



Google Developer Group
Incheon National University

Docker Start!

1기 장철희

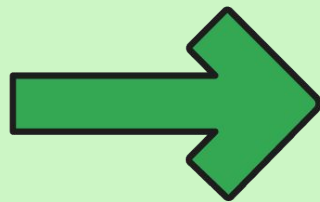


Table of Contents

1. 도커 소개
2. 도커 기본 명령어
3. Dockerfile 작성법
4. Docker Compose 소개
5. 데모 애플리케이션 배포(실습)
6. Q&A 및 마무리



<https://github.com/GDG-INU/gdg-inu-seminar>



Google Developer Groups

```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
  ),  
  Text(  
    'Ka  
    sty  
    c  
    ),  
    ),
```

Introduce Docker

도커 소개

Docker - 컨테이너를 다루는 플랫폼

컨테이너란?

- 호스트 OS의 커널을 공유하면서 애플리케이션과 실행에 필요한 라이브러리만 포함한 가볍고 빠른 격리 환경

그 전에 가상머신(VM)이란?

- 하드웨어를 가상화하여 하나의 물리적 서버를 논리적으로 여러 개 서버처럼 사용하는 기술 (무거움)

도커 소개

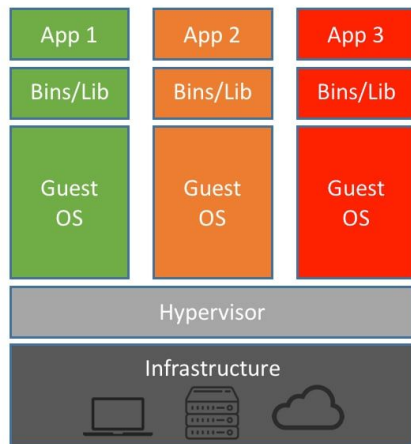
컨테이너 vs 가상머신

가상머신 (VM)

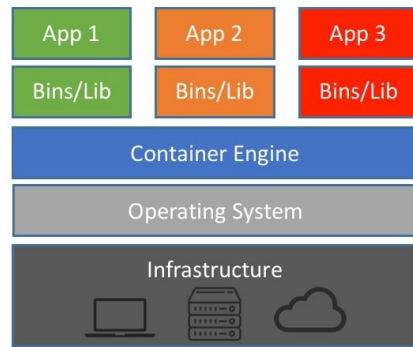
- 각 VM마다 독립적인 Guest OS 필요
- 무겁고 확장성 떨어짐
- 리소스 사용량이 많음

컨테이너

- 호스트 OS 리소스를 논리적으로 분리해 공유
- 가볍고 빠르게 작동 (게스트 OS 불필요)
- 동일한 인프라에서 더 많은 애플리케이션 실행 가능



