

빅데이터를 지탱하는 기술

Week 2

Linux & Docker

23.07.26 / 엄소은

CONTENTS

01. Linux

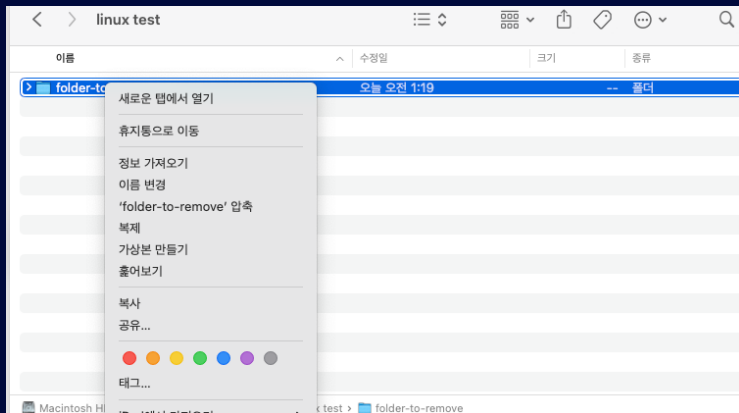
- 운영체제
- Linux 명령어 다루기

02. Docker

- Docker 개념
- Docker 명령어
- Docker 실습

Command Line 을 배우는 이유 ?

- cd, ls, conda, pip 등등 command line 을 배우는 이유가 무엇일까?
- 사용자 인터페이스(GUI) 를 사용하는 것보다 Command line(CLI) 를 사용해서 **훨씬 더 빨리** 할 수 있다 !
- 파일 삭제할 때를 보자.



폴더창 키고.. 삭제할 폴더 찾아서.. 오른쪽 마우스
클릭해서.. 휴지통으로 이동 ...

```
~ ➔ cd linux_test  
~/linux_test ➔ rmdir folder-to-remove  
~/linux_test ➔ mkdir folder-to-remove
```

명령어 3줄만으로 디렉토리 이동해서
폴더 삭제 & 폴더 만들기 할 수 있다 !!
익숙해지면 훨씬 빠르다!!

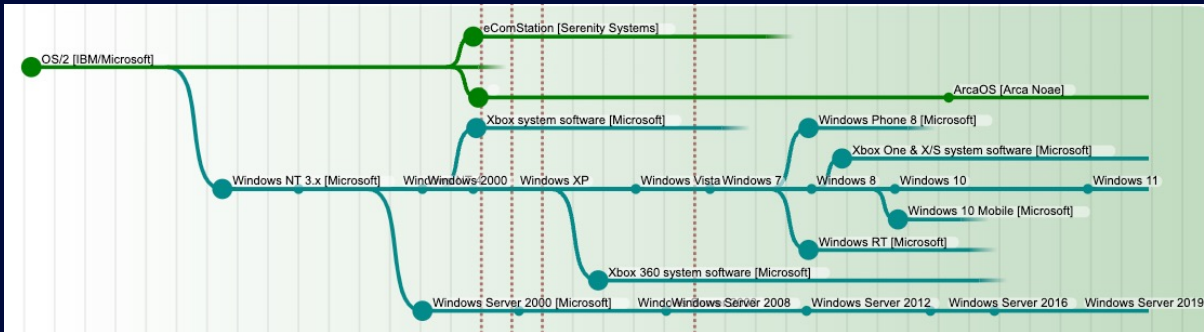
Command Line 을 배우는 이유 ?

- CLI에서 엄청나게 많은 일들을 할 수 있다 !
 - 서버를 시작하는 일, 숨긴 파일(.git) 을 다루는 일, 권한 업그레이드, 데이터베이스 접속 등등.
- 내 컴퓨터에 대한 더 많은 권한이 생긴다 !

1. Linux

운영체제(OS)

- 가장 많이 쓰는 운영체제인 Window 와 Mac OS 의 조상을 찾아보자 !!
- 같은 혈통일 경우 같은 명령어를 쓸 확률이 높다



Window
-> Microsoft NT



Mac OS
-> Unix

Unix 에서 나온 운영체제들은 Mac OS, Linux, Android, Chrome OS, PS4 OS 등이 있다.

