Docker Compose

Docker Compose

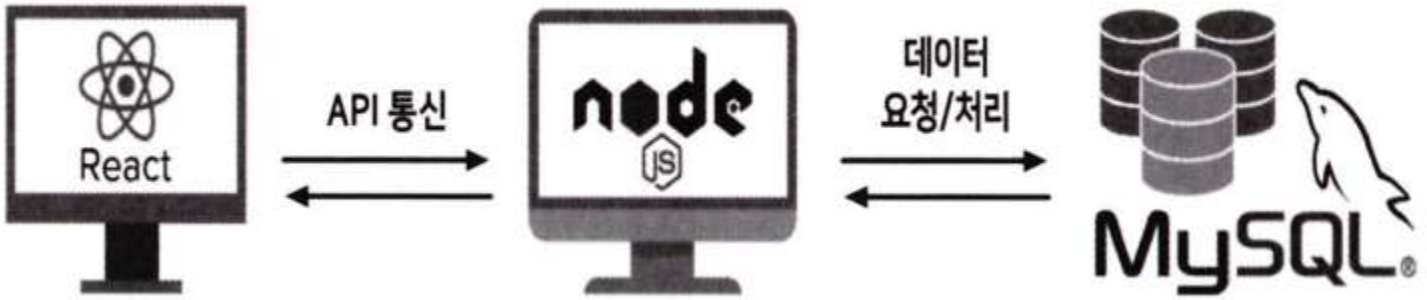
- ❖ docker-compose
  - ✓ 개요
    - 분산 애플리케이션을 기준으로 보면 DockerFile 스크립트는 애플리케이션의 한 부분을 패키징하는 수단
    - Web Frontend, Backend API, Database를 갖춘 애플리케이션을 패키징하려면 각 컴포넌트에 하나씩 세 개의 Dockerfile 스크립트가 필요한데 순서대로 각각의 컨테이너를 docker 명령을 통해 일일이 옵션을 지정해 가며 실행할 수도 있겠지만 이런 수동 프로세스는 온갖 실수와 오류의 원천이 되기 쉬움
    - 직접 실행하는 과정에서 조금만 옵션을 잘못 지정해도 애플리케이션이 정상적으로 동작하지 않거나 컨테이너 간 통신에 문제가 생길 수 있기 때문
    - 공통의 목적을 갖는 애플리케이션 스택을 yaml 코드로 정의해서 한 번에 서비스를 올리고 관리할 수 있는 도구가 docker compose
    - docker compose 파일은 애플리케이션의 원하는 상태 즉 모든 컴포넌트가 실행 중일 때 어떤 상태여야 하는지를 기술하는 파일
    - docker container run 명령으로 컨테이너를 실행할 때 지정하는 모든 옵션을 한데 모아 놓은 형식의 파일
    - docker compose 파일을 작성하고 나면 docker compose 도구를 사용해 애플리케이션을 실행
    - docker compose가 컨테이너, 네트워크, 볼륨 등 필요한 모든 docker 객체를 만들도록 docker API에 명령을 내림

Docker Compose

❖ docker-compose

✓ 개요

- 하나의 웹 애플리케이션을 생성하는 3-Tier 환경을 보면 애플리케이션 데이터를 저장하기 위해 MySQL 데이터베이스를 설정하고 API 애플리테이션 설정을 위해 백엔드 단에 플라스크나 Node를 생성하고 사용자 인터페이스(프론트엔드) 구성을 위해 React, Vue 등의 웹 프레임워크를 선택해서 구성할 수 있음



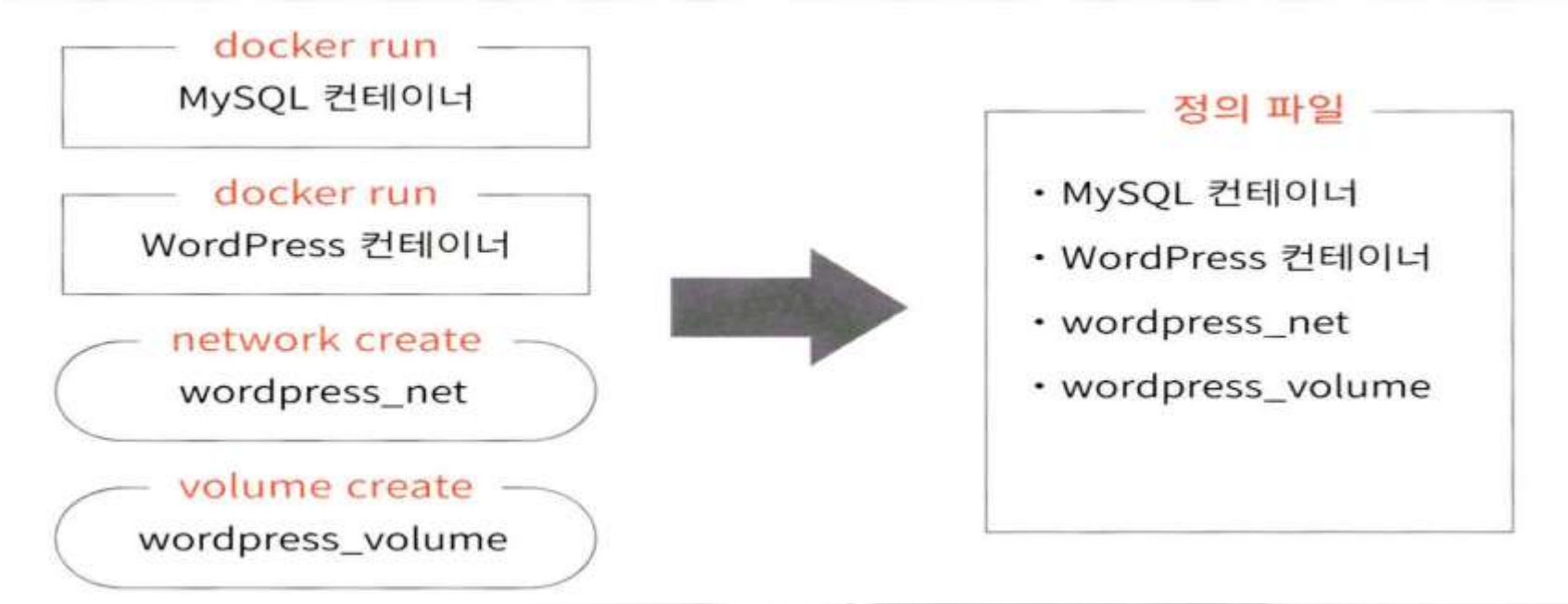
- 이러한 공통의 목적을 갖는 애플리케이션 스택을 도커 컴포즈 야믈 코드로 정의해서 한 번에 서비스를 올리고 관리할 수 있는 도구가 도커 컴포즈

Docker Compose

- docker-compose
  - ✓ 개요
    - 도커 컴포즈로 실행된 컨테이너는 독립된 기능을 가지며 공통 네트워크로 구성되기 때문에 컨테이너 간 통신이 쉬움
    - 도커 컴포즈는 테스트, 개발, 운영의 모든 환경에서 구성이 가능한 오케스트레이션 도구 중 하나이지만 다양한 관리 기능을 갖고 있지 않기 때문에 테스트와 개발환경에 적합
    - 실제 운영 환경은 많은 관리적 요소가 필요하기 때문에 도커 스웜이나 쿠버네티스와 같은 오케스트레이션 도구가 가지고 있는 자동 확장, 모니터링, 복구 등의 운영에 필요한 기능과 함께 사용하는 것을 권장

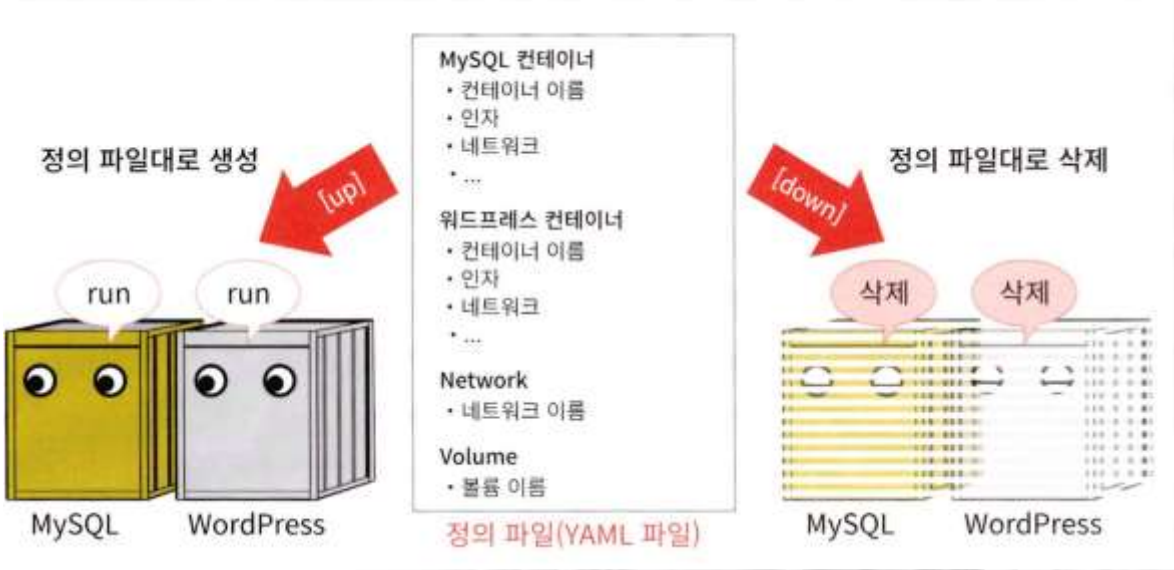
Docker Compose

- ❖ docker-compose
  - ✓ 개요
    - docker compose로 실행된 컨테이너는 독립된 기능을 가지며 공통 네트워크로 구성되기 때문에 컨테이너 간 통신이 쉽고 빠른 런타임 환경을 제공



