7/12

요약

Docker → Docker compose → kubernetes

docker(container)

dockerfile ——→ image ——→ container(단일 개체) ——→ service(compose 복합 개체) ——→

FROM (Base Image) → Tag x latest default 사용 Tag o 태그 적힌 버전 가져온다
시점에 따라서 ID Digest가 변경될 수 있다. 둘 다
따라서 특정 Digest를 사용하면 변경의 걱정 없이 항상 같은 image를 사용 가능하다.

docker hub pull 계정명/레포지토리명:Tag

Docker file 명령어 요약

RUN : Build 시에 수행할 명령
ADD : 파일 복사 URL 지원
COPY : 파일 복사 HOST에서만
USER : 설치 퍼미션 문제 때문에 설치 시에는 ROOT로 사용할 때는 다른 걸로 하는 경우가 생긴다
ENV : 환경변수(Default) 평문이기에 password 같은 것을 넣어선 안된다 .env 파일로 추후 주입하는 것이 맞다. 하드코딩 금지
ARG : Build 시에만 사용하는 변수

CMD → 인자
Entrypoint → 명령어

docker build -f filepath -p 이미지이름 or docker build -p 이미지이름 .

IMAGE 요약

Layer의 합

바닥에 base image 그 위에 변경사항들이 온다.

docker file의 명령어가 하나의 layer로 존재

cache의 존재로 apt—get—update같은 결과가 다른 명령어 할 때는 - -no -cache 사용

오랜만에 돌리는 상황이면 no cache를 하자

변경이 많은 것은 아래에 적은 것은 위에 캐시 최대한 사용

> 💡 항상 cache가 의도한 대로 사용되는지 생각해봐야 한다.

docker run 이미지명 [cmd]

layer들은 read only

컨테이너

맨 위 layer에 container layer가 하나 붙는다 읽기 쓰기가 가능한 형태 run하면서 붙는다
이게 붙으면서 container가 된다. 컨테이너가 삭제되면 이 layer가 삭제되기 때문에 모든 변경사항이 유실된다.

기본적으로 격리가 되어있기에 open을 하려면 -v로 volume 연결 -network로 네트워크 연결 -p로 port를 open해야 한다.

port는 host port : container port로 설정

nginx → default 80 port
container의 port는 격리되어 있고 하나의 service만 쓰기에 port명을 굳이 변경할 필요가 없음

host network 쓰는 것은 조심해야 겹칠 수 있기 때문에
attach 메인 프로세스 접근
exec 서브 프로세스 접근

Docker compose

yaml 파일을 어떻게 작성하는가 핵심..

(optional) version : 3.8
(optional이지만 웬만하면 적는다)name: 프로젝트 이름

name 우선순위

1순위 -p 옵션 이름

2순위 yaml 이름

3순위 디렉토리 이름

service :

어떠한 service가 들어가는지 명시

db, front, back ,server 같이 여러 service가 존재하고

또 front에 서버가 여러 개 있을 수도 있다.

stand alone이든 여러 개 있든 똑같다.

docker compose <project name> service

service 이름 :

image : 사용할 image:tag

Entrypoint : 이미지 entrypoint 오버라이드

command :

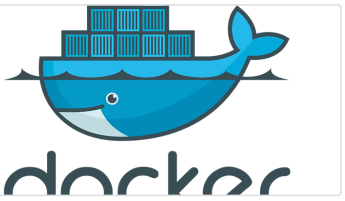
ports: port 개방

expose :

[Docker] ports와 expose의 차이

개발, 배포, 운영에 큰 장점이 있는 가상화/컨테이너 애플리케이션인 도커(docker)는 그 이점들이 있기 때문에 최근 많이 사용이 되고 있습니다. 도커를 사용함에 있어서 초보자들에게 매우 헷갈리는 부분으로 ports와 expose 설정이 있습니다. 두 설정 모두 컨테이너의 포트번호 설정

https://growd.tistory.com/77



envirement:

env-file: 대량으로 넣을 때 사용 파일을 직접 주입

환경 변수 따로 설정한게 1순위

env-file에서 맨 밑에 있는게 2순위로 높다

순서를 잘 기억해야..

networks:

-사용할이름

-net:

alias: 별칭 사용가능

-사용하고 싶은 dns이름

volume:

-host: container path

host를 사용하려면 상대경로 or 절대경로

이름만 적혀있으면 volume인 것

volume의 sub만 사용하는 건 못한다 volume 통짜로 해야함

host 연결

~/src : /code

volume 연결

src : /code

networks: bridge 네트워크만 사용가능하다.

