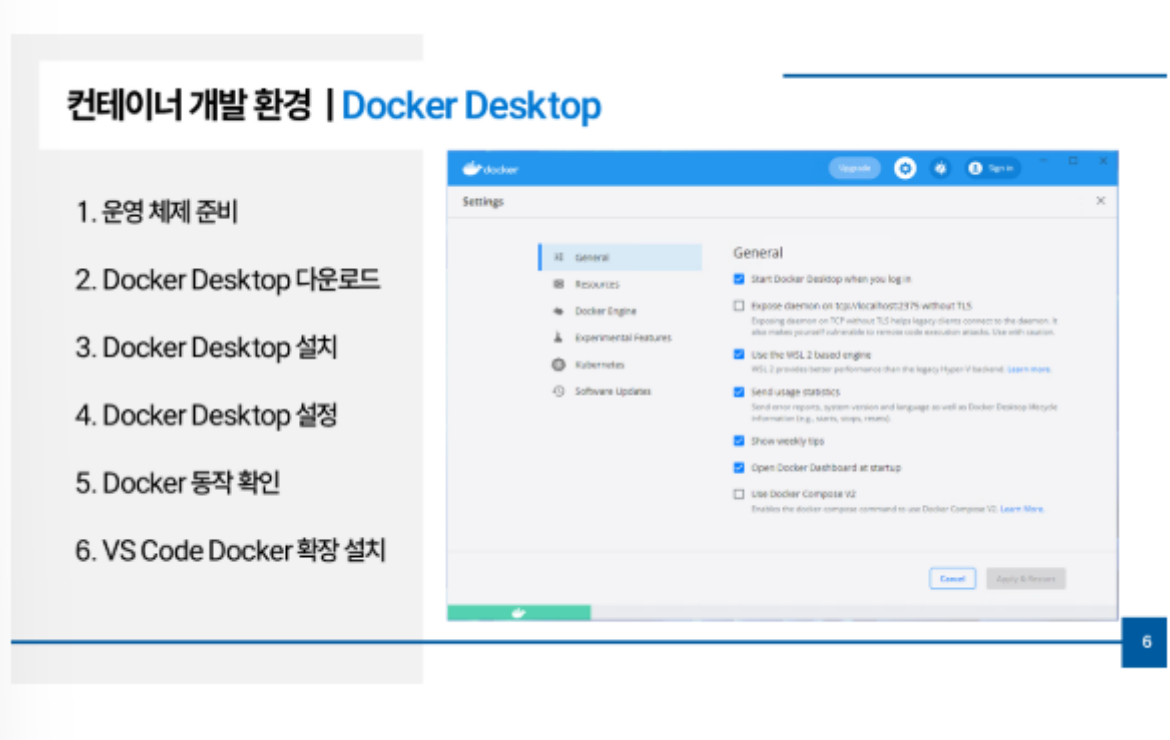
day_1

컨테이너 개발 환경



01_setup-docker_desktop

#1. Enable WSL 2

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux -NoRestart

#2. Enable 'Virtual Machine Platform'

Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform

#3. Linux 커널 업데이트 패키지 다운로드 및 설치

https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

#4. Set WSL 2 as default

wsl --set-default-version 2

#5. Azure CLI 설치

##Windows
https://docs.microsoft.com/ko-kr/cli/azure/install-azure-cli-windows?tabs=azure-cli

#6. 도커 데스크톱 다운로드

https://hub.docker.com/editions/community/docker-ce-desktop-windows

#7. 도커 데스크톱 설치 --> 다시 시작

컨테이너 기술 개요

구동되는 공간이 독립되어서 돌아간다. 앱과 앱이 돌아가는 공간을 격리하여 가상화 하여 돌아가는 것을 컨테이너라고 함.

