Docker

도커 이미지

- 컨테이너를 생성할 때 필요한 요소
- 가상 머신의 iso 파일과 비슷한 무엇인가
- 사용할 때는 읽기 전용으로 사용되며
- 도커 명령어로 내려받을 수 있음
- 이름 형식 : [저장소 이름]/[이미지 이름]:[태그]의 형태
  - bitnami/pytorch:2.0.1
  - bitnami/pytorch:latest
  - ubuntu:22.04

도커 허브

- https://hub.docker.com/

도커 컨테이너 생성/시작

- 도커 엔진 버전 확인
  - docker –v
- 도커를 쓸 수 있는 상황인가?
  - docker –ps
- 위에거는 틀렸고, 도커 데몬이 돌고 있는지 확인을 해야 합니다.
- 컨테이너 실행
  - docker run
  - docker run –i –t ubuntu:22.04
    - 종료 : exit
    - 나오기 : CTRL P, Q

도커 컨테이너 생성/시작 (2)

- 이미지 가져오기
  - docker pull
    - docker pull bitnami/pytorch
      - 기본으로 :latest tag를 가지고 오게 됨
  - 이미지 확인
    - docker images
- 컨테이너 생성
  - docker create
    - docker create -i -t --name mytorch bitnami/pytorch:latest
  - 이름을 생략하면?
    - 형용사_명사 형태로 이름은 자동 부여

도커 컨테이너 생성/시작 (3)

- 생성된 컨테이너 시작
  - docker start 이름
  - docker attach 이름
- 컨테이너 확인
  - docker ps
    - docker ps -a
    - docker ps -q
  - container id : 컨테이너 식별자
  - command : 컨테이너가 시작될 때 실행할 명령, 지정 가능
    - docker run -i -t ubuntu echo hello world
  - 이름 : 변경 가능 (중복 금지)
    - docker rename oldname newname

도커 컨테이너 삭제

- 삭제
  - docker rm 이름
    - docker rm mytorch
  - 실행중인 컨테이너는 삭제할 수 없으므로 정지
    - docker stop
    - docker rm -f : 강제삭제
  - 모두삭제
    - docker container prune
    - 실행중인 컨테이너는 삭제되지 않음
    - docker rm $(docker ps -a -q) *windows에서는 안 될 수도 있음
- 이미지 삭제 : docker rmi

컨테이너를 노출시키자

- 컨테이너의 네트워크 확인
  - docker run -i -t ubuntu 후 ifconfig
    - apt-get update / apt-get install net-tools
  - 컨터이너 내의 eth0을 동작하게 만드려면
    - 호스트의 IP/PORT를 컨테이너 eth0의 IP/PORT와 바인딩
- 포트 바인딩
  - -p 옵션 : -p 호스트:컨테이너
    - -p 80:80 → 호스트의 80번 포트는 컨테이너의 80번 포트로 바인딩 되어 사용
    - -p 192.168.35.253:8808:80 → 호스트의 192.168.35.253:8808은 컨테이너의 80번 포트로 바인딩 되어 사용

컨테이너를 노출시키자 (2)

- 실습
  - docker run -i -t -p 80:80 --name web ubuntu
  - 웹서버 설치 & 실행
    - apt-get update
    - apt-get install nginx
    - service nginx start

컨테이너 APP 구축

- 내가 mysql이 필요한데...
  - docker 컨테이너로 해 볼 수 있을까?
  - docker hub의 mysql을 방문
  - How to use this image...
- docker run의 -e option
  - 환경변수 설정
  - docker run -i -t -e username=edberg ubuntu
    - echo $username

컨테이너 APP 구축 (2)

- docker run -e MYSQL_ROOT_PASSWORD=... -e MYSQL_DATABASE=... mysql:latest
- docker run -d -e MYSQL_ROOT_PASSWORD=... -e MYSQL_DATABASE=... mysql:latest → detached
- 동작중인 docker의 무엇인가를 실행 : docker exec -i -t
  - docker exec -i -t name /bin/bash
  - mysql -u root -p, use databases

컨테이너 APP 구축 (3)

