Docker+CI/CD - 5주차

[목차]

00. 5주차 오늘 배울 것
    ✅ 목표
01. Docker Volume
    volume
    실습
02. Docker Network
    Docker Network 이해
03. Docker Image 심화
    Docker hub repositories에 image push
    Dockerfile 최적화
04. Container 가상화
    Container 기술
5주차 끝

> 💡 모든 토글을 열고 닫는 단축키
> Windows :
> `Ctrl` + `alt` + `t`
> Mac :
> `⌘` + `⌥` + `t`

---

00. 5주차 오늘 배울 것

> ✔️ **Docker 고급 사용법을 다룹니다.**

✅ 목표

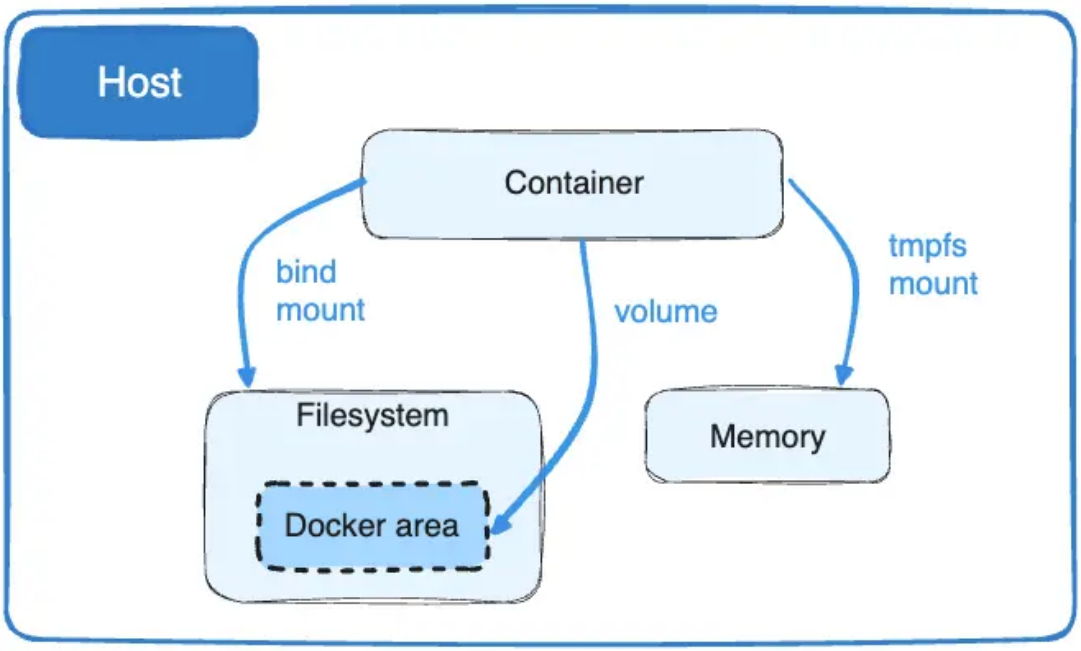
- Docker volume의 구조에 대해서 이해합니다.
- Docker network의 구조에 대해서 이해합니다.
- Docker Image 의 최적화에 대해서 이해합니다.
- Container 가상화에 대해서 이해합니다.

01. Docker Volume

✔ **Docker Container 를 실행하면서 데이터를 보존해요.**

▼ Volume 사용 이유

- **데이터가 남아있게 하려고(데이터 영속성)**: 컨테이너는 쓰고 나면 사라지는데, 볼륨을 쓰면 컨테이너가 사라져도 데이터는 그대로 남아요.
- **같은 데이터를 여러 컨테이너가 쓸 수 있게**: 볼륨 덕분에 여러 컨테이너가 같은 데이터를 함께 쓸 수 있어요. 이렇게 하면 데이터를 여기저기 복사할 필요가 없죠.
- **데이터 백업하고 옮기기 쉽게**: 볼륨을 쓰면 데이터를 백업하거나 다른 컴퓨터로 옮기기가 편해져요.
- **더 빨리 돌아가게**: 때로는 볼륨을 쓰는 게 데이터를 더 빠르게 처리할 수 있게 해줘요, 특히 많은 양의 데이터를 다룰 때 유용하죠.
- **데이터를 더 안전하게 보관하려고**: 볼륨을 사용하면 중요한 데이터를 컨테이너 밖에 따로 안전하게 보관할 수 있어요. 이렇게 하면 보안도 더 좋아지죠.
- **코드와 데이터를 따로 두려고**: 볼륨을 쓰면 코드를 바꿔도 데이터는 그대로 유지할 수 있어서, 개발이나 운영할 때 편리해요.
- Docker 공식 사이트에서는 volume, bind mount, tmpfs mount 중에 volume 을 가장 추천해요