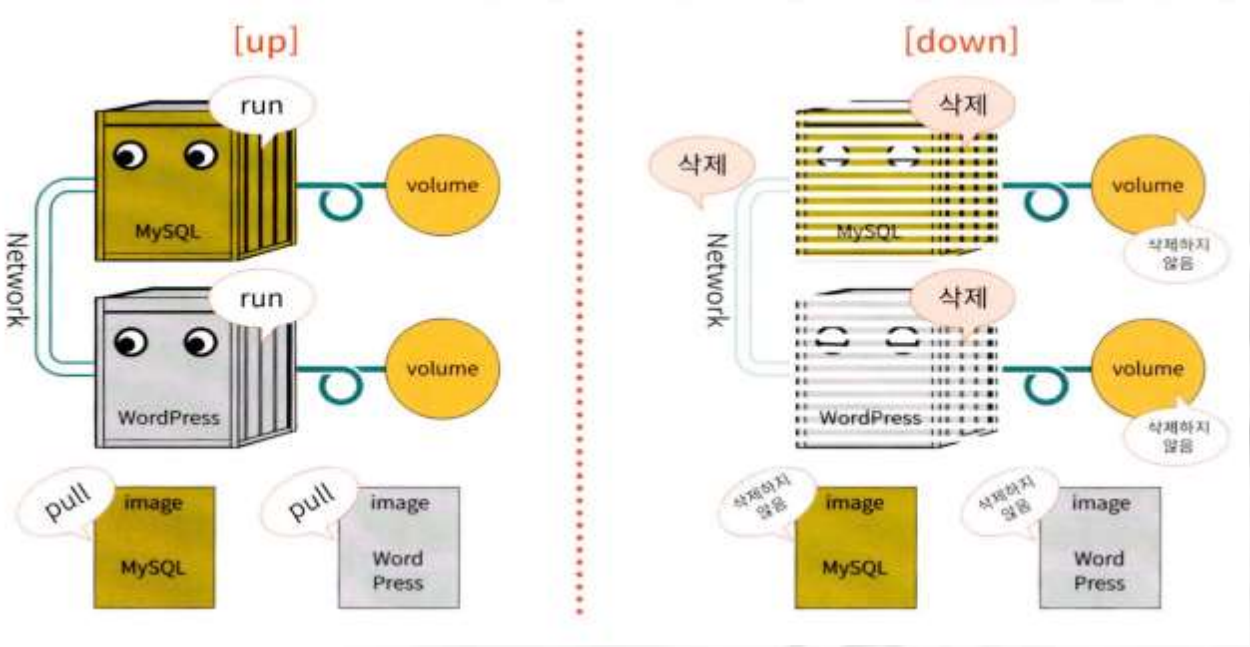
Docker Compose

- ❖ docker-compose
  - ✓ 개요
    - yaml 포맷으로 기술된 설정 파일을 이용해서 여러 Container 간의 실행이나 관계를 설정
    - 시스템을 일괄 실행 또는 일괄 종료 및 삭제 할 수 있는 도구
    - 컨테이너나 볼륨을 어떠한 설정으로 만들지에 대한 항목이 기재돼 있는데 작성 내용은 docker 명령어와 비슷하지만 docker 명령어는 아님



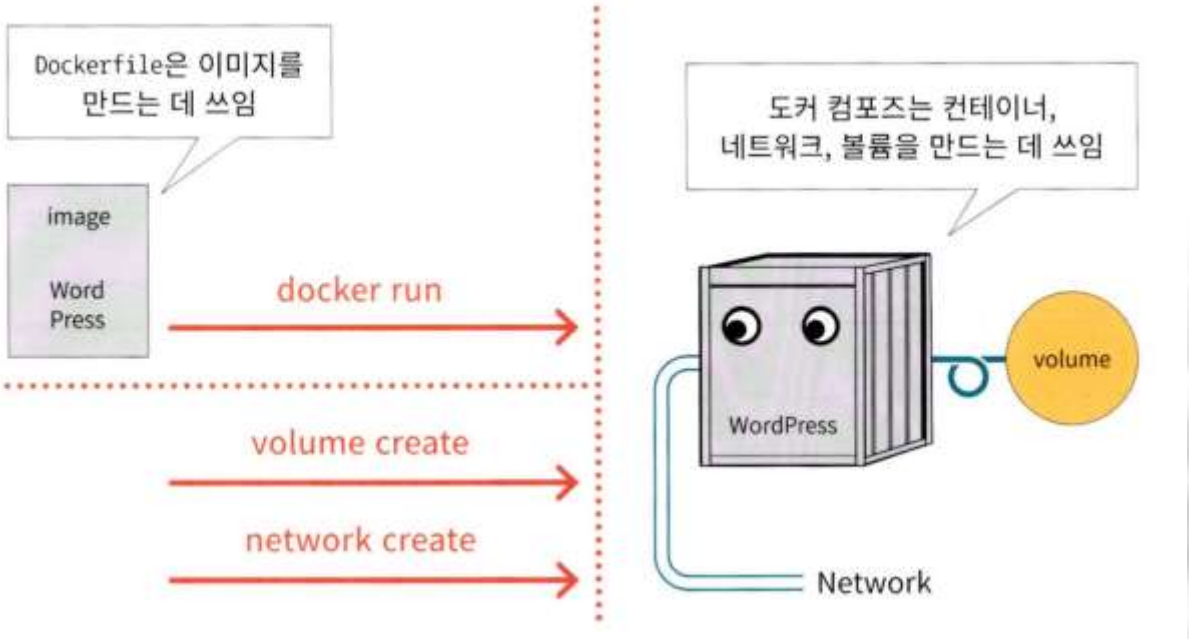
Docker Compose

- ❖ docker-compose
  - ✓ 개요
    - up 커맨드는 docker run 커맨드와 유사해서 정의 파일에 기재된 내용대로 이미지를 내려받고 컨테이너를 생성 및 실행하는데 정의 파일에는 네트워크나 볼륨에 대한 정의도 기재할 수 있어서 주변 환경을 한꺼번에 생성할 수 있음
    - down 커맨드는 컨테이너와 네트워크를 정지 및 삭제하는 것으로 볼륨과 이미지는 삭제하지 않고 컨테이너 와 네트워크 삭제없이 종료 만 하고 싶다면 stop 커맨드를 사용



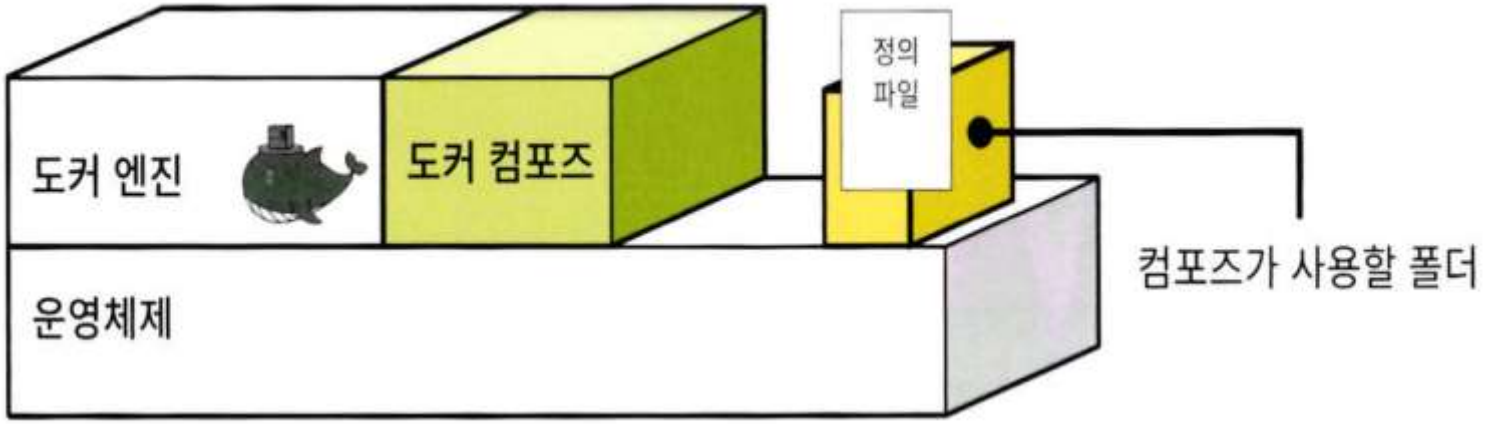
Docker Compose

- ❖ docker-compose
  - ✓ Docker Compose 와 Dockerfile
    - docker compose는 docker run 명령어를 여러 개 모아놓은 것과 같은데 컨테이너와 주변 환경을 생성하며 네트워크와 볼륨까지 함께 만들 수 있음
    - Dockerfile 스크립트는 이미지를 만들기 위한 것으로 네트워크나 볼륨은 만들 수 없음



Docker Compose

- docker-compose



Docker Compose

- 설치
  - ✓ Mac 이나 Windows에서는 Docker Desktop을 설치한 경우 별도의 설치 과정 필요 없음
  - ✓ 리눅스
    - 깃허브에서 다운로드: sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
    - 실행을 위한 권한 변경: sudo chmod +x /usr/local/bin/docker-compose
    - sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
    - sudo chown 계정 /usr/local/bin/docker-compose
    - docker-compose --version

Docker Compose

- ❖ docker compose 작성 및 실행
  - ✓ 컨테이너 생성 명령
    - docker run --name apa000ex1 -d -p 8080:80 httpd
  - ✓ Docker-Compose 활용
    - docker-compose.yaml 파일 작성

```
version: "3"
services:
    apa000ex2:
        image: httpd
        ports:
            - 8081:80
        restart: always
```

    - docker-compose 명령 수행

```
docker-compose up -d
```

    - 컨테이너 확인

```
docker ps
```

Docker Compose

- Maria DB 테스트
  - ✓ 일반적인 컨테이너 생성 명령

```
docker run -d ₩
  --name mariadb-server ₩
  -p 3306:3306 ₩
  -e MYSQL_ROOT_PASSWORD=your_password ₩
  -v ~/mariadb/data:/var/lib/mysql ₩
  --restart always ₩
  mariadb:10.4.6
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - docker-compose.yaml 파일 작성

```
version: '3.8'
services:
  mariadb:
    image: mariadb:10.4.6
    container_name: mariadb-server
    restart: always
    environment:
      # 루트 비밀번호 (반드시 수정해서 사용하세요)
      MYSQL_ROOT_PASSWORD: your_root_password
      # 초기 생성할 데이터베이스 이름 (선택 사항)
      MYSQL_DATABASE: my_database
      # 일반 사용자 계정 (선택 사항)
      MYSQL_USER: my_user
      MYSQL_PASSWORD: my_user_password
    ports:
      - "3306:3306"
    volumes:
      # 호스트의 ./data 폴더와 컨테이너 내부 데이터 폴더를 동기화
      - ./data:/var/lib/mysql
      # 초기 설정 파일이 있다면 ./init 폴더에 넣어두세요
      - ./init:/docker-entrypoint-initdb.d
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● docker-compose 명령어로 컨테이너 생성

      docker-compose up -d

      [+] up 2/2

      ✔ Network docker_default Created 0.0s

      ✔ Container docker-mariadb-1 Created

      ◆ 하나의 네트워크를 자동 생성

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 컨테이너 정보 확인

      docker-compose ps

      NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS

      docker-mariadb-1 mariadb "docker-entrypoint.s…" mariadb About a minute ago Up About a minute 0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp

      docker ps

      CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

      ca8f6ecb80e2 mariadb:10.4.6 "docker-entrypoint.s…" 44 seconds ago Restarting (1) 11 seconds ago docker-mariadb-1

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

      $ docker exec -it mariadb-server bash

      /# mysql -uroot –p

      MariaDB [(none)]> show databases;

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| my_database        |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.001 sec)
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

```
MariaDB [(none)]> use my_database;
Database changed

MariaDB [my_database]> create table item(item_id int, item_name varchar(10));
Query OK, 0 rows affected (0.007 sec)

MariaDB [my_database]> insert into item values(20, 'docker-ce');
Query OK, 1 row affected (0.002 sec)

MariaDB [my_database]> select * from item;
+---------+-----------+
| item_id | item_name |
+---------+-----------+
|      20 | docker-ce |
+---------+-----------+
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

