01

AWS EC2 배포

01. SpringBoot 배포 순서

- AWS EC2 인스턴스 생성
  - 키페어 저장
  - 탄력적 IP 생성 인스턴스에 연결
  - 보안그룹에서 인바운드 규칙 편집(3306, 8080, 3000, 80번 포트 개방)
- 터미널을 열어 AWS EC2 인스턴스 접속
  - chmod 400 "20241211_ServerKey.pem"
  - ssh -i "20241211_ServerKey.pem" ubuntu@ec2-43-202-46-116.ap-northeast-2.compute.amazonaws.com
- sudo su : 관리자 권한 취득
- apt update : 패키지 목록 최신 상태로 갱신
- apt upgrade
  - 이미 설치된 패키지들을 최신 버전으로 업그레이드

01. SpringBoot 배포 순서

- apt-cache search openjdk : 설치가능한 자바버전확인
- apt install openjdk-17-jdk : java 설치
- apt install mysql-server
- Mysql –u root –p 접속
  - ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY ‘1234';
  - Exit;

- SpringBoot의 IP를 EC2의 IP로 변경
- Spring Boot Build
  - Gradle – tasks- build- bookjar
  - 프로젝트경로/build/libs/안에 .jar 파일 확인

01. SpringBoot 배포 순서

- 다른 터미널을 열어 EC2에 파일 복사
  - scp -i 20241211_ServerKey.pem “springboot build 파일명.jar” ubuntu@ec2-43-202-46-116.ap-northeast-2.compute.amazonaws.com:/home/ubuntu
  - scp -i 20241211_ServerKey.pem “export 한 데이터베이스 테이블명.sql” ubuntu@ec2-43-202-46-116.ap-northeast-2.compute.amazonaws.com:/home/ubuntu

- EC2에서 mysql 접속
  - Use shop;
  - Source “export한 데이터베이스 테이블.sql”;
  - Quit;
- Java –jar “springboot build 파일명.jar”
- 로컬PC에서 EC2백앤드에 요청 테스트

02. React 배포 순서

- Frontend 요청 Ip 주소를 EC2의 아이피로 변경
- Frontend에서 npm run build
- Frontend 빌드 폴더안의 모든 내용을 EC2 /home/ubuntu/build안에 복사
  - scp -i 20241211_ServerKey.pem –r ./build ubuntu@ec2-43-202-46-116.ap-northeast-2.compute.amazonaws.com:/home/ubuntu

02. React 배포 순서

- nohup java -jar Auth2JWT-0.0.1-SNAPSHOT.jar > output.log 2>&1 & => 백앤드 서버 백그라운드 실행
  - Auth2JWT-0.0.1-SNAPSHOT.jar를 백그라운드에서 실행
  - 로그를 output.log에 저장
  - 2>&1 : 표준에러(stderr)를 표준출력(stdout)으로 리다이렉션하여 로그파일에 저장
  - tail -f output.log : 명령으로 실시간 로그 확인 가능

- apt install -y nodejs : serve 설치를 위해
- apt install npm
- npm install -g serve
- serve -s build
- 로컬에서 http://EC2 IP:3000으로 테스트

03. Nginx로 React 배포

- apt update
- apt install nginx
- systemctl start nginx
- systemctl status nginx
- rm /etc/nginx/sites-available/default
- rm /etc/nginx/sites-enabled/default
- cd /etc/nginx/sites-available/
- vi react-project.conf

03. Nginx로 React 배포

```
server {
    listen 80;
    server_name ec2 IP 주소;

    root /var/www/html;
    index index.html index.htm;

    location / {
        try_files $uri $uri/  /index.html;
    }
}
```