- 외부에서는 이 mysql을 어떻게 사용하지?
  - port mapping
    - --p 3306:3306
  - 컨테이너 끼리라면?
    - --link : --link 컨테이너이름:호스트이름
  - docker run -d --name mydb -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=testdb mysql:latest
  - docker run –i -t --link mydb:mysqlhost Ubuntu
    - apt-get update && apt-get install mysql-client
    - mysql –h mysqlhost -u root -p

컨테이너 APP 구축 (4)

- wordpress 블로그를 뛰워 봅시다.
  - step1. mysql 컨테이너 실행
    - 환경 변수로 설정해야 할 것
      - MYSQL_ROOT_PASSWORD
      - MYSQL_DATABASE
  - step2. wordpress 컨테이너 실행
    - 환경 변수로 설정해야 할 것
      - WORDPRESS_DB_HOST
      - WORDPRESS_DB_USER
      - WORDPRESS_DB_PASSWORD
      - WORDPRESS_DB_NAME
    - 그리고 LINK –p 8888:80

- container 1 : mysql image로 mydb container. 외부로 노출할 필요가 없어요. 내부에서 컨테이너 끼리만 통신을 하니까.
- 설정을 요구하는것
- -e MYSQL_ROOT_PASSWORD=password1234
- wordpress를 위해서 필요한 설정
- -e MYSQL_DATABASE=testdb
- --name mydb

- container 2 : wordpress image로 wp container. 80번 포트를 외부로 노출. 왜냐하면 외부에서 블로깅을 해야 하니까. --name wp
- 설정을 요구하는 것
- -e WORDPRESS_DB_HOST=db
- -e WORDPRESS_DB_USER=root
- -e WORDPRESS_DB_PASSWORD=password1234
- -e WORDPRESS_DB_NAME=testdb
- --link mydb:db
- -p 8888:80 (누가 8888번 포트로 호스트에 접근하면, 80번 컨테이너 포트로 바인딩)

- docker run –d -e MYSQL_ROOT_PASSWORD=password1234 -e MYSQL_DATABASE=testdb --name mydb mysql:latest

- docker run –d -e WORDPRESS_DB_HOST=db -e WORDPRESS_DB_USER=root -e WORDPRESS_DB_PASSWORD=password1234 -e WORDPRESS_DB_NAME=testdb --name wp -p 8888:80 --link mydb:db wordpress:latest

- docker container의 로그를 볼 수 있는 가장 쉬운 방법
  - docker logs container_id (혹은 이름)

데이터 저장

- 데이터는 유지될 것인가?
  - 생성된 이미지는 변경되지 않으며
  - 컨테이너는 휘발성
- Persistent 데이터를 사용하려면
  - 호스트 볼륨 공유
  - 볼륨 컨테이너
  - 도커 볼륨

컨테이너로 파일 copy

- docker cp filepath(local) container_name:path(in container)
  - docker cp cp.html web:/var/www/html
  - 호스트 → 컨테이너

- 반대방형 : 컨테이너에서 파일 copy
  - docker cp container_name:path(in container) path(local)
  - 컨테이너 → 호스트

