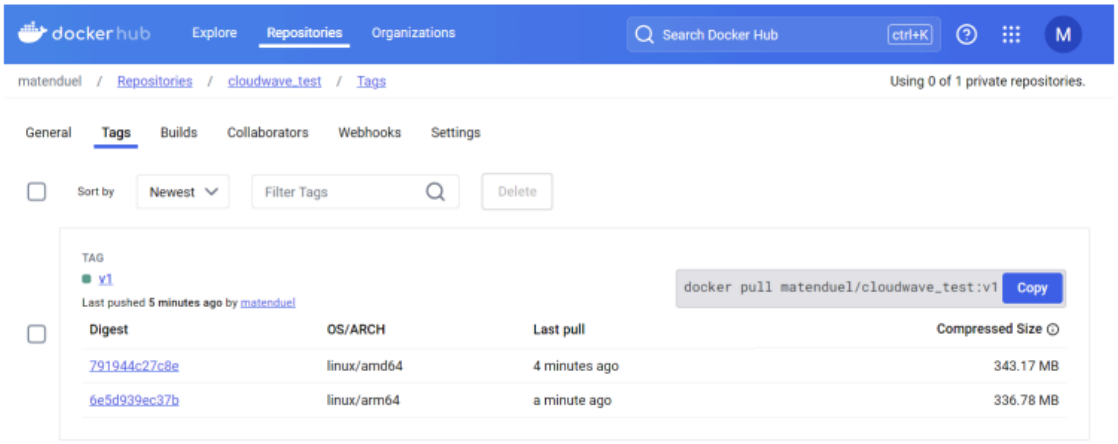
[Extra] `buildx`를 이용하여 `Docker Hub`에 `Multi platform` 이미지를 `Push` 하기

- [선택] `docker login`을 이용하여 `Registry`에 로그인하세요.
- 다음 조건을 만족하는 `builder`를 생성하세요
  - `linux/amd64`와 `linux/arm64`를 지원해야 합니다.
  - `docker-container`를 `driver`로 설정하세요
- 다음 조건을 만족하는 `Dockerfile`을 작성하세요.
  - `Digest`대신 `tag`를 사용하세요

  - `Multi-stage`를 이용하여 `amazon/aws-cli:2.17.9` 이미지에서 `aws-cli` 패키지를 복사하세요.
    - `aws-cli`는 `/usr/local/aws-cli/v2`에 설치되어 있습니다.
    - `aws` 명령어를 사용하기 위해서 다음과 같이 `Symbolic link`를 설정하세요.

```
ln -s /usr/local/aws-cli/v2/current/bin/aws /usr/local/bin/aws
ln -s /usr/local/aws-cli/v2/current/bin/aws_completer
/usr/local/bin/aws_completer
```

  - `aws-cli`를 제외한 나머지 패키지는 기존 방식대로 설치해도 상관 없습니다.
- `Docker Hub`에서 다음과 같이 `Multi-platform` 이미지가 `Push`되었는지 체크합니다.




```
COPY --from=amazon/aws-cli:2.17.9 /usr/local/aws-cli/v2 /usr/local/aws-cli/v2
RUN ln -s /usr/local/aws-cli/v2/current/bin/aws /usr/local/bin/aws
```

```
RUN ln -s /usr/local/aws-cli/v2/current/bin/aws_completer /usr/local/bin/aws_compl
```

docker buildx build --push --platform linux/amd64,linux/arm64 -t dlwpdnr213/cloudwave:practice.v2 .

```
# https://hub.docker.com/r/linuxserver/code-server
FROM linuxserver/code-server:4.90.3

# ENV for Code-server (VSCode)
ENV TZ="Asia/Seoul"
ENV PUID=1000
ENV PGID=1000

# Update & Install the packages
RUN apt-get update && apt-get -y upgrade

RUN apt install -y ca-certificates curl gnupg software-properties-common wget unzi

# Docker CLI
RUN install -m 0755 -d /etc/apt/keyrings  \
    && curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyring
    && chmod a+r /etc/apt/keyrings/docker.asc

RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docke
      $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
      tee /etc/apt/sources.list.d/docker.list > /dev/null

RUN apt-get update && apt-get install -y docker-ce docker-ce-cli

# Install AWS CLI from `amazon/aws-cli:2.17.9` image
COPY --from=amazon/aws-cli:2.17.9 /usr/local/aws-cli/v2 /usr/local/aws-cli/v2
RUN ln -s /usr/local/aws-cli/v2/current/bin/aws /usr/local/bin/aws
RUN ln -s /usr/local/aws-cli/v2/current/bin/aws_completer /usr/local/bin/aws_compl

# Terraform
RUN wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | tee /usr/sha
    && gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-ke
    && echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] htt

RUN apt update && apt-get install -y terraform=1.6.6-1

# Kubectl
RUN curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | gpg --dea
    && echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://

RUN apt-get update && apt-get install -y kubectl

# Kubectx & Kubens
RUN git clone https://github.com/ahmetb/kubectx /opt/kubectx  \
    && ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx  \
    && ln -s /opt/kubectx/kubens /usr/local/bin/kubens
```