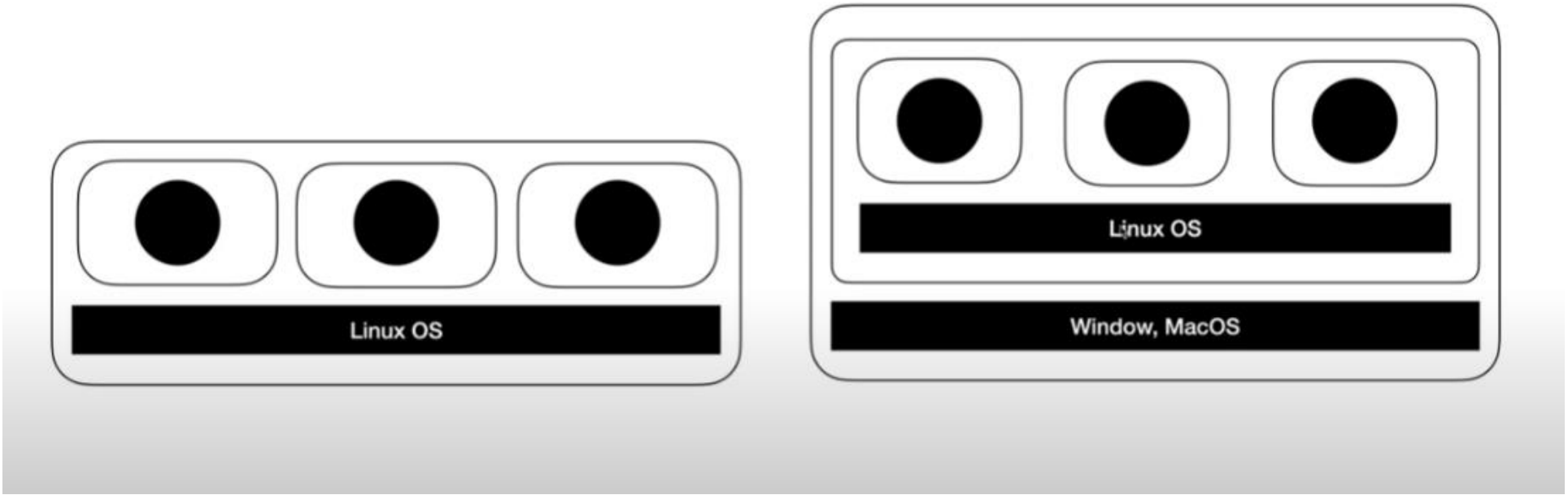
- ln -s /etc/nginx/sites-available/react-project.conf /etc/nginx/sites-enabled/react-project.conf : 링크 생성(바로가기)
- systemctl restart nginx
- systemctl status nginx
- chmod -R 755 /var/www/html
- cp –r ./build/* /var/www/html