호스트 볼륨 공유

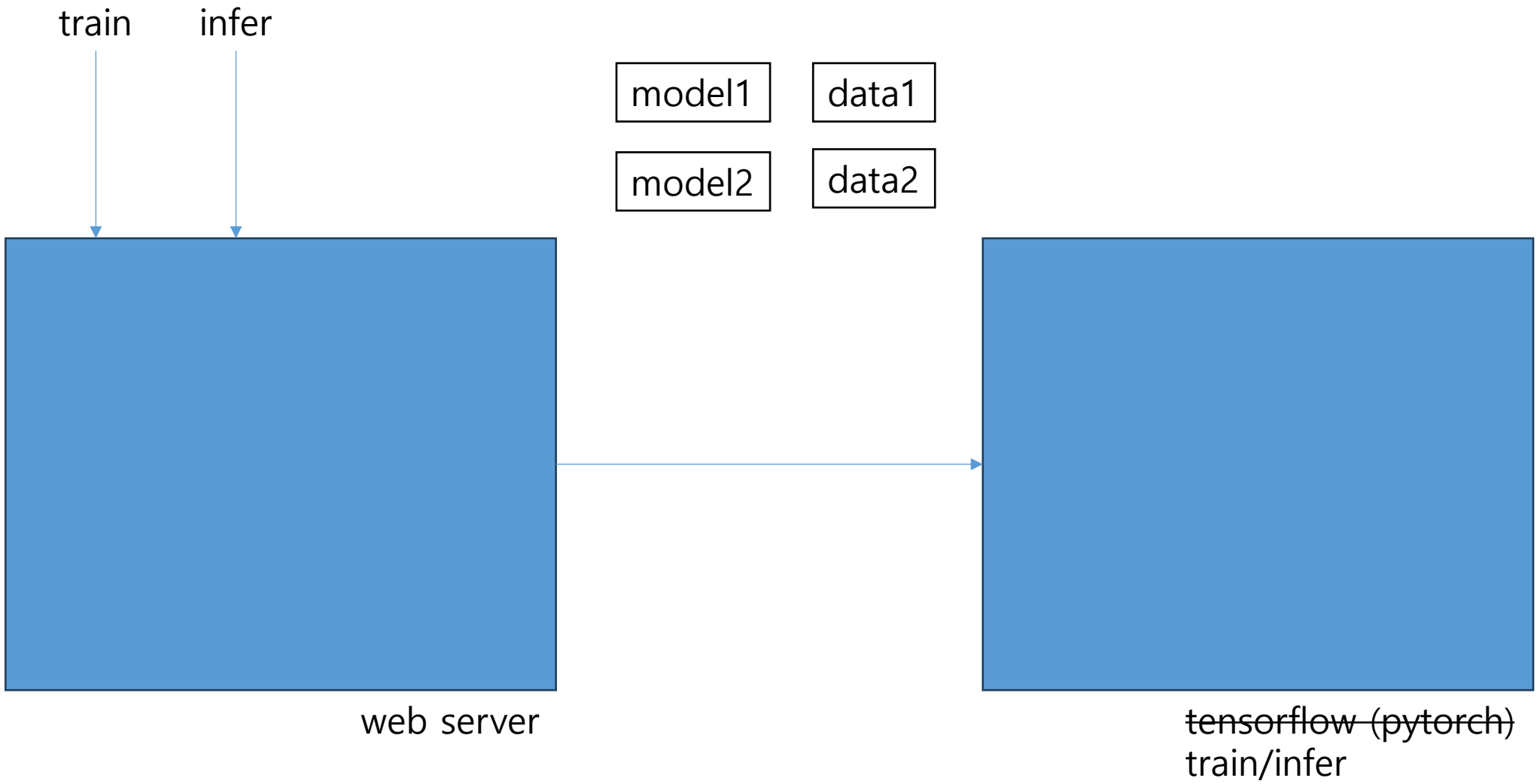
- -v [호스트디렉토리]:[컨테이너디렉토리]
  - docker run -i -t -v ./temp/docker:/root ubuntu
    - 윈도우라면 -v C:/temp/docker:/root 이런 식으로
  - 앞의 mysql에 -v /mydir:/var/lib/mysql 을 추가한다면?
- 컨테이너를 삭제하더라도 데이터는 유지됨
- 새로운 컨테이너에 기존의 호스트 디렉토리를 공유하면?
  - 컨테이너의 디렉토리는 호스트 디렉토리로 overwrite

볼륨 컨테이너

- --volumes-from : 다른 컨테이너와 볼륨을 공유
  - docker run -i -t --name u1 -v /temp/docker:/root ubuntu
  - CTRL-P, Q 후
  - docker run -i -t --volumes-from u1 ubuntu

도커 볼륨 (1)

