7/8

도커를 쓰는 이유?

- **격리 & 추상화**

컨테이너가 하나의 머신 안에 여러 개가 있다 하더라도 자원들이 다 격리되어 있다.

이런 격리가 굉장히 중요 stand alone

os - ubuntu db -mysql,postgres couple - golang

모두가 똑같은 결과값을 얻을 수 있는 것이 좋은 점 환경이 다 달라도

파이썬 자바 버전 왔다갔다 할 때 굉장히 중요 환경을 격리한다는 것 이미지만 갈아 쓰면 되니까

서비스의 추상화해서 쓸 수 있다..

이미지

From (BASE 이미지)로부터 한층 한층 쌓이는 것
BASE는 리드 온니이다. BASE는 절대 WRITE 안된다.

이미지 표기랑 실제 디스크 사용량은 다르다

IMAGE SIZE
UBUNTU 800M
UBUNTU ADD 800M
UBUNTU APT GET 810M
UBUNTU ~ 810M
이럴 때 저걸 다 더한 것이 아닌 810M만 디스크에 저장 나머지는 할당임

CACHE를 사용하면 변경사항이 일어났을 때 동일한 LAYER를 캐시된 걸 쓰고 변경사항만 빌드하면 된다.
문제점은 똑같은 명령어를 실행했을 때 결과 값이 다를 수 있다. 심지어 버전이 같더라도 보안 업데이트로 달라지는 경우도..

이전 LAYER가 조금이라도 차이가 있으면 그 이후 LAYER부터 똑같고 명령어가 같더라도 CACHE를 쓰는 것이 아니다.

여기서부터 도출되는 주의점 DOCKERFILE은 순서가 굉장히 중요하다.

EX1)
FROM ← 여기까지만 cache 사용된다. 아래 env가 계속 바뀌기때문
ENV VERSION ← 날짜마다 바뀜
RUN APT-GET UPDATE
CMD [”BASH”]
Entrypoint

도커파일을 작성할 때
변경이 빈번하게 일어나는 걸 밑바닥에 가깝게 두는게 좋다 그래야 빌드할 때 시간이 덜 드니까
캐시를 최대한 사용하니까
반대로 변경이 덜 일어나는 것은 위쪽으로

ENV는 오버라이드하기 때문에 위에 적는다.
그래도 ENV를 매번 바꿔야하는 상황이면 아래에
LABEL은 어디든 상관 없다.
CMD랑 Entrypoint는 어처피 맨 밑바닥
나머지를 어떻게 조합하느냐에 따라 달라진다..

RUN은 Build할 때

CMD 실행 시 런타임 때 (인자) 변경 가능

Entrypoint 실행 시 런타임 때 (명령어) 변경 불가능

따로따로 쓰면 상관 없다.

패키지는 생각보다 업데이트 잘 안한다. 버전 업데이트 됐을 때 안되는 경우가 많기 때문 따라서
패키지는 미리 설치하고 실행하는 편 RUN으로

landscape에서 현재 많이 쓰이는 유명한 오픈소스들 확인 가능

여기서 참조한 뒤 공부하고 내가 나중에 쓸 때 이걸 왜 썼는지 설명 가능해야

데브옵스의 1순위 도전과제는 지속 가능성이다.

데브 : 개발 옵스 : 배포

배포는 하루에도 굉장히 많이 일어난다.
이런 반복적인 작업(배포)를 자동화
자동화에 중요한 것은 추상화

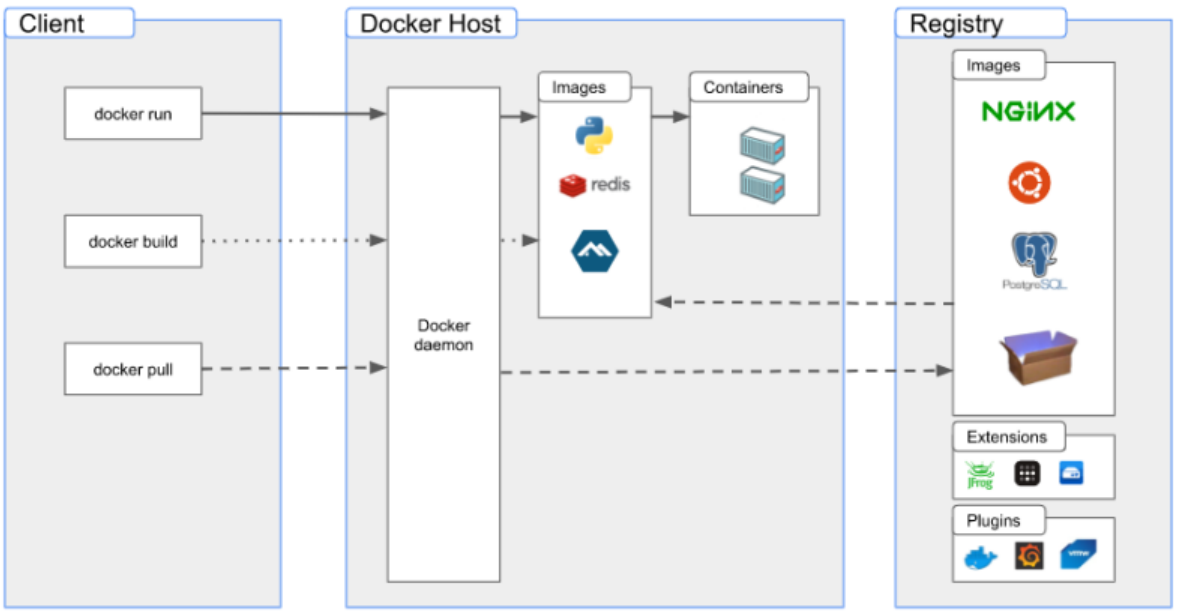
하드웨어 추상화 → vmware
os를 여러 개

서비스 추상화 → docker container
도커는 엄연히 컨테이너는 아니지만 붙여져서 자주 사용이 된다.

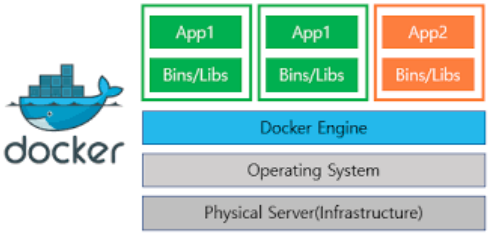
컨테이너 = service의 추상화

docker에서 중요한 것은 docker daemon

docker command를 보내면 이 데몬에서 처



레지스트리가 일종의 깃허브처럼 작동



컨테이너는 결국 layer 단위로 이루어져 있다.