```
MariaDB [(none)]> exit
Bye

root@54dda7eccaca:/# ls -l /var/lib/mysql/my_database/
total 104
-rw-rw---- 1 mysql mysql    65 Feb  5 00:53 db.opt
-rw-rw---- 1 mysql mysql   476 Feb  5 01:00 item.frm
-rw-rw---- 1 mysql mysql 98304 Feb  5 02:14 item.ibd

adam@original:~/docker$ sudo ls -l ./data/my_database
[sudo] password for adam:
total 104
-rw-rw---- 1 999 systemd-journal    65 Feb  5 00:53 db.opt
-rw-rw---- 1 999 systemd-journal   476 Feb  5 01:00 item.frm
-rw-rw---- 1 999 systemd-journal 98304 Feb  5 02:14 item.ibd
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

      $ docker-compose down

      WARN[0000] /home/adam/docker/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] down 2/2

      ✔ Container mariadb-server Removed
      1.8s

      ✔ Network docker_default Removed

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

      $ docker-compose up -d

      WARN[0000] /home/adam/docker/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] up 2/2

      ✔ Network docker_default Created 0.0s

      ✔ Container mariadb-server Created

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 영속성 확인 - docker ps 명령으로 컨테이너 이름 확인

```
$ docker exec -it mariadb-server bash
root@871a6fc74153:/# mysql -uroot -p

MariaDB [(none)]> use my_database
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed

MariaDB [my_database]> select * from item;
+---------+-----------+
| item_id | item_name |
+---------+-----------+
|      20 | docker-ce |
+---------+-----------+
1 row in set (0.003 sec)
```

Docker Compose

- ❖ Maria DB 테스트
  - ✓ Docker-Compose
    - ● 네트워크
      - ◆ docker-compose up 명령을 수행하면 자체 기본 네트워크가 생성되는데 up과 동시에 가장 먼저 기본 네트워크를 디렉터리명_default 이름으로 생성
      - ◆ 확인: docker network ls

NETWORK ID NAME DRIVER SCOPE

2ecc16a0a53b bridge bridge local

7447c531ba23 docker_default bridge local

c88439e4b52a host host local

b34edce41fb1 none null local

      - ◆ 이후 도커 컴포즈에 포함된 다중 컨테이너를 구동하고 이 네트워크에 컨테이너 서비스를 연결하게 되며 이 기본 네트워크를 통해 IP가 아닌 서비스명(컨테이너명)으로 서비스 간 통신을 할 수 있음

.

Docker Compose

- 작성 방법
  - ✓ 파일 이름: docker-compose.yaml, docker-compose.yml, compose.yaml, compose.yml 등
  - ✓ 주 항목
    - version
    - services
    - networks
    - volumes
  - ✓ 정의 내용
    - image
    - networks: --net
    - volumes: -v
    - ports: -p
    - environment: -e
    - depends_on: 다른 서비스에 대한 의존 관계
    - restart: 컨테이너 종료 시 재시작 여부(no, always, on-failure, unless-stopped)

Docker Compose

❖ 작성 방법

✓ 버전 정의

- yaml 코드 첫 줄은 버전을 명시

  version: '3.8'

- docker compose 및 docker 엔진 호환

컴포즈 버전	도커 엔진 릴리스	컴포즈 버전	도커 엔진 릴리스
3.8	19.03.0 ~	3.1	1.13.1 ~
3.7	19.03.0 ~	3.0	1.13.0 ~
3.6	18.02.0 ~	2.4	17.12.0 ~
3.5	17.12.0 ~	2.3	17.06.0 ~
3.4	17.09.0 ~	2.2	1.13.0 ~
3.3	17.06.0 ~	2.1	1.12.0 ~
3.2	17.04.0 ~	2.0	1.10.0 ~

Docker Compose

- ❖ 작성 방법
  - ✓ 버전 정의
    - 버전 선택은 docker 엔진 릴리스에 적합한 버전을 선택하는데 만일 버전이 맞지 않으면 에러 메시지를 출력
    - docker 버전 및 docker compose 버전 업데이트 속도가 잦은 편이니 종종 docker 도큐먼트를 참고하여 버전 변화를 참고

ERROR: Version in ''./docker-compose.yml" is unsupported. You might be seeing this error beca니se

you're using the wrong Compose file version. Either specify a supported version (e.g "2.2" or

"3.3") and place your service definitions under the 'services' key, or omit the 'version' key

and place your service definitions at the root of the file to use v은rsion 1.

For more on the Compose file format versions, see https://docs.docker.com/compose/compose-file/

Docker Compose

- ❖ 작성 방법
  - ✓ 버전 정의
    - ● 에러 발생 원인
      - ◆ 작성한 버전과 현재 docker compose 또는 docker 엔진 릴리스가 적합하지 않은 경우
      - ◆ docker compose 도구가 오래된 경우 - 새로운 버전으로 업데이트 필요
      - ◆ 버전 문제가 아닌 들여쓰기의 공백 수가 하위 레벨과 맞지 않은 경우

Docker Compose

- 작성 방법
  - ✓ 서비스 정의
    - docker compose를 통해 실행할 서비스를 정의
    - docker compose는 컨테이너 대신 서비스 개념을 사용
    - 상위의 version 명령과 동일 레벨로 작성되며 다중 컨테이너 서비스 실행을 목적으로 하기 때문에 복수형으로 작성되는 것에 유의
    - docker compose로 함께 실행된 모든 서비스는 docker compose 명령을 통해 통합 관리
    - services 하위에는 실행될 컨테이너 서비스를 작성하고 하위 레벨에 docker 명령 실행과 유사하게 컨테이너 실행에 필요한 옵션을 작성

```
version: '3.8'
services:
        myweb:
                image: nginx:latest
        mydb:
                image: mariadb:10.4.6
```

Docker Compose

- 작성 방법
  - ✓ 서비스 정의
    - 프로젝트에서 별도 이미지 개발없이 docker 허브에서 제공하는 오피셜 이미지를 사용하는 경우에는 image:nginx:latest와 같이 작성
    - 프로젝트에 필요한 애플리케이션 개발을 위해 Dockerfile을 작성하여 컨테이너를 실행하는 경우에는 미리 빌드해서 이미지명 태그를 명시해도 되지만 build 옵션을 사용하면 docker compose 실행과 함께 이미지를 빌드
    - build 옵션은 이미지 빌드에 필요한 Dockerfile의 경로를 지정하고 docker-compose.yaml 파일과 동일 경로에 위치한 경우에는 .을 이용해 Dockerffle이 같은 경로에 있음을 명시

```
version: '3.8'
services:
        web:
                build: .
```

Docker Compose

- 작성 방법
  - ✓ 서비스 정의
    - Dockerfile의 위치와 파일명이 다른 경우 context 와 dockerfile 옵션을 이용해 세부 내용을 정의

```
version: '3.8'
services:
    web:
        build:
            context: .
            dockerfile: ./compose/pyfla/Dockerfile-py
```

Docker Compose

- ❖ 작성 방법
  - ✓ 서비스 정의
    - ● 하위 옵션
      - ◆ container_name: docker run의 --name 옵션과 동일하며 생략 시 자동으로 부여되는데 디렉토리명_서비스명_n
      - ◆ ports: docker run의 -p 옵션과 동일하며 서비스 내부 포트와 외부 호스트 포트를 지정하여 바인드
      - ◆ expose: 호스트 운영체제와 직접 연결하는 포트를 구성하지 않고 포트를 노출하는데 필요 시 링크로 연결된 서비스와 서비스 간의 통신만 사용
      - ◆ networks: docker run의 --net(--network) 옵션과 동일하며 최상위 레벨의 networks에 정의된 네트워크 이름을 작성
      - ◆ volumes: docker run의 -v(--volume) 옵션과 동일한데 서비스 내부 디렉토리와 호스트 디렉토리를 연결하여 데이터 지속성 설정
      - ◆ environment: docker run의 -e 옵션과 동일한데 서비스 내부 환경 변수 설정에 이용하며 환경 변수가 많은 경우에는 파일(*.env)로 만들어 env_file 옵션에 파일명을 지정(envfile: ./envfile.env)
      - ◆ command: docker run의 마지막에 작성되는 명령어로 서비스가 구동 이후 실행할 명령어 작성

Docker Compose

- 작성 방법
  - ✓ 서비스 정의
    - 하위 옵션
      - restart: . docker run의 --restart 옵션과 동일하며 서비스 재시작 옵션 지정(no: 수동 재시작, always: 컨테이너 수동 제어를 제외하고 항상 재시작, on-failure: 오류가 있을 시 재시작)
      - depends_on: 서비스 간의 종속성을 의미하며 먼저 실행해야 하는 서비스를 지정하여 순서를 정한느데 이 옵션에 지정된 서비스가 먼저 시작됨

Docker Compose

- 작성 방법
  - ✓ 네트워크 정의
    - 다중 컨테이너들이 사용할 최상위 네트워크 키를 정의하고 이하 하위 서비스 단위로 이 네트워크를 선택할 수 있음
    - networks 옵션을 지정하지 않으면 자체 기본 네트워크를 자동으로 생성
    - 최상위 레벨에 networks 지정 시 해당 이름의 네트워크가 생성되고 대역은 172.x.x.x로 자동으로 할당되며 기본 드라이버는 브리지로 지정됨

```
(base) adam@ADAMui-MacBookPro ~ % docker network ls
NETWORK ID     NAME                 DRIVER    SCOPE
6d837d63ee70   bridge               bridge    local
6ec9a4b52a52   host                 host      local
523d4e3822d4   mariadb_default      bridge    local
25b3310bb09f   none                 null      local
65a8dbb93a2d   redmine000net2       bridge    local
44b7dfd88614   wordpress000net4     bridge    local
```

Docker Compose

- ❖ 작성 방법
  - ✓ 네트워크 정의
    - docker에서 생성한 기존 네트워크를 지정하는 경우에는 externel 옵션에 네트워크 이름을 작성

```
version: "3.9"
services:
  # ...
networks:
  default:
      external:
          name: vswitch-ap
```

Docker Compose

- 작성 방법
  - ✓ 볼륨 정의
    - 데이터의 지속성을 유지하기 위해 최상위 레벨에 볼륨을 정의하고 서비스 레벨에서 볼륨명과 서비스 내부의 디렉토리를 바인드
    - docker volume create와 동일하게 docker가 관리하는 가상 영역 (/var/lib/docker/volume)에 자동 배치
    - docker volume ls 명령을 통해서 확인 가능하고 docker volume inspect 볼륨명으로 세부 디렉토리 경로까지 알 수 있음