https://docs.docker.com/storage/volumes/

volume

> ✔ **Docker Volume을 통해 데이터를 안전하게 다루어요**

▼ volume

- **볼륨이 뭔가요?** 볼륨은 컴퓨터에 따로 만들어진 폴더 같은 거예요. Docker가 이 폴더를 관리해서 우리가 쓰는 데이터를 여기에 저장해요.
- **어디에 만들어지나요?** Docker는 보통 볼륨을 컴퓨터의 특정 폴더 (대부분 `/var/lib/docker/volumes/` 라는 곳) 에 만들어요.
- **볼륨의 좋은 점은 뭐가 있나요?**
  - **백업하고 옮기기 쉬워요**: 볼륨은 데이터를 다른 곳으로 옮기거나 백업하기가 편해요.
  - **Docker 명령어로 관리해요**: 컴퓨터에 있는 Docker 프로그램 명령어나 인터넷으로 연결된 API를 써서 볼륨을 관리할 수 있어요.
  - **어디서나 잘 돌아가요**: 볼륨은 리눅스나 윈도우, 어떤 컨테이너에서도 잘 작동해요.

    - **여러 컨테이너에서 안전하게 쓸 수 있어요**: 볼륨은 여러 컨테이너가 같이 쓰기에도 안전하게 만들어져 있어요.
    - **볼륨 드라이버로 더 많은 기능을 추가할 수 있어요**: 볼륨 드라이버를 쓰면 데이터를 다른 곳에 저장하거나 보안을 강화하는 등 여러 기능을 추가할 수 있어요.
    - **Mac이나 Windows에서 더 빠르게 돌아가요**: Mac이나 Windows 컴퓨터에서 Docker를 쓸 때 볼륨이 기본 저장 방식보다 훨씬 빠르게 돌아가요.
- → 이렇게 볼륨을 쓰면 데이터를 더 잘 다루고, 여러 컨테이너에서 공유하거나, 데이터를 안전하게 관리하는데 도움이 돼요.

▼ bind mount

- **바인드 마운트는 뭐예요?** 바인드 마운트는 컴퓨터의 특정 폴더나 파일을 Docker 컨테이너 안에서 직접 쓸 수 있게 해주는 거예요. 볼륨보다 기능이 좀 덜하지만, 특정 상황에서 유용해요.
- **어떻게 사용하나요?** 컴퓨터에서 어떤 폴더나 파일을 골라서 Docker 컨테이너에 '붙여넣는' 것과 비슷해요. 그러면 컨테이너 안에서도 그 파일이나 폴더를 마치 자기 것처럼 쓸 수 있어요.
- **어디에 있는 파일이나 폴더를 쓸 수 있나요?** 바인드 마운트는 컴퓨터 안의 정확한 위치(절대 경로)에 있는 파일이나 폴더를 사용해요. 그래서 컨테이너가 어디에 있든, 그 파일이나 폴더를 똑같이 쓸 수 있죠.
- → 바인드 마운트를 사용하면 컨테이너에서 컴퓨터의 특정 파일이나 폴더를 쉽게 접근하고 사용할 수 있어요. 이런 방식은 특정 개발 작업이나 데이터를 다룰 때 도움이 될 수 있어요.

▼ tmpfs mount

- **tmpfs 마운트는 뭔가요?** tmpfs 마운트는 컴퓨터 메모리를 사용해서 일시적인 데이터를 저장하는 방법이에요. 이 방법은 컨테이너 안에서 잠깐 필요한 데이터를 다룰 때 유용해요.
- **왜 사용하나요?**
    - **일시적인 데이터 저장**: 컨테이너가 임시 데이터를 만들 때, 이걸 영구적으로 저장하지 않고 메모리에만 잠시 두고 싶을 때 써요.
    - **컴퓨터의 파일 시스템 안 씀**: tmpfs 마운트는 컴퓨터의 일반 파일 시스템 대신 메모리를 사용해서 데이터를 저장해요.
    - **성능 향상**: 메모리를 사용하므로 데이터를 빠르게 읽고 쓸 수 있어서 컨테이너의 성능이 좋아져요.

    - **컨테이너끼리 공유 안 됨**: 이 방식으로 저장된 데이터는 해당 컨테이너에서만 사용할 수 있고, 다른 컨테이너와 공유는 안 돼요.
- → tmpfs 마운트는 컨테이너가 잠깐 필요한 데이터를 메모리에 저장하고 싶을 때 사용하면 좋아요. 이렇게 하면 데이터를 빠르게 처리할 수 있고, 컨테이너가 끝나면 자동으로 데이터도 사라져서 편리해요.