모든 변경사항이 누적된다 유실되는 것이 없다. 층처럼 쌓아져서 최종 변경사항이 윗단에 위치

도커는 격리된 환경 신기술이 아님 이전에 있던 기술을 편하게 사용하게끔 만들어 낸 것

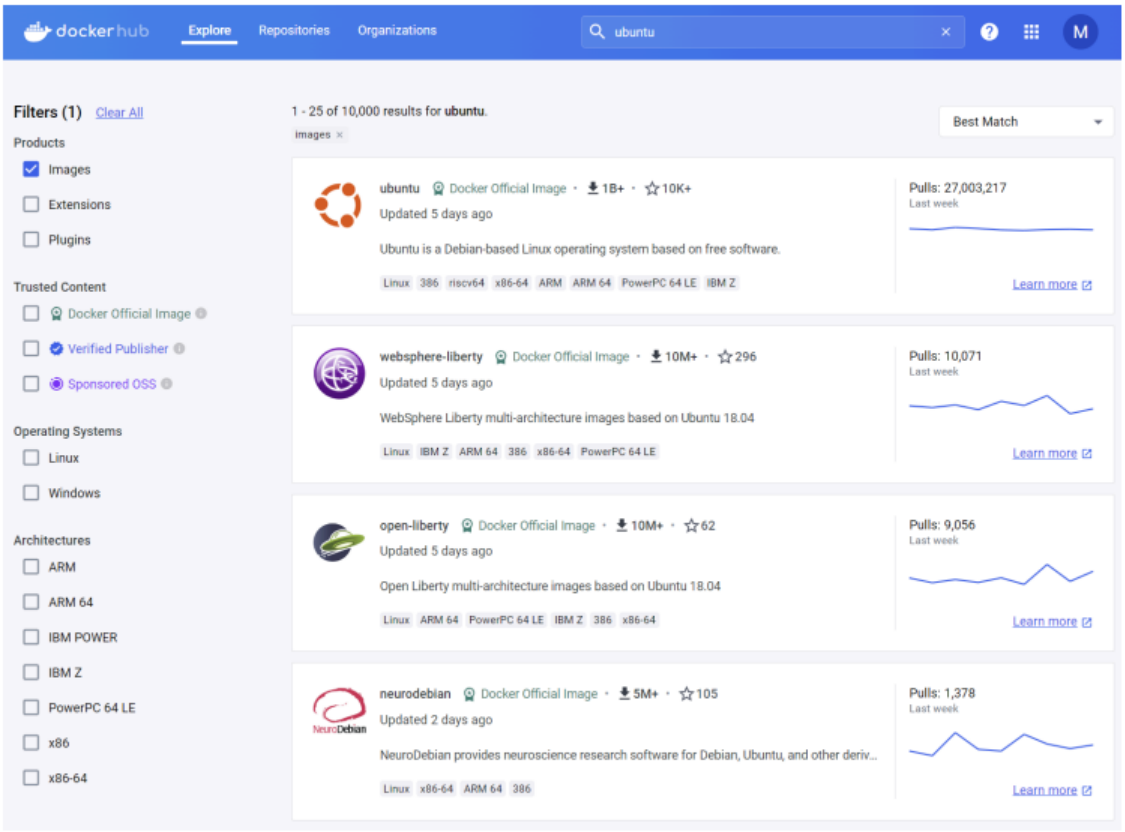
환경을 격리한다는 것은 실무에 있어 중요하다.
업무를 하다보면 쌓여있는 레거시때문에 힘들다.

컨테이너는 자바같은 언어에서 종속성 버전이란 것을 관리함에 있어 굉장히 편리

회사의 상황에 따라 도커 허브 사용 가능 불가능하다.

도커 허브를 통해 아키텍처 확인이 편리하기 때문에 도커 허브를 사용하는 것이 용이하다.

로컬에서 돌아가는 것이 ec2에서 안돌아간다. 아키텍처가 다르기 때문에



도커 허브에서 pull로 이미지 다운로드

```
docker pull [OPTIONS] REPOSITORY[:TAG|@DIGEST]
# Default
docker pull <REPOSITORY>:latest
# By Tag
docker pull <REPOSITORY>:<TAG>
# By digest
docker pull <REPOSITORY>@<DIGEST>
```

주요 옵션

Option	Short	Default	Description
`--all-tags`	`-a`		태그된 모든 이미지를 다운로드합니다.
`--platform`			이미지의 `platform`을 설정합니다.

all tags는 오류 가능성이 너무 많기에 안쓴다.

platform은 아키텍처를 확실하게 하기 위해 사용한다.

Tag & Digest

- `Tag`는 사용자가 특정 이미지에 부여한 값이므로, 같은 `Tag`라도 다운로드 시점에 따라 다른 이미지를 나타낼 수 있습니다.
  - ex) `latest` 태그
- `Digest`는 변경이 불가능한 고유의 값이며, 동일한 방식으로 생성된 이미지라면 동일한 `Digest`가 생성됩니다.

주의사항

- 동일한 `Tag`를 가진 이미지가 존재하는 경우, 이미지를 교체하지 않습니다.
- 동일한 이미지더라도 `Tag`가 다른 경우 이미지가 추가됩니다.
- 이미지마다 지원하는 `arch`가 다를 수 있습니다.
  - `arch` = arm, amd64, ...

tag는 일종의 깃허브에서 브랜치를 바꾸는 것임

다이제스트로 호출해야 변경 없이 안전하게 사용 가능 고유값이기에 충돌이 안나고 항상 동일

이미지 목록 보기

> https://docs.docker.com/engine/reference/commandline/images/

```
docker images [OPTIONS] [REPOSITORY[:TAG]]

# Alias
docker image list
docker image ls
```

로컬에 있는 이미지를 볼 때 사용하는 명령어

주요 옵션

Option	Short	Default	Description
`--all`	`-a`		중간 이미지(`intermediate image`)도 표시합니다.
`--digests`			`Digest`를 표기합니다.
`--filter`	`-f`		필터 조건에 맞는 이미지만 표시합니다.
`--no-trunc`			`sha256` 형식의 전체 이미지 ID를 표기합니다.

Option	Short	Default	Description
`--quiet`	`-q`		이미지 ID만 표기합니다.

태그 값 기준이 아닌 특정 날짜 기준으로 특정된 하나를 쓸려면 다이제스트를 사용해야..

```
$ docker images ubuntu
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       22.04     174c8c134b2a   2 weeks ago   77.9MB
ubuntu       latest    174c8c134b2a   2 weeks ago   77.9MB
```

tag는 alias이기 때문에 중복이 되는 것을 확인 가능하다.

주요 옵션

Option	Short	Default	Description
`--force`	`-f`		컨테이너가 존재하더라도 이미지를 삭제합니다.

주의사항

- 동일한 `Image ID`를 가진 이미지가 2개 이상인 경우 `Image ID`를 이용하여 삭제할 수 없습니다.
  - `-f` 옵션을 사용한 경우 모든 이미지를 삭제합니다.

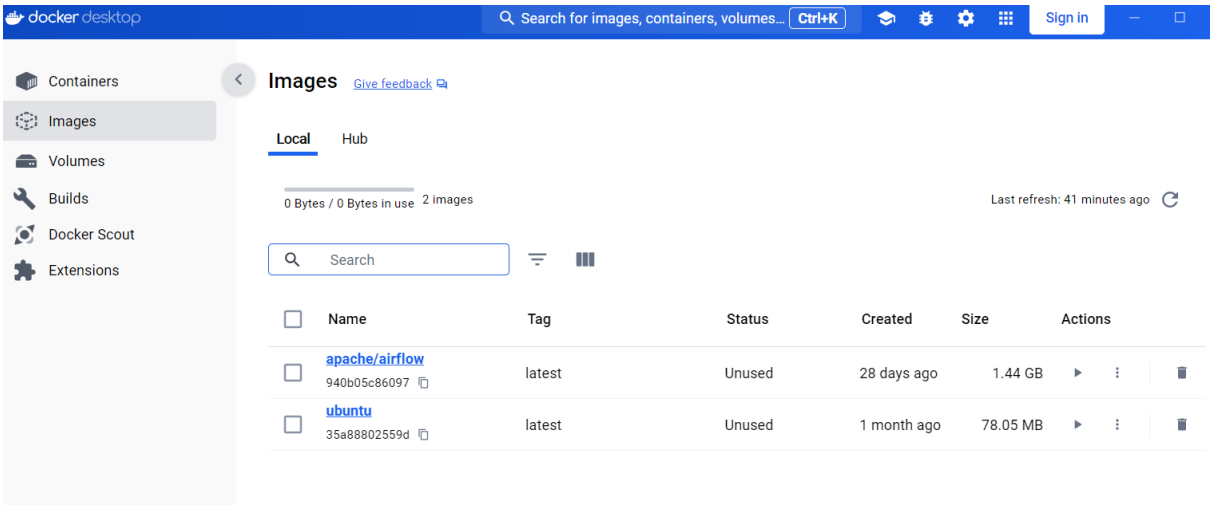
`dangling` 이미지

- `Repository`가 `none`으로 표기되는 이미지입니다.
- 일반적으로 다음과 같은 상황에서 발생합니다.
  - 동일한 이름(`Name` & `Tag`)을 가진 다른 이미지를 생성한 경우
  - `Multi-Stage` 빌드중 생성된 중간 이미지

```
$ docker images -f dangling=true
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    5b95739d14ad   About a minute ago   1.02GB
```

주요 옵션

Option	Short	Default	Description
`--all`	`-a`		`dangling` 이미지를 포함한 사용하지 않는 모든 이미지를 삭제합니다.
`--filter`			Provide filter values (e.g. `until=<timestamp>`)



gui로도 직접 삭제 가능하다.

