CLOUD SQUAD

클라우드 5주차

2025.05.28

BY 김대현

Contents

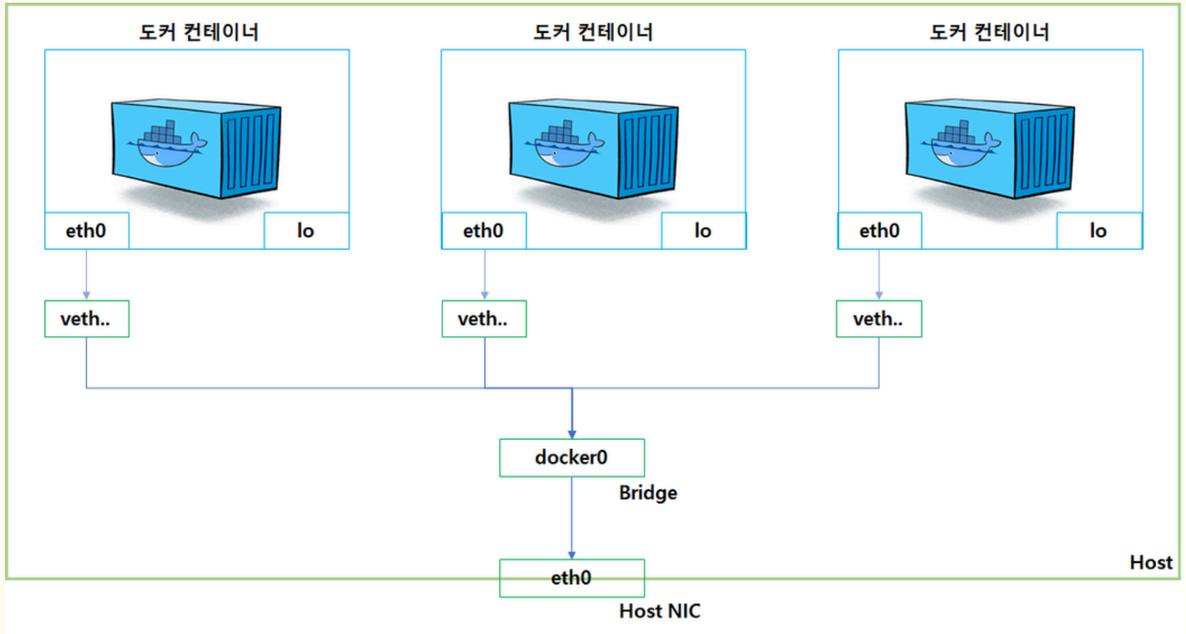
Docker Network

01 Docker Network

Docker Network

- **Docker 공식문서: https://docs.docker.com/reference/cli/docker/network/**

- **Docker Network**
- Docker 컨테이너 간의 통신을 관리하는 시스템. 컨테이너를 서로 연결하고 외부 네트워크와 통신할 수 있게 해주는 가상 네트워킹 인프라를 제공



01 Docker Network

Docker Network 종류

- 네트워크 드라이버 - Docker는 여러 네트워크 드라이버를 지원.
  - bridge: 기본 네트워크 드라이버로, 같은 호스트 내 컨테이너 간 통신에 사용
  - host: 컨테이너가 호스트의 네트워킹을 직접 사용
  - overlay: 다중 Docker 호스트 간의 통신을 위한 드라이버
  - macvlan: 컨테이너에 MAC 주소를 할당하여 물리적 장치처럼 작동하게 함
  - none: 모든 네트워킹을 비활성화
- 사용자 정의 네트워크 - 필요에 따라 사용자 정의 네트워크를 생성하여 애플리케이션의 네트워킹 요구 사항을 충족시킬 수 있습니다.
- 컨테이너 간 통신 - 같은 네트워크에 연결된 컨테이너는 서로 통신할 수 있으며, 컨테이너 이름을 사용하여 DNS 해결이 가능합니다.
- 포트 매핑 - 컨테이너의 포트를 호스트 시스템의 포트에 매핑하여 외부에서 컨테이너 서비스에 접근할 수 있게 합니다.
- 네트워크 격리 - 서로 다른 네트워크에 있는 컨테이너는 기본적으로 통신할 수 없어 보안을 강화합니다.