실습

✔ **Docker Volume을 활용하는 실습을 해요**

▼ 실습: 데이터 영속성

- 예제

```
docker volume create datavol
docker volume ls
docker container run -ti --rm -v datavol:/data alpine


# docker container 내에서
echo "볼륨 데모" > /data/demo.txt
exit

# Host Machine에서 alpine을 이용한 컨테이너 삭제 확인
docker container ls

# 새로운 이미지와 새로운 컨테이너에서 demo.txt 파일 확인
docker container run --rm -v datavol:/data ubuntu cat /
```

- host machine 의 디렉토리 확인

```
sudo apt update; sudo apt install -y tree;
sudo tree -a /var/lib/docker/volumes/datavol
```

- inspect 명령으로 volume 확인

```
docker volume inspect datavol
[
    {
        "CreatedAt": "2023-11-25T14:33:11+09:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/datavol/
        "Name": "datavol",
        "Options": null,
        "Scope": "local"
    }
]
```

▼ 실습: 암시적 볼륨 마운트

- 사용법

```
docker run -d --name mysqltest -v /var/lib/mysql mysql:
docker inspect mysqltest # 정보가 많음
docker inspect mysqltest | jq .[].Mounts # `Name` 을 확인
docker volume ls # `Name`을 확인
```

▼ 실습: bind mount를 사용해서 디렉토리 mount #1

- 사용법

```
cd ~
mkdir test-app
cd test-app
touch run.sh
chmod +x ./run.sh
docker run -ti --rm -v .:/app alpine

# docker container 내부에서
cd /app
ls -ahlvF
# run.sh 존재 확인
```

▼ 실습: readonly와 readwrite(default) 디렉토리 마운트

- 사용법

```
cd ~
mkdir readonly
mkdir readwrite
docker run -ti -v ~/readonly:/readonly:ro -v ~/readwrit

# Docker Machine 내부에서
echo "test" > /readonly/readonly.txt # 파일을 쓸 수 없음
echo "test" > /readwrite/readwrite.txt # 파일 쓰기 가능
exit

# Host Machine에서
cat ~/readwrite/readwrite.txt
```

▼ 실습: mysql 데이터를 보존해서 실행하기

- 사용법
  - mysql 컨테이너 실행

```
docker run -ti --rm -d --name mysqltest -e MYSQL_ROO
```

  - mysql 서버에 ssh 접속 및 mysql 접속

```
docker exec -ti mysqltest /bin/bash

mysql -h localhost -u root -p
```

  - mysql 명령

```
show databases;

use mysqltest;
create table mysqltest(id int, name varchar(50));
insert into mysqltest values(1, 'testname');
select * from mysqltest;
(Ctrl+D)
```

- mysql 서버의 ssh 접속 상태에서

```
ls -ahlvF /var/lib/mysql/mysqltest
exit
```

- Host Machine 에서

```
ls -ahlvF ~/mysqldata/mysqltest
```

- 동일한 데이터를 사용해서 다른 컨테이너 실행

```
docker stop mysqltest

docker run -ti --rm -d --name mysqltest2 -e MYSQL_RO
docker logs -f mysqltest2
docker exec -ti mysqltest2 /bin/sh

mysql -h localhost -u root -p

use mysqltest;
select * from mysqltest;
```

- docker 실행 시 환경 변수 리스트를 어디에서 확인하는지?
  - https://hub.docker.com/_/mysql

02. Docker Network

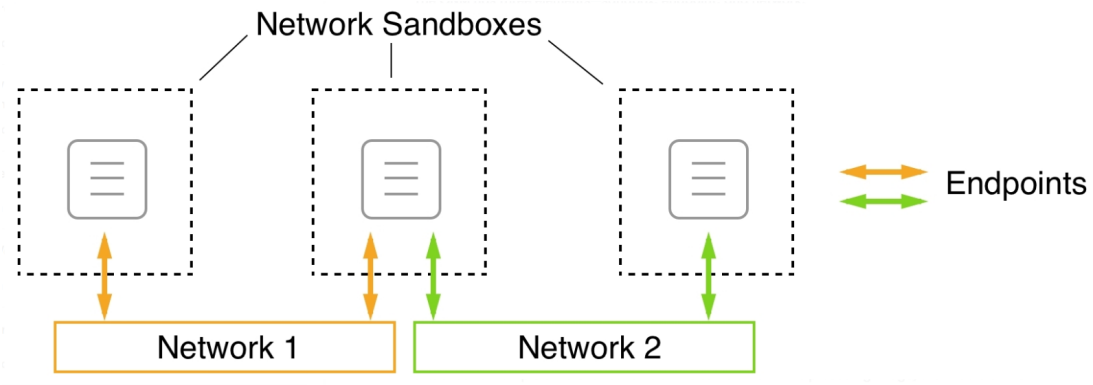
Docker Network 이해

✔️ **Docker Network에 대해 이해합니다.**

- 도커 네트워크는 도커 컨테이너끼리 서로 통신할 수 있게 도와주는 네트워크예요. 이걸 쓰면 컨테이너들이 서로 통신하는 걸 더 쉽게 할 수 있고, 보안도 더 강하게 만들 수 있어요.