> 💡 환경변수에 aws 키나 시크릿 키 등 개인 정보 절대 삽입 금지

탈취 시도도 굉장히 많고 직접 이미지 다운이 불가하더라도 inspect를 통해 조회가 가능하다.
실무에서도 이런 일들이 일어난다. 깃허브에 올라가는 것이 최악 커밋이 된 순간 삭제 불가능하기에
레포지토리를 다시 파야함

[예시] `ubuntu:22.04` 이미지 정보 조회하기

```
$ docker inspect ubuntu:22.04
[
    {
        "Id":
"sha256:174c8c134b2a94b5bb0b37d9a2b6ba0663d82d23ebf62bd51f74a2fd457333da",
        "RepoTags": [
            "ubuntu:22.04",
            "ubuntu:latest"
        ],
        "RepoDigests": [

 "ubuntu@sha256:6042500cf4b44023ea1894effe7890666b0c5c7871ed83a97c36c76ae560bb9b"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2023-12-12T11:38:59.637410824Z",
```

연습 문제

[연습] ARM용 Ubuntu 다운로드하기

다음 명령어를 이용하여 `ubuntu:22.04`의 `linux/arm`용 이미지를 다운로드할 수 있습니다.

```
$ docker pull ubuntu:22.04 --platform linux/arm
22.04: Pulling from library/ubuntu
Digest: sha256:6042500cf4b44023ea1894effe7890666b0c5c7871ed83a97c36c76ae560bb9b
Status: Downloaded newer image for ubuntu:22.04
docker.io/library/ubuntu:22.04
```

다운로드한 이미지의 `arch`를 `inpect`를 이용하여 확인합니다.

```
$ docker inspect ubuntu:22.04 -f "{{ .Architecture }}"
arm
```

[연습] PostgreSQL 이미지의 레이어 갯수 확인하기

> 조건
>
> - digest: `sha256:a2282ad0db623c27f03bab803975c9e3942a24e974f07142d5d69b6b8eaaf9e2`

제시된 `Digets`를 이용하여 다음과 같이 이미지를 다운로드합니다.

```
$ docker pull
postgres@sha256:a2282ad0db623c27f03bab803975c9e3942a24e974f07142d5d69b6b8eaaf9e2
```

다운로드한 이미지를 확인합니다.

```
$ docker images postgres
REPOSITORY   TAG             IMAGE ID       CREATED        SIZE
postgres     <none>          391a00ec7cac   13 days ago    425MB

# Digest도 함께 표시
$ docker images --digests postgres
```

다음과 같이 `Docker Id`를 이용하여 해당 이미지를 구성하고 있는 레이어들을 확인할 수 있습니다.

```
$ docker inspect 391a00ec7cac -f "{{range $v := .RootFS.Layers}}{{println $v}}
{{end}}"
sha256:92770f546e065c4942829b1f0d7d1f02c2eb1e6acf0d1bc08ef0bf6be4972839
sha256:94ef9904d4df70d048f10800d60a22f6df9f45fe3c4d2a12e485761b7d695892
sha256:65c0efddc8b86104633e023acee9707dfa41249636f3690d315c01c987da56fe
sha256:7e28e769eedfd948a6c6d1dd70ee5a4b0d14b4576f38bb23e64ee30d9afd4d48
sha256:7fad9b4e65a17abc549695d9529b286d97c2453cb2e43e288dcc9a31f87b01b0
sha256:b693edccc3930a496794bd45d6e0fb1d0e7a2d00c3ec72130fe944696479e379
sha256:4f0b1281c6dcf2ceca4094d9730ea0a6184894ca020f94ff43ede84742b4da03
sha256:059a984b41a09df6a5309e7928fdb61772b9db821cae4dad464b4091b126d78b
sha256:b885153181c241ae470af66c8ee62619bb667a0579f9a93fcbecaf1482bbafc3
sha256:931d2dae8d071197900c6b9fd55756e9db68d383fd863f9c1dc40e4e9545f46e
sha256:983e1162f18556cbac001e4a4ae18766a64d157ad14bfef23bdab9a836be4b63
sha256:e59308f2f1f587291d3de81d2df9893e8ee395981e5f753dc370f3b8eda44b24
sha256:39db9e416d9745d9160895485c1d22543edbcffb365bc7ececd44ceab38a9c67
```

Dockerfile ——————→ image ——————→ Container

명세서 빵틀 빵(상품)

ㅇ 레이어의 합

- 이미지는 조합의 개념
- 레이어를 얼마나 섞냐에 따라 컨테이너는 달라진다.
- 쓰기 읽기가 가능한 레이어가 붙어 최종 완성 그냥 이미지는 읽기만 가능하다.

쿠버네티스와 함께 사용시 컨테이너는 계속해서 죽고 실행되고의 반복이라 이전 log가 다 사라짐 이를 잘 알고 사용해야..

Container(일종의 Process) 컨테이너 실행시켰는데 프로세스가 하나도 없으면 자동으로 꺼진다

따라서 /bin/bash나 sleep 같은 것을 같이 날려서 process 1번으로 작동하게끔 해야한다.

이 process 1번이 꺼지거나 restart를 하면 컨테이너는 종료되고 재시작되지 않음

컨테이너를 멀티 프로세스 환경으로 만들지 않는다. 차라리 컨테이너를 여러 개 띄우면 된다.
멀티 컨테이너를 하면 됨. 단일 프로세스를 n개 띄운다.

docker run 이미지명 /bin/bash 이런 식으로 시작해야 한다.
항상 유념하고 사용해야..

1.2. Docker 컨테이너 관리

컨테이너 실행하기

> https://docs.docker.com/engine/reference/commandline/run/

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

주요 옵션

Option	Short	Default	Description
`--name`			컨테이너의 이름을 지정합니다.
`--detach`	`-d`		컨테이너를 백그라운드에서 실행합니다.
`--env`	`-e`		환경 변수를 설정합니다.
`--env-file`			환경 변수를 저장한 파일을 설정합니다.
`--expose`			포트 또는 포트 범위를 노출합니다.
`--publish`	`-p`		컨테이너의 포트를 공개합니다.
`--rm`			컨테이너가 종료되면 자동으로 삭제합니다.
`--interactive`	`-i`		`STDIN` 을 활성화합니다.
`--tty`	`-t`		`pseudo-TTY` 를 할당합니다.
`--volume`	`-v`		볼륨을 설정합니다.

- - - name

  이름을 지정해서 쓰는 경우가 많음 도커에서 자동으로 만들어지는 이름으론 분간이 힘들다.

- -d

  백그라운드에서 사용한다 실무에선 보안을 위함이기도 하고 cat같은 단순 명령어도 없는 깡통을 사용

- env-file

  환경 설정을 20개씩 하는 경우도 있는데 이때는 .envfile로 직접 주입

- rm

  test할 때 layer도 삭제해서 재생성할 때 이름 중복 문제를 조기에 방지

- -i

  콘솔 stdin 활성화

- -t
  pseudo -TTY 할당

- -v
  로그를 메인 서버에 보낼떄 볼륨을 공유한다. 볼륨용 서버를 하나 만든다

aws 인프라

vpc : 버츄얼 네트워크

그 vpc 안에 여러개의 리전을 만든다.
그 리전 안에서 또 subnet으로 퍼블릭하고 private으로 나뉜다.

퍼블릭 영역에 nat와 외부 서비스가 있다.
프라이빗이 이 퍼블릭 nat를 통해 외부와 소통

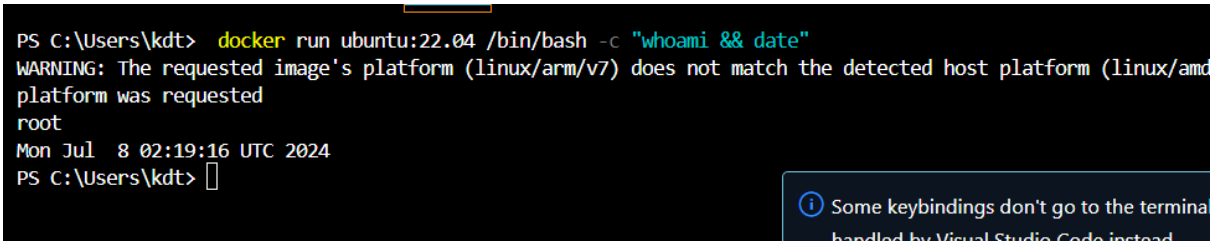
요금 때문에 모든 인프라를 한 곳에 몰빵해놓고 dr로 하나의 리전만 사용한다.
이는 실시간이 아니기에 가능한 행위

실시간이 굉장히 중요하면 어쩔 수 없이 2개 계속 사용해야..

db 같은 경우는 nat 연결이 필요 없다. 보안 그룹으로 프라이빗 내에서만 접근 가능하게 설정 가능

장애 사항 많이 나오는게 당연하다. 많이 나와도 대처하면 좋은 경험..