어떤 운영체제, 실행환경이 다른 것과 구분되어 서로 영향을 받지 않도록 만드는 것을 뜻함.

unix 초창기에 프로그램 작성시 문제가 발생하면 운영체제에 crash가 발생해서 컴퓨터 자체를 재부팅해야하는 문제가 있었음 -> 프로그램을 테스트하다가 운영체제에 영향을 주는 케이스가 발생하면 재부팅하지 않고 문제가 발생하는 부분만 떼어낼 수는 없을까?
ex) 그래픽카드의 드라이버를 잘못설치해서 블루스크린이 떠서 재부팅하는 것과같은 경우 (kernel 영역을 침범해서 운영체제에 crash가 발생해서 컴퓨터 전체에 영향을 미치는 경우)
컨테이너형 가상화 기술의 등장 ::->그래서 운영체제에 영향을 주지 않고 어떻게 실행시킬 수 있을까? 에 대한 해결책으로 container 형 가상화 기술이 탄생함 (1982, chroot), 근데 kernel 영역을 코딩해야 했기 때문에 보편적으로 모두가 사용할만한 기술은 아니었음.
본격적인 사용 : LXC(Linux Container, 2008) -> 그래도 좀 보편적으로 사용할 수 있게 됨 -> 이 기술이 로컬에서만 사용할때는 괜찮은데 다른 컴퓨터로 옮겨서 실행하면 네트워크를 타고 실행할때 문제, storage 상태에 따라서 문제, 보안등의 문제가 있었음.

가상머신 : 내가 쓰고있는 머신을 그대로 가상화 하는 것 (운영체제,cpu, 메모리 등을 똑같이 가지고 있어야 함), 만약 문제가 있는 하드웨어에서 문제가 없는 하드웨어로 옮길때 네트워크 트래픽을 많이 먹음, 운영체제 단에서 완벽한 격리된 상태로 돌아가니까 격리가 잘됨?
즉, 머신 자체를 격리시킨거
크기 : 최소 4GB ~

컨테이너 : 컨테이너∈ 내가만든 app, 운영체제의 일부(kernel을 포함하지는 않고, user land의 일부만 포함하는 구조임(파일시스템, app이 실행될 때 필요한 라이브러리 등))
크기 몇 MB규모로도 가능함 -> 그래서 네트워크 상에서도 빨리 통신되고 처리됨.
즉, app을 실행하기 위한 환경만 격리시킨 것
∴ VM보다는 작은

microservice 아키텍처 - 컨테이너를 잘 결합해서 프로젝트를 만들기 좋다고 하네요

만약 가상머신 내에서 windows, 엔터프라이즈 linux를 쓰면 비용을 내야함.
컨테이너는 운영체제의 일부(파일시스템 등)만 사용하기 때문에 적은비용, 컨테이너 데이터 크기도 작아서 오케스트레이션 하기 좋아서 개발자들이 많이 씀

도커 서비스 개요

도커 서비스 개요

▪도커(Docker)란?

✓컨테이너형 가상화 기술을 구현하기 위한 상주 애플리케이션과 이 애플리케이션을 조작하기위한 명령 줄 도구로 구성된 제품

✓초기에는 LXC를 커널의 가상화 기능 액세스 인터페이스로 사용

✓0.9x 부터 자체 개발한 libcontainer로 LXC 대체

▪도커의 등장

✓프랑스의 dotCloud의 내부 프로젝트로 시작(솔로몬 하익스)

✓Docker, Inc. 설립 - Y 컴비네이터 2010 여름 스타트업 인큐베이터 그룹 (Solomon Hykes/Sebastien Pahl)