- 도커 자체에서 제공하는 볼륨 기능
- 볼륨 생성
  - docker volume create --name tutorial
- 볼륨 확인
  - docker volume ls
- 도커 볼륨 마운트 : -v
  - docker run -i -t -v tutorial:/root/ ubuntu
- 그냥 쓰는 -v : 볼륨을 자동으로 생성
  - docker run -i -t -v /root Ubuntu
  - docker container inspect containe이름 : 컨테이너 각종 정보 조사, 특히 볼륨도 (Mounts/Source)


train
infer
model1
data1
model2
data2
web server
~~tensorflow (pytorch)~~
train/infer

도커 볼륨 (2)

- 도커 볼륨 삭제
  - docker volume rm 볼륨명
    - 실행중인 컨테이너에 연결된 경우는 삭제 안됨
  - docker volume prune
    - 모든 볼륨을 삭제하되, 단 한번도 파일이 담궈지지 않은 볼륨에 한하여
- --mount 옵션도 유사
  - --mount type=volume,source=tutorial,target=/root
  - --mount type=bind,source=C:/Temp/docker,target=/root
- -v는 여러개 쓸 수 있으며, 디렉토리 뿐 아니라 파일 단위로도 가능 : 디렉토리는 디렉토리라는 티를 내주자.

컨테이너 네트워크

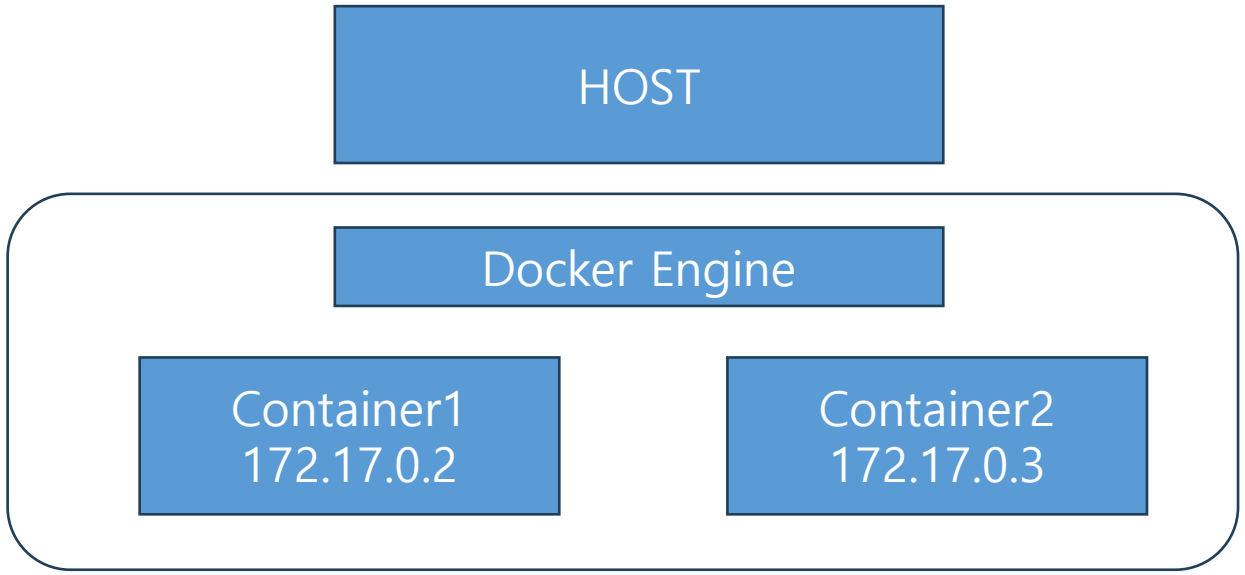
- 사용 가능한 네트워크 목록 확인
  - docker network ls
- 종류
  - bridge network : 컨테이너를 생성할 때 자동으로 연결되는 docker0 브릿지를 활용

컨테이너 네트워킹

- 컨테이너 내부에서 네트워크를 확인해 봅시다.
  - apt-get update && apt-get install net-tools
  - eth0 (첫 번째 이더넷 인터페이스)
  - lo (루프백 인터페이스)
- 내부 네트워크 테스트
  - 컨테이너1 - web
    - nginx 설치, 시작
  - 이 상황에서 컨테이너의 IP로 웹으로 접근이 가능할까요? (apt-get install net-tools)
  - 컨테이너2 - ubuntu
    - curl 설치 (apt-get update && apt-get install curl)
    - curl ip
  - 컨테이너 끼리는 서로 잘 통신이 됨