Docker compose

서비스 → 컨테이너의 조합

front, back db redis 컨테이너들을 조합해서 하나의 서비스로 제공

시작하기 전에

> https://docs.docker.com/compose/reference/

```
Usage:  docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker.

Options:
      --ansi string                Control when to print ANSI control characters
("never"|"always"|"auto") (default "auto")
      --compatibility              Run compose in backward compatibility mode
      --env-file stringArray       Specify an alternate environment file
  -f, --file stringArray           Compose configuration files
      --parallel int               Control max parallelism, -1 for unlimited
(default -1)
      --profile stringArray        Specify a profile to enable
      --project-directory string   Specify an alternate working directory
                                   (default: the path of the, first specified,
Compose file)
  -p, --project-name string        Project name
```

주요 옵션

Instruction	Short	Description
`env-file`		환경 변수를 저장한 파일을 설정합니다.
`file`	`-f`	사용할 `compose` 파일을 지정합니다.
`profile`		사용할 프로파일을 지정합니다.
`project-name`	`-p`	프로젝트 이름을 설정합니다.

주의사항

- `-f`를 통해 사용할 `compose` 파일을 지정하지 않는 경우, `docker-compose.yml` 또는 `docker-compose.yaml` 파일이 사용됩니다.
- `Docker compose` 파일의 `version`에 따라서 사용할 수 있는 기능과 변수명이 달라질 수 있습니다. 따라서, 서버에 설치된 `Docker Engine`의 버전과 호환되는지 체크하는 것이 좋습니다.
- 프로젝트 이름이 설정되지 않은 경우, 현재 디렉토리 이름이 프로젝트 이름으로 설정됩니다.

-f는 무조건 compose 옆에 적어야 한다.

Compose file 버전별 호환 목록

Compose file format	Docker Engine release
Compose specification	19.03.0+
3.8	19.03.0+
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+

안되면 확인할 때가 있다.

yaml

파일 포맷의 형태

key : value

딕셔너리 구조 json으로도 변환 가능

들여쓰기 할 때 두 칸씩

프로젝트 실행

> https://docs.docker.com/engine/reference/commandline/compose_up/

```
docker compose up [OPTIONS] [SERVICE...]
```

주요 옵션

Option	Short	Default	Description
`--abort-on-container-exit`			1개 이상의 컨테이너가 멈춘(`stopped`) 경우, 모든 컨테이너를 종료합니다. `-d`와는 함께 사용할 수 없습니다.
`--build`			컨테이너 실행 전에 이미지를 빌드합니다.
`--detach`	`-d`		컨테이너를 백그라운드에서 실행시킵니다.
`--force-recreate`			변경 여부와 관계없이 모든 컨테이너를 재생성합니다.
`--remove-orphans`			`compose` 파일에 정의되어 있지 않은 서비스를 위한 컨테이너를 삭제합니다.

주의사항

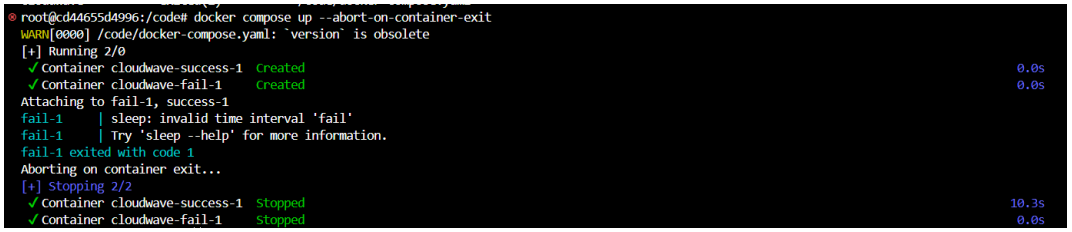
- 서로 다른 `docker-compose.yaml`의 프로젝트 명이 중복된 경우, 각 파일에 명시된 모든 `component`들이 적용됩니다.

[예시] `--abort-on-container-exit` 사용하기

```
# docker-compose.yaml
version: '3.8'
name: 'cloudwave'

services:
  success:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - sleep infinity
    networks:
      private: {}

  fail:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - sleep fail
    networks:
      private: {}

networks:
  private:
```

`fail` 서비스를 위한 컨테이너가 정상적으로 실행되지 않으므로, 다음과 같이 프로젝트의 모든 컨테이너가 종료되는 것을 확인할 수 있습니다.

```
$ docker compose up --abort-on-container-exit
[+] Building 0.0s (0/0)
         docker:desktop-linux
[+] Running 2/0
 ✓ Container cloudwave-success-1  Created
                          0.0s
 ✓ Container cloudwave-fail-1     Created
                          0.0s
Attaching to cloudwave-fail-1, cloudwave-success-1
cloudwave-fail-1     | sleep: invalid time interval 'fail'
cloudwave-fail-1     | Try 'sleep --help' for more information.
cloudwave-fail-1 exited with code 1
Aborting on container exit...
[+] Stopping 2/2
 ✓ Container cloudwave-success-1  Stopped
                         10.1s
 ✓ Container cloudwave-fail-1     Stopped
                          0.0s
```



프로젝트 목록

> https://docs.docker.com/engine/reference/commandline/compose_ls/

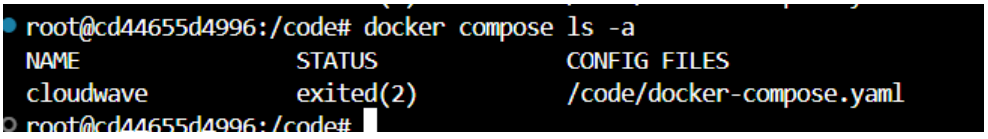
```
docker compose ls [OPTIONS]
```

주요 옵션

Option	Default	Short	Description
`--all`		`-a`	종료된 프로젝트를 포함한 모든 프로젝트를 표시합니다.
`--filter`			필터 조건을 추가합니다.
`--quiet`		`-q`	프로젝트의 이름만 표기합니다.