✓2013년 3월 오픈소스로 출시

10

도커 구성 요소와 도구

▪구성 요소

✓Docker Engine(run time): dockerd ← docker engine api

✓Docker Client (CLI): docker

✓Docker Objects: Docker container, Docker Image, Docker service

✓Docker Registry

▪도구

✓Docker Compose: docker-compose, docker-compose.yaml

✓Docker Swarm: docker swarm CLI, docker node CLI

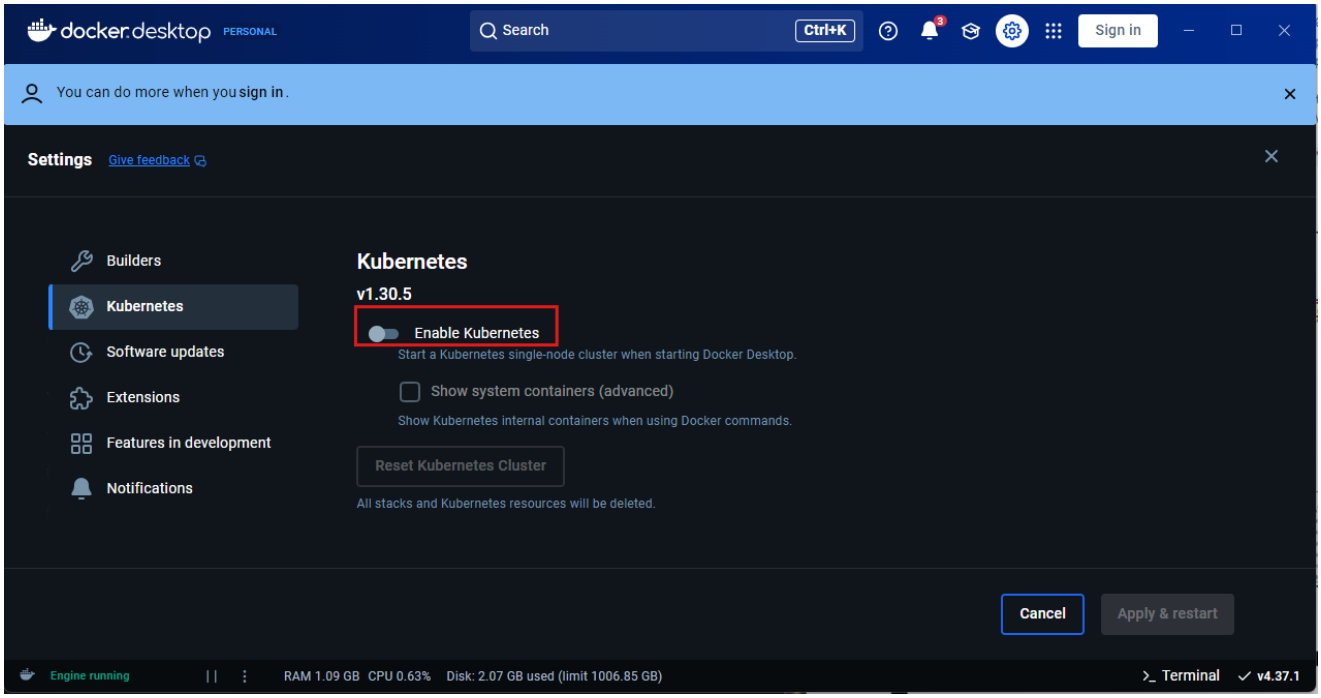
11

dockerd (docker daemon) <- 얘에 실행 명령을 내리는 애는 docker engine api
Docker Client (CLI) : docker의 cmd같은 느낌?
docker image : docker 컨테이너를 만드는 템플릿같은 거 (얘를 실행하면 컨테이너가 됨)
docker registry : 도커 깃 허브같은거

docker compose : one container - one app이 되면 좋은디, one app - multi container가 되는 상황에서
ex) app(wordpress) - container (word press engine, word press DB)인 상황에서 container 두 개가 잘 돌아가게 돌아가도록 설정 파일같은것을 만들어서 의존성을 잘 기술해두면 의존성이 있는 container들을 잘 다룰 수 있다(kill하고 싶을 때 의존성을 고려해서 kill을 하는 등)

docker swarm : 쿠버네티스같은 docker 오케스트레이션 도구

오케스트레이션 서비스 : 여러 곳에 올려진 도커들을 조율하는 일종의 load balancer같은거 같음...