컨테이너 네트워킹(2)

하지만 호스트는 몰라

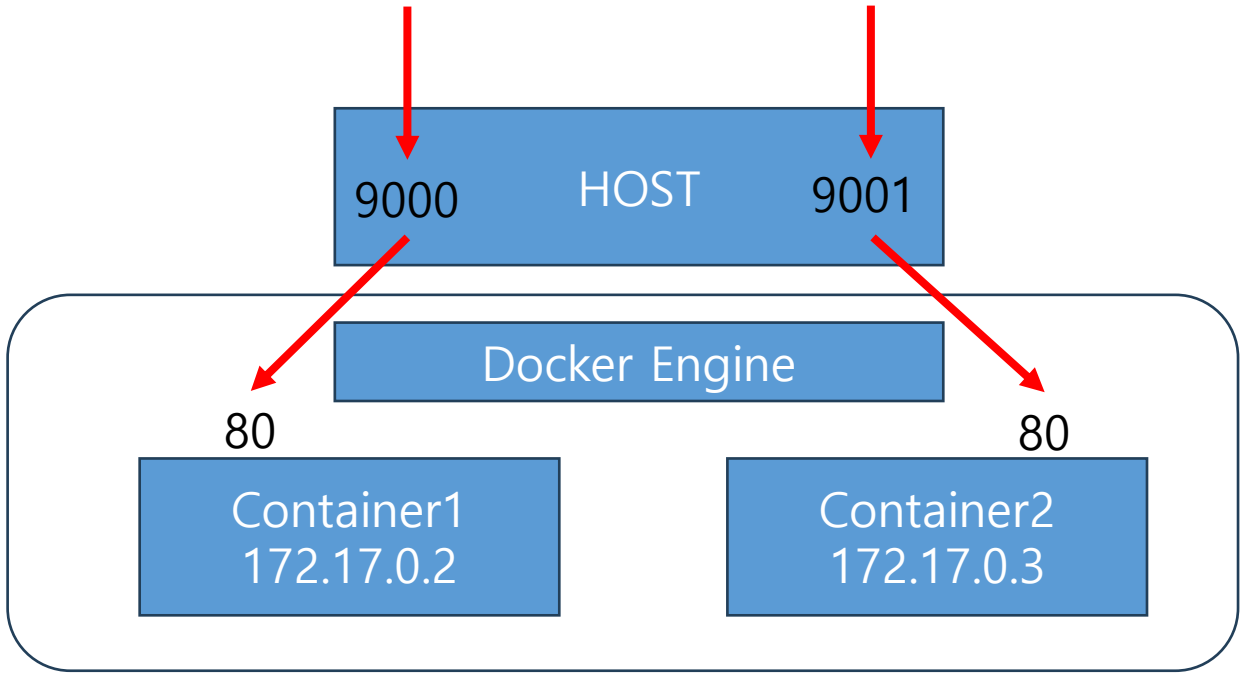


너희 끼리는 알어

대신 포트 바인딩을 하면
보내 줄 수는 있어

브리지 네트워크

컨테이너 네트워킹(3)



대신 포트 바인딩을 하면
보내 줄 수는 있어

C1 -p 9000:80
C2 -p 9001:80

컨테이너 네트워킹(4)

- 네트워크 확인
  - docker network ls
  - docker network inspect bridge
- 사용자 정의 브릿지 만들기
  - docker network create --driver bridge newbridge
  - 사용할 때는 --net 옵션으로 지정
    - docker run -i –t --net newbridge ubuntu
  - docker network disconnect 브릿지이름 컨테이너이름
  - docker network connect 브릿지이름 컨테이너이름

컨테이너 네트워킹(5)

- 호스트 네트워크
  - 호스트의 네트워크 환경을 그대로 사용
  - --net host로 컨테이너 생성
  - docker run –i –t --net host --name hostnetwork ubuntu
- None 네트워크
  - 네트워크를 사용하지 않음
  - --net none