Machine Virtualization



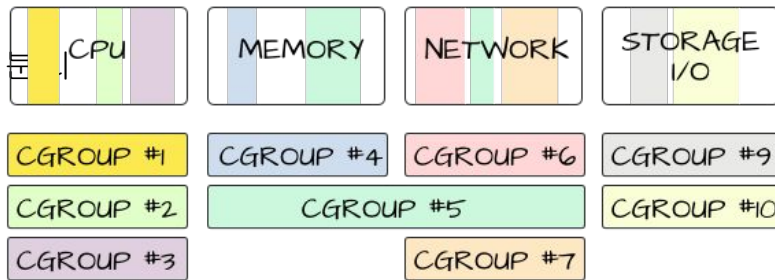
Containers

도커의 기본 구조

초기 도커는 리눅스 컨테이너 기술(LXC) 기반, 현재는 **libcontainer** 기반의 **runC** 사용

호스트 OS 위에서 동작하는 격리된 실행 환경을 제공하기 위해 아래 두 기술 사용

네임스페이스: 프로세스(PID), 네트워크, 파일시스템(MOUNT) 등



cgroup: 프로세스와 스레드를 그룹화해서 관리하는 기술

- 그룹별로 사용할 수 있는 컴퓨팅 리소스 리소스
(CPU, 메모리, 디스크 I/O 등) 할당/정의

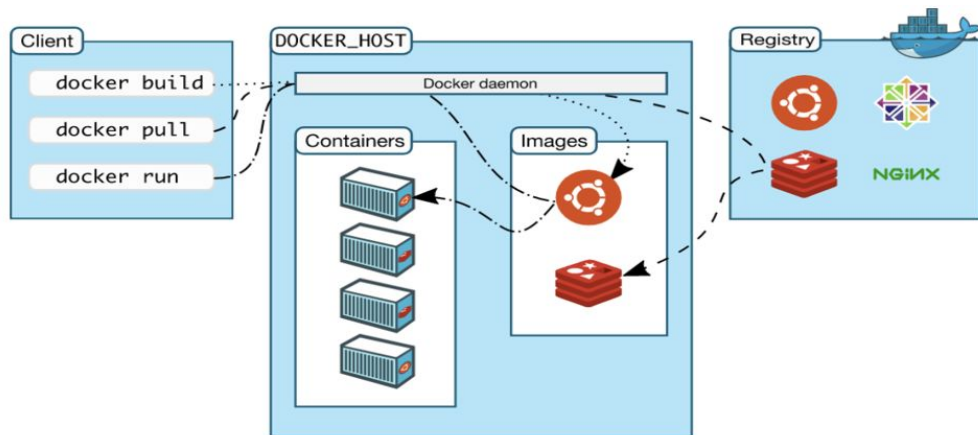
핵심 개념

이미지

- 애플리케이션 실행에 필요한 모든 파일이 포함된 읽기 전용 템플릿
- 특징: 레이어 구조-변경된 부분만 차분(레이어)으로 관리
- 이미지 공유를 통한 재사용성