나중에 Enable 시키면 로컬에서 쿠버네티스 실행가능인데, 로컬 자원이 부족하면 많이 느려질거임

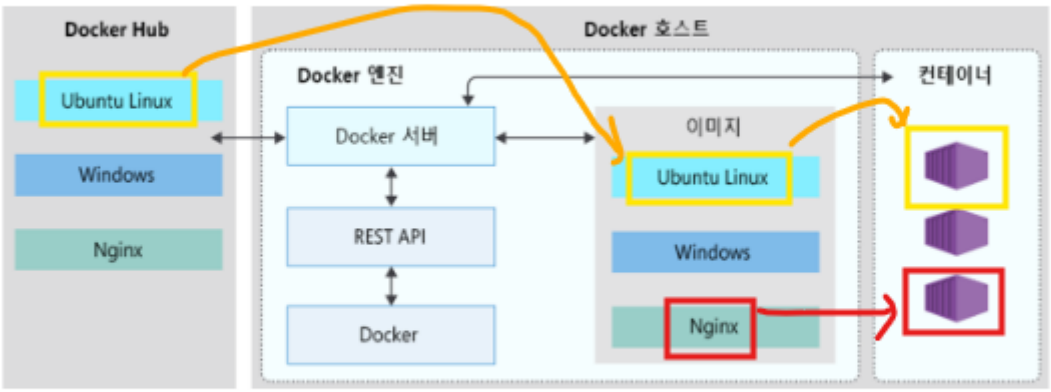
Dockerfile, Docker Image, Docker Container의 관계

12

- Dockerfile
  보라색 : instruction
  파란색 : instruction의 파라미터
  각 라인 명령어의 실행을 포토샵 layering이라고 생각할 수 있음.

container : 실행 중인 상태
image : 실행전 파일의 상태
file : instruction 집합

Docker 환경 아키텍처



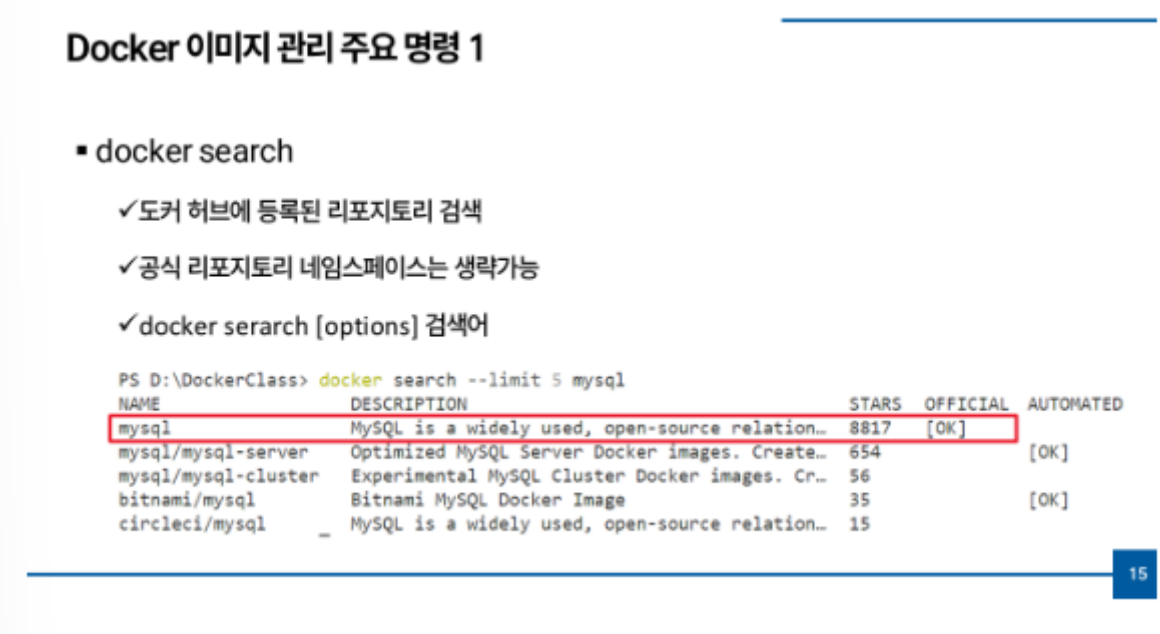
13

Docker hub ≈ git hub

docekr host ≈ local computer

docker server <-> rest api <-> docker ≈ local 에 설치된 git
container ≈ memory

- 이런 유사한 관계에서 볼 수 있듯이 과거의 기술, 현재 사용 중인 유사한 기술이 다른 기술에도 사용될 수 있는 유사성을 가지는 경우가 생각보다 많음.