[예시] 이름에 `cloud`가 포함된 모든 프로젝트 목록 보기

```
$ docker compose ls --filter "name=cloud" -a
NAME                STATUS              CONFIG FILES
cloud_wave          running(3)          <PROJECT_PATH>/docker-compose.yaml
cloud_wave2         exited(2)           <PROJECT_PATH>/docker-compose.yaml
```



docker compose ps는 내가 현재 위치한 폴더내에 있는 docker-compose.yaml을 기준으로 찾는다 따라서 이 파일이 없다면 오류가 난다.

docker compose는 따라서 내가 어디에 위치해있는지가 매우 중요 폴더 구분 잘해야..

컨테이너 목록

> https://docs.docker.com/engine/reference/commandline/compose_ps/

```
docker compose ps [OPTIONS] [SERVICE...]
```

주요 Options

Option	Short	Default	Description
`--all`	`-a`		종료된 컨테이너를 포함한 모든 컨테이너를 표시합니다.
`--quiet`	`-q`		컨테이너 ID만 표기합니다.
`--services`			서비스만 표기합니다.
`--status`			상태값을 기반으로 서비스를 필터링합니다. [paused | restarting | removing | running | dead | created | exited]

주의 사항

- CLI를 통해 전달된 `project-name` 또는 `configuration file`에 명시된 `name`을 기반으로 조회합니다.

[예시] 이름이 `cloud_wave`인 프로젝트의 실행중인 컨테이너 목록 보기

```
$ docker compose -p cloud_wave ps
NAME                    IMAGE                    COMMAND                  SERVICE
            CREATED              STATUS              PORTS
cloud_wave-frontend-1   nginx:latest             "/docker-entrypoint.…"
frontend            21 minutes ago      Up 21 minutes       0.0.0.0:80->80/tcp
cloud_wave-server-1     ubuntu:22.04             "/bin/bash -c 'sleep…"   server
             21 minutes ago      Up 21 minutes
db_postgres             postgres:16.1-bullseye   "docker-entrypoint.s…"
postgres            19 minutes ago      Up 19 minutes       5432/tcp
```

프로젝트 삭제

> https://docs.docker.com/engine/reference/commandline/compose_down/

```
docker compose down [OPTIONS] [SERVICES]
```

주요 옵션

Option	Short	Default	Description
`--remove-orphans`			`Compose` 파일에 정의되지 않은 서비스를 구성하는 컨테이너도 삭제합니다.
`--volumes`	`-v`		`Copose` 파일에 정의된 `anonymous` 볼륨을 같이 삭제합니다.

주의 사항

- 현재 디렉토리에 `docker-compose.yaml` 파일이 존재하지 않는 경우, `-p` 또는 `-f` 를 이용하여야 합니다.
- `--remove-orphans` 를 사용하지 않을 경우, `compose` 파일에 정의된 서비스를 구성하는 컨테이너만 삭제합니다.
- `external` 로 정의된 `network` 와 `volume` 은 삭제되지 않습니다.
- 다른 프로젝트에서 사용중인 리소스(`volume` , `network` , ...)는 삭제되지 않습니다.

[예시] 이름이 `cloud_wave`인 프로젝트 삭제하기

```
$ docker compose -p cloud_wave down
[+] Running 3/3
 ✓ Container cloud_wave-postgres-1  Removed
                          0.1s
 ✓ Container cloud_wave-frontend-1  Removed
                          0.0s
 ✓ Container cloud_wave-server-1    Removed
                          10.1s
```

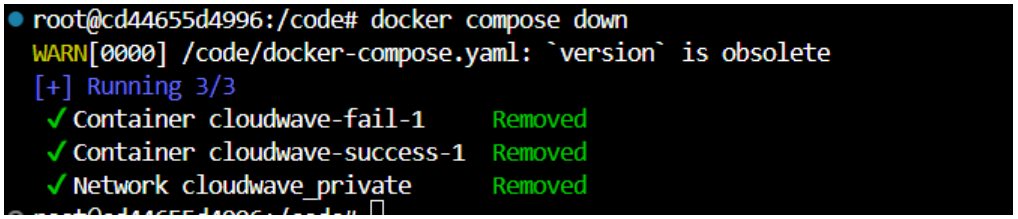
[예시] 프로젝트 삭제시 `volume`도 함께 제거하기

```
# docker-compose.yaml
version: '3.8'
name: 'cloud_wave'

services:
  success:
    image: ubuntu:22.04
```

```
    entrypoint: /bin/bash
    command:
      - -c
      - sleep infinity
    networks:
      private: {}
    volumes:
      - anonymous:/anonymous
      - named:/named
      - external:/external

networks:
  private:

volumes:
  anonymous:
  named:
    name: "named_volume"
  external:
    name: "external"
    external: true
```