▼ 개념 설명

- 실제 비즈니스 애플리케이션을 컨테이너화할 때, 여러 개의 컨테이너가 서로 협력해야 목표를 달성할 수 있어요. 그래서 각각의 컨테이너가 서로 소통할 수 있는 방법이 필요하죠. 이를 위해 컨테이너들 사이에서 데이터 패킷을 주고받을 수 있는 경로, 즉 네트워크를 설정해야 해요. 도커에서는 이런 네트워크를 쉽고 효율적으로 구축할 수 있도록 도와주는 간단한 네트워크 모델을 만들었어요. 이 모델을 컨테이너 네트워크 모델(CNM)이라고 부른답니다. 이 모델은 컨테이너 네트워크를 구현할 때 필요한 기본 요건들을 정해줘요.



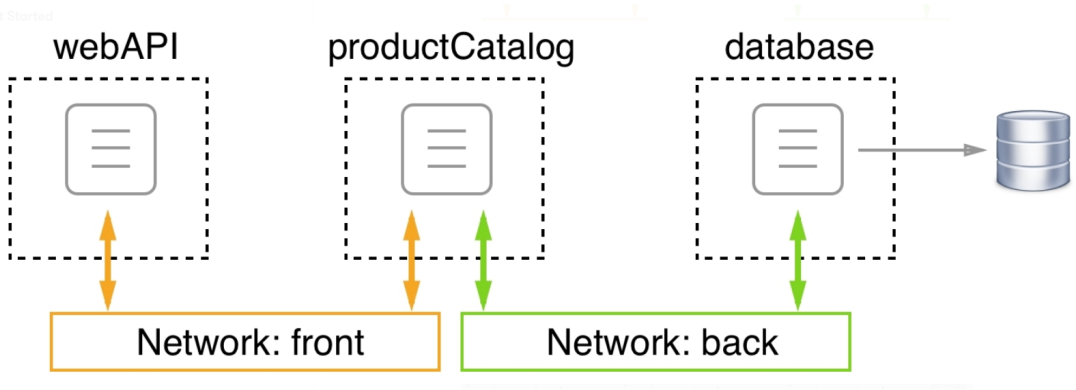
Docker Container Network Model

- **샌드박스**: 샌드박스는 컨테이너를 외부 세계로부터 완전히 분리해요. 바깥에서 들어오는 네트워크 연결은 이 샌드박스 안의 컨테이너로는 들어올 수 없죠. 그런데 컨테이너가 완전히 소통 불가능하면 큰 쓸모가 없겠죠? 그래서 여기에 엔드포인트가 필요해요.
- **엔드포인트**: 엔드포인트는 외부 세계와 샌드박스(컨테이너가 아닌) 사이의 연결점이에요. 이게 컨테이너를 안전하게 지켜주는 거죠. 엔드포인트는 샌드박스를 다음 요소인 네트워크에 연결해요.
- **네트워크**: 네트워크는 엔드포인트에서 다른 엔드포인트로, 결국은 컨테이너에서 다른 컨테이너로 데이터를 보내는 길이에요.
- 하나의 네트워크 샌드박스 안에는 여러 개의 엔드포인트가 있을 수 있어요. 즉, 하나의 샌드박스 안에 있는 컨테이너는 네트워크에 전혀 연결되지 않거나, 여러 네트워크에 동시에 연결될 수 있어요. 가령, 세 개의 샌드박스 중 하나는 엔드포인트를 통해 두 개의 네트워크에 동시에 연결된 상태일 수 있죠.
- 이 네트워크 모델은 꽤 범용적이에요. 개별 컨테이너가 네트워크에서 어디에서 작동하는지는 정하지 않아요. 모든 컨테이너가 한 호스트에서 실행될 수도 있고(로컬), 여러 호스트에 걸쳐 있을 수도 있어요(글로벌).
- 물론 CNM은 그냥 네트워크가 어떻게 작동하는지 설명하는 모델일 뿐이에요. 실제로 네트워크를 사용하려면 CNM을 구현해야 해요. 로컬이든 글로벌이든, CNM을

구현하는 여러 방법이 있답니다.

▼ 실제와 유사한 사례

- 우리에게 웹 API, 제품 카탈로그, 데이터베이스 세 개의 서비스로 이루어진 애플리케이션이 있다고 해요. 이때 우리는 웹 API가 제품 카탈로그와는 통신할 수 있지만, 데이터베이스와는 통신하지 못하게 하고 싶어요. 반면, 제품 카탈로그는 데이터베이스와도 통신할 수 있으면 좋겠죠. 이걸 해결하는 방법 중 하나는 웹 API와 데이터베이스를 서로 다른 네트워크에 두고, 제품 카탈로그는 두 네트워크에 모두 연결하는 거예요. 이렇게 하면 원하는 대로 통신 구조를 설정할 수 있어요.