Res ⊂ Repo
Repository는 중복되면 안되기 때문에 namespace(일종의 구분자)를 붙임, 근데 검색 시는 생략해도 됨

02_docker.azcil

#0. sudo 없이 도커 사용

```
sudo usermod -aG docker tony

sudo -su tony

sudo chmod 666 /var/run/docker.sock
```

#1. 이미지 검색

`docker search --limit 5  mysql`

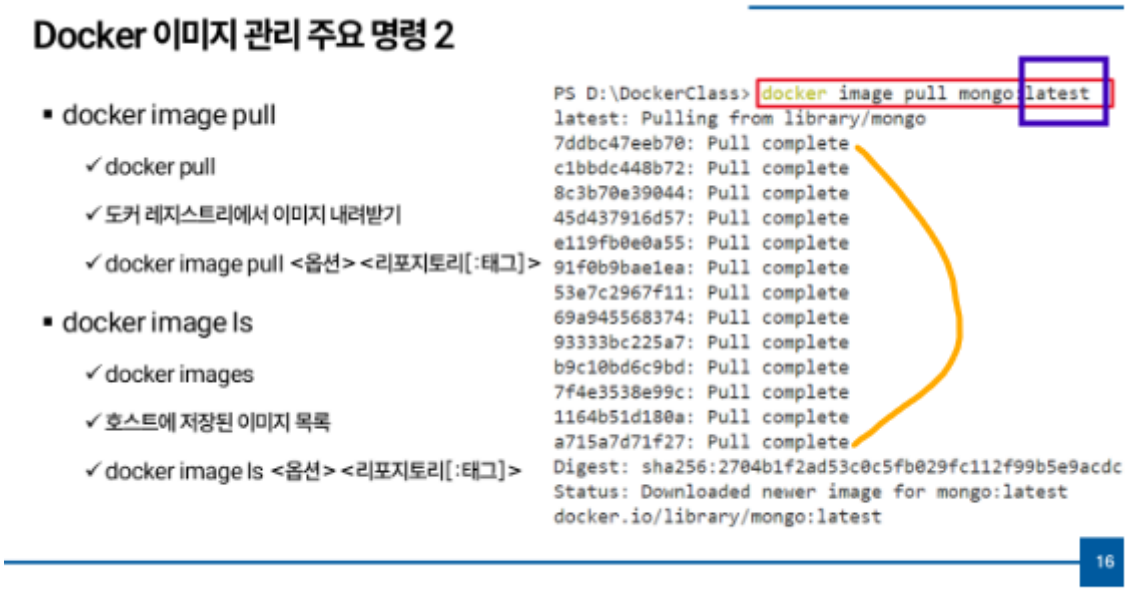
```
C:\Users\qw1\Desktop\Fly ai\DevOps\Container>docker search --limit 5  mysql
NAME                DESCRIPTION                                     STARS     OFFICIAL

mysql               MySQL is a widely used, open-source relation…   15583     [OK]
bitnami/mysql       Bitnami container image for MySQL               123
circleci/mysql      MySQL is a widely used, open-source relation…   30
cimg/mysql                                                          3
bitnamicharts/mysql Bitnami Helm chart for MySQL                    0

C:\Users\qw1\Desktop\Fly ai\DevOps\Container>
```

docker에 있는 image 중 악의적인 image도 있기 때문에 STARS가 많은, 공식적인 image를 선택해야함.