Docker Compose

- 작성 방법
  - ✓ 볼륨 정의

```
version: "3.9"
services:
  mydb:
    image: mysql:5.7
    container_name: mysql_app
    volumes:
      - db.data:/var/lib/mysql
...
  myweb:
    depends_on:
      - mydb
    image: wordpress:latest
    container_name: wordpress_app
    volumes:
      - web.data:/var/www/html
...
volumes:
  db.data: {}
  web_data: {} # {} 생략 가능 docker가 관리하는 볼륨 생성을 의미
.
```

Docker Compose

- ❖ 작성 방법
  - ✓ 볼륨 정의
    - 최상위 레벨에 volumes를 정의하지 않고 서비스 하위 레벨에 호스트의 절대 경로와 컨테이너의 디렉토리 경로를 직접 사용하는 바인드 마운트 방식도 가능

```
version: "3.9"
services:
  myweb:
    depends_on:
      - mydb
    image: wordpress:latest
    container_name: wordpress_app
    volumes:
      - web_data:/var/www/html
      - /home/adam/my_wp/myweb-log:/var/log
```

Docker Compose

❖ docker 명령어 와 docker compose yaml 코드 비교

요구사항	도커 명령	도커 컴포즈 옵션
컨테이너(서비스)명	--name	container_name:
포트 연결	-p	ports:
네트워크 구성	--net	networks:
재시작	--restart	restart:
볼륨	-v (--volume)	volumes:
환경 변수	-e	environment:
컨테이너 간 연결	--link 컨테이너명:서비스명	depends_on:
이미지	이미지명	image:

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 볼륨 생성 및 조회
        - ▪ docker volume create mydb_data

        mydb_data

        - ▪ docker volume create myweb_data

        myweb_data

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 볼륨 생성 및 조회
        - ▪ docker volume ls

```
DRIVER    VOLUME NAME
local
    04eb82e17284f91ac28bac45408cfae69b14873bfee1689f487eb0382894d037
local
    56dbed4ffc515d090fbaa82d046eef5e0f7eb8af7fa8de90044afee06650b7ff
local
    e6f07c7f86a1d6c7d2f4ad996e85d79e06bd6cc876deb373f790402bce565009
local     mydb_data
local     mysql-data-vol
local     myweb_data
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 볼륨 생성 및 조회
        - ▪ docker inspect --type volume mydb_data

```
[
    {
        "CreatedAt": "2023-10-09T01:49:25Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mydb_data/_data",
        "Name": "mydb_data",
        "Options": null,
        "Scope": "local"
    }
]
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 네트워크 생성 및 조회
        - docker network create myapp-net

5e2b89ee3bac80528d892dc4c87e0624bb622830bcd9b4e0ab187752700d85de

        - docker network ls

```
NETWORK ID     NAME                DRIVER    SCOPE
6d837d63ee70   bridge              bridge    local
6ec9a4b52a52   host                host      local
523d4e3822d4   mariadb_default     bridge    local
5e2b89ee3bac   myapp-net           bridge    local
25b3310bb09f   none                null      local
65a8dbb93a2d   redmine000net2      bridge    local
44b7dfd88614   wordpress000net4    bridge    local
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 네트워크 생성 및 조회
        - ▪ docker network inspect myapp-net


```
[
    {
        "Name": "myapp-net",
        "Id":
    "5e2b89ee3bac80528d892dc4c87e0624bb622830bcd9b4e0ab187752700
    d85de",
        "Created": "2023-10-09T01:51:23.148789502Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - docker 명령어
      - ◆ 네트워크 생성 및 조회


```
            "Config": [
                {
                    "Subnet": "172.22.0.0/16",
                    "Gateway": "172.22.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 컨테이너 생성

```
docker run -dit --name=mysql_app -v mydb_data:/var/lib/mysql --
    restart=always -p 3306:3306 --net=myapp-net -e
    MYSQL_ROOT_PASSWORD=wnddkd -e MYSQL_DATABASE=adam -e
    MYSQL_USER=adam -e MYSQL_PASSWORD=wnddkd mysql:8.0

docker run -dit --name=wordpress_app -v  myweb_data:/var/www/html -v
    ${PWD}/myweb-log:/var/log --restart=always -p 8888:80 --net=myapp-
    net -e W0RDPRESS_DB_HOST=mysql_app:3306 -e
    WORDPRESS_DB_NAME=adam -e WORDPRESS_DB_USER=adam -e
    WORDPRESS_DB_PASSWORD=wnddkd --link mysql_app:mysql
    wordpress:5.7
```

Docker Compose

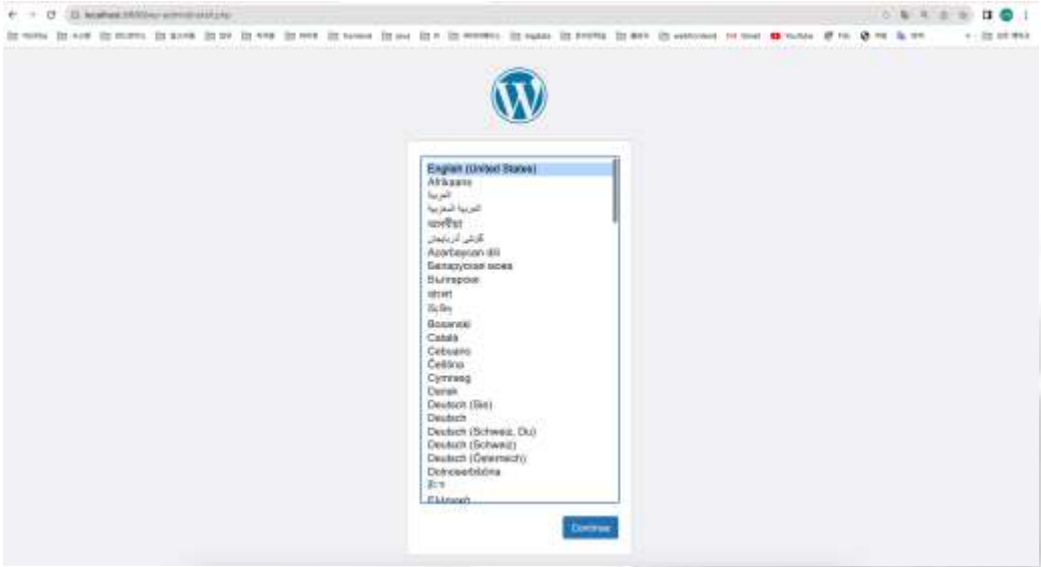
- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 확인

        docker ps

```
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS          PORTS                               NAMES

c4b996e40f4f   wordpress:5.7   "docker-entrypoint.s..."   27 seconds ago   Up
26 seconds   0.0.0.0:8888->80/tcp                wordpress_app

b145b55438ed   mysql:8.0       "docker-entrypoint.s..."   4 minutes ago    Up
4 minutes    0.0.0.0:3306->3306/tcp, 33060/tcp   mysql_app
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker 명령어
      - ◆ 확인
        - ▪ 웹 브라우저 확인: localhost:8888



Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ docker-compose.yaml 작성

```
version: "3.9"
services:
    mydb:
        #docker 허브에서 제공하는 mysql:8.0 이미지 선택
        image: mysql:8.0
        #서비스 컨테이너 이름 지정
        container_name: mysql_app
        #최상위 레벨에 정의 mydb_data 볼륨 지정
        volumes:
            - mydb_data:/var/lib/mysql
        #수동 제어를 제외한 컨테이너 종료 시 자동 재시작
        restart: always
        #호스트 운영체제와 컨테이너의 3306 포트를 바인드
        ports:
            - "3306:3306"
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ docker-compose.yaml 작성

```
#최상위 레벨에 정의한 backend-net을 기본 네트워크로 지정
networks:
    - backend-net
#서비스가 사용할 환경 변수를 지정한다
environment:
    MYSQL_ROOT_PASSWORD: wnddkd
    MYSQL_DATABASE: adam
    MYSQL_USER: adam
    MYSQL_PASSWORD: wnddkd
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ docker-compose.yaml 작성

```
# 두 번째 서비스 정의
myweb:
    # myweb 서비스가 실행되기 전에 mydb 서비스를 먼저 실행하는 의존성을 설정한다
    depends_on:
        - mydb
    # wordpress:5.7 이미지를 지정
    image: wordpress:5.7
    # 서비스 컨테이너 이름을 지정
    container_name: wordpress_app
    # 호스트 운영체제의 8888 포트와 컨테이너의 80 포트를 바인드
    ports:
        - "8888:80"
    # backend-net으로 mydb 서비스와 동일 네트워크로 지정하고
    # 외부 연결을 위한 네트워크를 위해 fronetend-net 지정을 가정
    # docker network connect - 명령으로 두 번째 네트워크를 추가하는 것과 같음
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ docker-compose.yaml 작성

```
networks:
    - backend-net
    - frontend-net
# 컨테이너 데이터 지속성을 위해 docker 볼륨 기법과 바인드 마운트 기법을 사용
volumes:
    - myweb_data:/var/www/html
    - ${PWD}/myweb-log:/var/log
# 수동 제어를 제외한 컨테이너 종료 시 자동 재시작
restart: always
#서비스가 사용할 환경 변수를 지정
environment:
    WORDPRESS_DB_HOST: mydb:3306
    WORDPRESS_DB_USER: adam
    WORDPRESS_DB_PASSWORD: wnddkd
    WORDPRESS_DB_NAME: adam
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ docker-compose.yaml 작성

```
# docker compose 애플리케이션이 사용할 네트워크 생성 - docker network create와 동일
networks:
    frontend-net: {}
    backend-net: {}

# docker compose 애플리케이션이 사용할 볼륨 생성 - docker volume create 와 동일하다
volumes:
    mydb_data: {}
    myweb_data: {}
```

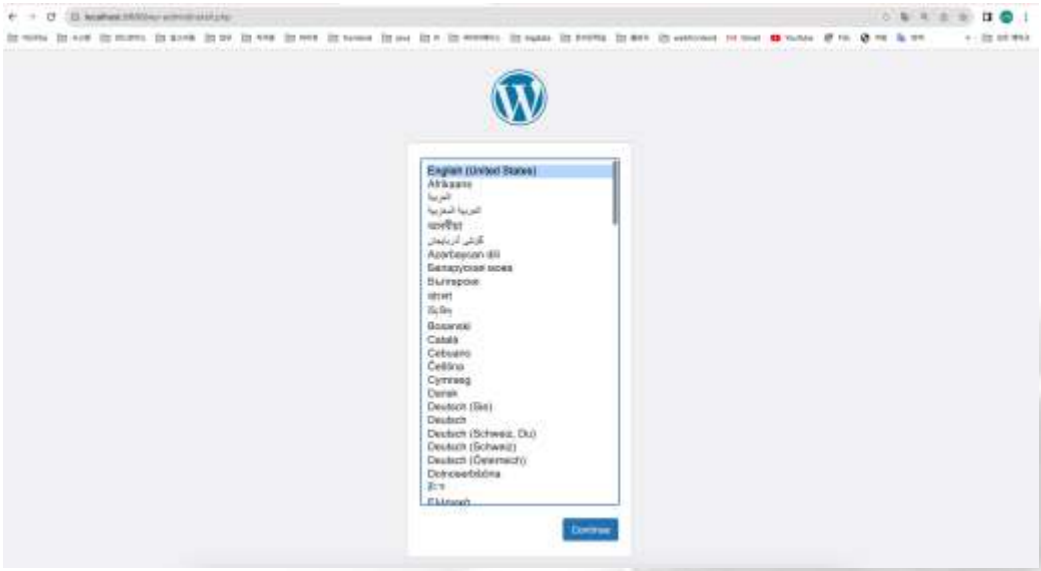
Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ 실행
        - ▪ docker-compose up -d
      - ◆ 확인
        - ▪ docker-compose ps

```
NAME                IMAGE               COMMAND                  SERVICE
CREATED             STATUS              PORTS

mysql_app           mysql:8.0           "docker-entrypoint.s..."   mydb
3 minutes ago       Up 3 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp

wordpress_app       wordpress:5.7       "docker-entrypoint.s..."   myweb
3 minutes ago       Up 3 minutes        0.0.0.0:8888->80/tcp
```