여러 네트워크에 연결된 컨테이너

- 각 네트워크는 외부에서 오는 권한 없는 접근으로부터 리소스를 보호해서 더 많은 보안을 제공해요. 그래서 애플리케이션을 만들고 운영할 때 여러 네트워크를 사용하는 게 좋아요. 그리고 서로 꼭 통신해야 하는 서비스들만 같은 네트워크에 두는 거죠. 방금 본 예제에서는 웹 API가 데이터베이스와 직접 통신할 필요가 없으니까, 이 둘을 다른 네트워크에 두었어요. 이렇게 하면 만약 해커가 웹 API를 해킹해도, 제품 카탈로그 서비스를 먼저 해킹하지 않고서는 데이터베이스에 접근하기 어렵게 됩니다.
- 다음과 같이 실제 구현해 볼게요.

```
# back, front 네트워크 생성
docker network create --driver=bridge back
docker network create --driver=bridge front

# 각 서비스를 생성 및 실행
docker run --name=webapi -itd --net=front ubuntu:14.04
docker run --name=catalog -itd --net=back ubuntu:14.04
docker run --name=database -itd --net=back ubuntu:14.04
```

```
# catalog 서비스는 기본 back 네트워크 뿐만 아니라 front 네트워크
docker network connect front catalog

# webapi 의 라우팅 테이블
docker exec webapi route

# catalog 의 라우팅 테이블
docker exec catalog route

# database 의 라우팅 테이블
docker exec database route

docker network inspect front # webapi / catalog
docker network inspect back # catalog / database


docker exec -it webapi bash
# 머신 안에서
ping -c 1 catalog # 가능
ping -c 1 database # 연결 불가능
exit

#
docker exec -it catalog bash
# 머신 안에서
ping -c 1 webapi
ping -c 1 database

# 리소스 정리
docker network disconnect front catalog

docker stop webapi catalog database

docker rm webapi catalog database

docker network rm back

docker network rm front
```

▼ Docker Network 종류

1. **브리지 네트워크**: 이건 Docker 쓸 때 가장 많이 보게 되는 기본 네트워크예요. 한 컴퓨터 안에서 여러 Docker 컨테이너가 서로 통신할 수 있게 해줍니다. 컨테이너를 이 네트워크에 붙이면, Docker가 알아서 IP 주소를 줘요. 그리고 이거, 호스트 밖의 다른 네트워크랑은 일단 따로 동작해요. 근데 포트 매핑이라는 걸 사용하면, 바깥에서도 컨테이너에 접근할 수 있게 할 수 있어요.
2. **공용 네트워크**: 사실 Docker에서 '공용 네트워크'라는 말은 흔히 안 쓰이는데요, 보통 외부에서 접근할 수 있게 하려면, 브리지 네트워크에 포트를 열어주거나, 아니면 '호스트 네트워크'라고 해서 컨테이너가 직접 우리 컴퓨터의 네트워크를 쓰게 하는 방법이 있어요.
3. **사설 네트워크**: 이건 좀 특별한 컨테이너끼리만 통신할 수 있는 네트워크예요. Docker에서는 사용자가 직접 네트워크를 만들고, 그 안에 원하는 컨테이너만 연결할 수 있어요. 이렇게 하면 다른 컨테이너나 외부 네트워크와는 격리되어서, 더 안전하게 정보를 주고받을 수 있죠.

▼ Docker Network 요약

- 단일 호스트에서 실행되는 컨테이너가 서로 통신하는 방법에 대해 배웠습니다.
- 컨테이너 네트워크의 요구 사항을 정의하는 CNM을 살펴보고, 브리지 네트워크와 같은 CNM의 여러 구현을 살펴보았습니다.
- 브리지 네트워크가 어떻게 작동하는지 자세히 살펴보고, 도커가 네트워크와 해당 네트워크에 연결된 컨테이너에 대해 제공하는 정보에 대해 알아보았습니다.