다른 네트워크 사용하려면 network_mode를 사용해라.

net: ← 내 yaml 파일에서 사용할 네트워크 변수 이름

name: private ← 실제 외부 네트워크 이름

external : true

label : 관리를 위한 목적 명시하는 것이 좋다 쿠버네티스를 위해

배포나 패치를 위해 이름 지정

라벨이 없으면 관리가 안된다.

volumes: 네트워크랑 구조가 똑같다.

vol1 : ← key 값으로 사용 내 yaml 파일에서 사용할 volume 이름

name : 외부 이름

external : true

label :

config : ——————————→ config map<쿠버네티스 objectin>

key :

file : 경로

secret: ——————————→ secret

	config	secret
	파일	파일
	외부 공개	비밀 값 password ap token
	암호화 x	암호화 x compose에서 작동 x docker swarm에서만 가능
마운트 되는 기본 위치	/ (root 기본 위치)	/run/secrets/

profile

특정 서비스는 compose 할 때 뜨지 않아도 되는 경우 profile 명시하여 뜨지 않게끔

profile이 존재하지 않으면 up 했을 때 항상 뜬다

디버깅 용도로 필요한 서비스들이 있다. 배포할때는 필요 x

service :

back :

db :

test :

profile:

-debug

docker compose up -d : back , db만 생성

docker compose -profile debug up -d : 다 생성

depend on

서비스 간에 우선순위가 생긴다.

pad는 최소 배포단위가 아니다. compose랑 유사할 뿐

anchor&alias

& alias.name * alias

key:value ——————————————————→ << *key = yaml

key2: *key = value

k1:v1

k2:v2

health check 가장 중요 devops 에선

서버가 의도한 대로 정상적으로 작동하는지 모니터링

기능(수치화)

평균 응답속도 , db

가치를 정상적으로 전달하는지 체크

1. 서버가 켜져있는지
2. 특정 주소로 get,post 날렸을 때 정상적으로 반응하는가

쿠버네티스

동적이다. 선언적이다. 가지고 있는 서버 중 하나를 띄우기 때문에 ip를 제대로 알고 사용하기 힘듦 그렇기에 dns를 사용한다. 다이렉트 ip는 사용 거의 불가능

multi node 여러 개의 서버

health check

2.3.5. health check

> https://docs.docker.com/engine/reference/builder/#healthcheck

```yaml
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 10s
  retries: 3
  start_period: 40s
  start_interval: 5s
```

- `healthcheck`를 이용하여 서비스의 상태를 확인할 수 있습니다.
- 시간은 `{value}{unit}` 형태로 작성하며, 지원하는 시간 단위는 다음과 같습니다.
  - `us(microseconds)`
  - `ms(milliseconds)`
  - `s(seconds)`
  - `m(minutes)`
  - `h(hours)`

Attributes	Description
`test`	컨테이너 상태 체크를 위한 명령어를 설정합니다. `NONE`으로 설정한 경우 `Dockerfile`에 정의된 `HEALTHCHECK`를 비활성화 합니다.
`interval`	상태 체크 간격을 의미합니다.
`timeout`	응답까지 걸리는 최대 시간을 의미합니다. 만약 해당 시간까지 응답이 도착하지 않으면 `fail`로 판단합니다.
`start_period`	컨테이너 실행까지 소요되는 시간을 의미합니다. 해당 시간동안에는 상태가 `fail`인 경우에도 카운트되지 않습니다.

Attributes	Description
`start_interval`	`start_period` 내에서의 체크 간격을 의미합니다.
`disable`	`HEALTHCHECK`를 비활성화 합니다. `test: ["NONE"]`과 동일합니다.

test로 서버 상태 체크

인터벌을 잘 설정해야..

time

연습 1

연습문제

[연습] Anchor & Alias 이용하여 `yaml` 파일 작성하기

```yaml
version: '3.8'

x-common:
  &common
  restart: always
  volumes:
    - source:/code
  environment:
    &default-env
    BY: "x-common"

services:
  ubuntu:
    <<: *common
    image: ubuntu:22.04
    environment:
      <<: *default-env
      FROM: "env definition"
    entrypoint: /bin/bash
    command:
      - -c
      - echo 'env from ${FROM}' && echo env from $${FROM}
    restart: no

volumes:
  source:
```