01 Docker Network

Docker Network

bridge (기본)

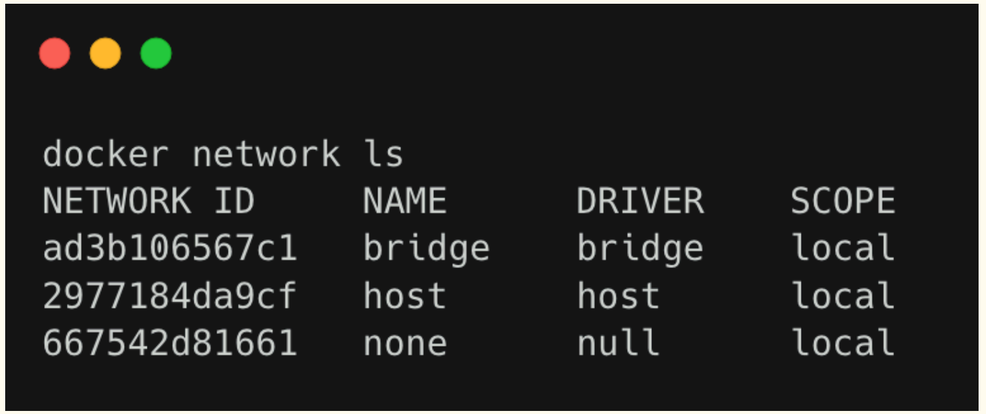
- Docker가 생성하는 기본 가상 네트워크.
- 컨테이너끼리 IP 또는 컨테이너 이름(DNS)으로 통신 가능.

host

- 컨테이너가 호스트 머신의 네트워크 네임스페이스를 그대로 사용.
- 네트워크 분리가 필요 없을 때(고성능 네트워크) 사용.

none

- 네트워크 완전 비활성화.
- 외부 통신이 전혀 필요 없는 컨테이너에 사용.



01 Docker Network

사용자 정의 Network 생성 & 삭제

- 기본 제공(bridge/host/none) 네트워크는 삭제할 수 없습니다.
- 삭제도 rm, prune으로 삭제 가능 (기본 제공 네트워크는 삭제 불가)

```
# 네트워크 생성
docker network create my-first-network

# 생성 확인
docker network ls

# 네트워크 삭제
docker network rm my-first-network
```

```
docker network rm my-first-network
my-first-network

docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
ad3b106567c1   bridge    bridge    local
2977184da9cf   host      host      local
667542d81661   none      null      local

docker network rm bridge
Error response from daemon: bridge is a pre-
defined network and cannot be removed
```

01 Docker Network

Container에 Network 연결

- 새 컨테이너를 실행하며 특정 네트워크에 연결하기
- 컨테이너 실행시 네트워크 옵션을 통해서 컨테이너 연결 가능

```
docker network create mbti-network
5d54f1726b1fcfb9d1874d33f55ea5b81c08956df37ae4004fb559cfd82bb207

docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
ad3b106567c1   bridge         bridge    local
2977184da9cf   host           host      local
5d54f1726b1f   mbti-network   bridge    local
667542d81661   none           null      local
```

```
docker container run \
> --name mbti \
> -e PORT=3000 \
> --rm \
> -d \
> --network mbti-network \
> toby/mbti:embedded-db
c499727720e4557
```

- network 연결 (만약 컨테이너 실행시 & run때 네트워크 옵션 지정 안했을때)

```
# docker network connect [생성 네트워크] [연결할 컨테이너]

docker network connect mbti-network mbti
```

01 Docker Network

Network정보 확인 (inspect)

- 컨테이너에 연결된 네트워크 정보 확인
  - inspect → 도커의 요소를 다루는 네트워크 정보

```
docker container inspect mbti
```