03. Docker Image 심화

Docker hub repositories에 image push

✔ **Docker Image를 Docker Hub에 올리는 방법을 알아봅니다.**

▼ push: 이미지 올리기

- registry: Dockerfile 을 통해 생성한 이미지나 docker commit 을 통해 생성된 이미지를 저장하는 곳
- Public registry: 공개적으로 사용할 수 있어 아무나 접근 가능
- Private registry: 특정인이나 특정 그룹만 접근 가능

- Docker Hub: hub.docker.com은 Docker 공식 Registry
- Docker Hub 내 나의 계정의 Registry에 push하기 위해서 docker 계정 생성 및 로그인 필수
- Docker image tag → push
  - hub.docker.com 에 본인 계정의 repositories에 생성한 이미지를 업로드하기 위해서는, 본인 계정을 이미지명 앞에 붙여야 docker push 수행 시 계정으로 찾아가 저장된다.

```
docker logout
docker login

docker image pull nginx:latest
docker image pull ubuntu:22.04

docker images

docker image tag nginx:latest nbcdocker0000/nginx-test:

docker push nbcdocker0000/nginx-test:1.0

docker images
docker image rm nbcdocker0000/nginx-test:1.0
docker container rm nginx-test --force
docker image rm nginx-test:1.0 --force

docker pull nbcdocker0000/nginx-test:1.0
docker run -d -p 8001:80 --name=nginx-test nbcdocker000
```

  - push 한 이미지를 웹사이트를 통해 확인

Dockerfile 최적화

▼ **왜 Dockerfile을 최적화해야 하나요?**

- **빠르게 만들기 위해서**: Docker 이미지를 만드는 시간을 줄이면, 일이 더 빨리 끝나고 다른 일에 시간을 더 쓸 수 있어요.

- **이미지 크기 줄이기**: 이미지가 작으면 저장 공간을 덜 차지하고, 다운로드나 전송할 때도 더 빨라져요.
- **재사용성 높이기**: 잘 만들어진 Dockerfile은 여러 번, 여러 곳에서 쓸 수 있어서 편리해요.
- **보안 강화**: 보안을 잘 고려해서 만들면 해킹 같은 문제에서 더 안전해져요.
- **유지보수 쉽게**: 잘 정리된, 깔끔한 Dockerfile은 나중에 수정하거나 업데이트할 때 훨씬 쉬워요.

▼ Java 환경 image build

- Dockerfile

```
# build 는 gradle 이미지에서 `builder`라는 이름으로.
FROM gradle:8.5-jdk21-alpine AS builder

WORKDIR /app
COPY ./ ./
RUN gradle clean bootJar


# App
FROM eclipse-temurin:21-jre-alpine

WORKDIR /app
COPY --from=builder /app/build/libs/spring-boot-0.0.1-S

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "spring-boot-0.0.1-SNAPSHOT
```

  - alpine 이미지를 사용해서 최소한의 라이브러리만 설치된 이미지를 사용하여 빌드
  - multi stage를 사용하여 application 실행 Container에 gradle이 설치되는 것을 방지

04. Container 가상화

Container 기술

✔ **Docker의 근간이 되는 Container에 대해서 알아보아요.**

▼ Container 기술

- **컨테이너란 뭐에요?**
  - 컨테이너는 앱을 실행하는 데 필요한 모든 걸 담은 작은 상자 같아요. 앱을 돌리는 데 필요한 코드, 프로그래밍 언어, 라이브러리 같은 것들이 포함돼 있죠.
  - Docker에서는 'Dockerfile'이라는 걸 사용해서 이런 컨테이너를 만들어요.
- **컨테이너는 어떤 점이 좋나요?**
  - 컨테이너는 컴퓨터 자원(예를 들어 CPU나 메모리)을 여러 앱들과 나눠 쓸 수 있게 해줘요.
  - 앱을 컨테이너에 넣으면 그 컨테이너가 어디에서 실행되든 똑같이 잘 돌아가요. 이건 마치 앱을 작은 상자에 넣어서 어디든 가져갈 수 있는 것과 비슷해요.
  - 컨테이너는 서로 독립적이라서 한 컨테이너가 문제가 생겨도 다른 컨테이너에는 영향을 주지 않아요.
  -

이렇게 컨테이너 기술을 사용하면 앱을 더 안정적이고 효율적으로 관리하고 실행할 수 있어요. 컨테이너 덕분에 앱을 어디서든 쉽게 실행하고 관리할 수 있죠.

▼ Container 이점

- **경량화란?**
  - 컨테이너는 컴퓨터의 운영체제(OS)를 여러 앱과 공유해요. 그래서 각 앱마다 따로 운영체제를 설치할 필요가 없어요. 이 덕분에 컨테이너 파일은 작고 가볍습니다. 그리고 크기가 작아서 빨리 시작할 수 있어요.
- **이동성과 플랫폼 독립성이란?**
  - 컨테이너는 앱을 실행하는 데 필요한 모든 것을 함께 담아요. 그래서 한 번 만든 앱을 노트북이든, 클라우드든, 다른 컴퓨터든 재설정 없이 그대로 옮겨 실행할 수 있어요.
- **현대적인 개발 및 아키텍처 지원이란?**
  - 컨테이너는 크기가 작고 여러 플랫폼에서 잘 작동해서 최신 개발 방식과 앱 패턴에 잘 맞아요. 이게 무슨 말이냐면, 새로운 코드를 조금씩 추가하거나 업데이

트할 때 매우 유용하다는 거예요.