#2. 이미지 내려 받기



- 파란색 : 버전 (latest : 최신)
- pull하는게 주황색으로 긴 이유 : 아까 말했던 layering -> 네트워크 부담이 적음

##Docker Hub 레지스트리에서 ASP.NET Sample 앱 이미지 검색

```
docker pull mcr.microsoft.com/dotnet/samples:aspnetapp

docker pull mcr.microsoft.com/azuredocs/aci-helloworld
```

#3. 이미지 목록 확인

```
docker image ls

docker images
```

#4. 이미지에 태그(새로운 이름) 추가

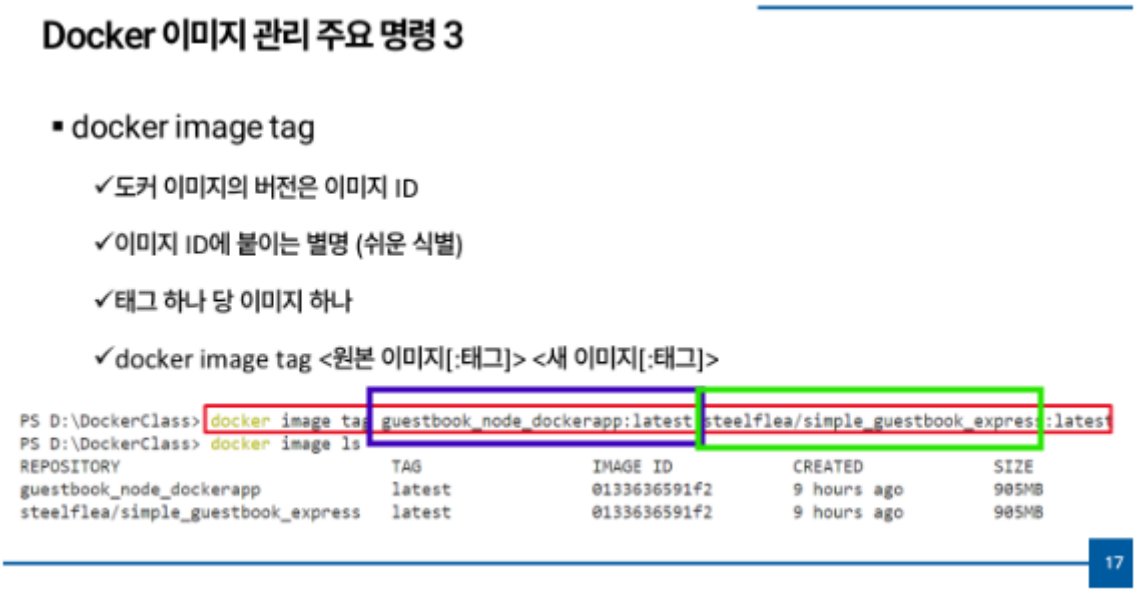


image ID를 보면 둘이 같은 image라는 것을 알 수 있음.
Tag 기능 : private resitory에 image를 올릴 때 많이 씀.

```
docker image tag mcr.microsoft.com/azuredocs/aci-helloworld
hello.docker/aci-helloworld:latest

docker tag mcr.microsoft.com/dotnet/samples:aspnetapp hello.docker/samples
```

`mcr.microsoft.com/azuredocs/aci-helloworld` -> `hello.docker/aci-helloworld:latest`
얘로 별명을 한거임 (iamge ID를 보면 동일할거임)

Dockerfile과 이미지 빌드

Dockerfile의 구조

Dockerfile 문법 1

- 전용 도메인 언어(명령, 인스트럭션)로 이미지 구성 정의
- FROM <이미지명>:<태그>
- RUN <도커 컨테이너 안에서 실행할 명령>
- COPY <원본 파일/디렉터리> <대상 디렉터리>
- CMD
  - ✓문법 1 – CMD ["명령1" "인자1" "인자 2" …]
  - ✓문법 2 – CMD <명령> <인자1> <인자2>

Dockerfile > ...

```
1  FROM node:8.9.3-alpine
2  RUN mkdir -p /usr/src/app
3  COPY ./app/* /usr/src/app/
4  WORKDIR /usr/src/app
5  RUN npm install
6  CMD node /usr/src/app/index.js
```