`--volumes`을 사용하는 경우, 다음과 같이 `external`로 설정된 볼륨을 제외한 나머지 2개 볼륨이 함께 삭제되는 것을 확인할 수 있습니다.

```
$ docker compose -p cloud_wave down --volumes
[+] Running 5/5
 ✓ Container cloud_wave-fail-1     Removed
                          0.0s
 ✓ Container cloud_wave-success-1  Removed
                          10.1s
 ✓ Volume named_volume             Removed
                          0.0s
 ✓ Volume cloud_wave_anonymous     Removed
                          0.0s
 ✓ Network cloud_wave_private      Removed
                          0.1s
```

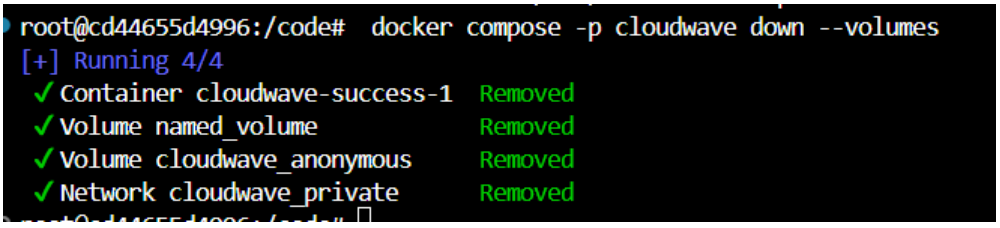


yaml 파일이 없는 경우엔 -p 옵션으로 이름 지정한 뒤 지운다

만약 yaml 파일에서 fail 부분을 주석 처리한 뒤 down을 시도하면 fail부분이 없다고 판단하여 succes만 지우게 되고 fail은 찌꺼기로 남게된다.

yaml 파일을 잘 관리해야..

docker volume create로 volume 먼저 만들어야 한다..




```
docker volume create external // volume 먼저 만들자
docker compose up -d // 만든다
docker compose ls // 만들어 진거 확인한다.
 docker compose -p cloudwave down --volumes //cloudwave 이름으로 만들어진거 지운다
```

재실행

재실행

> https://docs.docker.com/engine/reference/commandline/compose_restart/

```
docker compose restart [OPTIONS] [SERVICE...]
```

주의 사항

- 서비스를 명시하지 않은 경우, 모든 서비스에 대해서 종료된 컨테이너를 포함하여 재실행됩니다.
- 반영되지 않은 변경사항이 `compose` 파일에 존재하더라도, 해당 변경사항은 **반영되지 않습니다.**

재실행하고 싶은 서비스명만 명시하고 진행

종료 and 시작

종료 & 시작

> https://docs.docker.com/engine/reference/commandline/compose_stop/
>
> https://docs.docker.com/engine/reference/commandline/compose_start/

```
# 종료
docker compose stop [OPTIONS] [SERVICE...]
# 시작
docker compose start [SERVICE...]
```

[예시] `cloud_wave` 프로젝트 종료하기

```
$  docker compose -p cloud_wave stop
[+] Stopping 2/3
 ✓ Container cloud_wave-postgres-1  Stopped
                          0.1s
 ✓ Container cloud_wave-server-1    Stopped
                         10.2s
 ✓ Container cloud_wave-frontend-1  Stopped
                          0.1s
```

[예시] `cloud_wave` 프로젝트 실행하기

```
$ docker compose -p cloud_wave start
[+] Running 3/3
 ✓ Container cloud_wave-frontend-1  Started
                          0.6s
 ✓ Container cloud_wave-postgres-1  Started
                          0.4s
 ✓ Container cloud_wave-server-1    Started
                          0.5s
```

```
root@cd44655d4996:/code# docker compose -p cloudwave stop
[+] Stopping 1/1
 ✓ Container cloudwave-success-1  Stopped
root@cd44655d4996:/code# docker compose -p cloudwave start
[+] Running 1/1
 ✓ Container cloudwave-success-1  Started
```

💡 -p를 쓰는 것을 습관화하자 다른 서비스가 멈추지 않게

Docker compose build

서비스 빌드하기

> https://docs.docker.com/engine/reference/commandline/compose_build/

```
docker compose build [OPTIONS] [SERVICE...]
```

주요 옵션

Option	Short	Default	Description
`--build-arg`			빌드시에 사용할 `ARG`를 설정합니다.
`--builder`			사용할 `builder`를 설정합니다.
`--no-cache`			이미지 빌드시 `Cache`를 사용하지 않습니다.
`--pull`			항상 이미지를 새로 다운로드 받습니다.
`--push`			서비스의 이미지를 `Push` 합니다.