[예시] Ubuntu 에서 명령어 실행하기



[예시] 80 포트를 이용하여 nginx 서비스하기

```
$ docker run -d -p 80:80 nginx:latest
# Result
$ CURL localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

-p를 안주면 접근을 할 수가 없다. 반드시 해줘야

docker run -d -p 80:80 nginx:latest

앞의 port가 local host의 port 뒤의 것이 컨테이너의 port이다..

컨테이너 목록 보기

```
docker ps [OPTIONS]

# Include stopped container
docker ps -a
```

컨테이너가 돌아감을 보장한다면 어느 환경에서든 돌아간다는 것을 의미 컨테이너 단위 테스트가 중요

java 4버전을 사용하는 옛날 컴퓨터도 도커를 설치한 다음 자바 11을 사용해도 문제가 없다.

이미지의 크기가 크면 복구하는데에 있어 시간이 계속 길어지게 된다. 데브 옵스 입장에서는 이미지의 크기가 작게끔 해야 한다.

컨테이너 로그 확인하기

```
docker logs [OPTIONS] <NAME_OR_ID>
```

주요 옵션

Option	Short	Default	Description
`--follow`	`-f`		계속 로그를 표시합니다.
`--tail`	`-n`	`all`	마지막 `n` 개의 로그를 표시합니다.
`--timestamps`	`-t`		`timestamp` 를 표기합니다.
`--since`			특정 시간 이후에 발생한 로그만 표시합니다. - timestamp (e.g. `2013-01-02T13:23:37Z`) - relative (e.g. `42m` , `10s` , ...)
`--until`			특정 시간 이전에 발생한 로그만 표시합니다. - timestamp (e.g. `2013-01-02T13:23:37Z`) - relative (e.g. `42m` , `10s` , ...)

- -f

  계속해서 실행되고 있는 컨테이너면 -f로 계속 로그를 모니터링

- - -tail

  마지막 로그만 확인

[예시] `nginx`의 마지막 로그 확인하기

`nginx` 컨테이너를 실행합니다.

```
$ docker run --rm -d -p 80:80 --name nginx nginx:latest
```

다음 명령어를 통해 `nginx` 컨테이너에 API를 호출합니다.

```
$ curl 127.0.0.1:80
```

`nginx`의 마지막 1개 로그를 확인합니다.

```
$ docker logs --tail 1 nginx
172.17.0.1 - - [21/Dec/2023:15:47:34 +0000] "GET / HTTP/1.1" 200 615 "-"
"curl/8.4.0" "-"
```

```
PS C:\Users\kdt> docker run --rm -d -p 80:80 --name nginx nginx:latest
d5c66d7ec9893a86d081d7ce7ca544579d9f79c7970cf87c90425486ff55c02e
docker: Error response from daemon: driver failed programming external connectivity on endpoint nginx (455660d20012145447a7e67465b
4453023a43c03442c1d50f2eb5e77b8b53191): Bind for 0.0.0.0:80 failed: port is already allocated.
PS C:\Users\kdt> docker run --rm -d -p 80:80 --name nginx nginx:latest
43a10060dcc1ed37b588c9f539ed6ca73df902c6b750b695541c90941a2b196b
```

```
PS C:\Users\kdt> curl 127.0.0.1:80


StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE html>
                    <html>
                    <head>
                    <title>Welcome to nginx!</title>
                    <style>
                    html { color-scheme: light dark; }
                    body { width: 35em; margin: 0 auto;
                    font-family: Tahoma, Verdana, Arial, sans-serif; }
                    </style...
RawContent        : HTTP/1.1 200 OK
                    Connection: keep-alive
                    Accept-Ranges: bytes
                    Content-Length: 615
                    Content-Type: text/html
                    Date: Mon, 08 Jul 2024 02:40:06 GMT
                    ETag: "6655da96-267"
                    Last-Modified: Tue, 28 May 2024 ...
Forms             : {}
Headers           : {[Connection, keep-alive], [Accept-Ranges, bytes], [Content-Length, 615], [Content-Type, text/html]...}
Images            : {}
InputFields       : {}
Links             : {@{innerHTML=nginx.org; innerText=nginx.org; outerHTML=<A href="http://nginx.org/">nginx.org</A>; outerText=n
```

```
PS C:\Users\kdt> docker logs --tail 1 nginx
172.17.0.1 - - [08/Jul/2024:02:40:06 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Windows NT; Windows NT 10.0; ko-KR) Windows
PowerShell/5.1.22621.3672" "-"
```

[예시] 특정 기간의 로그 확인하기

다음과 같이 매초마다 날짜를 출력하는 컨테이너를 실행합니다.

```
$ docker run --name clock -d busybox sh -c "while true; do $(echo date); sleep 1;
done"
```

현재 시간으로부터 `20초` 전부터 `10초` 전까지 발생한 로그를 다음과 같이 확인합니다.

```
$ docker logs -f --since 20s --until 10s clock && echo current time is %time%
Fri Dec 29 18:44:37 UTC 2023
Fri Dec 29 18:44:38 UTC 2023
Fri Dec 29 18:44:39 UTC 2023
Fri Dec 29 18:44:40 UTC 2023
Fri Dec 29 18:44:41 UTC 2023
Fri Dec 29 18:44:42 UTC 2023
Fri Dec 29 18:44:43 UTC 2023
Fri Dec 29 18:44:44 UTC 2023
Fri Dec 29 18:44:45 UTC 2023
Fri Dec 29 18:44:46 UTC 2023
current time is  3:44:57.61
```

```
PS C:\Users\kdt>  docker run --name clock -d busybox sh -c "while true; do $(echo date); sleep 1; done"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Status: Downloaded newer image for busybox:latest
88a0a8849df3e518cc8fac3c112d8ba29d9bbf9a2155096a789e3eb142d3057b
```

```
PS C:\Users\kdt>  docker logs -f --since 20s --until 10s clock
Mon Jul  8 02:44:26 UTC 2024
Mon Jul  8 02:44:27 UTC 2024
Mon Jul  8 02:44:28 UTC 2024
Mon Jul  8 02:44:29 UTC 2024
Mon Jul  8 02:44:30 UTC 2024
Mon Jul  8 02:44:31 UTC 2024
Mon Jul  8 02:44:32 UTC 2024
Mon Jul  8 02:44:33 UTC 2024
Mon Jul  8 02:44:34 UTC 2024
Mon Jul  8 02:44:35 UTC 2024
```

echo 명령어가 windows 환경에서는 동작하지 않는 것처럼 보임..

컨테이너 접속하기

Attach

> https://docs.docker.com/engine/reference/commandline/attach/

```
docker attach [OPTIONS] CONTAINER
```

현재 실행중인 컨테이너의 터미널의 `STDIN`, `STDOUT`, `STDERR`에 접근합니다.

주의사항

- `Ctrl+C`를 통해 프로세스를 종료하게 되면 컨테이너도 종료됩니다.
- 컨테이너를 종료하지 않고 접속을 종료하기 위해서는 `Ctrl + P + Q`를 사용해야합니다.

Exec

> https://docs.docker.com/engine/reference/commandline/exec/

```
docker exec -it [OPTIONS] <ID_OR_NAME> /bin/bash|sh|...
```

현재 실행중인 컨테이너에서 새로운 명령어를 실행합니다.

주의사항

- 메인 프로세스가 실행되는 동안에만 명령어를 실행할 수 있습니다.
- `exec`로 실행한 명령은 컨테이너 재실행(`start`)시 실행되지 않습니다.
- 일부 컨테이너는 `bash`, `sh`와 같은 `shell`이 없을 수 있습니다.
  - ex) `Scracth` 이미지, ...
- `-it` 옵션이 없다면 원격으로 명령어를 실행하고 프로세스가 종료됩니다.

```
# `-it` 옵션이 없는 경우
$ docker exec clock sh -c "echo $$"
3031
$ docker exec clock sh -c "echo $$"
3039
$ docker exec clock sh -c "echo $$"
3047

# `-it` 옵션을 사용한 경우
$ docker exec -it clock sh
root@963c4fdec415:/ # echo $$
3221
root@963c4fdec415:/ # echo $$
3221
```

`Attach`와 `Exec` 차이점

	Attach	Exec
접근 터미널	컨테이너의 `STDIN`, `STDOUT`, `STDERR`	새로운 process의 Terminal
접속 종료 방법	`Ctrl + p + q`	`Ctrl + c`
PID	1 (Main Process)	!= 1

[예시] `Attach`와 `Exec`를 각각 사용하여 `PID` 출력하기

다음 명령어를 사용하여 `ubuntu` 컨테이너를 실행합니다.