컨테이너

- 이미지의 실행 가능한 인스턴스
- 격리된 프로세스, 네트워크, 파일시스템 제공



<https://www.qa.com/resources/blog/what-is-docker/>

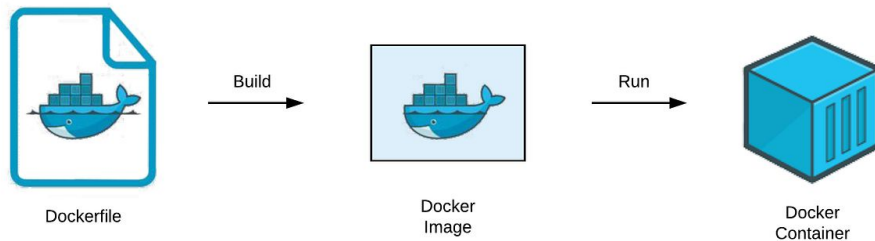
핵심 개념

Dockerfile

- 이미지를 코드로 정의하는 설정 파일
- 베이스 이미지, 환경변수, 명령어 등 정의

레지스트리

- 이미지를 저장하고 공유하는 저장소
- Docker Hub, AWS ECR 등



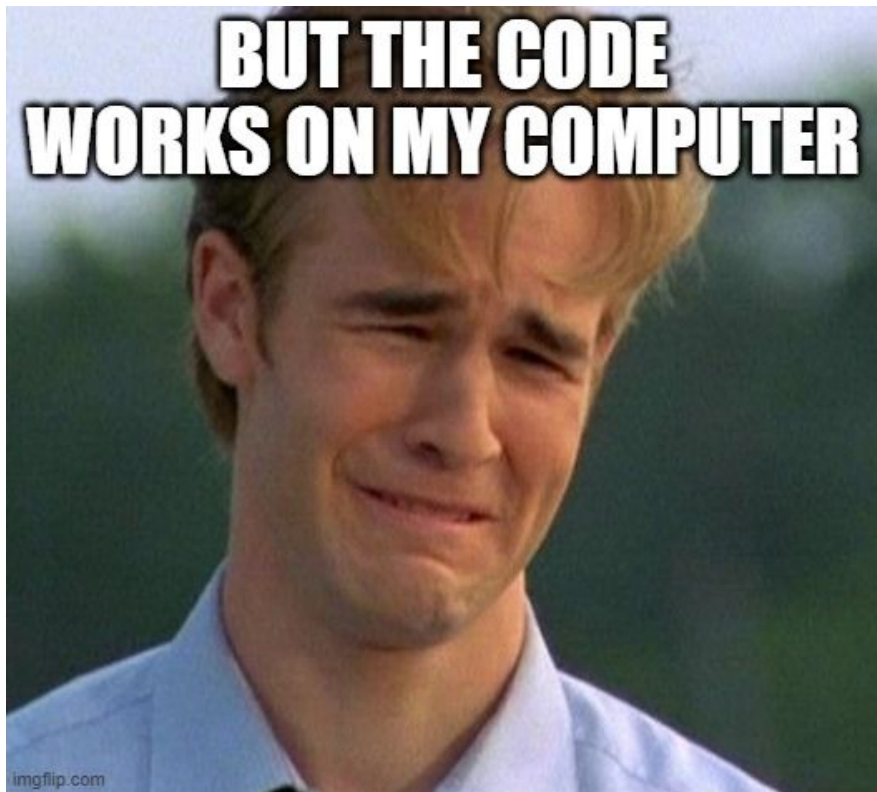
Elastic Container Registry

<https://medium.com/swlh/understand-dockerfile-dd11746ed183>

필요성

개발 환경 표준화

- "내 컴퓨터에서는 작동하는데?"
- 개발, 테스트, 운영 환경의 일관성 확보



필요성

작은 단위의 개발

- 각 컨테이너를 독립적으로 배포, 개발, 테스트
- 마이크로서비스 아키텍처에 적합

현대 애플리케이션 개발에 적합

- 빠른 개발 주기, 빠른 배포, 빠른 피드백
- 배포환경 및 의존성 문제 해결



```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
),  
Text(  
  'Ka  
  sty  
  c  
  ),  
),  
),
```

Let's Practice!

이미지 관련 명령어

```
# 이미지 다운로드
docker pull <이미지명>[:태그]

# 이미지 목록 확인
docker images
# 또는
docker image ls

# 이미지 삭제
docker rmi <이미지명>
```

docker pull - 이미지 다운로드

docker image ls - (로컬)이미지 목록 확인

docker rmi - (로컬) 이미지 삭제

컨테이너 관련 명령어

```
# 컨테이너 실행
docker run [옵션] <이미지명> [명령어]

# 실행 중인 컨테이너 목록
docker ps

# 또는
docker container ls

# 모든 컨테이너 목록 (정지된 것 포함)
docker ps -a

# 컨테이너 정지
docker stop <컨테이너ID/이름>

# 컨테이너 삭제
docker rm <컨테이너ID/이름>
```

docker run - 컨테이너 실행

docker ps - 실행 중인 컨테이너 목록 출력

docker ps -a - 실행/정지된 모든 컨테이너 목록

docker stop - 컨테이너 정지

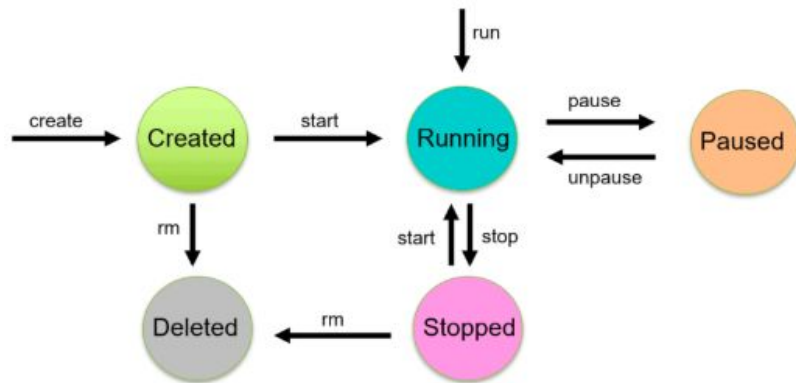
docker rm - 컨테이너 삭제

컨테이너 실행 옵션

옵션	설명	예시
<code>-d</code>	백그라운드 모드로 실행 (detached)	<code>docker run -d nginx</code>
<code>-p</code>	포트 포워딩 (호스트:컨테이너)	<code>docker run -p 8080:80 nginx</code>
<code>-v</code>	볼륨 마운트 (호스트:컨테이너)	<code>docker run -v /host/path:/container/path nginx</code>
<code>-e</code>	환경변수 설정	<code>docker run -e MYSQL_ROOT_PASSWORD=secret mysql</code>
<code>--name</code>	컨테이너 이름 지정	<code>docker run --name web-server nginx</code>
<code>--rm</code>	컨테이너 종료 시 자동 삭제	<code>docker run --rm ubuntu ls</code>
<code>-it</code>	대화형 터미널 연결	<code>docker run -it ubuntu bash</code>
<code>--network</code>	네트워크 연결	<code>docker run --network my-network nginx</code>
<code>--restart</code>	재시작 정책 설정	<code>docker run --restart always nginx</code>