02

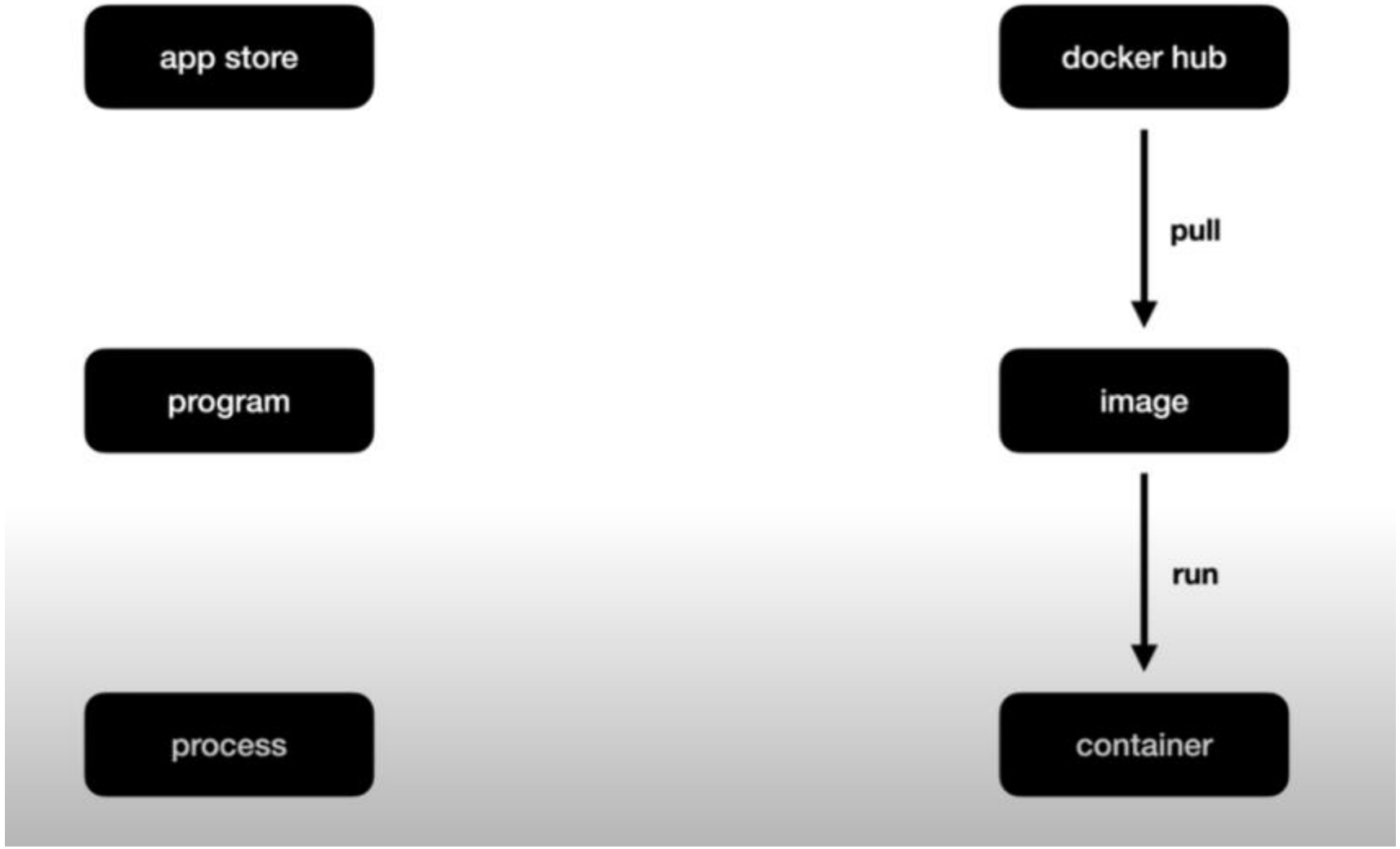
docker

01. docker

애플리케이션과 그 실행 환경을 컨테이너라는 가벼운 단위로 패키징하여 어디서나 동일하게 실행할 수 있게 해주는 플랫폼



01. docker


app store
program
process
docker hub
pull
image
run
container

02. docker 기본 명령어

- Image 다운로드
  - docker pull 이미지 이름
  - docker pull ubuntu:18.04
- 다운로드된 이미지 보기
  - docker images
- 이미지 삭제
  - docker rmi 이미지 이름 또는 아이디
  - docker rmi unbuntu
- 이미지 실행하여 컨테이너 생성
  - docker run 이미지 이름 또는 아이디
  - docker run –d tomcat
  - docker run –d –-name mytomcat tomcat
  - docker run –d –p 80:8080 tomcat
  - docker run –it ubuntu bash
  - docker run –dit ubuntu

02. docker 기본 명령어

- 컨테이너 리스트 보기
  - docker ps
  - docker ps –a
- 컨테이너 중지
  - docker stop 컨테이너 이름 또는 아이디
- 컨테이너 다시 시작
  - docker start 컨테이너 이름 또는 아이디
- 컨테이너 삭제
  - docker rm 컨테이너 이름 또는 아이디
- 현재 중지된 컨테이너 모두 삭제
  - docker rm $(docker ps -aq -f status=exited)