Docker Compose

- ❖ docker 명령어 와 docker compose yaml 코드 비교
  - ✓ 데이터베이스 단은 mysql:8.0을 이용하고 웹 단은 wordpress:5.7 이미지를 이용한 웹 애플리케이션 서비스 구성
    - ● docker compose 활용
      - ◆ 확인
        - ▪ 웹 브라우저 확인



Docker Compose

- 컨테이너 간의 통신
  - ✓ 외부에서 도커 내부의 컨테이너에 접근하고자 하는 경우에는 도커 컨테이너를 생성할 때 Port Forwarding을 수행해서 접근
  - ✓ 하나의 도커 엔진에 만들어진 컨터이너 간에는 도커 엔진에서 부여한 IP를 이용해서 접근 가능 - docker inspect network bridge 명령

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ 레디스 컨테이너 실행
    - docker run --name myredis -d -p 6379:6379 redis
  - ✓ 가상 환경 생성
    - python3 -m venv ./{your venv name}
    - 가상 환경 활성화
      - Linux, Mac: source {your venv name}/bin/activate
      - Windows: {your venv name}/Scripts/activate
  - ✓ 패키지 설치 및 실행
    - (venv)$ pip install flask
    - (venv)$ pip install redis
  - ✓ 라이브러리의 의존성을 파일에 내보내기
    - pip freeze > requirements.txt

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ app.py 파일을 생성하고 작성: host의 IP는 현재 컴퓨터의 IP 나 localhost

```
import time
import redis
from flask import Flask

app = Flask(__name__)

def web_hit_cnt():
    with redis.StrictRedis(host='localhost', port=6379) as conn:
        return conn.incr('hits')
@app.route('/')
def hello():
   cnt = web_hit_cnt()
   print(cnt)
   return '''<h1 style="text-align:center; color:black;">docker-compose application:
Flask & Redis</h1>
<p style="text-align:center; color:deepskyblue;">container service.</p>
<p style="text-align:center; color:deepskyblue;">Web access count : {}
times</p>'''.format(cnt)
```

Docker Compose

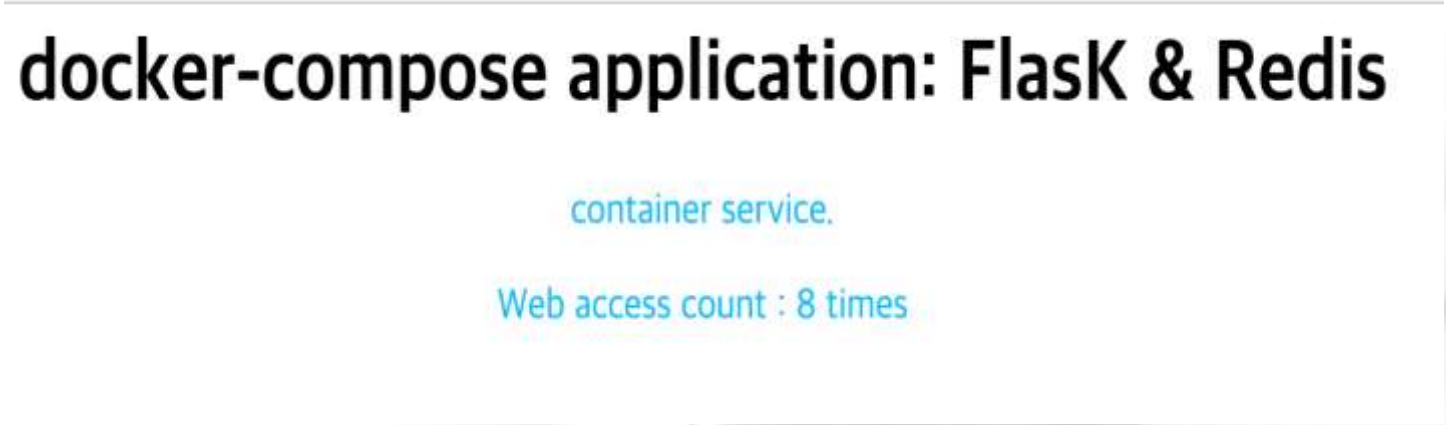
- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ app.py 파일을 생성하고 작성

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000, debug=True)  #app.run(debug=True) 개발 모드
```

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ 방화벽 재실행
    - sudo ufw allow 9000
    - sudo systemctl restart ufw
  - ✓ 실행

    python app.py
  - ✓ 확인: localhost:9000



Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ Dockerfile을 이용한 이미지 생성 및 실행
    - Dockerfile 작성

```
FROM python:3.10-slim
ENV LIBRARY_PATH=/lib:/usr/lib
ENV FLASK_APP=py_app
ENV FLASK_ENV=development
EXPOSE 9000
WORKDIR /py_app
COPY . .
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ Dockerfile을 이용한 이미지 생성
    - app.py 파일의 redis 접속 IP를 수정

      docker inspect network bridge 명령으로 확인해서 수정
    - 이미지 생성

      docker build -t flaskapp .
    - 컨테이너 실행

      docker run -dit -p 9000:9000 --name=flaskapp flaskapp

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ Docker Compose를 이용한 실행
    - 기존 컨테이너 와 이미지 삭제

      docker stop flaskapp

      docker rm flaskapp

      docker rmi flaskapp

      docker stop myredis

      docker rm myredis

      docker rmi redis

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ Docker Compose를 이용한 실행
    - docker-compose.yaml 작성

```
version: '3'
services:
  redis:
    image: redis:latest
    ports:
      - 6379:6379
    restart: always
  flask:
    build: .
    ports:
      - 9000:9000
    depends_on:
      - redis
    restart: always
```

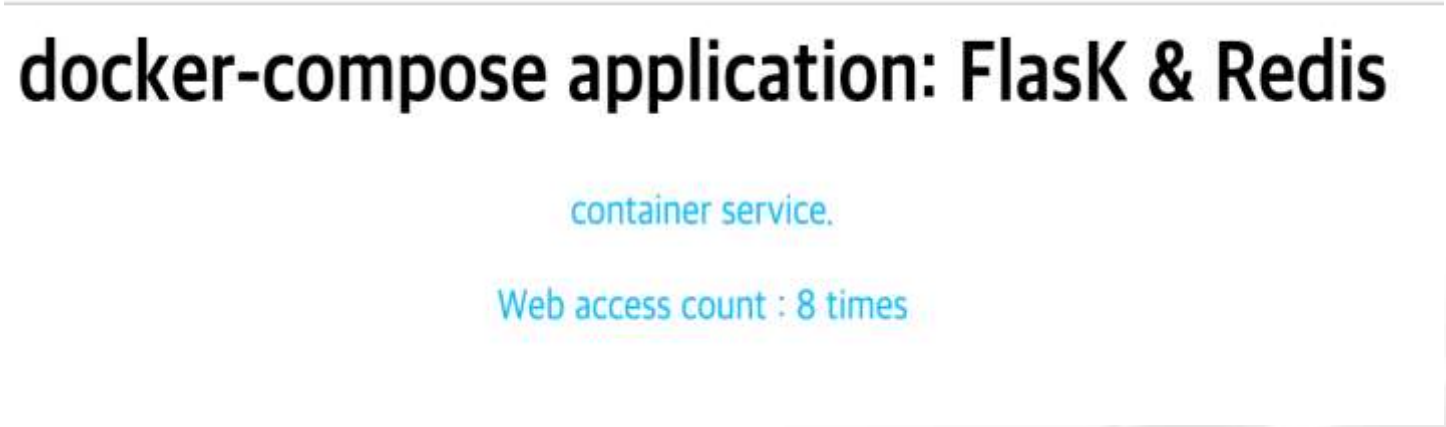
Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ Docker Compose를 이용한 실행
    - app.py 파일의 데이터베이스 접속 위치를 서비스 이름으로 수정

```
import time
import redis
from flask import Flask

app = Flask(__name__)

def web_hit_cnt():
    with redis.StrictRedis(host='redis', port=6379) as conn:
        return conn.incr('hits')
@app.route('/')
def hello():
   cnt = web_hit_cnt()
   print(cnt)
   return '''<h1 style="text-align:center; color:black;">docker-compose applica>
Flask & Redis</h1>
<p style="text-align:center; color:deepskyblue;">container service.</p>
<p style="text-align:center; color:deepskyblue;">Web access count : {} times</p>

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=9000, debug=True)  #app.run(debug=True) 개발자
```

Docker Compose

- 플라스크의 웹 프레임워크로 웹 페이지를 만들고 redis 인메모리 데이터베이스에서 웹 페이지 액세스 카운트를 캐시하여 페이지에 출력
  - ✓ 컨테이너 생성: docker-compose up -d
  - ✓ 확인



Docker Compose

- ❖ docker compose 명령
  - ✓ docker-compose 명령어
    - build: Dockerfile을 이용한 빌드 또는 재빌드
    - config: docker compose 구성 파일의 내용 확인
    - create: 서비스 생성
    - down: docker compose 자원을 일괄 정지 후 삭제
    - events: 컨테이너에서 실시간으로 이벤트 정보를 수신
    - exec: 실행 중인 컨테이너에 명령 실행
    - help: 도움말
    - images: 사용된 이미지 정보
    - kill: docker 킬 명령 과 동일, 컨테이너 강제 정지
    - logs: 컨테이너의 실행 로그 정보 출력
    - pause: 컨테이너 서비스 일시 정지
    - port: 포트 바인딩된 외부로 연결된 포트 출력
    - ps: 실행 중인 컨테이너 서비스 출력
    - pull: 서비스 이미지 가져오기
    - push: 서비스 이미지 올리기
    - restart: 컨테이너 서비스 재시작

Docker Compose

- ❖ docker compose 명령
  - ✓ docker-compose 명령어
    - rm: 정지된 서비스 제거
    - run: 실행 중인 컨테이너에 일회성 명령어 실행
    - scale: 컨테이너 서비스에 대한 컨테이너 수 설정
    - start: 컨테이너 서비스 시작
    - stop: 컨테이너 서비스 중지
    - top: 실행 중인 프로세스 출력
    - unpause: 컨테이너 서비스 일시 정지 해제
    - up: 컨테이서 서비스 생성 과 시작
    - version: 버전 정보 표시 및 종료