- 출력 중 "NetworkSettings" → "Networks" 항목에서
  - IPAddress : 컨테이너의 IP 주소
  - DNSNames : 사용자 정의 네트워크에서만 컨테이너 이름(DNS) 제공

```
"Networks": {
  "my-first-network": {
    "IPAddress": "172.18.0.2",
    "DNSNames": ["mbti", "..."]
  }
}
```

- 실제로 출력되는 네트워크 정보 하단에 존재


```
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "b32db204759767c852dc481024c9ae481d0459d4ddc8978a2bc11d937a3d8e4f",
            "SandboxKey": "/var/run/docker/netns/b32db2047597",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "",
            "Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "",
            "IPPrefixLen": 0,
            "IPv6Gateway": "",
            "MacAddress": "",
            "Networks": {
                "mbti-network": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": [
                        "c499727720e4"
                    ],
                    "MacAddress": "02:42:ac:13:00:02",
                    "NetworkID": "5d54f1726b1fcfb9d1874d33f55ea5b81c08956df37ae4004fb559cfd82bb207",
                    "EndpointID": "13c2c661bb72955d82dbf8d65b923e5eab3ef018b9bae6cce5988659a54d4dd6",
                    "Gateway": "172.19.0.1",
                    "IPAddress": "172.19.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null,
                    "DNSNames": [
                        "mbti",
                        "c499727720e4"
                    ]
                }
            }
        }
    }
]
```

01 Docker Network

Docker Network 연결

- curlimages/curl 이미지를 같은 네트워크에 연결하여 다른 컨테이너에 요청

```
# 실패 예: 네트워크 미연결
docker run --rm curlimages/curl curl my-app:3000/api/healthcheck

# 성공 예: 같은 네트워크 지정
docker run --rm --network my-first-network curlimages/curl \
  curl my-app:3000/api/healthcheck
```

- IP 주소 직접 요청

```
docker run --rm --network my-first-network curlimages/curl \
  curl 172.18.0.2:3000/api/healthcheck
```

01 Docker Network

Docker Network 연결 (how?)

- 컨테이너 외부에서 통신해보기
  - (다른 컨테이너에서 요청 보내보기)

```
# 이미지 다운로드

docker image pull curlimages/curl
Using default tag: latest
latest: Pulling from curlimages/curl
cb8611c9fe51: Pull complete
c1a172bb1214: Pull complete
4ca545ee6d5d: Pull complete
Digest:
sha256:c1fe1679c34d9784c1b0d1e5f62ac0a79fca01fb6377cdd33e90473c6f9f
9a69
Status: Downloaded newer image for curlimages/curl:latest
docker.io/curlimages/curl:latest
```

- container 실행
  - 당연히 안됨. mbti container와 같은 네트워크에 연결이 안되었기 때문

```
▶ docker container run \
> --rm \
> curlimages/curl \ # 이미지 이름
> curl mbti:3000/api/healthcheck # 명령어 컨테이너:포트번호/api
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0curl: (6) Could not resolve host: mbti
```

01 Docker Network

Docker Network 연결 (how?)

- 같은 네트워크에 연결

```
docker container run \
> --rm \
> --network mbti-network \
> curlimages/curl \
> curl mbti:3000/api/healthcheck
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--  0     0    0
    0    0     0      0      0 --:--:-- --:--:-- --:--100    15    0    15    0
0    132      0 --:--:-- --:--:-- --:--:--   131
{"status":"ok"}
```

- ip address에 요청 보내기

```
docker container run \
--rm \
--network mbti-network \
curlimages/curl \
curl -s 172.19.0.2:3000/api/healthcheck

{"status":"ok"}%
```

- 네트워크 옵션 없이 기본 (bridge) 네트워크에 연결
  - (컨테이너 재실행)

```
docker container run \
> --name mbti \
> -e PORT=3000 \
> --rm \
> -d \
> daehyunbigbread/mbti:embedded-db
59350dff1915a5e06b8306db209037aba4867652ca654835791479d93707df11
```

01 Docker Network

Docker Network 연결 (how?)