02. docker 기본 명령어

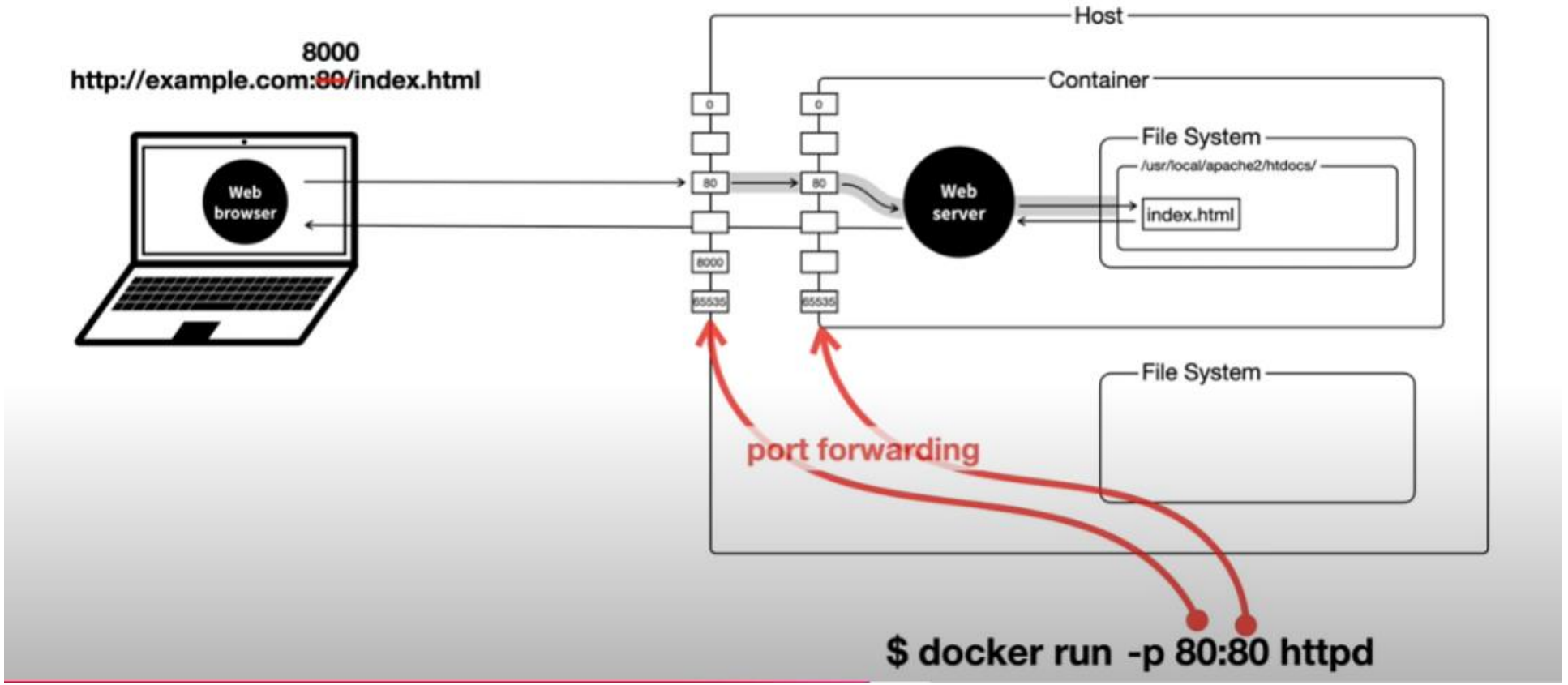
- 현재 실행 또는 중지중인 모든 컨테이너 삭제
  - docker rm -f $(docker ps -aq)
- 현재 docker의 모든 이미지 삭제
  - docker rmi $(docker images –q)
- 현재 실행 중인 컨테이너의 로그 보기
  - docker logs 컨테이너 아이디
  - docker logs –follow 컨테이너 아이디

03. docker 컨테이너의 생명주기

- 운영체제(예:ubuntu)와 같은 정적 프로그램
  - docker run ubuntu, docker run –d ubuntu 와 같이 실행하면 실행되었다가 바로 종료
  - docker run –dit ubuntu나 docker run –it ubuntu bash 로 실행하여야 컨테이너가 유지됨

- 웹서버(예: httpd, tomcat, nginx)와 같은 동적 프로그램
  - 주로 server의 역할을 하는 프로그램으로 한번 run 시키면 종료 시키기 전까지 컨테이너 유지
  - docker run tomcat : tomcat 서버를 Foreground로 실행
  - docker run –d tomcat : Background로 실행
  - docker run –d –p 80:8080 tomcat : 포트 포워딩으로 실행

