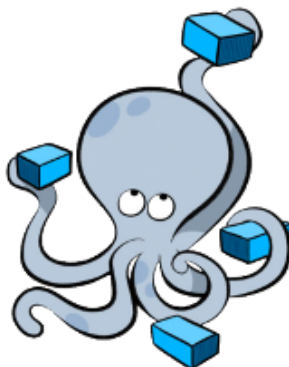


Dockercompose



Try Docker Compose

Check out this tutorial on how to use Docker Compose from defining application dependencies to experimenting with commands.

 <https://docs.docker.com/compose/gettingstarted/>



<https://github.com/docker/compose>

- Docker Compose는 Docker를 사용하여 멀티 컨테이너 Docker 애플리케이션을 정의하고 실행하기 위한 도구이다.
- Docker Compose를 사용하면 YAML 파일을 사용하여 애플리케이션의 서비스, 네트워크 및 볼륨을 구성할 수 있습니다.
- Docker Compose는 각 서비스의 이미지를 빌드하고, 컨테이너를 시작하며, 각 컨테이너 간의 네트워크를 설정합니다. 이를 통해 애플리케이션에 필요한 모든 구성 요소를 하나의 명령으로 실행할 수 있습니다.

위에서 링크된 Docker Compose 공식 GitHub 페이지에서 Docker Compose의 자세한 사용 방법과 YAML 파일 작성 방법 등을 참고하실 수 있습니다.

- Docker-compose 버전 확인

```
> docker compose --version
```

참고

`docker-compose` 와 `docker compose` 는 동일한 명령어입니다. 이전에는 `docker-compose` 를 사용했지만, Docker 버전 20.03부터는 `docker compose` 가 새로운 구문으로 추가되었습니다. 이 구문의 사용법은 `docker-compose` 와 같습니다. 따라서 두 명령어는 동일한 기능을 수행합니다.

YML 예시

- Docker Compose를 사용하여 멀티 컨테이너 Docker 애플리케이션을 구성하는 방법은 YAML 파일을 작성하는 것입니다. 각 서비스에 대한 구성을 작성하고, 컨테이너 간에 공유할 네트워크와 볼륨을 지정할 수 있습니다. 아래는 YAML 파일의 예시입니다.

```
# docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
  redis:
    image: "redis:alpine"
```

위의 예시에서, `version` 은 Compose 파일 포맷의 버전을 지정합니다. `services` 는 애플리케이션의 각 서비스를 정의합니다. `web` 과 `redis` 는 서비스의 이름입니다. `build` 는 Dockerfile을 사용하여 이미지를 빌드하는 방법을 지정합니다. `ports` 는 호스트와 컨테이너 간의 포트 매핑을 정의합니다. `volumes` 는 호스트와 컨테이너 간의 파일 시스템 공유를 정의합니다. `image` 는 Docker Hub에서 가져올 이미지의 이름을 지정합니다.

YML 파일에서 더 많은 구성 옵션을 사용할 수 있습니다. Compose 파일에 대한 자세한 내용은 Docker Compose 공식 문서를 참조하십시오.

위의 예시처럼 각 서비스의 이름과 해당 서비스에서 사용할 이미지, 포트 매핑, 파일 시스템 공유 등을 정의할 수 있습니다. 이러한 구성은 도커 컴포즈를 사용하여 로컬 머신에서 애플리케이션을 실행하고, 배포할 때 유용합니다.

또한, YML 파일을 사용하면 애플리케이션의 다양한 환경에서 사용할 수 있도록 설정을 변경할 수 있습니다. 예를 들어, 개발, 스테이징, 프로덕션 환경에서 각각 다른 구성으로 설정할 수 있습니다.

Docker Compose는 각 서비스의 이미지를 빌드하고, 컨테이너를 시작하며, 각 컨테이너 간의 네트워크를 설정합니다. 이를 통해 애플리케이션에 필요한 모든 구성 요소를 하나의 명령으로 실행할 수 있습니다. 더 자세한 Docker Compose의 사용 방법과 YAML 파일 작성 방법을 알고 싶다면, 위에서 제공한 Docker Compose 공식 GitHub 페이지나 Docker Compose 공식 문서를 참고하십시오.