```
$ docker run --rm -it -d --name server ubuntu:22.04
```

Attach 사용한 경우

`PID` 출력 시 메인 프로세스인 `1`이 출력됩니다.

```
$ docker attach server
root@89834e856287:/# ps -aux
root@89834e856287:/# echo $$
1
```

Exec 사용한 경우

`PID` 출력 시 `1`이 아닌 다른 값이 출력되는 것을 볼 수 있습니다.

```
$ docker exec -it server /bin/bash
root@89834e856287:/# echo $$
21
```

```
PS C:\Users\kdt>  docker attach server
root@4d9f58540919:/# ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.4  0.0 4331320 11944 pts/0   Ssl  03:06   0:00 /usr/bin/qemu-arm /bin/bash /bin/bash
root        14  0.0  0.0 4331044 9104 ?        Rl+  03:07   0:00 ps -aux
root@4d9f58540919:/# echo $$
1
```

```
PS C:\Users\kdt> docker exec -it server /bin/bash
root@e84c2abbe48c:/# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4628  3752 pts/0    Ss+  03:09   0:00 /bin/bash
root           9  0.0  0.0   4628  3692 pts/1    Ss   03:09   0:00 /bin/bash
root          17  0.0  0.0   7064  1608 pts/1    R+   03:10   0:00 ps -aux
root@e84c2abbe48c:/# echo $$
9
```

대부분 attach를 사용하지 않고 exec를 사용한다. attach 안쓴다고 봐도 무방

컨테이너 명령어 실행하기

> https://docs.docker.com/engine/reference/commandline/exec/

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

주의사항

- 명령어는 `Working Directory`를 기준으로 실행되므로 파일 사용 시 주의가 필요합니다.

```
$ docker exec clock sh -c "pwd"
/

$ docker exec --workdir /bin clock sh -c "pwd"
/bin
```

```
root@e84c2abbe48c:/# exit
exit
PS C:\Users\kdt>  docker exec clock sh -c "pwd"
/
PS C:\Users\kdt> docker exec --workdir /bin clock sh -c "pwd"
/bin
PS C:\Users\kdt>
```

명령어 실행시 exec를 통해서 실행하는 경우가 대부분

컨테이너 종료 & 시작

> https://docs.docker.com/engine/reference/commandline/stop/
>
> https://docs.docker.com/engine/reference/commandline/start/

```
# 종료
docker stop <NAME_OR_ID>
# 시작
docker start <NAME_OR_ID>
```

주의사항

- `Dockerfile`을 통해 실행되지 않은 프로세스들은 종료 후 재시작할 경우 살아나지 않습니다.

```
$ docker run -d --name server ubuntu
$ docker exec -d server /bin/bash -c "while true; do $(echo date); sleep 1; done"
$ docker exec server ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
```

```
root         1     0  0 16:57 pts/0    00:00:00 /bin/bash
root         9     0  0 17:05 pts/1    00:00:00 /bin/sh
root       225     0  0 17:08 ?        00:00:00 /bin/bash -c while true; do
$(echo date); sleep 1; done

$ docker stop server
$ docker start server
$ docker exec server ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 17:11 pts/0    00:00:00 /bin/bash
root         9     0  0 17:11 ?        00:00:00 ps -ef
```

컨테이너 중지 & 재실행

> https://docs.docker.com/engine/reference/commandline/pause/
>
> https://docs.docker.com/engine/reference/commandline/unpause/

```
# 중지
docker pause <NAME_OR_ID>
# 재실행
docker unpause <NAME_OR_ID>
```

주의사항

- 재실행하는 경우 정지되어 있던 `모든` 프로세스가 재실행됩니다.

```
$ docker run -d --name server ubuntu
$ docker exec -d server /bin/bash -c "while true; do $(echo date); sleep 1; done"
$ docker exec server ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 16:57 pts/0    00:00:00 /bin/bash
root         9     0  0 17:05 pts/1    00:00:00 /bin/sh
root       225     0  0 17:08 ?        00:00:00 /bin/bash -c while true; do
$(echo date); sleep 1; done

$ docker pause server
$ docker unpause server
$ docker exec server ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 16:57 pts/0    00:00:00 /bin/bash
root         9     0  0 17:05 pts/1    00:00:00 /bin/sh
root       225     0  0 17:08 ?        00:00:00 /bin/bash -c while true; do
$(echo date); sleep 1; done
```

pause랑 unpause는 거의 사용안한다. 하지만 start와 구분은 할 줄 알아야.

컨테이너 삭제

> https://docs.docker.com/engine/reference/commandline/rm/

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

주요 옵션

Option	Short	Default	Description
`--force`	`-f`		`SIGKILL` 시그널을 사용하여 실행중인 컨테이너를 삭제합니다.
`--volumes`	`-v`		컨테이너의 `anonymous volumes`도 함께 삭제합니다.

Tip

종료된 모든 컨테이너를 삭제하려면 다음과 같이 사용하면 됩니다.

```
$ docker rm $(docker ps -a -q -f status=exited)
```

gui 사용하는게 더 편하다.

컨테이너 리소스 사용량 조회

> https://docs.docker.com/engine/reference/commandline/stats/

```
docker stats [OPTIONS] [CONTAINER...]
```

주요 옵션

Option	Short	Default	Description
`--all`	`-a`		모든 컨테이너를 표시합니다.
`--no-stream`			결과를 지속적으로 업데이트하지 않습니다.

[예시] `ubuntu` 컨테이너 리소스 사용량 조회하기

`ubuntu:22.04`를 이용하여 `server`를 실행합니다.

```
$ docker run --rm -it -d --name server ubuntu:22.04
```

`server`가 사용중인 리소스는 다음과 같이 확인할 수 있습니다.

```
$ docker stats --no-stream server
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O
BLOCK I/O   PIDS
5c2817a20a00   server    0.00%     896KiB / 15.56GiB   0.01%     586B / 0B   0B /
0B     1
```

```
PS C:\Users\kdt>  docker run --rm -it -d --name server ubuntu:22.04
4793324490fa5b2c064d7814ab81f8a9cb52400ee53cb99b24619176d762ed2e
PS C:\Users\kdt>  docker stats --no-stream server
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O
BLOCK I/O   PIDS
4793324490fa   server    0.00%     876KiB / 15.47GiB   0.01%     746B / 0B
0B / 0B     1
PS C:\Users\kdt>
```

연습 문제

[연습] Ubuntu 실행 및 Package 업데이트

> 조건
>
> - Ubuntu 이미지 Tag = 22.04
> - Container 이름 = server
> - 업데이트 이후에도 컨테이너는 실행중이어야 합니다.

Ubuntu 실행하기

```
$ docker run -it -d --name server ubuntu:22.04
89834e856287c0875da2a9dddc5aa2905a8ad601f9348e014855badc9b345ccc
```

Package 업데이트하기

방법 1 - Exec 사용하기

- Interactive 모드

```
$ docker exec -it server /bin/bash
root@89834e856287:/# apt-get update && apt-get ugrade
...
```

- Non-Interactive

```
# Single Command
$ docker exec server apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
...
$ docker exec server apt-get upgrade
Reading package lists...
Building dependency tree...
...
```

```
# Or Multiple Command
$ docker exec server /bin/bash -c "apt-get update && apt-get upgrade"
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
...
Reading package lists...
Building dependency tree...
...
```

[실습] PostgreSQL DB에 Table 생성하기

> 조건
>
> - `postgres:16.1-bullseye` 이미지를 사용합니다.
> - 컨테이너의 이름은 `psql_db`로 설정합니다.
> - Container 생성시 다음 환경변수가 설정되어야 합니다.
>   - POSTGRES_PASSWORD

1. `PostgreSQL` DB 컨테이너를 실행하세요
2. `exec`를 사용하여 컨테이너 터미널에 접근하세요
3. 터미널에서 `psql -U postgres`를 입력하여 `PostgreSQL`에 접속하세요
4. 다음 `Query`를 실행하세요

```
CREATE TABLE IF NOT EXISTS cloud_wave (
    id SERIAL PRIMARY KEY,
    timestamp timestamp
);
```

5. `\dt`를 실행하여 다음과 같이 생성한 테이블이 보이는지 확인하세요