- **사용률 향상이란?**
  - 개발자와 운영자는 컨테이너를 사용해서 컴퓨터의 CPU나 메모리를 더 효율적으로 쓸 수 있어요. 앱의 각 부분을 따로따로 배치하거나 크기를 조절할 수 있기 때문에, 전체 앱을 확장하는 것보다 더 유연하게 관리할 수 있어요.

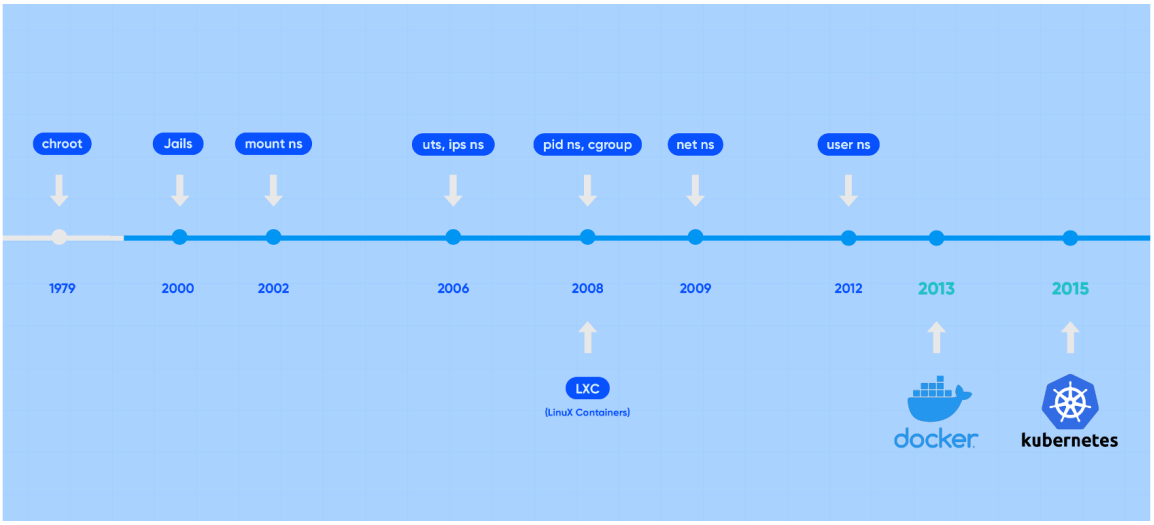
이렇게 컨테이너를 사용하면 앱을 가볍고 유연하게 만들 수 있고, 어디서든 잘 돌아가게 할 수 있어요. 현대적인 개발 방식에도 잘 맞고, 컴퓨터 자원도 더 잘 활용할 수 있죠.

▼ Container 용도

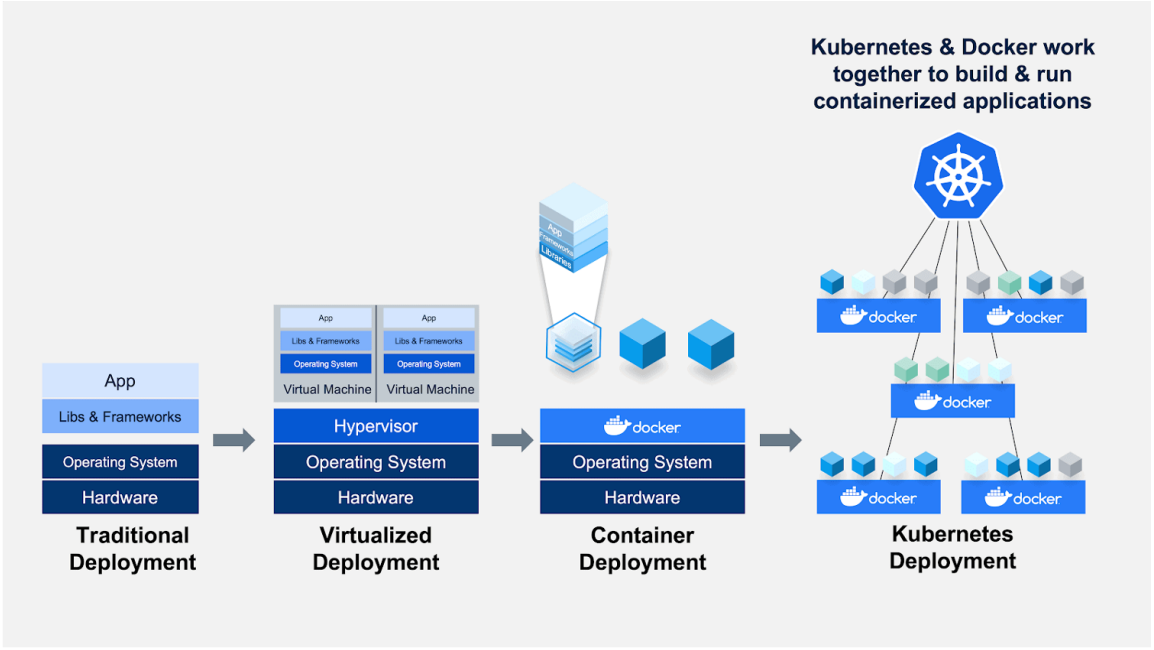
- **마이크로서비스에 좋아요**: 컨테이너는 작고 가벼워서, 앱을 여러 작은 부분으로 나누어 관리하는 마이크로서비스 아키텍처에 딱 맞아요. 각 부분이 서로 독립적으로 잘 돌아가게 할 수 있죠.
- **DevOps에 유용해요**: 소프트웨어를 만들고 운영하는 방식인 DevOps를 사용하는 팀에게 컨테이너는 아주 유용해요. 마이크로서비스 아키텍처와 잘 맞아서, 앱을 더 빠르고 효율적으로 개발하고 배포할 수 있어요.
- **하이브리드, 멀티클라우드에 이상적이에요**: 컨테이너는 어디서든 잘 돌아가요. 노트북이든, 회사 데이터 센터든, 클라우드든 상관 없어요. 이런 점 때문에 여러 클라우드 서비스를 섞어 쓰는 하이브리드나 멀티클라우드 환경에서도 잘 맞아요.
- **클라우드 마이그레이션에 도움돼요**: 많은 기업이 클라우드로 옮겨가면서 앱을 컨테이너로 바꾸는 방법을 많이 써요. 이렇게 하면 앱을 클라우드로 쉽게 옮길 수 있고, 관리도 편해져요.