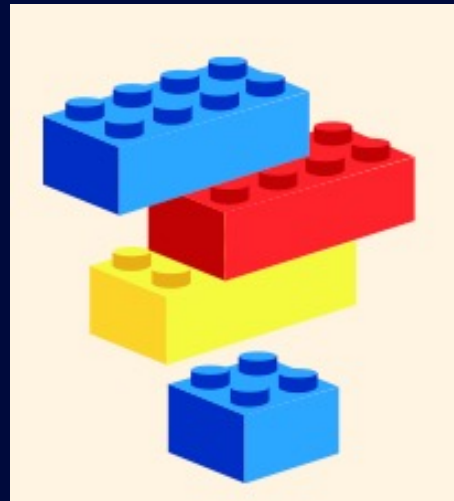
Unix 에 대해 간단하게 알아보자

- 1960년대에 Bell Labs 에서 만들어진 Unix 는 굉장히 많은 운영체제들의 조상 !
- 이때 만든 **디자인 철학**이 지금까지도 내려온다



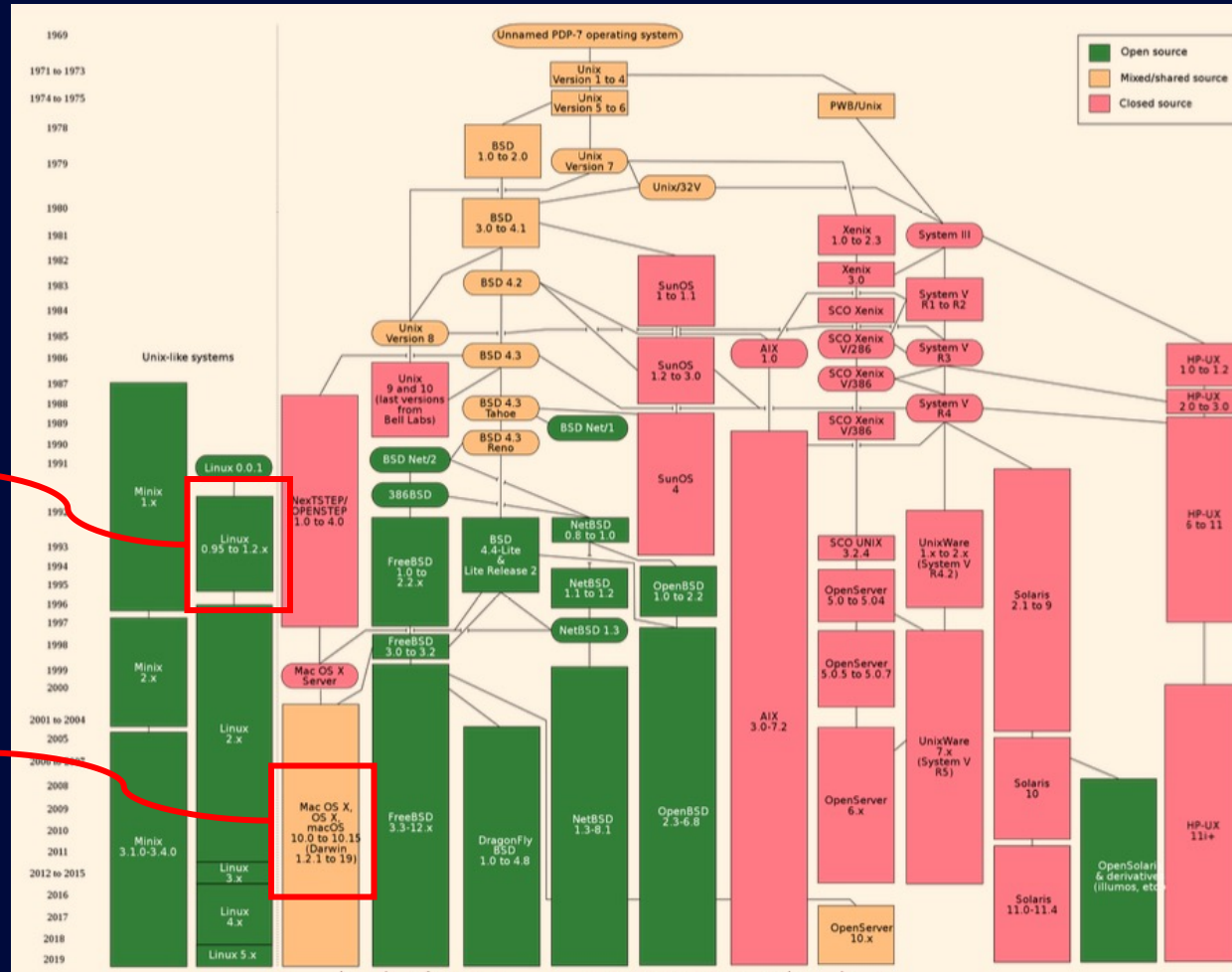
Unix Philosophy

작은 프로그램들을 만들고,
명령어들로 이것을 조합해서
더 복잡한 일을 하게 하는 것



1. Linux

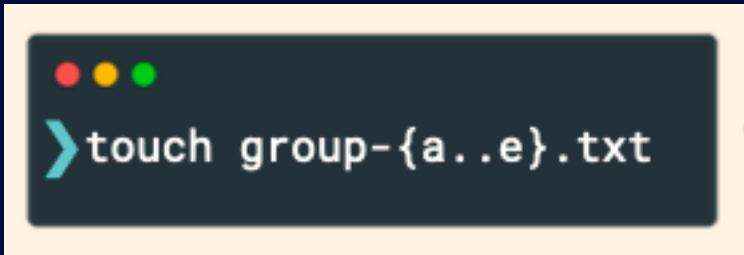
Unix 에 대해 간단하게 알아보자



- 빨간색 : closed software
- 초록색 : Open source

1. Linux

Terminal, Shell, OS ?



Terminal

입출력 환경
그냥 CLI 를 입력하는 곳
사용자와 상호작용하는 인터페이스

→
(터미널에서 셸을