Docker Compose

- ❖ docker compose 명령
  - ✓ 컨테이너 와 주변 환경 생성 명령어
    - docker-compose –f 파일경로 up 옵션
    - 옵션
      - -d, --detach: 백그라운드로 실행
      - --build: 컨테이너를 실행하기 전에 이미지를 빌드
      - --force-recreate: 설정 또는 이미지가 변경되지 않더라도 컨테이너를 재생성
      - -t, -timeout: 컨테이너를 종료할 때의 타임아웃 설정으로 기본은 10초
      - --scale SERVICE=NUM: 컨테이너의 수를 변경
      - --no-color: 화면 출력 내용을 흑백으로함
      - --no-deps: 링크된 서비스를 실행하지 않음
      - --no-create: 컨테이너가 이미 존재할 경우 다시 생성하지 않음
      - --no-build: 이미지가 없어도 이미지를 빌드하지 않음
      - --abort-on-container-exit: 컨테이너가 하나라도 종료되면 모든 컨테이너를 종료
      - --remove-orphans: 컴포즈 파일에 정의되지 않은 서비스의 컨테이너를 삭제

Docker Compose

- ❖ docker compose 명령
  - ✓ 이전에 수행한 docker-compose 정지 후 삭제
    - docker-compose down

      WARN[0000] /home/adam/docker/flaskapp/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] Running 3/3

      ✔ Container flaskapp-flask-1 Removed 0.1s

      ✔ Container flaskapp-redis-1 Removed 0.1s

      ✔ Network flaskapp_default Removed 0.0s

Docker Compose

- ❖ docker compose 명령
  - ✓ 컨테이너 생성
    - ● docker-compose.yaml 파일 작성

```
version: '3.8'
services:
  server_web:
    image: httpd:2
  server_db:
    image: redis:latest
```

Docker Compose

- ❖ docker compose 명령
  - ✓ scale up
    - ● docker-compose up -d

```
[+] Running 6/6
 ✔ server_web 5 layers [⣿⣿⣿⣿⣿]      0B/0B      Pulled                    4.2s
   ✔ e886f0f47ef5 Already exists                                        0.0s
   ✔ d738112ed1f3 Already exists                                        0.0s
   ✔ 9a10d47f6b07 Already exists                                        0.0s
   ✔ 33fb7e051279 Already exists                                        0.0s
   ✔ d301d23958b1 Already exists                                        0.0s
[+] Running 3/3
 ✔ Network scale_default          Created                              0.0s
 ✔ Container scale-server_db-1    Started                              0.2s
 ✔ Container scale-server_web-1   Started                              0.2s
```

Docker Compose

- ❖ docker compose 명령
  - ✓ scale up
    - ● docker-compose ps

```
NAME                IMAGE               COMMAND                  SERVICE
CREATED             STATUS              PORTS
scale-server_db-1   redis:latest        "docker-entrypoint.s…"   server_db
About a minute ago  Up About a minute   6379/tcp
scale-server_web-1  httpd:2             "httpd-foreground"       server_web
About a minute ago  Up About a minute   80/tcp
```

Docker Compose

- ❖ docker compose 명령
  - ✓ scale up
    - docker-compose down
    - docker-compose up --scale server_db=3 --scale server_web=3 -d

      [+] Running 7/7

      ✔ Network scale_default Created 0.0s

      ✔ Container scale-server_db-3 Started 0.6s

      ✔ Container scale-server_db-1 Started 0.3s

      ✔ Container scale-server_web-3 Started 0.4s

      ✔ Container scale-server_web-1 Started 0.5s

      ✔ Container scale-server_web-2 Started 0.2s

      ✔ Container scale-server_db-2 Started 0.5s

Docker Compose

- ❖ docker compose 명령
  - ✓ scale up
    - ● docker ps

```
NAME                IMAGE               COMMAND                  SERVICE
CREATED             STATUS              PORTS
scale-server_db-1   redis:latest        "docker-entrypoint.s…"   server_db
About a minute ago  Up About a minute   6379/tcp
scale-server_db-2   redis:latest        "docker-entrypoint.s…"   server_db
About a minute ago  Up About a minute   6379/tcp
scale-server_db-3   redis:latest        "docker-entrypoint.s…"   server_db
About a minute ago  Up About a minute   6379/tcp
scale-server_web-1  httpd:2             "httpd-foreground"       server_web
About a minute ago  Up About a minute   80/tcp
scale-server_web-2  httpd:2             "httpd-foreground"       server_web
About a minute ago  Up About a minute   80/tcp
scale-server_web-3  httpd:2             "httpd-foreground"       server_web
About a minute ago  Up About a minute   80/tcp
```

Docker Compose

- ❖ docker compose 명령
  - ✓ scale up
    - docker-compose up --scale server_db=1 --scale server_web=3 -d
    - docker-compose ps

```
NAME                    IMAGE          COMMAND                  SERVICE      CREATED
STATUS              PORTS
docker-server_db-1      redis:latest   "docker-entrypoint.s…"   server_db    About a
minute ago    Up About a minute   6379/tcp
docker-server_web-1     httpd:2        "httpd-foreground"       server_web   About a
minute ago    Up About a minute   80/tcp
docker-server_web-2     httpd:2        "httpd-foreground"       server_web   About a
minute ago    Up About a minute   80/tcp
docker-server_web-3     httpd:2        "httpd-foreground"       server_web   About a
minute ago    Up About a minute   80/tcp
```

Docker Compose

- ❖ docker compose 명령
  - ✓ docker compose 삭제
    - ● docker-compose [–f 파일경로] down [옵션]
    - ● 옵션
      - ◆ --rmi 종류: 삭제 시에 이미지도 삭제하는 옵션으로 all을 설정하면 모든 이미지가 삭제되고 local로 지정하면 커스텀 태그가 없는 이미지만 삭제
      - ◆ -v, -volumns: 기재된 볼륨을 삭제
      - ◆ --remove-orphans: 컴포즈 파일에 정의되지 않은 서비스의 컨테이너도 삭제

Docker Compose

- ❖ docker compose 명령
  - ✓ docker compose 삭제
    - docker-compose down

```
[+] Running 5/5
 ✔ Container docker-server_db-1    Remove...    0.1s
 ✔ Container docker-server_web-3   Remov...     1.1s
 ✔ Container docker-server_web-1   Remov...     1.2s
 ✔ Container docker-server_web-2   Remov...     1.2s
 ✔ Network docker_default          Removed      0.0s
```

Docker Compose

- ❖ docker compose 명령
  - ✓ 컨테이너 중지 명령어
    - docker-compose [–f 파일경로] stop [서비스이름]: 서비스 이름을 기재하면 특정 서비스만 중지 가능
    - docker-compose up -d

      WARN[0000] /home/adam/docker/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] Running 3/3

      ✔ Network docker_default Created 0.1s

      ✔ Container docker-server_web-1 Start... 0.2s

      ✔ Container docker-server_db-1 Starte... 0.2s
    - docker-compose stop

      WARN[0000] /home/adam/docker/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] Stopping 2/2

      ✔ Container docker-server_web-1 Stopp... 1.1s

      ✔ Container docker-server_db-1 Stoppe... 0.1s

Docker Compose

- ❖ docker compose 명령
  - ✓ 컨테이너 시작 명령어
    - docker-compose [–f 파일경로] start [서비스이름]
    - docker-compose start

      WARN[0000] /home/adam/docker/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion

      [+] Running 2/2

      ✔ Container docker-server_web-1 Start... 0.2s

      ✔ Container docker-server_db-1 Starte... 0.2s

Docker Compose

- docker compose 명령
  - ✓ 컨테이너 로그 확인
    - docker-compose logs -f

Docker Compose

- ❖ docker compose 명령
  - ✓ docker compose 야믈 파일 설정 확인
    - ● docker-compose config

```
name: docker
services:
  server_db:
    image: redis:latest
    networks:
      default: null
  server_web:
    image: httpd:2
    networks:
      default: null
networks:
  default:
    name: docker_default
```

Docker Compose

- ❖ docker compose 명령
  - ✓ 일시 정지(pause) 및 일지 정지된 컨테이너 서비스 재실행(unpause)
    - docker-compose pause

      [+] pause 2/2

      ✔ Container docker-server_db-1 Paused
      0.0s

      ✔ Container docker-server_web-1 Paused
    - docker-compose ps

      NAME IMAGE COMMAND SERVICE CREATED
      STATUS PORTS

      docker-server_db-1 redis:latest "docker-entrypoint.s..." server_db 3 minutes
      ago Up 3 minutes (Paused) 6379/tcp

      docker-server_web-1 httpd:2 "httpd-foreground" server_web 3 minutes
      ago Up 3 minutes (Paused) 80/tcp

Docker Compose

- ❖ docker compose 명령
  - ✓ 일시 정지(pause) 및 일지 정지된 컨테이너 서비스 재실행(unpause)
    - docker-compose unpause

      [+] unpause 2/2

      ✔ Container docker-server_db-1 Unpaused 0.0s

      ✔ Container docker-server_web-1 Unpaused 0.0s
    - docker-compose ps

      NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS

      docker-server_db-1 redis:latest "docker-entrypoint.s..." server_db 4 minutes ago Up 3 minutes 6379/tcp

      docker-server_web-1 httpd:2 "httpd-foreground" server_web 4 minutes ago Up 3 minutes 80/tcp

Docker Compose

- ❖ Load Balancer
  - ✓ 개요
    - 서버에 가해지는 부하(=로드)를 분산(=밸런싱)해주는 장치 또는 기술
    - 클라이언트와 서버풀(Server Pool, 분산 네트워크를 구성하는 서버들의 그룹) 사이에 위치하며 한 대의 서버로 부하가 집중되지 않도록 트래픽을 관리해 각각의 서버가 최적의 퍼포먼스를 보일 수 있도록 해주는 기술
    - 리소스 활용도를 최적화하고 처리량을 최대화하며 지연 시간을 줄이고 내결함성(fault tolerance) 구성을 보장하기 때문에 안정적인 시스템 운영에 도움이 됨
    - docker는 HaProxy, Nginx/Apache Load Balancer 등 외부 서비스와 컨테이너를 결합한 Load Balancing 기능 구현이 가능