- 보면 Networks 아래 bridge 확인 가능
  - DNSNames: Null → 기본제공 되는 네트워크는 DNS 이름 제공 안함
  - ip로 연결 가능 (다만 컨테이너가 실행될때 마다 달라질수도)
  - 다만 DNS이름으로 하면 Container 이름을 기본값으로 가지기 때문에 ip주소보단 사용하는게 편리
  - 사용자 정의 네트워크를 사용하는것이 권장됨.

```
docker container inspect mbti

......

        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"acd9f571c9b8f9e35816ab2b4e29b6eb5849ea6ce875518dc14151ca46a55d33",
            "SandboxKey": "/var/run/docker/netns/acd9f571c9b8",
            "Ports": {},
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"cfc9d71d20a13fa1a62305bd3f51f62b4c568adebc73720e5ff204a6809bcaea",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "NetworkID":
"ad3b106567c1701295646be195836515caf69564dc61f38f4ee0aa5215c6b16e",
                    "EndpointID":
"cfc9d71d20a13fa1a62305bd3f51f62b4c568adebc73720e5ff204a6809bcaea",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DriverOpts": null,
                    "DNSNames": null
                }
            }
        }
    }
]
```

01 Docker Network

Port Binding (포트 바인딩)

- **Port Mapping & Port Binding이라고 함**
- 포트 바인딩은 컴퓨터나 서버에서 네트워크 포트를 특정 프로세스에 연결하는 과정.
  - 쉽게 말해서, 특정 포트 번호를 통해 들어오는 데이터나 요청을 어떤 애플리케이션이 처리할지 지정

- **Port Mapping (-p 또는 -P)**
  - -p 호스트포트:컨테이너포트 로 호스트와 컨테이너 간 포트 연결
  - 예: -p 4000:3000 → 호스트 4000번 포트가 컨테이너 3000번 포트로 라우팅

```
docker run -d --name my-app -p 4000:3000 my-image
# 외부에서는 http://호스트IP:4000 으로 접근
```

```
# 컨테이너 실행시에도 포트 바인딩 가능

docker container run \
> --name mbti \
> -e PORT=3000 \
> --rm \
> -d \
> -p 4000:3000 \
> toby/mbti:embedded-db
164d739a9ac74b2bc618a661d73d3bf4156e2ff4b2d4f14f4787f0c2100f14a3
```

01 Docker Network

Port Binding (포트 바인딩)

- 확인해보기 (PORTS 에서 확인가능)

```
docker container ls
CONTAINER ID   IMAGE                      COMMAND           CREATED         STATUS         PORTS                    NAMES
164d739a9ac7   toby/mbti:embedded-db   "npm run start"   5 seconds ago   Up 5 seconds   0.0.0.0:4000->3000/tcp   mbti
```

- Inspect로도 확인 가능 (이렇게 PortBindings 옵션에서 확인 가능)

```
docker container inspect mbti
[
    {
        "Id": "164d739a9ac74b2bc618a661d73d3bf4156e2ff4b2d4f14f4787f0c2100f14a3",
        "Created": "2025-01-07T13:05:52.446824711Z",
        "Path": "npm",

....

            "PortBindings": {
                "3000/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "4000"
                    }
                ]
            },
```

01 Docker Network

Port Binding (포트 바인딩)

- **브라우저에서 요청할땐, Host Port인 4000번으로 해야 확인** 가능 → {"status":"ok"}
  - 포트 공개시 호스트만이 아닌 같은 네트워크에 속한 컨테이너도 포트를 통하여 통신 가능

- 컨테이너 내부에서 호스트 요청 보낼시엔 로컬호스트에서 컨테이너 자신을 가리키기 때문에 이렇게 사용

```
host.docker.internal:4000/api/healthcheck
```

```
docker network create other-network
beb50a8d0ed2369467c4ff2148a51ff19f87fdfd77cb6cb98e73f9b70054e421

docker container run \
> --rm \
> --network other-network \
> curlimages/curl \
> curl host.docker.internal:4000/api/healthcheck
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0{"status"100    15    0    15    0     0    597      0 --:--:-- --:--:-- --:--:--
600
```