주의 사항

- `yaml` 파일에 명시된 서비스의 `image`를 이름으로 한 이미지가 생성됩니다.

```yaml
version: "3.8"
services:
  server:
    image: compose:build.v1
    build:
      #dockerfile_inline: |
      #  FROM ubuntu:22.04
      #  RUN apt-get update && apt-get upgrade
      # Or Use `dockerfile`
      dockerfile: "./Dockerfile"
    entrypoint: /bin/bash
    command:
      - -c
      - "sleep 3600"
    restart: no
```

```
root@cd44655d4996:/code# docker compose build
WARN[0000] /code/docker-compose.yaml: `version` is obsolete
[+] Building 0.2s (6/6) FINISHED
 => [server internal] load build definition from Dockerfile
 => => transferring dockerfile: 93B
 => [server internal] load metadata for docker.io/library/ubuntu:22.04
 => [server internal] load .dockerignore
 => => transferring context: 2B
 => [server 1/2] FROM docker.io/library/ubuntu:22.04
 => CACHED [server 2/2] RUN apt-get update && apt-get upgrade
 => [server] exporting to image
 => => exporting layers
 => => writing image sha256:8f07a6005b24f0ad3c281abfce75fdd3d3679dda1f62d383b974d23705cab90c
 => => naming to docker.io/library/compose:build.v1
```

로그 확인하기

프로젝트 로그 확인하기

> https://docs.docker.com/engine/reference/commandline/compose_logs/

```
docker compose logs [OPTIONS] [SERVICE...]
```

주요 옵션

Option	Short	Default	Description
`--follow`	`-f`		계속 로그를 표시합니다.
`--tail`	`-n`	`all`	마지막 `n` 개의 로그를 표시합니다.
`--timestamps`	`-t`		`timestamp` 를 표기합니다.
`--since`			특정 시간 이후에 발생한 로그만 표시합니다. - timestamp (e.g. `2013-01-02T13:23:37Z`) - relative (e.g. `42m` , `10s` , ...)

Option	Short	Default	Description
`--until`			특정 시간 이전에 발생한 로그만 표시합니다. - timestamp (e.g. `2013-01-02T13:23:37Z`) - relative (e.g. `42m` , `10s` , ...)

주의사항

- `--tail` 은 전체가 아닌 각 컨테이너별 로그 수를 의미합니다.

[예시] `cloud_wave` 프로젝트의 모든 서비스의 로그를 2개 출력하기

```
$ docker compose -p cloud_wave logs -n 2
cloud_wave-frontend-1  | 2023/12/27 08:28:52 [notice] 1#1: start worker process
23
cloud_wave-frontend-1  | 2023/12/27 08:28:52 [notice] 1#1: start worker process
24
db_postgres            | 2023-12-27 08:28:52.215 UTC [29] LOG:  database system
was shut down at 2023-12-27 08:28:51 UTC
db_postgres            | 2023-12-27 08:28:52.219 UTC [1] LOG:  database system is
ready to accept connections
```

프로세스 확인하기

프로세스 확인하기

> https://docs.docker.com/engine/reference/commandline/compose_top/

```
docker compose top [SERVICES...]
```

[예시] `cloud_wave` 프로젝트의 컨테이너별 프로세스 확인하기

```
$ docker compose -p cloud_wave top
cloud_wave-frontend-1
UID     PID     PPID    C    STIME   TTY   TIME       CMD
root    38700   38649   0    08:58   ?     00:00:00   nginx: master process nginx
-g daemon off;
uuidd   38783   38700   0    08:58   ?     00:00:00   nginx: worker process

uuidd   38784   38700   0    08:58   ?     00:00:00   nginx: worker process

uuidd   38785   38700   0    08:58   ?     00:00:00   nginx: worker process


cloud_wave-server-1
UID    PID     PPID    C    STIME   TTY   TIME       CMD
root   38513   38472   0    08:58   ?     00:00:00   sleep 3600
```

```
db_postgres
UID   PID     PPID    C    STIME   TTY   TIME       CMD
999   38671   38622   0    08:58   ?     00:00:00   postgres

999   38786   38671   0    08:58   ?     00:00:00   postgres: checkpointer

999   38787   38671   0    08:58   ?     00:00:00   postgres: background writer

999   38789   38671   0    08:58   ?     00:00:00   postgres: walwriter

999   38790   38671   0    08:58   ?     00:00:00   postgres: autovacuum launcher

999   38791   38671   0    08:58   ?     00:00:00   postgres: logical replication
launcher
```

config 확인

Config 확인

> https://docs.docker.com/engine/reference/commandline/compose_config/

```
docker compose config [OPTIONS] [SERVICE...]
```

[예시] `cloud_wave`의 `config` 확인하기

```
$ docker compose -p cloud_wave config
name: cloud_wave
services:
  frontend:
    configs:
    - source: server-config
    image: nginx:latest
    networks:
      private: null
    ports:
    - mode: ingress
      target: 80
      published: "80"
      protocol: tcp
    restart: always
  ...
networks:
    ...
volumes:
  db-data:
    name: db-data
    labels:
      description: PostgreSQL 16.1 volume
secrets:
    ...
configs:
  ...
```

삭제

종료한 서비스 삭제

> https://docs.docker.com/engine/reference/commandline/compose_rm/

```
docker compose rm [OPTIONS] [SERVICE...]
```

주요 Options

Option	Short	Default	Description
`--all`	`-a`		Deprecated - no effect
`--force`	`-f`		Don't ask to confirm removal
`--stop`	`-s`		Stop the containers, if required, before removing
`--volumes`	`-v`		Remove any anonymous volumes attached to containers

[예시] 프로젝트 삭제하기

```yaml
version: '3.8'
name: 'cloud_wave'

services:
  success:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - sleep infinity
    networks:
      private: {}
  fail:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - sleep fail
    networks:
      private: {}

networks:
  private:
```