Docker Compose

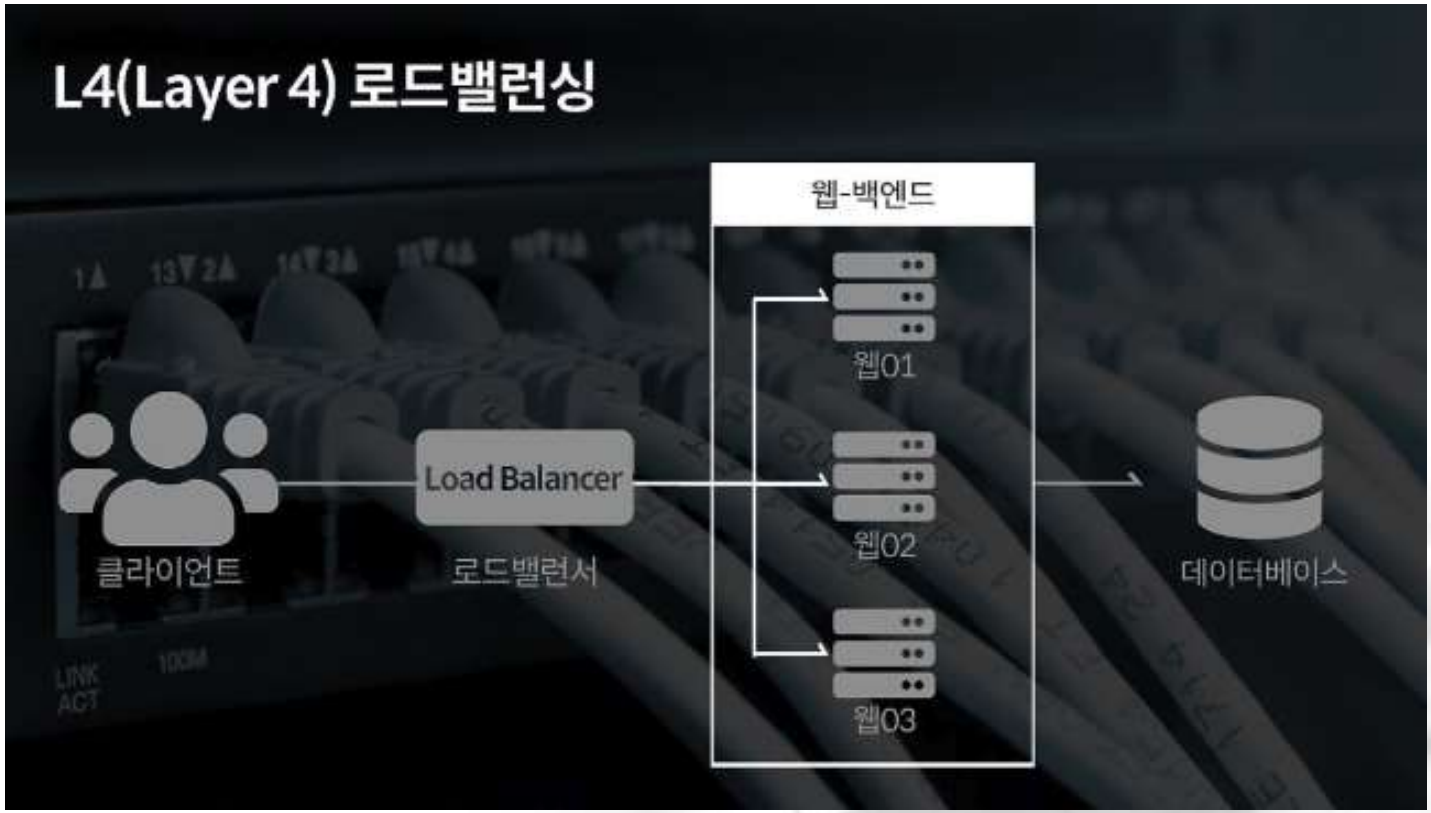
❖ Load Balancer

✓ Load Balancing 연결 알고리즘

연결 알고리즘	설명
라운드-로빈 (Round-Robin, RR)	클라이언트 요청을 서버 가중치를 고려하여 구성된 서버에 균등 배분하는 방식. 경로 보장 안 됨 (기본값).
최소 연결 (least connections)	현재 연결된 클라이언트 수가 가장 적은 서버로 요청 전달. 경로 보장 안 됨.
IP 해시(IP hash)	해시 키를 이용하여 IP별 INDEX를 생성하여 동일 IP 주소는 동일 서버로의 경로 보장. 해당 서버 장애 시 주소 변경됨. 균등 배분 보장 안 됨.
일반 해시(general hash)	사용자가 정의하는 키(IP, Port, URI 문자열 등)를 이용한 서버 지정 방식.
최소 시간(least time)	요청에 대한 낮은 평균 지연시간을 다음 지시자를 기준으로 계산하여 서버 지정. • header: 서버의 첫 번째 바이트를 받는 시간 기준 • last_byte: 서버에서 전체 응답을 받는 시간 기준 • last_byte_inflight: 불완전 요청을 고려한 전체 응답을 받는 시간 기준
무작위(random)	요청에 대한 무작위 서버를 선택하며 특정 기준을 두고 부합되는 서버 선택도 가능.

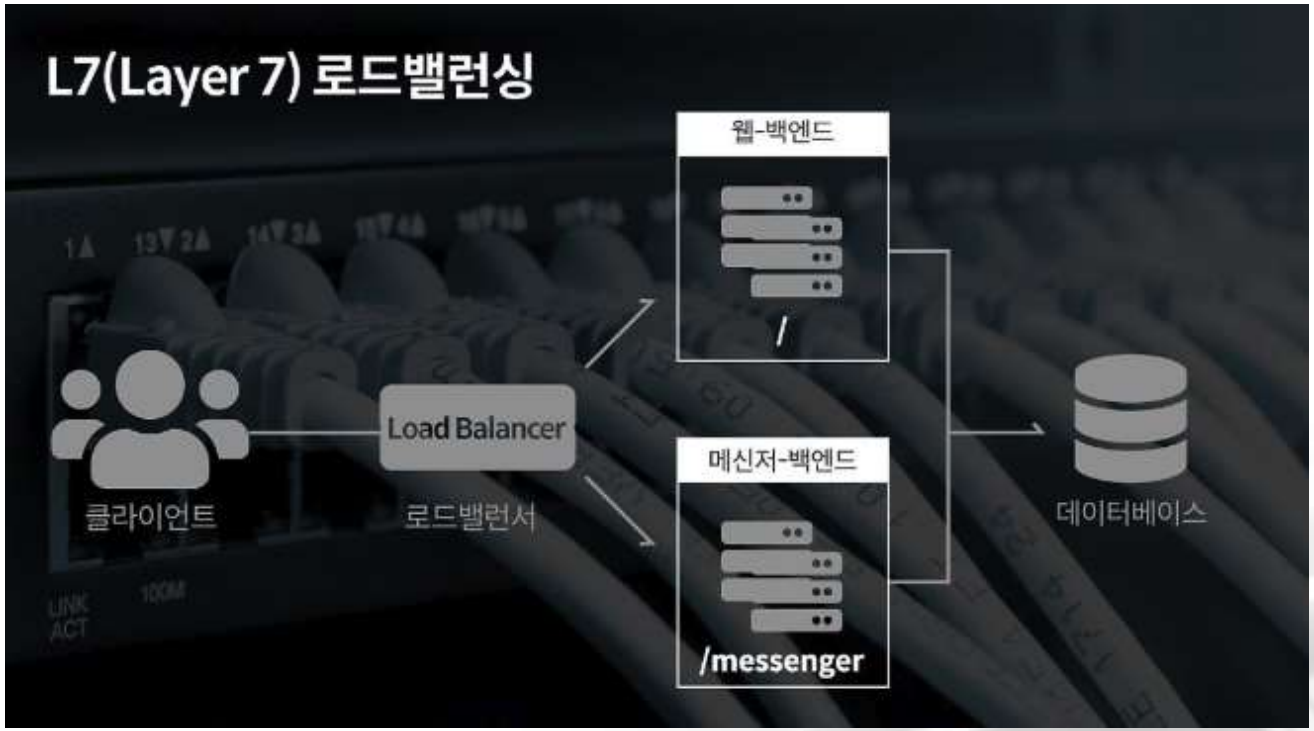
Docker Compose

- Load Balancer
  - ✓ 종류



Docker Compose

- ❖ Load Balancer
  - ✓ 종류



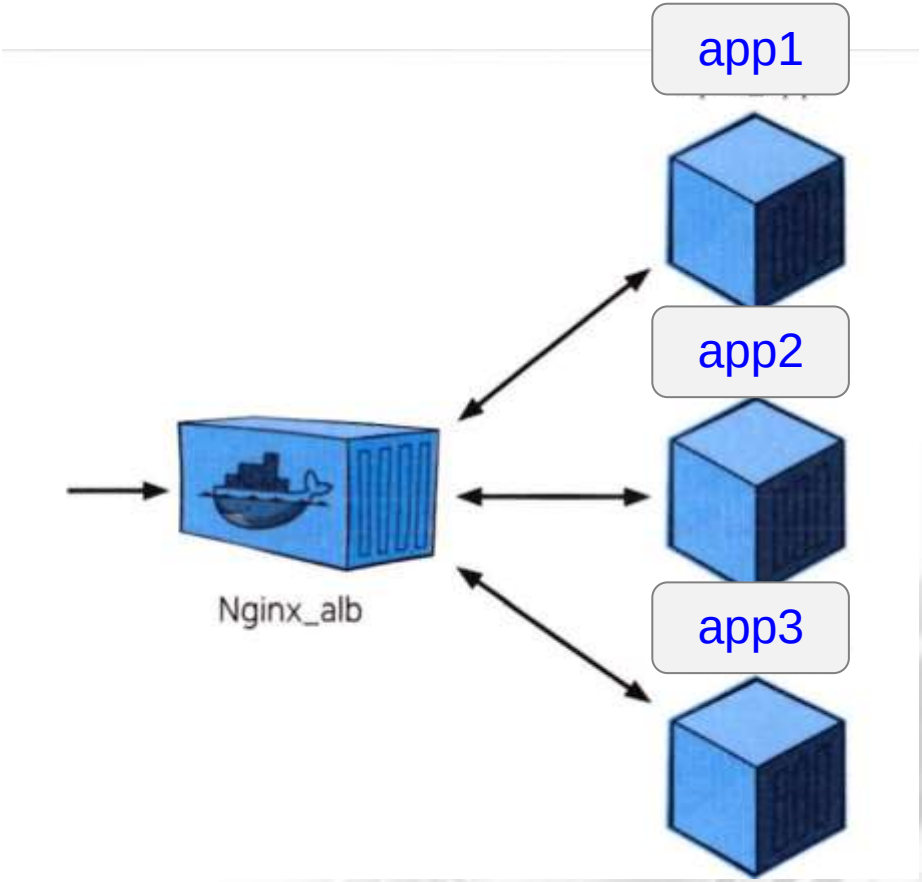
Docker Compose

- ❖ Load Balancer
  - ✓ nginx Load Balancing 관련 파라미터

연결 알고리즘	설명
weight	가중치(weight) 설정을 통해 서버 간 요청 분배(기본값=1).
max_conns, queue	최대 클라이언트 연결 수 지정과 대기열 생성.
max_fails	최대 실패 횟수 지정을 통해 임계치 도달 시 해당 서버를 분배 대상에서 제외.
fail_timeout	응답 최소 시간 지정으로 실패 횟수를 카운트함. 보통 max_fails와 함께 사용함.
backup	backup 키워드가 있는 서버는 평소에 동작하지 않고, 모든 서버가 동작하지 않을 때 사용.
down	down 키워드가 있는 서버는 사용하지 않음. ip_hash인 경우만 사용.