01 Docker Network

Expose

- **expose → 도커파일에 작성 or Container 실행시 옵션으로도 사용 가능** (옵션값: 포트의 값으로 부여)
  - 퍼블리시 옵션과 혼동 가능 → 퍼블리시: 실제 포트 공개, Expose: 포트 공개 X (표준 기술이 아니기 때문)
  - 포트 공개 방식은 플랫폼마다 달라질수도 있기 때문에, 어떤 포트를 공개를 해야하는지 컨테이너 정보에 기록.
  - 컨테이너 정보에 기록하는 역할 (expose)

```
docker run -d --expose 3000 my-image
# inspect 시 ExposedPorts에만 나타남
```

01 Docker Network

Expose

- 컨테이너 내부에서 실행되는 서비스를 호스트 시스템이나 외부 네트워크에서 접근할 수 있도록 설정하는 것.
- Dockerfile 또는 --expose 옵션으로 컨테이너 내부 포트를 메타데이터에 기록만 함
- 실제 외부 노출은 되지 않음(포트 바인딩과 별도)

```
docker container run \
> --name mbti \
> -e PORT=3000 \
> --rm \
> -d \
> --expose 3000 \
> toby/mbti:embedded-db
39757d22477ec51f18227c992d481dfb0c56e1fa053e2f0c364cf9515d7778a9
```

- inspect로 확인
- Config 속성 아래에 ExposedPorts로 확인가능
  - 다만 실제로 포트가 공개된건 아님 (접속 불가)

```
docker container inspect mbti
[
    {
        "Id":
"39757d22477ec51f18227c992d481dfb0c56e1fa053e2f0c364cf9515d77
78a9",
        "Created": "2025-01-07T13:21:22.330375461Z",

.....

        "Config": {
            "Hostname": "39757d22477e",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "3000/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,

.....

]
```

Docker Compose

02 Docker compose

Docker Compose

- **Docker 공식문서:**
  https://docs.docker.com/reference/compose-file/
  https://docs.docker.com/reference/cli/docker/compose/

- **Docker Compose: 여러 컨테이너(서비스)를 하나의 설정 파일로 정의, 관리할 수 있는 도구**
- docker CLI를 개별적으로 호출하지 않고, 프로젝트 단위로
  - 이미지 빌드(Build)
  - 컨테이너 실행·종료
  - 네트워크·볼륨 생성·삭제 등을 일괄 처리할 수 있음
- 기본적으로 docker-compose.yaml(또는 docker-compose.yml) 파일을 읽어서 동작

02 Docker compose

Docker Compose 주요 이점

- **통합된 설정**
  - 컨테이너별 image·build·environment·ports·volumes·networks 등을 YAML로 선언
- **프로젝트 격리**
  - Compose 프로젝트마다 네트워크·볼륨을 자동으로 네임스페이스화
  - **서로 다른 프로젝트가 충돌 없이 공존**
- **간단한 명령어**
  - up·down·exec 등으로 여러 컨테이너 관리
- **의존성 처리**
  - depends_on으로 컨테이너 간 시작 순서 지정 가능

02 Docker compose

Docker Compose

- **docker-compose를 실행하는 단위를 프로젝트 라고 함.**
  - **프로젝트 이름 설정**
  - yaml → 들여쓰기로 하위 디렉토리 지정