20

- FROM
  - `node` : 이미지 명 / `8.9.3-alpine` : 버전
- RUN : build하는 시점에서 실행
  - `-p` : 폴더를 만들 때 parents 폴더도 만들어라
- COPY :
- WORKDIR : 작업 디렉토리 지정
- RUN
  - `npm install` : 노드 모듈 생성
- CMD : container화 될때 실행
  - `node /usr/src/app....` : 노드 명령어로 실행
  - 문법 1의 "명령" 을 쓰는 경우는 명령에 공백이 있는 경우

Dockerfile 문법 1

- ADD
  - ✓문법 1 – ADD <원본> <대상>
  - ✓문법 2 (공백이 있는 경우) – ADD ["<원본>", "<대상>"]
- ENV <환경변수명> <값>
- EXPOSE <포트1> <포트2>
- WORKDIR <작업 디렉터리 경로>
- LABEL <이미지를 만든 사람에 대한 정보>

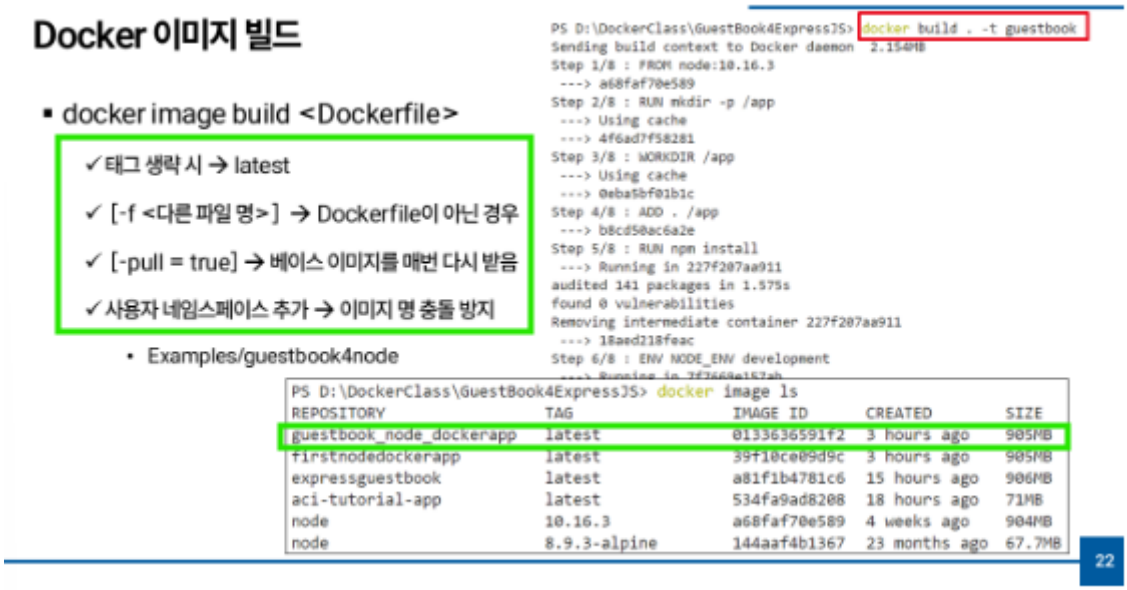
Dockerfile > ...

```
1  FROM node:10.16.3
2  RUN mkdir -p /app
3  WORKDIR /app
4  ADD . /app
5  RUN npm install
6  ENV NODE_ENV development
7  EXPOSE 3000 80
8  CMD ["npm","start"]
```