참고

yaml 파일과 yml 파일은 동일한 파일 형식이며, 파일 확장자만 다릅니다. 따라서 두 파일은 동일한 방식으로 사용할 수 있습니다. 그러나 yaml 파일이 더 널리 사용되고 있습니다.

docker-compose up 명령어

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

- docker-compose.yml 파일에 맞춰 container를 설치, 실행
- 기존에 CLI에 직접 길게 -v, -e, -p 옵션 등을 치며 하던 내용을 yml파일로 작성해 컨테이너 실행에 활용

```
# docker-compose.yml 예시(mysql, wordpress 설치)

version: '3'
services:
  db:
    image: mysql:5.7
    volumes:
      - ./mysql:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    image: wordpress:latest
    volumes:
      - ./wp:/var/www/html
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
```

mysql
image이름 : tag명
-v 옵션 (데이터 유실되지 않도록 mount)

Container가 자동으로 죽게되면, docker가 다시 띄워주는 역할
-e 옵션

word press


docker-compose 문법

version: '3'

- docker-compose.yml 파일의 명세 버전
- docker-compose.yml 버전에 따라 지원하는 도커 엔진 버전도 다름

Compose file versions and upgrading


Compose file reference

 <https://docs.docker.com/compose/compose-file/compose-versioning/>



Docker Compose와 버전별 특징 : NHN Cloud Meetup

도커는 이제 대부분의 개발자 노트북이나 PC에 하나씩은 설치되어있는 필수품이 되어가는데요 편하고 유용한 도커를 좀 더 유익하고 편하게 사용할 수 있는 도구인 Docker Compose에 대해서 알기 쉽게 정리하는 시간을 가졌습니다. Compose를

 <https://meetup.nhncloud.com/posts/277>



Compatibility matrix

There are several versions of the Compose file format – 1, 2, 2.x, and 3.x

This table shows which Compose file versions support specific Docker releases.

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+
...	...

services

- 실행할 컨테이너 정의
- `docker run --name django`과 같다고 생각할 수 있음

image

- 컨테이너에 사용할 이미지 이름과 태그
- 태그를 생략하면 latest
- 이미지가 없으면 자동으로 pull

ports:

```
- "8000:8000"
```

- 컨테이너와 연결할 포트(들)
- {호스트 포트}:{컨테이너 포트}

environment:

```
- MYSQL_ROOT_PASSWORD=somewordpress: '3'
```

- 컨테이너에서 사용할 환경변수(들)
- {환경변수 이름}:{값}

volumes:

```
- ./app:/app
```

- 마운트하려는 디렉터리(들)
- {호스트 디렉터리}:{컨테이너 디렉터리}

restart: always

- 재시작 정책
- restart: "no"
- restart: always
- restart: on-failure
- restart: unless-stopped

build:

```
context: .
```

```
dockerfile: ./compose/django/Dockerfile-dev
```

- 이미지를 자체 빌드 후 사용
- image 속성 대신 사용함
- 여기에 사용할 별도의 도커 파일이 필요함

docker-compose 명령어

up

- docker-compose.yml에 정의된 컨테이너를 실행

```
> docker compose up
> docker compose up -d #docker run의 -d 옵션과 동일
> docker compose up --force-recreate #컨테이너를 새로 만들기
> docker compose up --build #도커 이미지를 다시 빌드(build로 선언했을 때만)
```

start

- 멈춘 컨테이너를 재개

```
> docker compose start docker-compose start wordpress #wordpress 컨테이너만 재개
```

restart

- 컨테이너를 재시작

```
> docker compose restart
> docker compose restart ubuntu #wordpress 컨테이너만 재시작
```

stop

- 컨테이너 멈춤

```
> docker compose stop
> docker compose stop wordpress #wordpress 컨테이너만 멈춤
```

down

- 컨테이너를 종료하고 삭제

```
> docker compose down
```

logs

- 컨테이너의 로그

```
> docker compose logs docker-compose logs -f #로그 follow
```

ps

- 컨테이너 목록

```
> docker compose ps
```

exec

- 실행 중인 컨테이너에서 명령어 실행

```
> docker compose exec {컨테이너 이름} {명령어}  
> docker compose exec wordpress bash
```

build

- 컨테이너 build 부분에 정의된 내용대로 빌드
 - build로 선언된 컨테이너만 빌드

```
docker compose build  
docker compose build wordpress #wordpress 컨테이너만 build
```

참고

<https://github.com/docker/awesome-compose>

Enabling GPU access with Compose

GPU support in Compose

 <https://docs.docker.com/compose/gpu-support/>