프로젝트의 모든 컨테이너 목록을 다음과 같이 확인합니다.

```
$ docker compose -p cloud_wave ps -a
NAME                   IMAGE          COMMAND                  SERVICE   CREATED
       STATUS                  PORTS
cloud_wave-fail-1      ubuntu:22.04   "/bin/bash -c 'sleep…"   fail      6
seconds ago   Exited (1) 6 seconds ago
cloud_wave-success-1   ubuntu:22.04   "/bin/bash -c 'sleep…"   success   6
seconds ago   Up 6 seconds
```

해당 프로젝트를 삭제하면 다음과 같이 종료된 컨테이너만 삭제되는 것을 볼 수 있습니다.

```
$ docker compose -p cloud_wave rm
? Going to remove cloud_wave-fail-1 Yes
[+] Removing 1/0
 ✓ Container cloud_wave-fail-1  Removed
                        0.0s
```

[예시] 프로젝트 종료 후 삭제하기

`-s` 또는 `--stop`을 이용하여 프로젝트를 삭제하면, 다음과 같이 삭제 전 컨테이너를 종료하는 것을 확인할 수 있습니다.

```
$ docker compose -p cloud_wave rm -s
[+] Stopping 2/2
 ✓ Container cloud_wave-fail-1     Stopped
                        0.0s
 ✓ Container cloud_wave-success-1  Stopped
                       10.1s
? Going to remove cloud_wave-fail-1, cloud_wave-success-1 Yes
[+] Removing 2/0
 ✓ Container cloud_wave-success-1  Removed
                        0.0s
 ✓ Container cloud_wave-fail-1     Removed
                        0.0s
```

```yaml
version: '3.8'
services:
  ubuntu:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - "apt-get update && apt-get upgrade && apt-get install -y curl && sleep 360
    restart: no
  nginx:
    image: nginx:latest
    # expose 대신 ports를 사용합니다.
    # expose:
    #  - 80
    ports:
      - 80:80
    restart: always
```

연습1

연습문제

[연습] 프로젝트 실행 및 `Network` 확인하기

다음 파일을 `example.yaml` 로 저장합니다.

```
version: '3.8'

services:
  ubuntu:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
```

```
      - "apt-get update && apt-get upgrade && apt-get install -y curl && sleep
3600"
    restart: no

  nginx:
    image: nginx:latest
    expose:
      - 80
    restart: always
```

명령어를 이용하여 `project` 를 실행합니다.

```
$ docker compose -f example_1.yaml -p ex1 up -d
[+] Building 0.0s (0/0)
[+] Running 3/3
 ✓ Network ex1_default     Created
 ✓ Container ex1-nginx-1   Started
 ✓ Container ex1-ubuntu-1  Started
```

다음과 같이 `ex1_default` 라는 네트워크가 새로 생성된 것을 확인할 수 있습니다.

```
$ docker network ls
NETWORK ID     NAME             DRIVER    SCOPE
50f36ebc860b   bridge           bridge    local
5480e9b993ea   ex1_default      bridge    local
```

다음과 같이 생성된 `ex1-ubuntu-1` 컨테이너의 정보를 조회하면 `docker run` 을 통해서 실행하였을 때와 다른점을 확인해볼 수 있습니다.

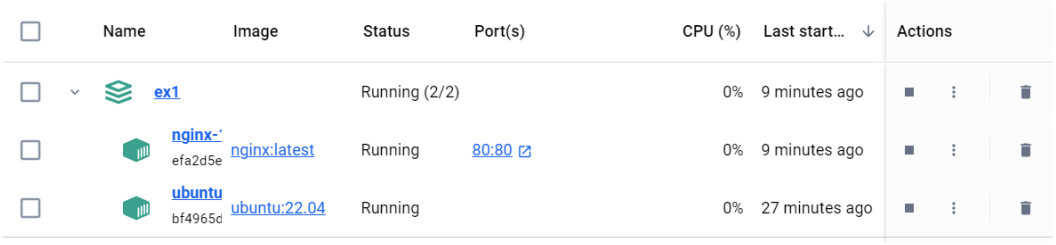
- `default` 네트워크가 아닌 `ex1_default` 네트워크만 할당되어 있습니다.
- `service` 의 이름과 컨테이너의 이름이 `alias` 로 등록되어 있는 것을 확인할 수 있습니다.

```
$ docker inspect ex1-ubuntu-1 -f "{{ println .NetworkSettings.Networks }}"
map[ex1_default:0xc000000240]

$ docker inspect ex1-ubuntu-1 -f "Alias:{{ println
.NetworkSettings.Networks.ex1_default.Aliases }}IP:{{ println
.NetworkSettings.Networks.ex1_default.IPAddress }}"
Alias:[ex1-ubuntu-1 ubuntu f5cfe44b6bee]
IP:172.21.0.3
```

실제로 `ubuntu` 서버에서 다음과 같이 `curl` 명령어를 사용하면, 정상적으로 `nginx` 서버에서 결과를 얻어 오는 것을 확인할 수 있습니다.

```
$ curl nginx:80
<!DOCTYPE html>
<html>
<head>
```

```
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```



해당 우분투서버에 접근한 뒤

docker exec -it ex1-ubuntu-1 /bin/bash

curl nginx:80

연습 2

[연습] 프로젝트 업데이트 - 서비스 포트 추가하기

위에서 생성한 `ex1` 프로젝트의 `nginx` 서비스는 외부에 포트가 열려있지 않아 다음과 같이 `host`에서 접근할 수 없는 상태입니다.