21

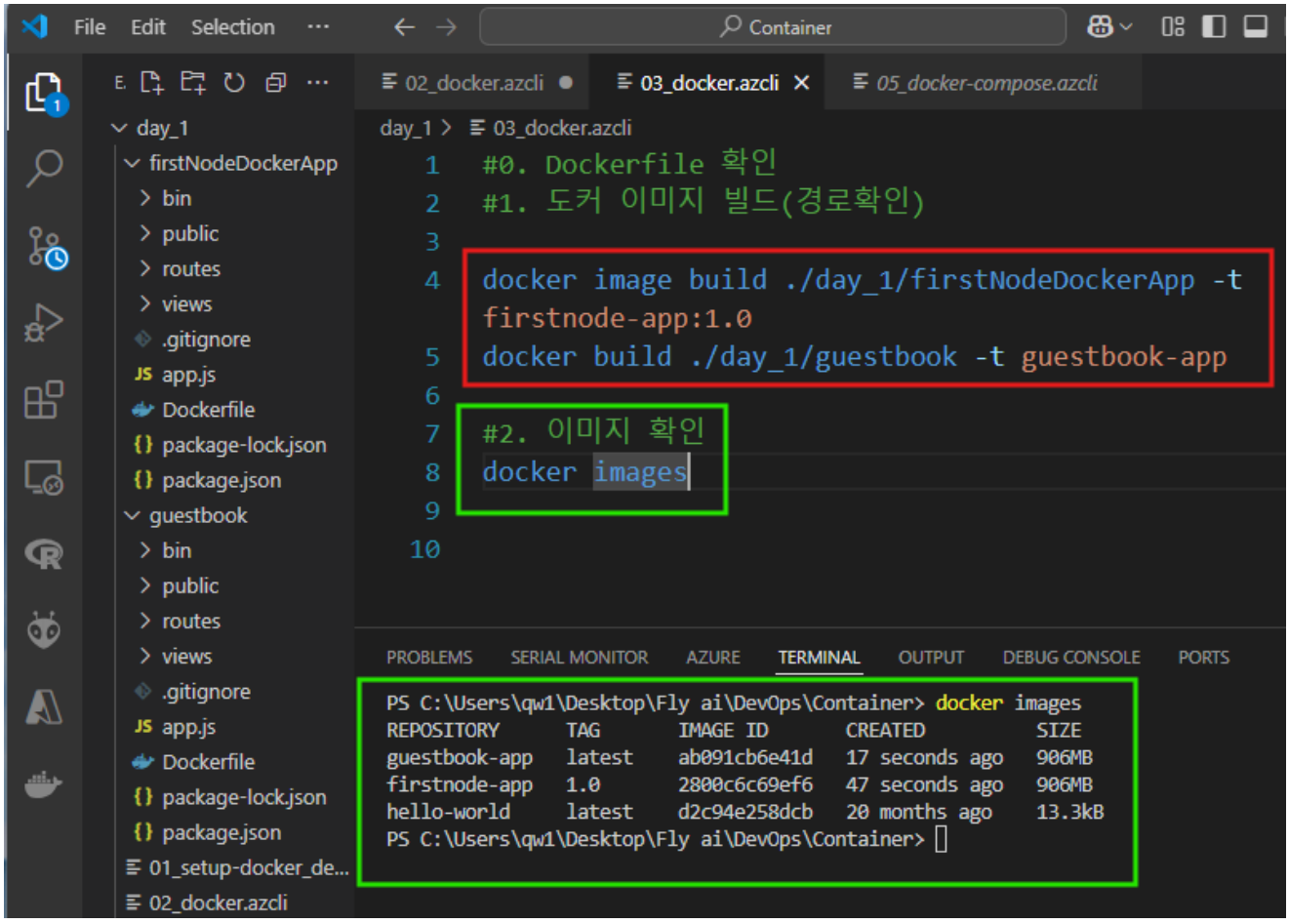
- ADD
  - `. /app` : `.` (원본에서) `/app` 을 추가함

ADD는 원본이 Remote 위치일 경우에 사용 (ex 원본이 GitHub에 있는 상황)



Dockerfile은 관례 상 파일명을 `Dockerfile`로 함
근데 만약 다른 이름을 할려면 `-f<다른 파일명>`

`docker build . -t guestbook` 에서 `.`은 현재 위치



- 빨간색 실행하고 초록색 실행하면 `guestbook-app, firstnode-app`이 images에 생김
참고 만드는 환경에 따라서 가지고 오는 라이브러리가 달라서 SIZE가 다를 수 있음 (윈

도우 10과 윈도우 11에서 가져오는 라이브러리 버젼같은게 조금씩 다를 수 있음), 파일 시스템마다 할당사이즈가 달라서 다르게 보일 수도 있음.