컨테이너 실행

```
$ docker run hello-world  
  
$ docker run -it ubuntu bash  
  
$ docker images hello-world
```



```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
),  
Text(  
  'Ka  
  sty  
  c  
  ),  
),
```

Introduce Dockerfile

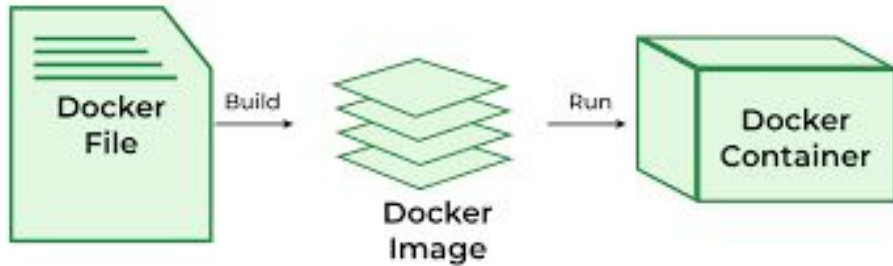
Dockerfile이란?

이미지를 코드로 정의하는 설정 파일

Docker 이미지를 생성하기 위한 프로세스(순서)를 정의

위에서 아래로 순서대로 실행하여 이미지가 만들어짐

- 베이스 이미지, 환경변수, 데몬 실행 정보, 미들웨어나 OS 설치/설정 등 컨테이너 구성정보를 기술



<https://www.geeksforgeeks.org/what-is-dockerfile/>

Dockerfile 지시어

```
FROM <베이스 이미지>
WORKDIR <작업 디렉토리>
COPY <호스트 경로> <컨테이너 경로>
RUN <명령어>
ENV <환경변수명>=<값>
EXPOSE <포트>
CMD ["실행 명령어"]
```

지시어	설명
FROM	베이스 이미지 지정
WORKDIR	작업 디렉토리 설정
COPY	파일/디렉토리 복사
RUN	명령 실행 (이미지 빌드 시)
ENV	환경변수 설정
EXPOSE	컨테이너가 리스닝할 포트 지정
CMD	컨테이너 실행 시 기본 명령 지정
ENTRYPOINT	컨테이너 실행 시 항상 실행되는 명령

Dockerfile

훑어보기

```
# Nginx 공식 이미지를 베이스로 사용
FROM nginx:alpine

# 작업 디렉토리 설정
WORKDIR /usr/share/nginx/html

# 기존 기본 파일 삭제
RUN rm -rf ./*

# 로컬 html 파일들을 컨테이너의 웹 루트 디렉토리로 복사
COPY ./html/ .

# 포트 80 노출
EXPOSE 80

# Nginx 실행
CMD ["nginx", "-g", "daemon off;"]
```

ENTRYPOINT vs CMD

함께 사용 시 동작

```
ENTRYPOINT ["python"]
CMD ["app.py"]
```

위와 같이 설정하면:

- 컨테이너는 기본적으로 `python app.py` 를 실행
- `docker run myimage script.py` 로 실행하면 `python script.py` 가 됨
- ENTRYPOINT는 그대로 유지되고 CMD만 대체됨

Dockerfile 빌드

이미지 빌드

```
docker build -t my-website .
```

컨테이너 실행 (80번 포트 매핑)

```
docker run -d -p 8080:80 my-website
```

효율적인 Dockerfile 작성 팁

1. 베이스 이미지 최적화

가능하면 작은 이미지 사용 (alpine 등)