```
postgres=# \dt
              List of relations
 Schema |        Name        | Type  |  Owner
--------+--------------------+-------+----------
 public |     cloud_wave     | table | postgres
(1 row)
```

```
PS C:\Users\kdt> docker run -e POSTGRES_PASSWORD=1234 -it -d --name psql_db p
ostgres:16.1-bullseye
```

💡 /bin/bash로 접근하면 오버라이드해버린다. db 명령어를 따라서 안붙이고 해야..

password를 환경 설정 해주고 name을 설정해준다. -d로 백그라운드 실행

15	RUN /bin/sh -c set -eux; dpkg-divert --add ...	9.93 KB	
16	RUN /bin/sh -c mkdir -p /var/run/postgres...	128 B	
17	ENV PGDATA=/var/lib/postgresql/data	0 B	
18	RUN /bin/sh -c mkdir -p "$PGDATA" && ch...	170 B	
19	VOLUME [/var/lib/postgresql/data]	0 B	
20	COPY docker-entrypoint.sh docker-ensure...	5.42 KB	
21	RUN /bin/sh -c ln -sT docker-ensure-initdb...	185 B	
22	ENTRYPOINT ["docker-entrypoint.sh"]	0 B	
23	STOPSIGNAL SIGINT	0 B	
24	EXPOSE map[5432/tcp:{}]	0 B	
25	CMD ["postgres"]	0 B	

25의 postgres 부분을 /bin/bash가 오버라이드 해버리는 것

> -it를 붙이고 실행해야 꺼지지 않는다.

```
PS C:\Users\kdt>  docker exec -it psql_db  /bin/bash
```

```
root@01c15bfce522:/# psql -U postgres
psql (16.1 (Debian 16.1-1.pgdg110+1))
Type "help" for help.
```

```
postgres=# CREATE TABLE IF NOT EXISTS cloud_wave (
id SERIAL PRIMARY KEY,
timestamp timestamp
);
CREATE TABLE
postgres=# \dt
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | cloud_wave | table | postgres
(1 row)
```

DOCKER 이미지 생성하기

```
# docker build [OPTIONS] PATH | URL | - [-f <PATH_TO_FILE>]
$ docker build . [-f <PATH_TO_FILE>]

# or
$ docker buildx build [OPTIONS] PATH | URL | - [-f <PATH_TO_FILE>]
```

주요 옵션

Option	Short	Default	Description
`--build-arg`			`ARG`를 설정합니다.
`--file`	`-f`		`Dockerfile`의 경로를 지정합니다.
`--label`			라벨을 추가합니다.
`--no-cache`			이미지 빌드시 캐시를 사용하지 않습니다.
`--platform`			`platform`을 지정합니다.
`--pull`			관련된 이미지를 저장 유무에 관계없이 `pull` 합니다.
`--tag`	`-t`		이름과 `Tag`를 설정합니다.

💡 docker build 파일 주소가 실행 명령 .은 현재 폴더를 의미

- - - no - cache

  패키지 업데이트를 해야 하는데 cache를 사용하면 이전에 패키지 업데이트했던 명령어와 같다고 판단해서 문제가 발생 해당 상황에는 캐시를 사용하지 말아야

- - - platform

  플랫폼을 명시해야 할 때 사용

- - -pull

  이미지를 저장 유무에 관련없이 pull 옛날에 받은게 있는데 또 받아야 할 경우 이 옵션을 주지 않으면 무시 따라서 이렇게 호출해야..

Dockerfile

Instruction	Description
FROM	사용할 Base 이미지를 설정합니다.
ARG	build 에 사용할 변수를 설정합니다.
ENV	환경 변수를 설정합니다.
ADD	파일 또는 폴더(directory)를 추가합니다.
COPY	파일 또는 폴더(directory)를 복사합니다.
LABEL	라벨을 추가합니다.
EXPOSE	포트를 외부에 노출합니다.
USER	User 와 Group 을 설정합니다.
WORKDIR	Working Directory 를 변경합니다.
RUN	명령어를 실행합니다.
CMD	기본 명령어를 정의합니다.
ENTRYPOINT	필수로 실행할 명령어를 정의합니다.

주의사항

- -f 를 통해 사용할 Dockerfile 을 명시하지 않는 경우, 파일 이름이 Dockerfile 인 파일이 사용됩니다.
- Dockerfile 내 모든 상대 경로들은 build 시 입력된 Path 를 기준으로 계산됩니다.
- cache 를 사용할 경우, apt-get 을 통해 설치한 패키지 버전이 최신이 아닐 수 있습니다.

- add

  링크에 있는 파일 다운로드 받아서 넣어줄 수 있다.

- copy

  로컬에 있는 파일만 가능 귀찮을땐 다 add만 써도 된다.

- run
  항상 빌드타임에 실행될 때
  cmd나 entrypoint는 빌드해서 실행될 때

- WORKDIR
  워킹 디렉토리 명시

- Entrypoint
  일반적인 환경에서 오버라이드 안된다.

run과 cmd 엔트리포인트 구분 반드시 해야

https://velog.io/@inhalin/ENTRYPOINT-CMD-차이

- RUN
  이미지를 빌드하는 과정에서 실행

- CMD, Entrypoint
  컨테이너를 실행하는 순간 실행
  CMD는 도커 RUN 할 때 변경이 가능 오버라이드가 가능
  Entrypoint는 변경이 불가능 오버라이드가 불가능

우분투 베이스	CMD	Entrypoint	결과
	echo hello	x	hello
	x	echo world	world
	echo hello	echo world	world echo hello
	hello world	echo	hello world
	Info	Docker	Docker Info가 실행
	60	sleep	sleep 60이 실행

항상 엔트리 포인트가 1순위 따라서 echo라는 entrypoint 커맨드 뒤에 echo hello가 붙는 것

두 개를 같이 사용하면 cmd는 인자 Entrypoint는 커맨드이다.

배포만큼 중요한 것이 롤백이다. 롤백이 바로바로 되어야한다. 오류 없이 한번에 배포되는 경우는 거의 없기에

많이 사용하는 `base` 이미지

- Scratch 이미지

> https://hub.docker.com/_/scratch

`binary`를 실행하기 위한 최소한의 이미지입니다.
`Dockerfile`에서 사용했을 시, 별도의 `layer`가 추가되지 않습니다.

cat 같은 명령어도 없다 bin 파일만 실행

창문과 문이 없는 집

- Alpine linux

> https://hub.docker.com/_/alpine

`Alpine` 리눅스는 용량이 5Mb 이하로 매우 가벼우며 보안성이 뛰어납니다.
많은 이미지들이 `alpine`을 기반으로한 경량화된 이미지를 함께 제공합니다.

뺄만한 거 다 뺀 리눅스

창문과 문이 하나만 있는 집

쿠버네티스 배포할 때는 수십개의 이미지도 사용하는데 이미지는 최대한 가벼운 것이 좋다

- Distroless 이미

> https://github.com/GoogleContainerTools/distroless

`Distroless` 이미지는 애플리케이션 실행에 필요한 런타임 종속성만 포함되어있습니다.
`bash` 와 같은 셸도 포함하고 있지 않아 보안성이 매우 뛰어납니다.

순수 운영을 위한 보안적인 이미지

이렇게 빡세게 안하면 그냥 우분투 쓰는 경우도 있다.

Multi-stage builds

> https://docs.docker.com/build/building/multi-stage/

일반적으로 어플리케이션을 실행하기 위한 환경(`runtime dependencies`)과 빌드용 환경(`build-time dependencies`)은 같지 않습니다. 따라서 `Production` 환경에서 보안성 증대 및 이미지 경량화등을 위해 `Multi-stage build`를 활용하여 어플리케이션 코드를 `build`한 이후, `runtime`을 위한 이미지로 옮겨 사용하는 경우가 많습니다.

from이 2개 이상 들어간 것.

jvm이나 c를 컴파일 하고 싶다. 컴파일을 하고 나온 bin 파일을 실행할 때 gcc는 필요하지 않다. 런타임할 때와 빌드 할 때의 종속성을 다르게 하겠다.

사용 예시

- 보안 강화 및 이미지 경량화를 위해 빌드용 이미지와 배포용 이미지를 따로 사용하는 경우

```
# 'golang:1.19-alpine' 이미지를 이용하여 Go Applicatoin을 컴파일 합니다.
FROM golang:1.19-alpine as build
WORKDIR /app
COPY src ./
RUN CGO_ENABLED=0 go build -o main

---
# 위에서 컴파일한 소스코드만 'Scratch' 이미지로 가져옵니다.
FROM scratch as release
COPY --from=build /app/main /app/
WORKDIR /app
CMD ["/app/main"]
```

- 필요한 `Package` 또는 파일을 다른 이미지에서 가져오는 경우

```
# 'nginx:latest' 이미지에서 'nginx.conf' 파일을 가져옵니다.
COPY --from=nginx:latest /etc/nginx/nginx.conf /nginx.conf
```

연습문제

[연습] Go 서버 이미지 제작하기

> 다음 3개 이미지를 기반으로 이미지를 빌드합니다.
>
> - Ubuntu
> - Alpine
> - Scratch

전체적인 폴더 구조는 다음과 같습니다.