이렇게 컨테이너는 마이크로서비스, DevOps, 클라우드 환경에 매우 유용해요. 앱을 더 잘 관리하고, 어디서든 잘 돌아가게 할 수 있어서, 현대적인 소프트웨어 개발에 큰 도움이 되죠.

▼ Container의 발전

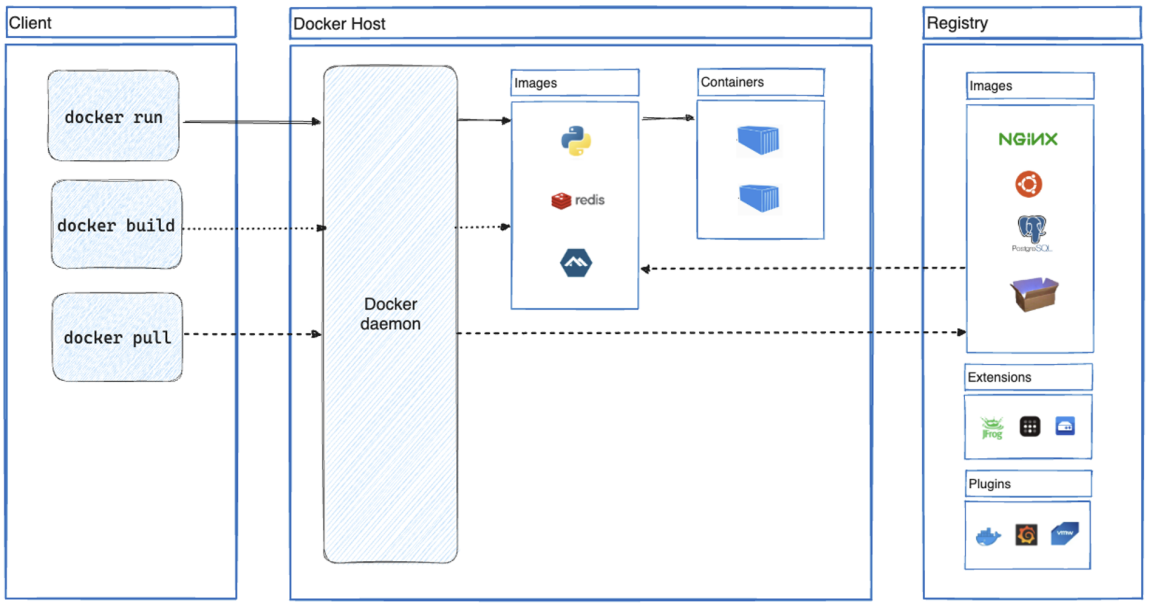


https://tech.kakaoenterprise.com/154



https://k21academy.com/docker-kubernetes/docker-and-kubernetes/

▼ Docker Architecture



https://docs.docker.com/get-started/overview/

5주차 끝

👉 전체 강의자료 보기

📕 Docker+CI/CD