2. 레이어 최소화

여러 RUN 명령을 하나로 합치기

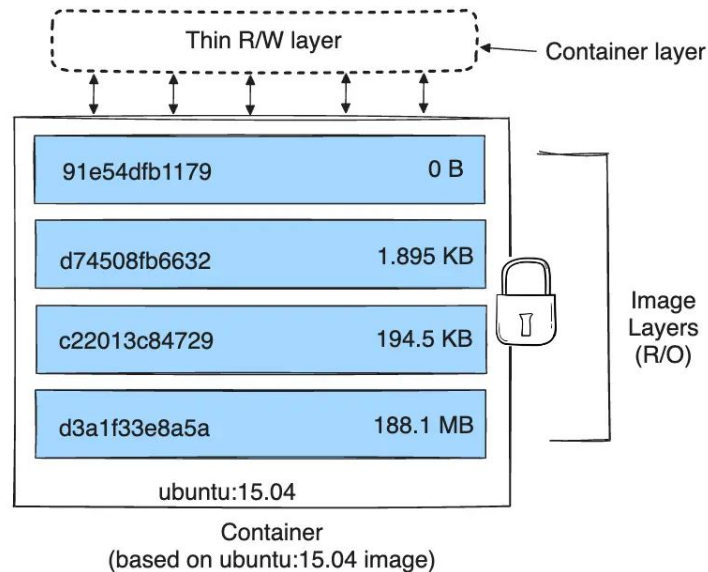
빈번히 변경되는 레이어는 마지막에 배치

3. .dockerignore 활용

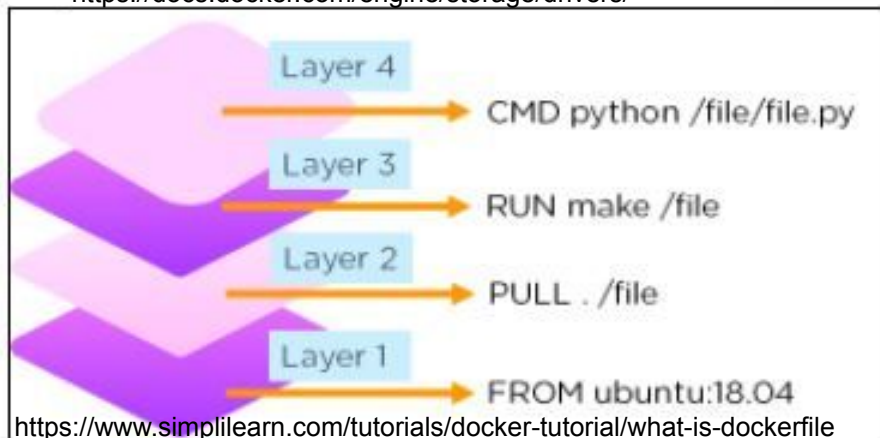
불필요한 파일 제외 (node_modules, .git 등)

4. 캐시 활용하기

종속성 설치와 코드 복사 분리



<https://docs.docker.com/engine/storage/drivers/>



<https://www.simplilearn.com/tutorials/docker-tutorial/what-is-dockerfile>

```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
  ),  
  Text(  
    'Ka  
    sty  
    c  
    ),  
    ),
```