```
$ curl localhost:80
curl: (7) Failed to connect to localhost port 80 after 2237 ms: Couldn't connect
to server
```

`host`에서도 접근할 수 있도록, 다음과 같이 `example.yaml`를 수정합니다.

```
# exaple.yaml
version: '3.8'

services:
  ubuntu:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - "apt-get update && apt-get upgrade && apt-get install -y curl && sleep
3600"
    restart: no

  nginx:
    image: nginx:latest
    # expose 대신 ports를 사용합니다.
    ports:
      - 80:80
```

```
    restart: always
```

다음 명령어를 이용하여 프로젝트를 업데이트 합니다.

```
$ docker compose -f example.yaml -p ex1 up -d
```

다시 `host`에서 접근을 시도하면 다음과 같이 정상적으로 접근이 되는 것을 확인할 수 있습니다.

```
$ curl localhost:80
```

cmd 혹은 ubuntu 호스트 환경에서 날린다

```
C:\Users\kdt>curl localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

연습 3

[연습] 이미지 빌드 후 프로젝트 실행하기

`server.dockerfile`와 `docker-compose.yaml` 파일을 다음과 같이 작성하고 저장합니다.

`폴더 구조`

```
.
├── Dockerfile
└── docker-compose.yaml
```

`server.dockerfile`

```
FROM ubuntu:22.04
RUN apt-get update && apt-get upgrade
```

`docker-compose.yaml`

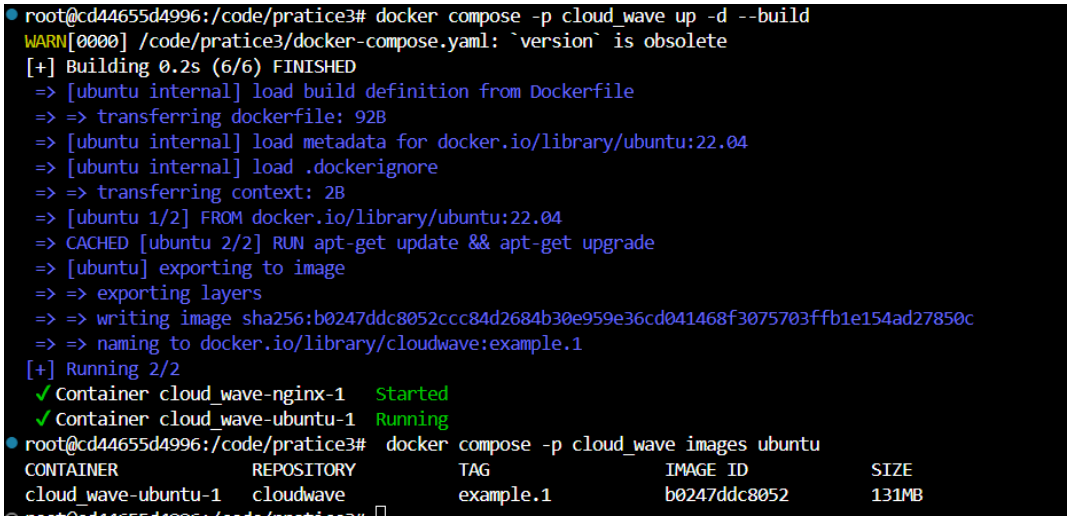
```
version: '3.8'

services:
  ubuntu:
    image: cloudwave:example.1
    build:
      dockerfile: "server.dockerfile"
    entrypoint: /bin/bash
    command:
      - -c
      - "sleep 3600"
    restart: no

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    restart: always
```

`--build` 옵션을 이용하여 다음과 같이 프로젝트를 실행합니다.

```
$ docker compose -p cloud_wave up -d --build

[+] Building 0.0s (6/6) FINISHED
       docker:desktop-linux
 => [ubuntu internal] load .dockerignore
                          0.0s
 => => transferring context: 2B
                          0.0s
 => [ubuntu internal] load build definition from server.dockerfile
                          0.0s
 => => transferring dockerfile: 99B
                          0.0s
 => [ubuntu internal] load metadata for docker.io/library/ubuntu:22.04
                          0.0s
 => [ubuntu 1/2] FROM docker.io/library/ubuntu:22.04
                          0.0s
 => CACHED [ubuntu 2/2] RUN apt-get update && apt-get upgrade
                          0.0s
 => [ubuntu] exporting to image
                          0.0s
 => => exporting layers
                          0.0s
 => => writing image
sha256:9fe790212381bd4b76c35994d028ce911c774b9719577f36ffbc6e8817eefe28
    0.0s
 => => naming to docker.io/library/cloudwave:example.1
                          0.0s
[+] Running 3/3
 ✓ Network cloud_wave_default     Created
                          0.0s
 ✓ Container cloud_wave-nginx-1   Started
                          0.0s
 ✓ Container cloud_wave-ubuntu-1  Started
                          0.0s
```

`docker images` 또는 `docker compose images`를 통해 생성된 이미지를 확인해볼 수 있습니다.

```
$ docker compose -p cloud_wave images ubuntu
CONTAINER             REPOSITORY          TAG                 IMAGE ID
 SIZE
cloud_wave-ubuntu-1   cloudwave           example.1           9fe790212381
 125MB
```