사용하는 것임)



Shell

터미널에서 친 명령어를 받아
실행하고 처리한다
=운영체제에게 넘기고
결과를 받아온다
(명령어 해석기)

→

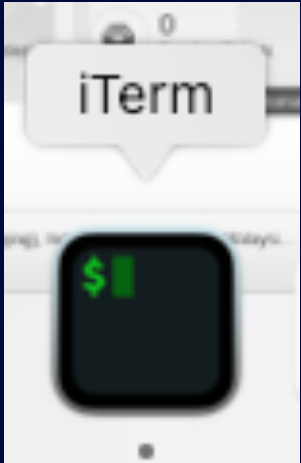


OS

실제 작업을 수행한다

1. Linux

Terminal, Shell, OS ?



나와 상호작용 하는 창
= iTerm(terminal)

```
~ zsh --version  
zsh 5.9 (arm-apple-darwin21.3.0)  
  
~ bash --version  
GNU bash, version 3.2.57(1)-release (arm64-apple-darwin22)  
Copyright (C) 2007 Free Software Foundation, Inc.
```

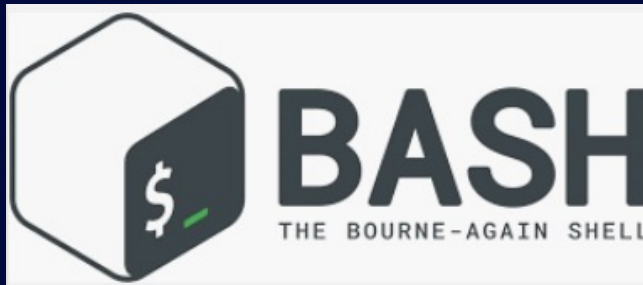
iTerm 에 깔려있는 프로그램 = Shell
(Bash, zsh 등등)