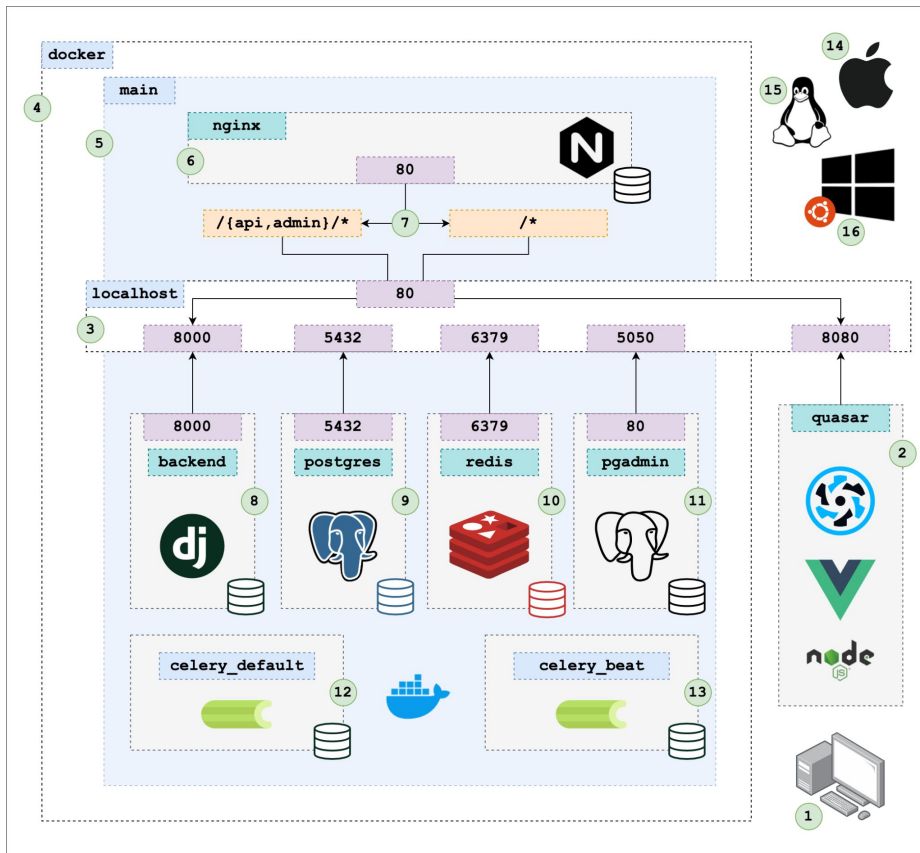
Docker Compose

Docker Compose란?

여러 컨테이너로 구성된 애플리케이션을 정의하고 실행하는 도구

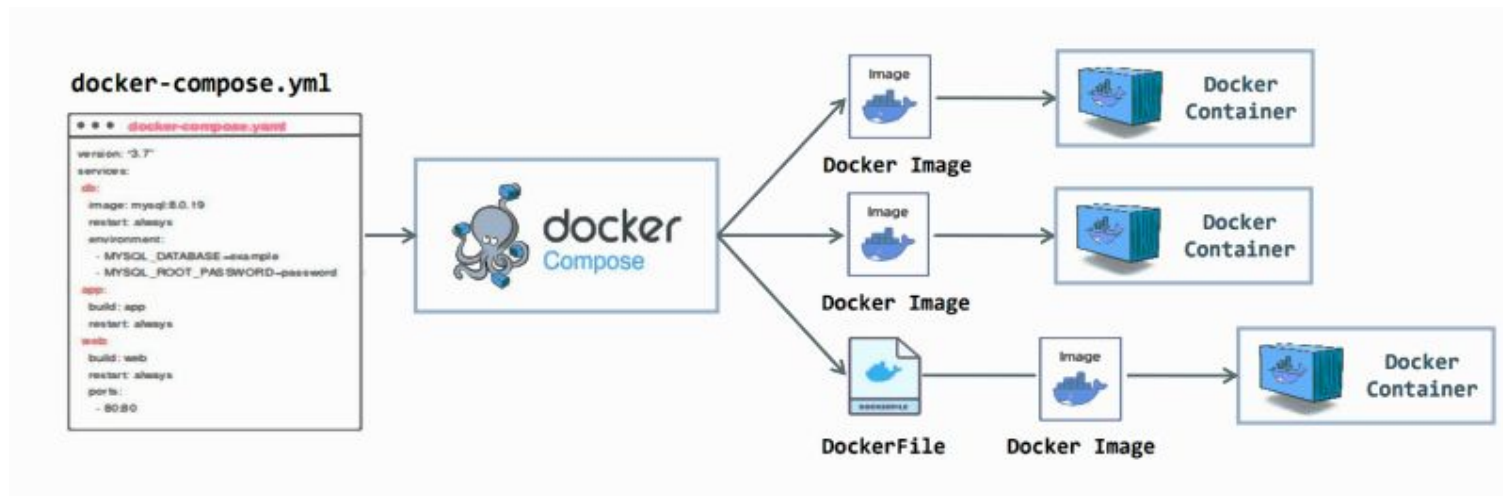
주요 특징:

- YAML 파일로 다중 컨테이너 정의
- 서비스 간 의존성 관리
- 네트워크 및 볼륨 설정
- 한 번에 여러 컨테이너 시작/중지



Docker Compose

기본개념



<https://blog.devops.dev/what-and-why-of-docker-compose-dc95314c74b8>

Docker Compose 작성법

Compose 파일 구조

서비스 정의: 이미지, 포트, 환경변수, 의존성 등

네트워크: 컨테이너 간 통신 정의

볼륨: 데이터 영속성 관리

환경변수 정의

섹션	설명	사용 사례
<code>services</code>	컨테이너 서비스 정의 섹션	모든 애플리케이션 구성 요소(웹서버, 데이터베이스 등)를 정의하는 최상위 섹션
<code>image</code>	사용할 이미지 지정	기존 Docker 이미지 사용 시 (예: 공식 MySQL, Nginx 이미지 등)
<code>build</code>	Dockerfile로 빌드할 경로 지정	커스텀 이미지를 빌드해야 할 때 (예: 자체 애플리케이션 코드)
<code>ports</code>	포트 매핑 (호스트:컨테이너)	외부에서 컨테이너 서비스에 접근할 수 있도록 포트 노출 시
<code>volumes</code>	볼륨 마운트 지정	데이터 지속성 유지, 설정 파일 마운트, 코드 변경사항 실시간 반영 시
<code>environment</code>	환경변수 설정	컨테이너 내부 애플리케이션 구성 (DB 연결 정보, API 키 등)
<code>depends_on</code>	서비스 의존성 정의	서비스 시작 순서 제어 (예: DB가 먼저 시작된 후 웹 서버 시작)
<code>networks</code>	네트워크 연결 설정	컨테이너 간 통신 그룹화, 네트워크 격리 필요 시
<code>restart</code>	재시작 정책	서비스 안정성 보장 (충돌 시 자동 재시작, 호스트 재부팅 시 자동 시작)
<code>container_name</code>	컨테이너 이름 지정	자동 생성된 이름 대신 식별하기 쉬운 고정 이름 필요 시
<code>healthcheck</code>	컨테이너 상태 확인 설정	애플리케이션 실제 동작 여부 확인, 오케스트레이션 도구의 자동 복구 지원
<code>volumes</code> (최상위)	볼륨 정의 섹션	여러 서비스에서 공유할 볼륨 정의, 외부 볼륨 사용 시
<code>networks</code> (최상위)	네트워크 정의 섹션	커스텀 네트워크 설정 (서브넷, 게이트웨이 등), 외부 네트워크 통합 시

Docker Compose 작성법

Compose 파일 구조

서비스 정의: 이미지, 포트, 환경변수, 의존성 등

네트워크: 컨테이너 간 통신 정의

볼륨: 데이터 영속성 관리

환경변수 정의

```
services:
  web:
    build: ./app
    container_name: flask-app
    ports:
      - "5001:5000"
    environment:
      - DB_HOST=db
      - DB_USER=user
    volumes:
      - ./app:/app
    depends_on:
      - db
    restart: always
    networks:
      - app-network

  db:
    image: mysql:8
    container_name: mysql-db
    environment:
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    volumes:
      - db_data:/var/lib/mysql
    ports:
      - "3308:3308"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

volumes:
  db_data:
```

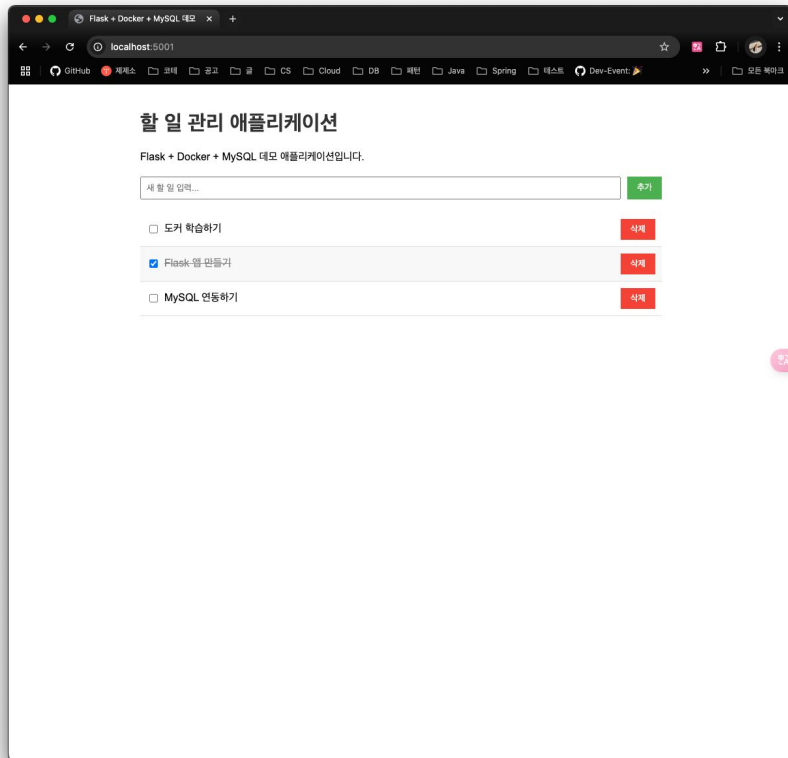
```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
),  
Text(  
  'Ka  
  sty  
  c  
  ),  
),
```