04. docker 작동원리



05. docker 컨테이너 접근

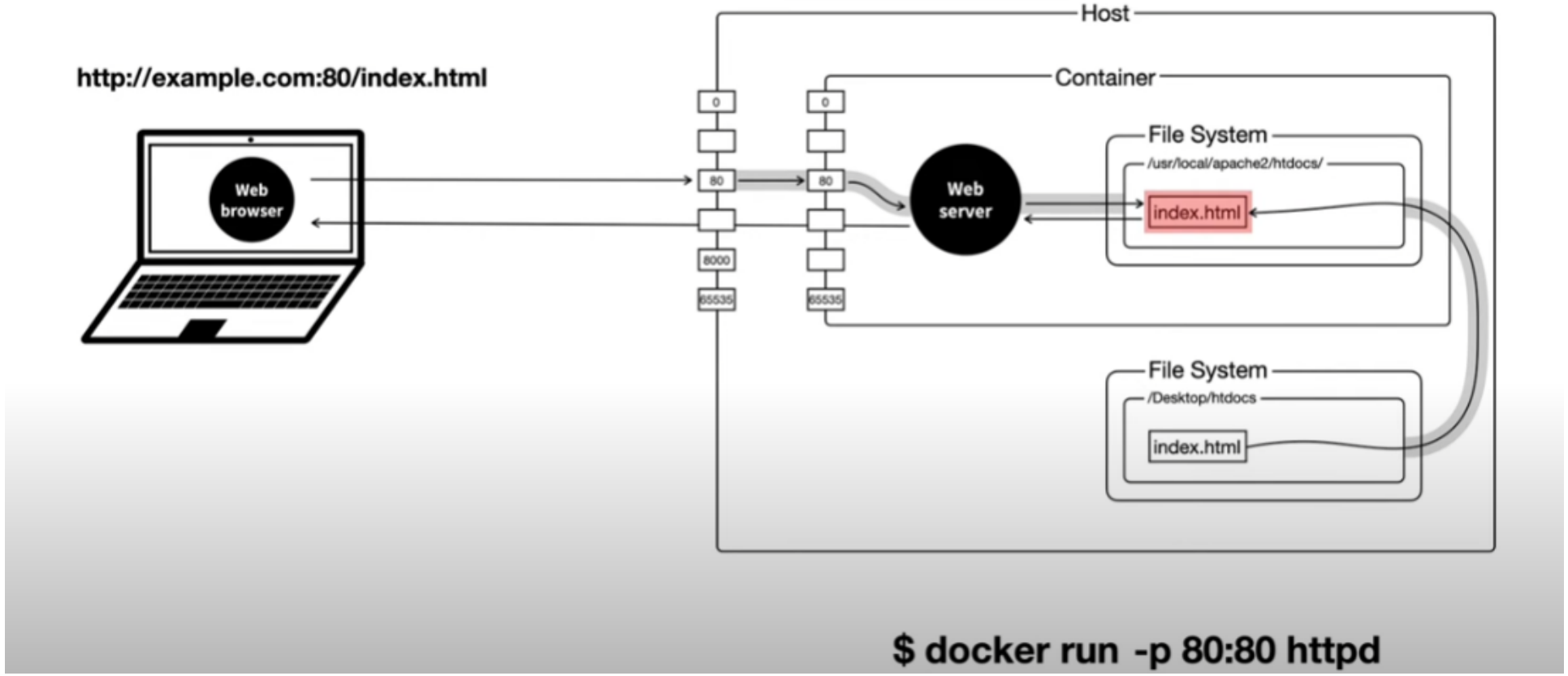
Background에 실행중인 컨테이너 접근

- Docker exec –it 컨테이너 이름 또는 아이디 bash
- docker run -d -p 8080:80 httpd
- docker exec –it ef86 bash

컨테이너 접근 종료

- Ctl + P + Q
  - docker run –it ubuntu bash 와 같이 컨테이너 생성과 동시에 접근 했을때 컨테이너를 종료하지 않고 접근을 종료
- Exit
  - docker run –it ubuntu bash로 접근했을 때는 컨테이너가 종료됨
  - docker exec –it … 로 접근했을 때는 컨테이너 유지