zsh : oh-my-zsh라는 커뮤니티 제공


bash: unix 체제 기본 shell

1. Linux

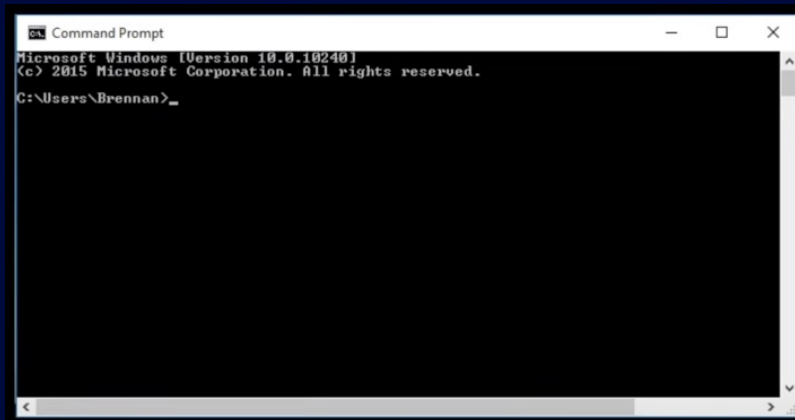
Shell



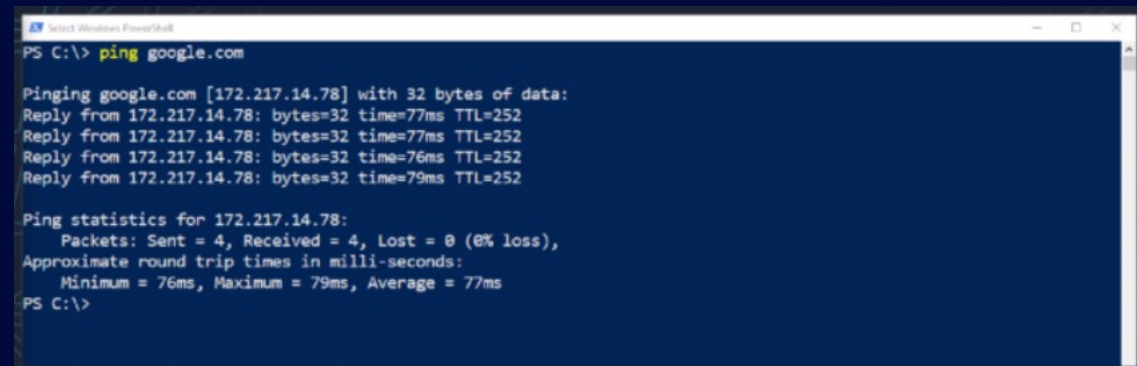
BASH

```
ryo@Chadui-MacBookPro:~  
- 51555fc [gnu-utils] Include `ghostname` if present (#10867)  
Bug fixes:  
- ed411d3 [httpie] Remove unnecessary `https` alias (#10856)  
- a26d72b [init] Check for unsafe directories in `fpath` (#10672)  
- a879ff1 [nvm] Support path from Apple Silicon Homebrew (#10875)  
You can see the changelog with `omz changelog`  
  
Hooray! Oh My Zsh has been updated!  
To keep up with the latest news and updates, follow us on Twitter: @ohmyzsh  
Want to get involved in the community? Join our Discord: Discord server  
Get your Oh My Zsh swag at: Planet Argon Shop  
base at 20:55:38
```

zsh(z-shell)



CMD



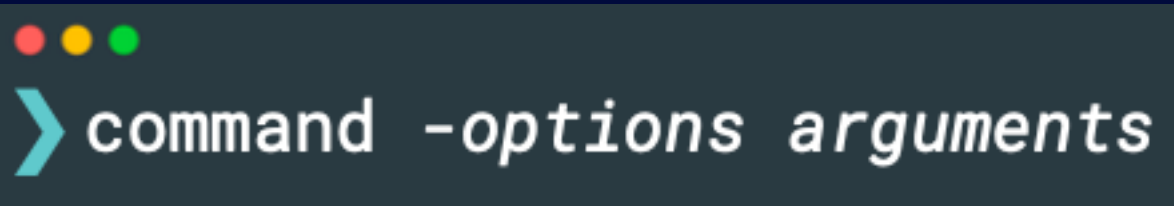
Powershell

-> Window 에서 가장 강력한 shell

1. Linux

Command 기본

1. 모든 command 는 (거의) option 과 함께 사용된다



```
~> python --help  
usage: /Library/Frameworks/Python.framework/Versions/3.6/bin/python [options]
```