컨테이너 네트워킹(6)

- 컨테이너 네트워크
  - 다른 컨테이너의 네트워크 환경을 공유
  - --net container:컨테이너 이름
- 네트워크 알리아스 설정
  - 특정 호스트 이름으로 여러 컨테이너에 접근 가능
  - --net-alias
  - docker run –i –t –d --net newbridge –-net-alias aliastest ubuntu (3번)
  - docker run –i –t --net newbridge ubuntu
  - apt-get update; apt-get install iputils-ping
  - ping –c 1 aliastest

컨테이너 로깅

- 컨테이너 로그 확인
  - docker logs 컨테이너 이름
  - 마지막 몇 줄만 읽고 싶을 때는 --tail n
  - --log-opt max-file=, --log-opt max-size= 로 로그 크기 제한 가능
- 로그를 syslog로 보내기
  - --log-driver=syslog

컨테이너 리소스

- 기본 설정으로 컨테이너는 호스트의 리소스를 무제한으로 사용
- 제한 방법
  - 메모리 : --memory="1g"
    - 1g를 초과하면 종료되므로, 적절하게 설정이 필요
    - mega : m
    - swap : --memory-swap
  - CPU
    - --cpu-shares n : 컨테이너의 상대적 cpu 사용 할당
      - 예를 들어 --cpu-shares 1000, --cpu-shares 500의 두 컨테이너는 2:1로 cpu를 사용
    - --cpuset-cpus : 여러 core 시스템에서 특정 코어를 결정
      - --cpuset-cpus=2 → 3번 core, --cpuset-cpus="1,3" → 2, 4번 코어, --cpuset-cpus="1-3"→2, 3, 4번 코어
    - --cpu-period, --cpu-quota : period중 quota만을 사용하겠다는 선언
      - --cpu-period=100000 --cpu-quota=25000 : 100ms에서 25ms만 쓰겠다.
    - --cpus : cpu 중 몇 개를 점유하겠다는 선언 (0.5 → 코어 반개, 2 → 코어 2개)

컨테이너 리소스(2)

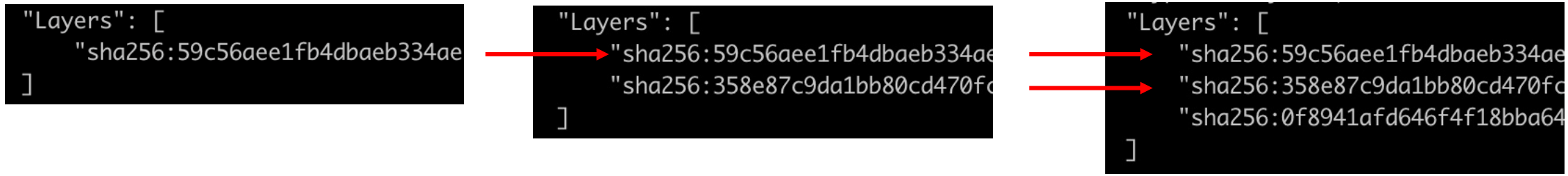
- 제한 방법
  - Block I/O
    - 쓰기 속도 제한 : --device-write-bps device명:제한 (--device-write-bps /dev/disk1s5s1:1mb → 해당 디스크의 쓰기 속도를 1MB로 제한)
    - 읽기 속도 제한 : --device-read-bps
    - 상대 속도 제한 : --device-write-iops, --device-read-iops

도커 이미지 만들기

- docker commit
  - ubuntu 실행 (--name ubuntu_with_tools)
  - apt-get update; apt-get install net-tools; apt-get install nginx
  - 빠져 나온 후 docker commit –a "edberg" –m "ubuntu with tools" ubuntu_with_tools ubuntu_test:first
  - docker images로 확인
- 그 위에 다시 만들기
  - ubuntu_test 이미지로 컨테이너 실행 (--name ubuntu_with_tools2)
  - apt-get install python3
  - 빠져 나온 후 docker commit
- 이미지 추출 : docker save –o filename image
  - 추출한 이미지 로드 : docker load –i filename

도커 이미지 구조

- docker inspect ubuntu, ubuntu_test:first, ubuntu_test:second



  - 저장도 Layer 단위로 저장
- ubuntu image를 삭제하면?
  - untagged?
- ubuntu_test:second를 삭제하면?