```
.
├── src
|   ├── go.mod
|   └── main.go
└── Dockerfile
```

go.mod

```
module example/hello
go 1.19
```

main.go

```
package main

import (
    "io"
    "net/http"
    "log"
)

func HelloServer(w http.ResponseWriter, req *http.Request) {
    io.WriteString(w, "Hello, Worlds!\n")
}

func main() {
    http.HandleFunc("/", HelloServer)
    log.Fatal(http.ListenAndServe(":80", nil))
}
```

Debian (bullseye)을 이용한 Dockerfile

```
FROM golang:1.19-bullseye

WORKDIR /app

COPY src ./

RUN CGO_ENABLED=0 go build -o main

CMD ["/app/main"]
```

Alpine을 이용한 Dockerfile

```
FROM golang:1.19-alpine

WORKDIR /app

COPY src ./

RUN CGO_ENABLED=0 go build -o main

CMD ["/app/main"]
```

Scratch를 이용한 Dockerfile

```
FROM golang:1.19-alpine as build
WORKDIR /app
COPY src ./
RUN CGO_ENABLED=0 go build -o main

FROM scratch as release
COPY --from=build /app/main /app/
WORKDIR /app
CMD ["/app/main"]
```

`Dockerfile`들을 이용하여 이미지를 생성합니다.

```
$ docker build -t go:<TAG_NAME> .
```

생성된 이미지는 다음과 같이 실행하여 확인해볼 수 있습니다.

```
$ docker run -d -p 80:80 go:scratch
$ curl localhost:80
Hello, Worlds!
```

각 이미지별 크기는 다음과 같이 확인할 수 있습니다.

```
$ docker build -t go:<TAG_NAME> .
$ docker images go
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
go           debian    7b5f70dd7edf   About a minute ago   1.02GB
go           alpine    ac6365a46643   6 minutes ago        378MB
go           scratch   97bcb5fd82c5   26 seconds ago       6.47MB
```

```
C:\programming> curl localhost:80
Hello, Worlds!

C:\programming> curl localhost:81
Hello, Worlds!

C:\programming> curl localhost:82
Hello, Worlds!
```

```
PS C:\programming>  docker images go
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
go           scratch   f7d2ff6cc0c8   2 seconds ago        6.47MB
go           alphine   cefc5085cfdd   About a minute ago   378MB
go           debian    8c1376264322   4 minutes ago        1.02GB
PS C:\programming>  docker run -d -p 80:80 go:scratch
fe877103d0c270b18d5d74242804684cb7d7af612153e214500bcc28f16d0508
PS C:\programming>  docker run -d -p 81:80 go:alphine
6daff2f2b42c42914c9392db2b9c0e2382f92220a3b62033ddabf8545cb2596a
PS C:\programming>  docker run -d -p 82:80 go:debian
4ed7779f7d0cba63f19095d8febd3e83eb8840de9f38dedf7f0012bd159f044b
PS C:\programming>
```

빌드할 때 -f 옵션으로 dockerfile이 아닌 다른 이름으로도 빌드가 가능하다.

dockerfile은 default일 뿐이지 반드시 이걸로 해야하는 것은 아니다.

빌드할 때 .이 아닌 절대 경로를 적어도 괜찮다

[연습] 실습용 `ubuntu` 이미지 제작하기

파일 구조는 다음과 같습니다.

```
.
├── install_docker_engine.sh
└── Dockerfile
```

install_docker_engine.sh

```
apt-get update && apt-get upgrade
apt-get install -y ca-certificates curl gnupg
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
apt-get update && apt-get upgrade
```

Dockerfile

```
FROM ubuntu:22.04

RUN mkdir -p /scripts
COPY install_docker_engine.sh /scripts

WORKDIR /scripts

RUN chmod +x install_docker_engine.sh
RUN ./install_docker_engine.sh

RUN apt-get install -y docker-ce docker-ce-cli
```

다음 명령어를 통해서 `cloudwave:base.v1` 이미지를 빌드합니다.

```
$ docker build -t cloudwave:base.v1 .
```

정상적으로 `docker`가 설치되었다면 다음과 같은 결과를 얻을 수 있습니다.

```
$ docker run cloudwave:base.v1 docker version
Client: Docker Engine - Community
 Version:           24.0.7
 API version:       1.43
 Go version:        go1.20.10
 Git commit:        afdd53b
 Built:             Thu Oct 26 09:07:41 2023
 OS/Arch:           linux/amd64
 Context:           default
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker
daemon running?
```

install docker script.sh 수정 버전

```
apt-get update
apt-get -y install ca-certificates curl
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docke
chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
apt-get install -y docker-ce docker-ce-cli
```

```
PS C:\programming\test2>  docker run cloudwave:base.v1 docker version
Client: Docker Engine - Community
 Version:           27.0.3
 API version:       1.46
 Go version:        go1.21.11
 Git commit:        7d4bcd8
 Built:             Sat Jun 29 00:02:33 2024
 OS/Arch:           linux/amd64
 Context:           default
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
```

add로 수정한 코드

```
FROM ubuntu:22.04
RUN mkdir -p /scripts
# COPY install_docker_engine.sh /scripts
ADD https://raw.githubusercontent.com/matenduel/cloudwave/main/docker-practice/doc
WORKDIR /scripts
# RUN chmod +x install_docker_engine.sh
RUN chmod +x ./docker_install_script.sh
# RUN ./install_docker_engine.sh
RUN ./docker_install_script.sh
RUN apt-get install -y docker-ce docker-ce-cli
```

Docker 이미지 업로드 하기

1.4. Docker 이미지 업로드 하기

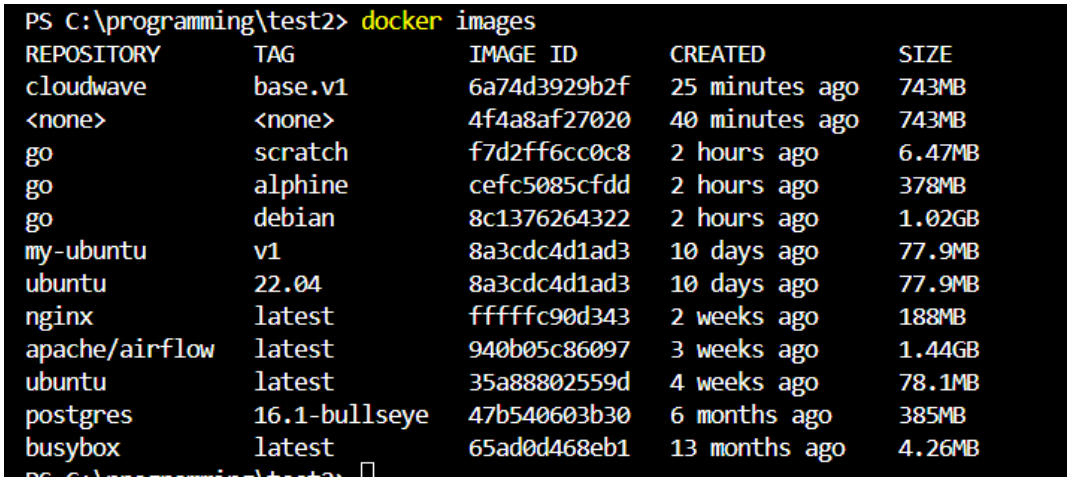
이미지 태그 변경하기

> https://docs.docker.com/engine/reference/commandline/tag/

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

[예시] 이미지 이름 변경하기

```
$ docker tag ubuntu:22.04 my-ubuntu:v1
$ docker images
REPOSITORY                          TAG             IMAGE ID       CREATED
SIZE
my-ubnutu                           v1              174c8c134b2a   9 days ago
 77.9MB
ubuntu                              22.04           174c8c134b2a   9 days ago
 77.9MB
```



이미지 업로드하기

```
docker push [OPTIONS] NAME[:TAG]
```

주요 옵션

Option	Short	Default	Description
`--all-tags`	`-a`		Push all tags of an image to the repository

주의사항

- `Docker hub`가 아닌 `ECR` 또는 자체 구축한 `Hub`와 같이 별도의 `registry`에 업로드하려는 경우, 이름은 `full image name` 형식으로 작성되어야 합니다.
- `Push`전 `docker login`을 통해서 사용할 `registry`에 인증해야 합니다.

Full image name 포맷

```
[HOST[:PORT_NUMBER]/]PATH
# namespace를 명시하는 경우
[HOST[:PORT_NUMBER]/][NAMESPACE/]REPOSITORY
```

`full image name`의 예시는 다음과 같습니다.