06. docker volume 연결



06. docker volume 연결

컨테이너와 호스트의 File System을 연결

- docker run -d -p 8080:80 -v d:₩temp₩html:/usr/share/nginx/html/ nginx
- d:₩temp₩html와 컨테이너의 /usr/share/nginx/html/를 연결
- 컨테이너의 /usr/share/nginx/html/에 들어가면 d:₩temp₩html를 접근할 수 있음

컨테이너와 docker named volume 연결

- docker volume create my-volume
- docker run –d –p 80:80 –v my-volume:/usr/share/nginx/html nginx
  - my-volume과 /usr/share/nginx/html 연결
  - 이 경로에서 파일을 변경하면 my-volume에 적용
  - 현재 컨테이너가 종료나 삭제가 되고 다른 컨테이너를 생성하면서 my-volume을 연결하면 저장된 내용을 다시 적용

07. docker hub 에 이미지 upload 실습

- docker hub 가입
- docker hub에 repository 생성
- docker run -dit ubuntu
- docker exec -it 93d8(컨테이너 아이디)
- apt update
- apt install vim
- cd home
- mkdir kim
- cd kim
- vi jinah
- exit

07. docker hub 에 이미지 upload 실습

- docker commit 0419 closer19/vim-ubuntu:1.0
  - 현재 실행중인 컨테이너(0419)를 이미지로 저장
- docker images
- docker push closer19/vim-ubuntu:1.0
- docker rm -f $(docker ps -aq)
- docker rmi $(docker images -q)
- docker run -dit closer19/vim-ubuntu:1.0
  - closer19/vim-ubuntu:1.0가 삭제되었으므로 docker hub에서 pull 받아서 컨테이너 생성

08. Dockerfile 작성 예제

- **Dockerfile 생성**

```
FROM httpd
COPY ./webapp /usr/local/apache2/htdocs
CMD ["httpd-foreground"]
```

- **같은 위치에 webapp폴더 생성**
- **webapp폴더안에 index.html생성**
- **docker build -t webserver:1.0 ./**
- **docker run -d -p 8080:80 webserver:1.0**
- **docker exec -it c667 bash**

08. Dockerfile 작성 예제

Dockerfile 생성

```
FROM ubuntu
RUN apt update
RUN apt install -y nginx
WORKDIR /var/www/html
COPY ./webapp/index.html ./index.html
ENTRYPOINT [ "nginx", "-g", "daemon off;" ]
```

ubuntu안에 nginx 설치

nginx –g deaemon off

- nginx 를 foreground로 실행 명령

08. Dockerfile 작성 예제

- **Dockerfile/nginx.conf 파일 생성**

```
FROM nginx
COPY ./webapp /usr/share/nginx/html
COPY ./conf/nginx.conf /etc/nginx/conf.d/default.conf
```

```
server {
    listen 100;
    server_name localhost;

    location / {
        root /usr/share/nginx/html;
        index index.html;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
     }}
```

09. DB Dockerfile

```
FROM mysql:8.0

COPY shop_authentbl.sql /docker-entrypoint-initdb.d

ENV MYSQL_ROOT_PASSWORD=1234
ENV MYSQL_DATABASE=shop
```

10. Backend Dockerfile

```
FROM openjdk:17-jdk-slim

WORKDIR /app

COPY . .

RUN chmod +x ./gradlew
RUN ./gradlew bootJar

ENV JAR_PATH=/app/build/libs
RUN mv ${JAR_PATH}/*.jar /app/app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]
```

11. nginx.conf

```
server {
    listen 80;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api/ {
        proxy_pass http://172.17.0.1:8080;
    }
}
```

12. Frontend Dockerfile

```
FROM node:alpine as build
WORKDIR /app

COPY package.json package-lock.json ./
RUN npm install --silent



COPY . /app
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY ./nginx/nginx.conf /etc/nginx/conf.d/default.conf
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

13. Docker hub

- **Database build 및 push**
  - docker build –t closer19/dockerdb .
  - docker push closer19/dockerdb
- **Backend build 및 push**
  - docker build –t closer19/dockerbackend .
  - docker push closer19/dockerbackend
- **Frontend build 및 push**
  - docker build –t closer19/dockerfrontend .
  - docker push closer19/dockerfrontend