- 도커 컴포즈 yaml 파일 실행 명령어 (root에서)
  - 특정 경로 지정 원할시 -f 옵션 사용
  - 빌드 관련 옵션 설정시? 컴포즈가 실행될때 이미지가 없다면 토대로 빌드.
    - 만약 이미지가 있다면? 빌드 하지 않음. 최신 코드로 빌드 하지 않음
    - 이럴때 docker compose 명령어를 사용할때 —build 옵션 제공

```
docker compose up

docker compose -f ./docker-compose.yaml up -d --build
```

```yaml
# (1) 프로젝트 이름 (생략 시 디렉토리명)
name: mbti

# (2) 서비스 정의
services:
  app:
    image: toby/mbti:mysql
    build:
      context: .                # 도커 빌드 컨텍스트
      dockerfile: ./Dockerfile # Dockerfile 경로
      args:
        - NODE_VERSION=20.15.1
      pull: true                # 항상 최신 베이스 이미지 가져오기
    container_name: app         # 컨테이너 이름 고정
    environment:                # 환경변수
      - PORT=3000
      - DB_HOST=db
      - DB_PORT=3306
      - DB_NAME=db_mbti
      - DB_USERNAME=user_mbti
      - DB_PASSWORD=pw_mbti
    networks:
      - mbti-net                # 네트워크 연결
    ports:
      - "4000:3000"             # 호스트:컨테이너 포트 매핑

  db:
    image: mysql:8.3.0
    container_name: db
    environment:
      - MYSQL_ROOT_PASSWORD=root1234
      - MYSQL_DATABASE=db_mbti
      - MYSQL_USER=user_mbti
      - MYSQL_PASSWORD=pw_mbti
    networks:
      - mbti-net
    volumes:
      - mbti-vol:/var/lib/mysql

# (3) 네트워크 선언
networks:
  mbti-net:
    name: mbti-net           # 실제 네트워크 이름

# (4) 볼륨 선언
volumes:
  mbti-vol:
    name: mbti-vol           # 실제 볼륨 이름
```

02 Docker compose

Docker Compose 기본 구조 (숫자 순서)

```yaml
# ──────────────────────────────────────────────
# (1) Compose 파일 포맷 버전
#     - v2, v3 등 여러 버전이 있으며, 주로 '3.8' 이상을 사용
version: '3.8'

# ──────────────────────────────────────────────
# (2) 환경변수 파일(.env) 지정 (선택)
#     - Compose 실행 시 자동으로 .env 읽어서 ${VAR} 치환
# env_file:
#   - .env
```

- github repo에 자료 첨부 (Week 5)

```yaml
# ──────────────────────────────────────────────
# (3) 서비스 정의
services:
  #
  # │ 서비스 이름 (이름대로 컨테이너가 생성됨)
  app:
    # ├ 이미지 사용
    image: my-org/my-app:latest
    # ├ 없으면 빌드
    build:
      context: .
      dockerfile: Dockerfile
      args:
        # build-arg KEY=VALUE 형식
        - NODE_VERSION=20
    # ├ 컨테이너 이름 고정
    container_name: app
    # ├ 환경변수
    environment:
      - NODE_ENV=production
      - PORT=3000
    # ├ 포트 매핑: "호스트:컨테이너"
    ports:
      - "4000:3000"
    # ├ 볼륨 마운트: "호스트경로:컨테이너경로"
    volumes:
      - ./app-data:/usr/src/app/data
    # ├ 네트워크 연결
    networks:
      - front-net
    # └ (선택) 서비스 시작 순서 보장
    depends_on:
      - db
```

```yaml
  #
  db:
    image: mysql:8.0
    container_name: db
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=mydb
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    volumes:
      - db-data:/var/lib/mysql
    networks:
      - front-net

# ──────────────────────────────────────────────
# (4) 사용자 정의 네트워크
networks:
  front-net:
    driver: bridge

# ──────────────────────────────────────────────
# (5) 볼륨 (영구 저장 영역)
volumes:
  db-data:
    name: myproject-db-data
```