연습 2

[연습] depend_on 을 이용하여 DB 생성 후 application 실행하기

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

services:
  postgres:
    image: postgres:16.1-bullseye
    environment:
      - POSTGRES_PASSWORD=mysecretpassword
```

```yaml
  server:
    image: ubuntu:22.04
    stdin_open: true # docker run -i
    tty: true        # docker run -t
    depends_on:
      - postgres

  pgadmin:
    image: dpage/pgadmin4:7.4
    environment:
      - PGADMIN_DEFAULT_EMAIL=user@sample.com
      - PGADMIN_DEFAULT_PASSWORD=SuperSecret
    depends_on:
      - postgres
      - server
    profiles:
      - debug
```

종합 실습 1

종합 문제

[실습-1] `code-server` 실행을 위한 `docker-compose.yaml` 작성하기

- 다음과 같이 `secret`을 생성 및 정의합니다.
  - 다음 명령어를 이용하여 `password.txt`를 생성합니다.

    > 임의의 텍스트 파일을 메모장등을 이용해서 생성해도 무방합니다.

    ```
    # Window - CMD
    $ echo %RANDOM% > password.txt

    # Linux
    $ openssl rand -base64 14 > password.txt
    ```

  - 위에서 생성한 `password.txt` 파일을 `docker-compose.yaml`에 정의합니다.
    - 이름은 `code-server-password`로 설정합니다.
- 다음과 같이 `volume`을 생성 및 정의합니다.
  - `local_code` 이름으로 `volume`을 생성합니다.
  - 생성한 `volume`을 `docker-compose.yaml`에 정의합니다.
- `.env` 파일을 다음과 같이 생성합니다.

```
DEFAULT_WORKSPACE=/code
PUID=1000
PGID=1000
```

- `code-server` 서비스를 다음과 같이 정의합니다
  - 이미지는 `Docker Hub`에 `Push` 했던 `cloudwave:practice.v1`를 사용합니다.

  - 컨테이너 이름은 `ide`로 설정합니다.
  - 다음과 같이 환경 변수를 설정합니다.

    ```
    FILE__PASSWORD: /run/secrets/<SECRET_NAME>
    ```

  - `.env` 파일을 이용하여 추가로 환경 변수를 설정합니다.
  - `/code` 디렉토리를 `Working directory`로 설정합니다.
  - 로컬 머신의 `8443` 포트와 컨테이너의 `8443` 포트를 바인딩합니다.
  - 다음 `secret`을 사용할 수 있도록 설정합니다.
    - `code-server-password`
  - 다음과 같이 `volume`을 정의합니다.
    - 로컬 머신의 `/var/run/docker.sock`를 컨테이너의 `/var/run/docker.sock`로 마운트합니다.
    - `volume`의 `local_code`를 컨테이너의 `/code/local`에 마운트합니다.
- 작성한 `docker-compose.yaml`을 이용하여 프로젝트를 실행합니다.
- 브라우저에서 `localhost:8443`을 입력하고 `VSC`에 접속합니다.
  - 비밀번호는 생성한 `password.txt` 파일을 보고 확인합니다.

```yaml
version: "3.8"
name: 'test_ide'
services:
  ide:
    image: dlwpdnr213/cloudwave:practice.v1
    environment:
      FILE__PASSWORD: /run/secrets/code_server_password
      #working_dir: $DEFAULT_WORKSPACE
    env_file:
      - .env
    # entrypoint: /bin/bash
    # command:
    #   - -c
    #   - "sleep infinity"
    secrets:
      - source: code_server_password
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - v1:/code/local
    ports:
      - "8443:8443"
secrets:
  code_server_password:
    file: ./password.txt
volumes:
  v1:
    name: local_code
    external: true
```

종합 실습 2

[실습-2] `git-sync` 서비스 추가하기

- 다음과 같이 `volume`을 추가로 생성 및 정의합니다.
  - `remote_code` 이름으로 `volume`을 생성합니다.
  - 생성한 `volume`을 `docker-compose.yaml`에 정의합니다.
- `GitSync` 서비스를 다음과 같이 정의합니다.
  - 이미지는 `registry.k8s.io/git-sync/git-sync:v4.1.0`를 사용합니다.
  - `source_code` 볼륨을 `/tmp`에 마운트합니다.
  - 다음과 같이 환경 변수를 설정합니다.