연습 4

[연습] 서비스별로 마지막 5개 로그 출력 후 `follow`하기

```
$ docker compose logs -n 5 -f
ex1-ubuntu-1  | Processing triggers for ca-certificates (20230311ubuntu0.22.04.1)
...
ex1-nginx-1   | 2023/12/25 16:17:36 [notice] 1#1: start worker process 41
ex1-nginx-1   | 2023/12/25 16:17:36 [notice] 1#1: start worker process 42
ex1-nginx-1   | 2023/12/25 16:17:36 [notice] 1#1: start worker process 43
ex1-nginx-1   | 2023/12/25 16:17:36 [notice] 1#1: start worker process 44
ex1-ubuntu-1  | Updating certificates in /etc/ssl/certs...
ex1-ubuntu-1  | 0 added, 0 removed; done.
ex1-ubuntu-1  | Running hooks in /etc/ca-certificates/update.d...
ex1-ubuntu-1  | done.
ex1-nginx-1   | 172.21.0.3 - - [25/Dec/2023:16:27:38 +0000] "GET / HTTP/1.1" 200
615 "-" "curl/7.81.0" "-"
...
```

```
docker compose -p ex1 logs -n 5
```

연습 5

[연습] 프로젝트 삭제하기

다음 `docker-compose.yaml` 파일을 이용하여 프로젝트를 실행합니다.

```
# docker-compose.yaml
version: "3.8"
name: "cloud_wave"

services:
  frontend:
    image: nginx:latest
    ports:
      - "80:80"
    networks:
      - private
    configs:
      - server-config
    restart: always

  server:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - "sleep 3600"
    networks:
      - private
      - db
    restart: no

  postgres:
    container_name: db_postgres
    image: postgres:16.1-bullseye
    networks:
      - db
    expose:
      - 5432
    volumes:
      - db-data:/var/lib/postgresql/data:rw
    secrets:
      - db_password
    env_file:
      - .env
    environment:
      DB_PASSWORD_PATH: /run/secrets/db_password
    restart: always


volumes:
  db-data:
    name: "db-data"
    labels:
      description: "PostgreSQL 16.1 volume"

configs:
  server-config:
    file: "server.conf"

# Secret not encrypted when swarm is off
secrets:
  db_password:
    file: "secret.txt"

networks:
  private:
    name: "private"
  db:
    name: "db"
```

```
$ docker compose up -d
[+] Building 0.0s (0/0)
           docker:desktop-linux
[+] Running 6/6
 ✓ Network db                        Created
                                0.0s
 ✓ Network private                   Created
                                0.0s
 ✓ Volume "db-data"                  Created
                                0.0s
 ✓ Container db_postgres             Started
                                0.0s
 ✓ Container cloud_wave-frontend-1   Started
                                0.0s
 ✓ Container cloud_wave-server-1     Started
                                0.0s
```

`docker-compose.yaml`에서 `frontend` 서비스를 제거합니다.

```
# docker-compose.yaml
version: "3.8"
name: "cloud_wave"

services:
  server:
    image: ubuntu:22.04
    entrypoint: /bin/bash
    command:
      - -c
      - "sleep 3600"
    networks:
      - private
      - db
    restart: no

  postgres:
    container_name: db_postgres
    image: postgres:16.1-bullseye
    networks:
      - db
    expose:
      - 5432
    volumes:
      - db-data:/var/lib/postgresql/data:rw
    secrets:
      - db_password
    env_file:
      - .env
    environment:
      DB_PASSWORD_PATH: /run/secrets/db_password
    restart: always


volumes:
  db-data:
    name: "db-data"
    labels:
      description: "PostgreSQL 16.1 volume"

configs:
  server-config:
    file: "server.conf"

# Secret not encrypted when swarm is off
secrets:
  db_password:
    file: "secret.txt"

networks:
  private:
    name: "private"
  db:
    name: "db"
```

이후, 다음과 같이 삭제를 시도하면 `frontend` 서비스와 `volume`을 제외한 나머지 컴포넌트에 대해서만 삭제를 시도하는 것을 확인할 수 있습니다.

```
$ docker compose -p cloud_wave down
[+] Running 4/4
 ✓ Container db_postgres             Removed
                                0.1s
 ✓ Container cloud_wave-server-1     Removed
                               10.2s
 ! Network private                   Resource is still in use
                                0.0s
 ✓ Network db                        Removed
                                0.0s
```

다음과 같이 `docker-compose.yaml`에서 제거된 `frontend` 서비스가 살아있는 것을 확인할 수 있습니다.

```
$ docker compose ps
NAME                    IMAGE          COMMAND                  SERVICE
  CREATED          STATUS          PORTS
cloud_wave-frontend-1   nginx:latest   "/docker-entrypoint.…"   frontend   5
minutes ago   Up 5 minutes   0.0.0.0:80->80/tcp
```

다음과 같이 `--volumes`와 `--remove-orphans` 옵션을 이용하면, 남아있는 컨테이너와 볼륨을 삭제할 수 있습니다.

```
$ docker compose down --volumes --remove-orphans
[+] Running 3/3
 ✓ Container cloud_wave-frontend-1   Removed
                                0.2s
 ✓ Volume db-data                    Removed
                                0.0s
 ✓ Network private                   Removed
                                0.1s
```