Docker Compose

- docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축



Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● 디렉토리 생성
      - ◆ mkdir alb
      - ◆ cd alb
      - ◆ mkdir nginx_alb pyfla_app1 pyfla_app2 pyfla_app3

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● nginx 설정
      - ◆ cd nginx_alb
      - ◆ Dockerfile 작성

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● nginx 설정
      - ◆ nginx 설정 파일 작성(nginx.conf)

```
upstream web-alb{
        server 172.17.0.1:5001;
        server 172.17.0.1:5002;
        server 172.17.0.1:5003;
}
server {
   listen 80;
        location / {
                proxy_pass http://web-alb;
        }
}
```

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● python 애플리케이션 설정
      - ◆ cd ../pyfla_app1
      - ◆ 파이썬 코드 작성: nano pyfla_app.py

```
from flask import request, Flask
import json
app = Flask(__name__)
@app.route("/")
def index():
  return "Web Application [1]" + "₩n"

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

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● python 애플리케이션 설정
      - ◆ Dockerfile 작성: nano Dockerfile

```
FROM python:3
WORKDIR /
COPY . .
RUN pip install flask
ENTRYPOINT ["python3"]
CMD ["pyfla_app.py"]
```

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● python 애플리케이션 설정
      - ◆ pyfla_app2 와 pyfla_app3 디렉토리에 방금 전 생성한 2개 파일을 복사하고 pyfla_app.py 파일의 출력 부분에서 1만 2나 3으로 수정

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - alb 디렉토리에 docker-compose 파일 작성

```
version: "3"
services:
  pyfla_app1:
    build: ./pyfla_app1
    ports:
      - "5001:5000"
  pyfla_app2:
    build: ./pyfla_app2
    ports:
      - "5002:5000"
  pyfla_app3:
    build: ./pyfla_app3
    ports:
      - "5003:5000"
```

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - ● alb 디렉토리에 docker-compose 파일 작성

```
nginx:
  build: ./nginx_alb
  ports:
    - "8080:80"
  depends_on:
    - pyfla_app1
    - pyfla_app2
    - pyfla_app3
```

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)
    - 서비스 생성: docker-compose up --build -d
    - 방화벽에서 8080, 5001, 5002, 5003 포트에 접근 할 수 있도록 설정
    - 다른 cmd 창을 열어서 curl localhost:8080 을 여러번 접속하면서 서로 다른 결과가 나오는지 확인

Docker Compose

- ❖ docker compose를 이용한 Load Balancer 구성
  - ✓ nginx 설정 변경
    - 알고리즘 변경(least_conn, ip_hash, hash $request_uri consistent, random two least_conn)

```
upstream myapp {
    least_conn;
    server 10.10.10.1:80;
    server 10.10.10.2:80;
}
```

    - 가중치 설정

```
upstream myapp {
    server 10.10.10.1:80 weight=3; # 3배 더 많은 요청을 받음
    server 10.10.10.2:80 weight=1;
}
```

Docker Compose

- docker compose를 이용한 Load Balancer 구성
  - ✓ nginx를 이용한 컨테이너 Load Balancer 구축(flask)

환경 변수 삽입 및 관리

- 환경 변수를 다루는 방법
  - ✓ Compose 파일에 직접 입력
  - ✓ 쉘 환경 변수로 등록
  - ✓ 환경 변수 파일로 구성
  - ✓ Dockerfile에 환경 변수를 직접 삽입

환경 변수 삽입 및 관리

- Compose 파일에 직접 입력
  - ✓ 방법
    - MySQL 이미지로 컨테이너를 구동시킬 때엔 MySQL의 관리자(root) 계정 비밀번호를 설정할 MYSQL_ROOT_PASSWORD 변수값이 반드시 필요한데 이럴 땐 배포용 Compose 파일에서 services.<서비스명>.environment 부분에 해당 내용을 변수명: 값 또는 - 변수명=값 형태로 넣어주면 됨
  - ✓ MySQL 8 이미지에 MYSQL_ROOT_PASSWORD 환경 변수값을 password로 지정하여 둔 Compose 파일

```
services:
  mysql:
    image: mysql:8
    restart: unless-stopped
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: password
```

환경 변수 삽입 및 관리

- 쉘 환경 변수로 등록
  - ✓ 쉘 환경 변수 설정

```
export MYSQL_ROOT_PASSWORD=password
```

  - ✓ MySQL 8 이미지에 MYSQL_ROOT_PASSWORD 환경 변수값을 password로 지정하여 둔 Compose 파일

```
services:
  mysql:
    image: mysql:8
    restart: unless-stopped
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
```

환경 변수 삽입 및 관리

- 쉘 환경 변수로 등록
  - ✓ 실제 내용 확인: docker-compose convert

```
name: mysql
services:
  mysql:
    environment:
      MYSQL_ROOT_PASSWORD: password
    image: mysql:8
    networks:
      default: null
    ports:
      - mode: ingress
        target: 3306
        published: "3306"
        protocol: tcp
    restart: unless-stopped
networks:
  default:
    name: mysql_default
```

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 개요
    - 배포용 Compose 파일에 환경 변수를 하나하나 다 적어넣거나 매번 쉘에서 변수로 직접 선언하는 방법은 다소 비효율적
    - 필요한 환경 변수 항목들만 골라내어 별도의 파일로 구성해 둔다면 환경 변수 관리를 보다 효율적으로 할 수 있음
    - Docker Compose에서 환경 변수 정보들을 떼어 별도의 파일로 구성할 때 가장 간편한 방법은 Compose 파일이 위치한 동일 경로에 .env 파일을 따로 구성하는 것
    - .env 파일을 이용하는 방식은 간편하기는 하지만 Docker Compose에서 docker compose up 명령을 수행할 때에만 활용 가능
  - ✓ env 파일 문법
    - 각 줄마다 변수명=값의 형태로 입력
    - 주석 처리는 # 문자를 이용
    - 비어있는 줄은 무시
    - 따옴표 처리('', "")는 불필요한데 입력된 따옴표는 위치에 따라 변수명이나 값의 일부분으로 간주

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 이전에 만든 환경 변수 삭제

    unset MYSQL_ROOT_PASSWORD
  - ✓ 하나의 파일 사용
    - ● docker-compose 파일과 동일한 위치에 .env 파일을 생성하고 작성

      MYSQL_ROOT_PASSWORD=password

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 하나의 파일 사용
    - 실제 내용 확인: docker-compose convert

```
name: mysql
services:
  mysql:
    environment:
      MYSQL_ROOT_PASSWORD: password
    image: mysql:8
    networks:
      default: null
    ports:
      - mode: ingress
        target: 3306
        published: "3306"
        protocol: tcp
    restart: unless-stopped
networks:
  default:
    name: mysql_default
```

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 실행 시 파일 지정하기
    - 경우에 따라 여러 개의 환경변수 파일로 나누는 일이 필요할 수 있는데 개발 환경(dev)과 운영 환경(prod)에 따라 주입해야 할 값들이 다른 경우엔 각각의 환경에 맞는 별도의 env_file을 구성한 뒤 배포할 때 --env-file <파일경로> 플래그로 불러올 파일을 직접 지정할 수 있음
    - 환경 변수 파일 생성

      $ cat ./config/.env.dev

      MYSQL_ROOT_PASSWORD=passwd_dev

      $ cat ./config/.env.prod

      MYSQL_ROOT_PASSWORD=passwd_prod

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 실행 시 파일 지정하기
    - 적용

      docker compose --env-file ./config/.env.dev config

```
name: mysql
services:
  mysql:
    environment:
      MYSQL_ROOT_PASSWORD: passwd_dev
    image: mysql:8
    networks:
      default: null
    ports:
      - mode: ingress
        target: 3306
        published: "3306"
        protocol: tcp
    restart: unless-stopped
networks:
  default:
    name: mysql_default
```

환경 변수 삽입 및 관리

- ❖ 환경 변수 파일로 구성
  - ✓ 서비스 별로 환경 변수 파일 설정
    - 각 서비스(컨테이너)에 대한 환경 변수 파일을 별도로 관리해야 할 수도 있는데 이런 경우에는 Compose 파일에 명시된 각 서비스 안에 env_file 항목을 함께 포함시키게 되는데 이 항목을 따로 명시하지 않았다면 기본적으로는 env_file: .env이 적용된 상태
    - 예시

```
services:
  mysql:
    env_file: # 파일이 여러 개라면 리스트 형태로 삽입한다.
      - a.env
      - b.env
    image: mysql:8
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
```

환경 변수 삽입 및 관리

- Dockerfile에 환경 변수를 직접 삽입
  - ✓ 환경 변수를 YAML이나 별도의 외부 파일로 담아두는 것이 꺼려지는 경우 Dockerfile에 ARG 또는 ENV를 이용하여 환경 변수를 직접 삽입한 뒤 나만의 이미지를 빌드하여 사용
  - ✓ Dockerfile

```
FROM mysql:8
ENV MYSQL_ROOT_PASSWORD password
```

환경 변수 삽입 및 관리

- 우선 순위
  1. Compose 파일에 직접 입력한 값
  2. 쉘 환경변수로 등록한 값
  3. 환경변수 파일로 입력된 값(.env 등)
  4. Dockerfile을 통해 삽입된 값

환경 변수 삽입 및 관리

- 한계점
  - ✓ Docker Compose에서의 환경변수 관리 방법에는 공통적인 단점이 하나 있는데 모든 정보가 평문(plain text)으로 저장된다는 것인데 이처럼 중요한 보안 정보가 평문 상태로 호스트에 남아있도록 하는 것은 보안 측면에서 결코 바람직하지 않음
  - ✓ 운영 환경에서는 이를 보완하기 위한 다양한 방법
    - 외부에 노출되지 않는 프라이빗 레지스트리를 구축한 뒤 보안 정보가 포함된 전용 이미지를 빌드하면서 운영하는 방법을 고려할 수 있는데 이 경우 해당 보안 정보가 이미지의 특정 레이어에 계속 남게 되므로 이미지와 레지스트리 관리에 신중해야 함
    - 쿠버네티스(Kubernetes), 도커 스웜(Docker Swarm) 등 컨테이너 오케스트레이션 도구에서는 이러한 보안 정보를 조금 더 안전히 다룰 수 있는 시크릿(Secret)을 제공하며 AWS, Azure 등 주요 클라우드 서비스 제공자들이 지원하는 컨테이너 인스턴스들 역시 환경 변수를 설정할 때 이러한 시크릿(Secret) 기능을 간편히 이용할 수 있도록 배려하고 있음
    - 더욱 강화된 형태의 보안이 필요하다면 하시코프 볼트(HashiCorp Vault) 등의 솔루션을 고려