```
GITSYNC_REPO=https://github.com/matenduel/cloudwave
GITSYNC_ROOT=/tmp/git
GITSYNC_REF=main
GITSYNC_DEPTH=1
GITSYNC_ONE_TIME=1
```

  - `profile`이 `init`인 경우에만 실행되도록 설정하세요.
- `code-server` 서비스에 다음 사항을 추가로 정의합니다.
  - `volume`의 `remote_code`를 컨테이너의 `/code/remote`에 마운트합니다.
  - `depend_on`을 이용하여 `GitSync` 컨테이너가 생성된 다음 컨테이너가 실행되도록 설정하세요.
    - `GITSYNC_ONE_TIME`가 `1`이라면 `service_completed_successfully`로 설정하세요.
    - `GITSYNC_ONE_TIME`가 설정되지 않았다면 `service_started`로 설정하세요.

```yaml
version: "3.8"
name: 'test_gitsync_ide'
services:
  GitSync:
    image: registry.k8s.io/git-sync/git-sync:v4.1.0
    volumes:
      - v2:/tmp
    environment:
      - GITSYNC_REPO=https://github.com/matenduel/cloudwave
      - GITSYNC_ROOT=/tmp/git
      - GITSYNC_REF=main
      - GITSYNC_DEPTH=1
    profiles:
      - init
  ide:
    image: dlwpdnr213/cloudwave:practice.v1
```

```
    environment:
      FILE__PASSWORD: /run/secrets/code_server_password
    env_file:
      - .env

    secrets:
      - source: code_server_password
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - v1:/code/local
      - v2:/code/remote
      - config:/config
    depends_on:
      GitSync:
        condition:  service_started

    ports:
      - "8444:8443"




secrets:
  code_server_password:
    file: ./password.txt
volumes:
  v1:
    name: local_code
    external: true
  v2 :
    name: remote_code
    external: true
  config :
    name : config
    external: true
```

종합 실습3

[실습-3] `build`를 이용한 `docker compose` 프로젝트 생성하기

- 다음 조건을 만족하는 `Dockerfile`을 작성합니다.
  - `ubuntu:22.04` 이미지를 베이스로 사용합니다.
  - `apt-get update`와 `apt-get upgrade`를 수행해야 합니다.
  - `apt`를 이용해서 `dnsutils`와 `wget`을 설치합니다.
- `ubuntu` 서비스를 다음과 같이 정의합니다.
  - 컨테이너의 이름을 `server`로 설정합니다.
  - 이미지 이름은 `cloudwave:ubuntu.dig.v1`으로 설정합니다.
  - 위에서 만든 `Dockerfile`을 기반으로 빌드되도록 설정합니다.
  - 컨테이너가 정지되지 않도록 다음과 같이 `command`를 설정합니다.
    - `sleep infinity`
- `web-app` 서비스를 다음과 같이 정의합니다.
  - 이미지는 `nginx:latest`를 사용합니다.
  - `replicas`를 3으로 설정합니다.
  - 80번 포트를 `Expose` 합니다.
- 작성한 `docker-compose.yaml`을 이용하여 이미지를 빌드한 다음 프로젝트를 실행합니다.
- `server` 컨테이너에 접속하여 다음 명령어를 실행합니다.

```
dig web-app
```

```
#Dockerfile
FROM ubuntu:22.04
RUN apt-get update && apt-get upgrade
RUN apt-get install -y dnsutils  && apt-get install -y wget
```

```
#docker-compose.yaml

version: "3.8"
services:
  server:
    image:  cloudwave:ubuntu.dig.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 infinity"
    restart: no
  web-app:
    image: nginx:latest
    expose:
      - 80
    deploy:
      replicas: 3
    restart: always
```

```
# dig web-app

; <<>> DiG 9.18.24-0ubuntu0.22.04.1-Ubuntu <<>> web-app
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34022
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;web-app.                       IN      A

;; ANSWER SECTION:
web-app.                600     IN      A       172.25.0.3
web-app.                600     IN      A       172.25.0.4
web-app.                600     IN      A       172.25.0.5

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
;; WHEN: Fri Jul 12 03:11:57 UTC 2024
;; MSG SIZE  rcvd: 94

#
```

docker exec test3-server-1 bin/bash -c "dig web-app”