```
# Example - ECR
<aws_account_id>.dkr.ecr.ap-northeast-2.amazonaws.com/cloudwave:v1

# Example - private registry
my.registry.com:5000/cloudwave/spring:v1
```

컴포넌트	설명
HOST	`registry`의 HOST을 의미합니다.
PORT_NUMBER	`registry` 서버의 포트를 의미합니다.
NAMESPACE	논리적 구분 단위입니다. 값이 설정되지 않은 경우, `library`로 설정됩니다.
REPOSITORY	이미지의 이름을 의미합니다.

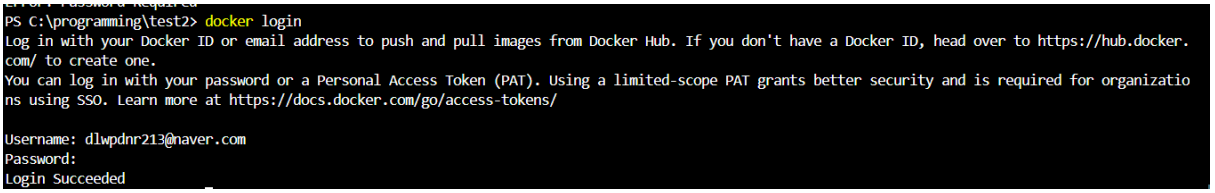
`Registry` 로그인하기

```
docker login [OPTIONS] [SERVER[:PORT]]
```

주요 옵션

Option	Short	Default	Description
`--password`	`-p`		password
`--password-stdin`			`STDIN` 을 통해 password를 입력받습니다.
`--username`	`-u`		Username

Docker hub login이다



연습 문제

[연습] Docker hub에 이미지 업로드하기

`Docker hub`에 로그인합니다.

```
$ docker login -u <USERNAME>
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: <USERNAME>
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

`ubuntu:22.04` 이미지의 이름을 `<USERNAME>/cloudwave:ubuntu.22.04`로 변경합니다.

```
$ docker tag ubuntu:22.04 <USERNAME>/cloudwave:ubuntu.22.04
```

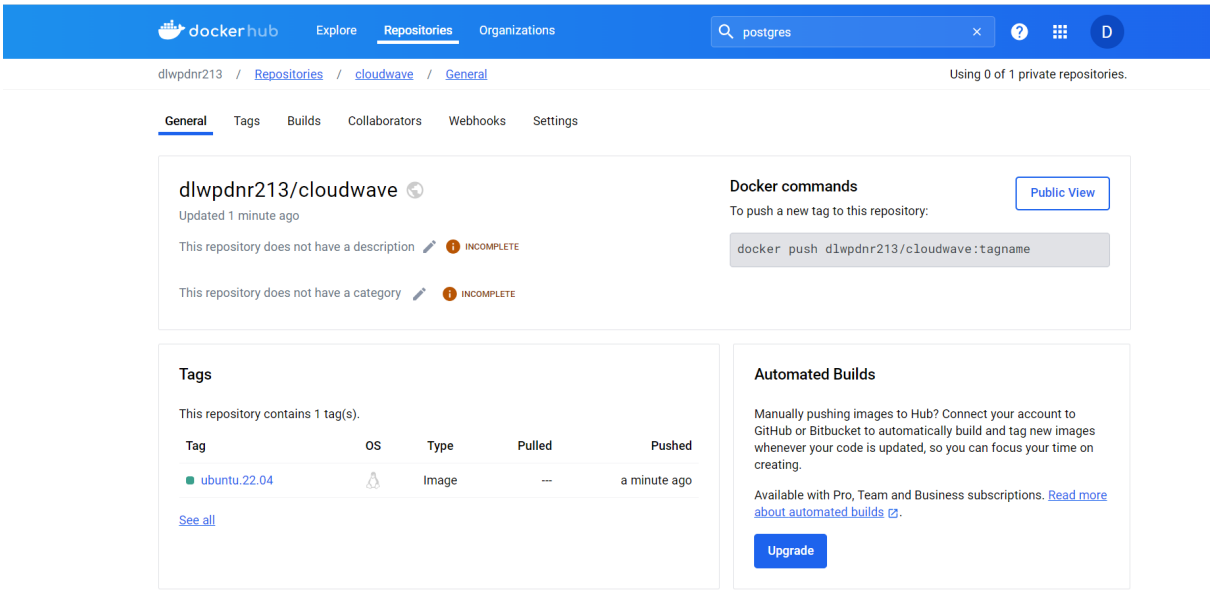
이미지를 `Push`합니다.

```
$ docker push <USERNAME>/cloudwave:ubuntu.22.04
```

업로드한 이미지는 `search` 명령어를 통해 다음과 같이 확인할 수 있습니다.

```
$ docker search <USERNAME>/cloudwave
NAME                    DESCRIPTION   STARS     OFFICIAL   AUTOMATED
<USERNAME>/cloudwave                  0
```

```
PS C:\programming\test2> docker push dlwpdnr213/cloudwave:ubuntu.22.04
The push refers to repository [docker.io/dlwpdnr213/cloudwave]
931b7ff0cb6f: Mounted from library/ubuntu
ubuntu.22.04: digest: sha256:7d44d9cfa17f15b3d5d66212420bf808b7d654ded57173e5197acc74897ed431 size: 529
PS C:\programming\test2> docker search dlwpdnr213/cloudwave
NAME                    DESCRIPTION   STARS     OFFICIAL
dlwpdnr213/cloudwave                  0
```



[실습] `Arg`를 이용하여 `base`이미지 변경하기

1. `Go 서버 이미지 제작하기`에서 사용한 `Dockerfile`을 수정합니다.
    1. `ARG`이름은 `OS`로 설정합니다.
    2. `ARG`를 이용하여 `Base 이미지`를 입력받아야 합니다.
    3. `ARG`의 값을 환경변수(`BASE`)에 저장합니다.
2. `--build-args` 옵션을 사용하여 `ubuntu:22.04`, `alpine:latest`를 기반으로한 이미지를 생성합니다.

```
# Alpine을 이용한 Dockerfile
ARG OS
FROM $OS
ENV BASE=$OS
WORKDIR /app
COPY src ./
RUN CGO_ENABLED=0 go build -o main
CMD ["/app/main"]
```

build 시 실행할 커맨드 코드 docker build --build-arg OS=golang:1.19-bullseye -t test:ubuntu .

docker images로 만들어진거 확인 가능

```
PS C:\programming\test3> docker images
REPOSITORY            TAG             IMAGE ID       CREATED              SIZE
test                  ubuntu          0c7c603560d7   About a minute ago   1.02GB
cloudwave             base.v1         6a74d3929b2f   7 hours ago          743MB
<none>                <none>          4f4a8af27020   7 hours ago          743MB
go                    scratch         f7d2ff6cc0c8   8 hours ago          6.47MB
go                    alphine         cefc5085cfdd   8 hours ago          378MB
go                    debian          8c1376264322   8 hours ago          1.02GB
dlwpdnr213/cloudwave  ubuntu.22.04    8a3cdc4d1ad3   10 days ago          77.9MB
ubuntu                22.04           8a3cdc4d1ad3   10 days ago          77.9MB
nginx                 latest          fffffc90d343   2 weeks ago          188MB
apache/airflow        latest          940b05c86097   4 weeks ago          1.44GB
ubuntu                latest          35a88802559d   4 weeks ago          78.1MB
postgres              16.1-bullseye   47b540603b30   6 months ago         385MB
```