Demo

Docker Compose로 간단한 ToDo 리스트 배포하기

Flask + MySQL로 ToDo List 배포하기

- Github 레포지토리 코드 참고
- <https://github.com/GDG-INU/gdg-inu-seminar/tree/main/2025-03/docker/code/toDoList>



예제 소개 (Dockerfile)

ToDo List 서버 애플리케이션 패키징
(Containerizing)

```
FROM python:3.9-slim

WORKDIR /app

# 의존성 설치 레이어 분리
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션 코드 복사
COPY . .

# 포트 노출
EXPOSE 5000

# 컨테이너 실행 명령
CMD ["python", "app.py"]
```

예제 소개 (docker-compose.yml)

```
services:
  web:
    build: ./app
    container_name: flask-app
    ports:
      - "5001:5000"
    environment:
      - DB_HOST=db
      - DB_USER=user
      - DB_PASSWORD=password
      - DB_NAME=flaskdb
    volumes:
      - ./app:/app
    depends_on:
      - db
    restart: always
    networks:
      - app-network
```

```
db:
  image: mysql:8
  container_name: mysql-db
  environment:
    - MYSQL_ROOT_PASSWORD=rootpassword
    - MYSQL_DATABASE=flaskdb
    - MYSQL_USER=user
    - MYSQL_PASSWORD=password
  volumes:
    - db_data:/var/lib/mysql
  ports:
    - "3308:3308"
  networks:
    - app-network
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-p$MY!"]
    interval: 10s
    timeout: 5s
    retries: 5

networks:
  app-network:
    driver: bridge

volumes:
  db_data:
```

알아두면 좋아요

이미지 최적화

- 다단계 빌드(multi-stage builds) 활용
- 적절한 베이스 이미지 선택 (alpine 등)
- 불필요한 패키지 제거

백엔드 개발자를 위한 팁

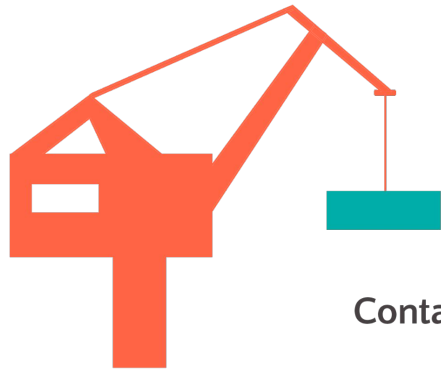
- CI/CD 파이프라인 통합
- Java 애플리케이션의 경우 Jib 같은 도구 활용
- 디버깅 전략 수립 (로그 볼륨 마운트 등)
- 테스트 자동화에 도커 활용 (TestContainers)
- 환경변수 노출 주의!! (JAR 파일조심)

```
FROM golang:1.23 AS build
WORKDIR /src
COPY <<EOF /src/main.go
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
EOF
RUN go build -o /bin/hello ./main.go

FROM scratch
COPY --from=build /bin/hello /bin/hello
CMD ["/bin/hello"]
```



Jib

Containerize your Java application.

Thank you!

연락처: ironhee8005@gmail.com

LinkedIn: <https://www.linkedin.com/in/jang-chulhee-810b78294/>

참고 자료

[Docker] 도커 컨테이너의 동작 원리 (LXC, namespace, cgroup)

Docker 공식 문서: <https://docs.docker.com/>

Docker Compose 문서: <https://docs.docker.com/compose/>

도커 치트시트:

<https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf>