02 Docker compose

Docker Compose

- 실행하면? 아래처럼 나옴

```
▶ docker compose -f ./docker-compose.yaml up -d --build

[+] Building 2.1s (10/10) FINISHED                                docker:desktop-linux
 => [app internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 647B                                              0.0s
 => [app internal] load metadata for docker.io/library/node:20.15.1               1.5s
 => [app auth] library/node:pull token for registry-1.docker.io                   0.0s
 => [app internal] load .dockerignore                                             0.0s
 => => transferring context: 89B                                                  0.0s
 => [app internal] load build context                                             0.6s
 => => transferring context: 82.47MB                                              0.6s
 => [app 1/4] FROM docker.io/library/node:20.15.1@sha256:6326b52a508f0d99ffdbfaa  0.0s
 => CACHED [app 2/4] COPY . /app                                                  0.0s
 => CACHED [app 3/4] WORKDIR /app                                                 0.0s
 => CACHED [app 4/4] RUN npm ci && npm run build                                  0.0s
 => [app] exporting to image                                                      0.0s
 => => exporting layers                                                           0.0s
 => => writing image sha256:f7f6a352d262dc8338cb90c45f95cf5253c7ee2041232aa5663d  0.0s
 => => naming to docker.io/daehyunbigbread/mbti:mysql                             0.0s
[+] Running 2/4
 ⠿ Network mbti-net   Created                                                     0.3s
 ⠿ Volume "mbti-vol"  Created                                                     0.2s
 ✔ Container app      Started                                                     0.2s
 ✔ Container db       Started                                                     0.2s
```

- docker compose exec [서비스 이름]
  - 컨테이너에 접속 가능

```
docker compose exec app sh
# exit

▶ docker compose exec db sh
sh-4.4# exit
exit
```

- docker compose 종료시?
  - docker compose down 으로 진행
  - 실행중인 모든 컨테이너 종료 & 삭제
- 단 volume은 자동으로 삭제 되지 않음.
  - volume도 같이 삭제 하기 위해선 -v 옵션 추가

02 Docker compose

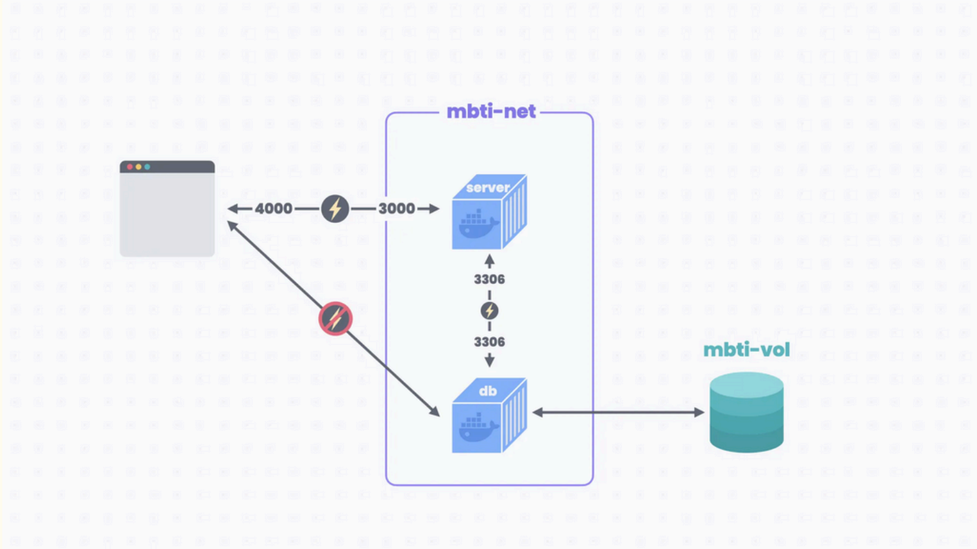
자주쓰는 Compose 명령어

명령어	설명
docker compose up	설정된 서비스(컨테이너)들을 빌드·실행 (foreground)
docker compose up -d --build	백그라운드 실행, 이미지 최신으로 재빌드
docker compose ps	실행 중인 서비스 목록 확인
docker compose logs [서비스명]	특정 서비스의 로그 스트림 표시
docker compose exec <서비스명> sh	해당 컨테이너 쉘 접속
docker compose down	실행 중인 모든 컨테이너·네트워크 중단 및 삭제
docker compose down -v	위 명령에 더해, 볼륨까지 함께 삭제

03 과제

과제. MBTI Service로 컨테이너 구성하기

- **구성해야 하는 컨테이너**는 다음과 같습니다.