도커 이미지 배포

- Option1 : save / load 또는 export / import
- Option2 : 도커 허브 이미지 저장소
  - push/pull만 하면 됨
  - Private 무료 사용은 제한이 있음
  - Public은 너무나 Open되어 있음
- Option3 : 나만의 도커 저장소를 활용
- Option4 : github 등을 통해 빌드 가능한 상태로 배포
  - 빌드 = 도커 이미지를 만든다

도커 허브 이미지 저장소

- 회원 가입 / 로그인
- 저장소 생성
- 내가 만든 이미지를 repository에 올릴 수 있도록 준비
  - docker tag 기존이미지이름:태그 새_이미지_이름:tag
    - docker tag ubuntu_test:first edberg1974/tutorial:0.0
    - tagging이지 삭제가 아님
- docker login
- docker push 새_이미지_이름:tag
  - docker push edberg1974/tutorial:0.0
- 내려받을 때는 : docker pull 이미지이름:tag
  - login이 필요 없음 (public 저장소 이니까)

나만의 도커 저장소

- Docker Private Registry
  - docker에서 제공하는 'registry' 이미지를 사용해 컨테이너를 만들면 됩니다.
  - docker run –d –p 5000:5000 --restart always --name registry registry:latest
  - 단, 이 경우 HTTPS 접속이 필요
    - 만약 이를 회피하고 싶을 때 Docker 옵션을 바꾸면 됨
    - --insecure-registry=ip:port
  - push / pull에 url 사용
    - docker push 127.0.0.1:5000/my_private_image:0.0
  - https를 쓰고 싶다?
    - 인증서를 발급 받고, nginx 등 웹 서버에 설정

도커 이미지 만들기 (다시)

- 개발한 어플리케이션을 컨테이너화 한다면?
  - 어떤 베이스 이미지로 컨테이너를 생성
  - 애플리케이션을 위한 환경 설정
  - 소스 코드를 입력하고 동작 여부를 확인
  - 컨테이너를 이미지로 만들고 커밋
- 이 과정을 자동화 할 수 있는 방법
  - Dockerfile에 이 과정을 기록하고
  - 빌드등을 실행

Docker build

- 간단한 python code – numpytest.py

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Docker build (2)

- Dockerfile

```
FROM ubuntu:22.04
LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
RUN apt-get update
RUN apt-get install python3
WORKDIR /app
COPY numpytest.py .

CMD ["python3", "numpytest.py"]
```

Docker build (3)

- 도커 빌드 : docker build –t numpytest:0.0 ./

```
Dockerfile:4
--------------------
   2 |     LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
   3 |     RUN apt-get update
   4 | >>> RUN apt-get install python3
   5 |     WORKDIR /app
   6 |     COPY numpytest.py .
--------------------
ERROR: failed to solve: process "/bin/sh -c apt-get install python3" did not com
plete successfully: exit code: 1
```

```
FROM ubuntu:22.04
LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
RUN apt-get update
RUN apt-get -yq install python3
WORKDIR /app
COPY numpytest.py .

CMD ["python3", "numpytest.py"]
```

Docker build (4)

```
(.venv) edberg:~/work/flyai/dockerbuild > docker build -t numpytest:0.0 ./
[+] Building 0.0s (10/10) FINISHED                          docker:desktop-linux
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 261B                                       0.0s
 => [internal] load metadata for docker.io/library/ubuntu:22.04            0.0s
 => [1/5] FROM docker.io/library/ubuntu:22.04                              0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 70B                                           0.0s
 => CACHED [2/5] RUN apt-get update                                        0.0s
 => CACHED [3/5] RUN apt-get -yq install python3                           0.0s
 => CACHED [4/5] WORKDIR /app                                              0.0s
 => CACHED [5/5] COPY numpytest.py .                                       0.0s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:a6fa630ea798                                   0.0s
 => => naming to docker.io/library/numpy                                   0.0s

What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quick
view
```

```
(.venv) edberg:~/work/flyai/dockerbuild > docker run numpytest:0.0
Traceback (most recent call last):
  File "/app/numpytest.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

Docker build (5)

```
FROM ubuntu:22.04
LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
RUN apt-get update
RUN apt-get -yq install python3 python3-pip
RUN pip3 install numpy
WORKDIR /app
RUN
COPY numpytest.py .