- 다음의 절차로 **과제를 진행**해보세요.

1. **mbti 이미지 빌드**
2. **db 컨테이너 실행**
3. **app 컨테이너 실행**
4. **테스트**

Tip: 기존에 했던 과제들을 떠올려 보세요..!!
한것들 다시 한번해서 합치기만 하면 됩니다!

Repo 주소 (사용할 Client - MBTI 테스트 서비스)
: https://github.com/hufs-pnp/25-Cloud-mbti

설명

- db 컨테이너와 app 컨테이너는 같은 네트워크(mbti-net)에 연결되어 있어요.
- app 컨테이너는 외부 네트워크에서 4000번 포트로 접속할 수 있어야 해요. 반면 db 컨테이너는 외부 네트워크에서 접근할 수 없어야 해요.
- db 컨테이너는 볼륨(mbti-vol)을 활용해서 컨테이너가 삭제되어도 데이터를 유지할 수 있어야 해요.

03 과제

과제. MBTI Service로 컨테이너 구성하기 조건

App Container - Dockerfile로 Build

- **베이스 이미지**
  - **레포지토리: node**
  - **태그: 20.15.1 -** 베이스 이미지의 태그는 Dockerfile 작성 시점이 아닌 이미지를 빌드하는 시점에 정합니다.
- **이미지 내 소스 코드 경로: /apps/mbti**
- **환경 변수**
  - **PORT: 3000**
  - **DB_HOST: localhost**
  - **DB_PORT: 3306**
  - **DB_NAME: db_mbti**
  - **DB_USERNAME: user_mbti**
  - **DB_PASSWORD: pw_mbti**
- **node_modules 디렉토리는 복사 대상에서 제외하기**
- **빌드할 이미지 태그: latest, mysql**
- **이미지 레포지토리: 계정명/mbti**

03 과제

과제. MBTI Service로 컨테이너 구성하기 조건

DB Container - MySQL Docker Image를 Docker Hub에서 받아와서 컨테이너 실행

- **컨테이너 이름: db**
- **이미지: mysql:8.3.0**
- **환경 변수**
  - **MYSQL_ROOT_PASSWORD: pw_root**
  - **MYSQL_DATABASE: db_mbti**
  - **MYSQL_USER: user_mbti**
  - **MYSQL_PASSWORD: pw_mbti**
- **컨테이너 종료 시 자동 삭제**
- **백그라운드 실행**

Hint

- db 컨테이너의 데이터가 저장되는 경로는 /var/lib/mysql이예요. v 옵션으로 mbti-vol 볼륨과 연결합니다.
- -network 옵션으로 mbti-net 네트워크에 연결합니다.
- db 컨테이너는 외부에서 접속이 불가능해야 해서 p 옵션을 사용하지 않아요.

03 과제

과제. MBTI Service로 컨테이너 구성하기 조건

App Container - Build 된 Docker Image를 컨테이너 실행

- 컨테이너 이름: app
- 이미지: 계정명/mbti:mysql
- 환경 변수
  - PORT: 3000
  - DB_HOST: db
  - DB_PORT: 3306
  - DB_NAME: db_mbti
  - DB_USERNAME: user_mbti
  - DB_PASSWORD: pw_mbti
- 컨테이너 종료 시 자동 삭제
- 백그라운드 실행

Hint (참고 내용)

- Docker Image Build, Container 실행, Volume, Network

03 과제

제출 방식

- **홈페이지 실행해서 한번 해보고, MBTI Test한 화면 캡처해서 제출**
  - **Page Link 제출 하는곳: Cloud Squad Notion - Follow up System (6주차)**
- **Due Date: 6/4 (수 19:30 까지)**
- **과제 하다가 오류 있으면 바로 DM으로 저에게 말해주세요! (오류가 있을수도 있습니다.... ㅈㅅ....)**

다음 미팅

- **Squad Meeting Week 6**
  - 일시: 06.11 (수) 20:00 ~ 21:00
  - 예정시간: 50~60분
  - 내용: **AWS IAM, VPC**