CMD ["python3", "numpytest.py"]
```

```
(.venv) edberg:~/work/flyai/dockerbuild > docker run numpytest:0.0
[1 2 3 4 5]
```

- 빌드 시에는 cache를 사용
  - 사용하지 않으려면 docker build --no-cache –t 이미지

Docker build (6)

- ENTRYPOINT – 컨테이너를 실행가능한 무엇인가로 만듦

  ENTRYPOINT ["python"]

  CMD ["app.py"]
- pip3 freeze : 필요한 패키지의 목록을 생성
  - pip3 freeze > requirements.txt
  - (In Dockerfile)
    - COPY requirements.txt .
    - RUN pip3 install –r requirements.txt
    - 주의할 점 : 쓸모없는 모듈들 주의

Docker build (7)

- COPY는 directory 단위로도 COPY가 가능
  - requirements.txt와 *.py를 app 디렉토리를 만들어 옮긴 후

```
FROM ubuntu:22.04
LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
RUN apt-get update
RUN apt-get -yq install python3 python3-pip
COPY app /app
WORKDIR /app
RUN pip3 install -r requirements.txt

ENTRYPOINT ["python3"]
CMD ["numpytest.py"]
```

  - 디렉토리가 꼬였다 싶을땐?
    - docker run --entrypoint /bin/bash imagename

Docker build (8)

- 멀티스테이지 빌드 - 2개 이상의 FROM
  - 이미지 크기를 줄이기 위한 용도

```
FROM golang
ADD main.go /root
WORKDIR /root
RUN go build -o /root/app /root/main.go

FROM ubuntu:latest
WORKDIR /root
COPY --from=0 /root/app .
CMD ["./app"]
```

```
FROM golang as gobuild
ADD main.go /root
WORKDIR /root
RUN go build -o /root/app /root/main.go

FROM ubuntu:latest
WORKDIR /root
COPY --from=gobuild /root/app .
CMD ["./app"]
```

Docker build (9)

- 명령어들
  - EXPOSE : 이미지에서 노출할 포트를 지정
  - ENV : 환경변수 설정
    - ENV DB_HOME /db
    - docker run / create의 –e 옵션은 이를 덮어 씀
  - ARG : 도커 빌드 시에, 즉 Docker file 내에서 사용할 변수
    - ARG pypath=pythonfiles
    - WORKDIR $(pypath) → WORKDIR pythonfiles
  - USER : 지정한 사용자 권한으로 작업
    - RUN grouopadd –r student && useradd –r –g student eddy
    - USER eddy

Docker build (10)

- 명령어들 (2)
  - ADD와 COPY의 차이
    - ADD는 외부 URL이나 TAR File 내부에서도 꺼내서 넣을 수 있음
    - ADD http://any.com/file.html /html
    - ADD file.tar /root/file
- .dockerignore 파일
  - docker build시 무시해야 할 파일과 디렉토리를 지정
  - .gitignore와 유사한 컨텍스트
    - 경로는 빌드 컨텍스트의 루트를 '/'로 사용

Docker build (11)

- 주의사항
  - Image 사이즈를 줄이기 위한 노력이 필요
  - 레이어 수를 줄이는 것이 좋음
    - RUN … && … && …

```
FROM python:3.9.17-slim
LABEL maintainer "edberg1974 <edberg.s@gmail.com>"
COPY app /app
WORKDIR /app
RUN pip3 install -r requirements.txt

ENTRYPOINT ["python3"]
CMD ["numpytest.py"]
```