긴 option 은 --(dash 2개)

```
~> python -h  
usage: /Library/Frameworks/Python.framework/Versions/3.6/bin/python [options]
```

짧은 option 은 -(dash 1개)

2. command 에서 대소문자 구분은 중요하다

```
~> Python
```

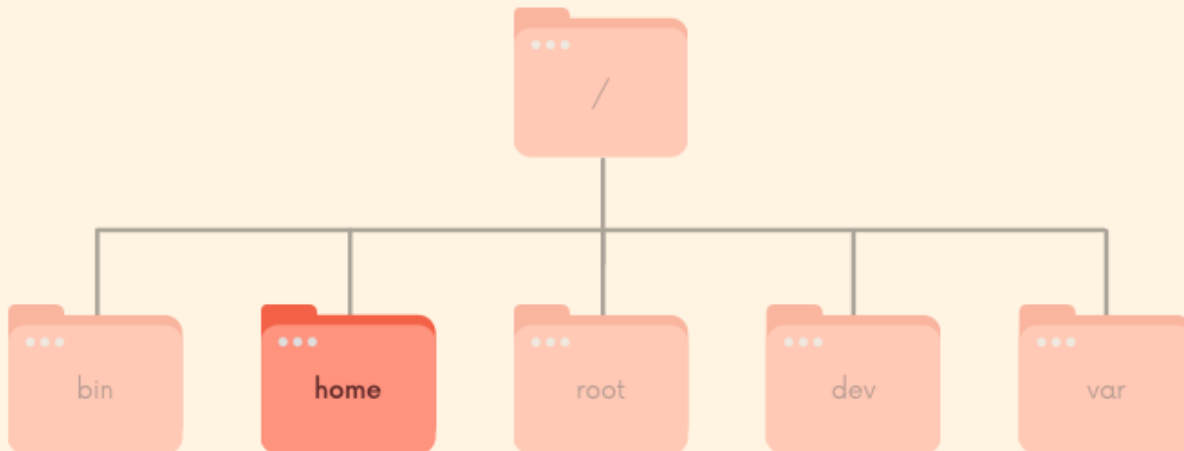
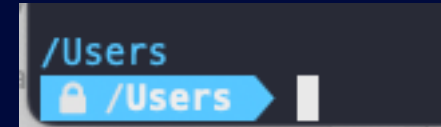
1. Linux

파일 시스템 탐색



The Root Directory

The starting point for the file system is the root folder.
We call it the root, but its actual directory name is "/"



- / : Root directory
- ~ : Home directory

1. Linux

파일 시스템 탐색

```
~ ➔ pwd  
/Users/soeun-uhm
```

pwd : 내가 지금 어느 위치에 있나?

```
~/yonsei/DSL ➔ ls  
DSL 22-2  
DSL_23-1  
Vision Transformer
```

ls : 그 디렉토리 내에 있는
모든 콘텐츠를 보고 싶을 때

```
~/yonsei/DSL ➔ ls -a  
.  
..  
.DS_Store  
.Rhistory  
DSL 22-2  
DSL_23-1  
Vision Transformer
```

ls -a : 숨겨진 폴더도 보여줌

```
~/yonsei/DSL ➔ ls -l  
total 8  
drwxr-xr-x 10 soeun-uhm staff 320 1 23 2023 DSL 22-2  
drwxr-xr-x 31 soeun-uhm staff 992 7 25 18:20 DSL_23-1  
drwxr-xr-x 12 soeun-uhm staff 384 11 17 2022 Vision Transformer
```

ls -l : 콘텐츠에 대한 자세한 정보도 보여줌

1. Linux

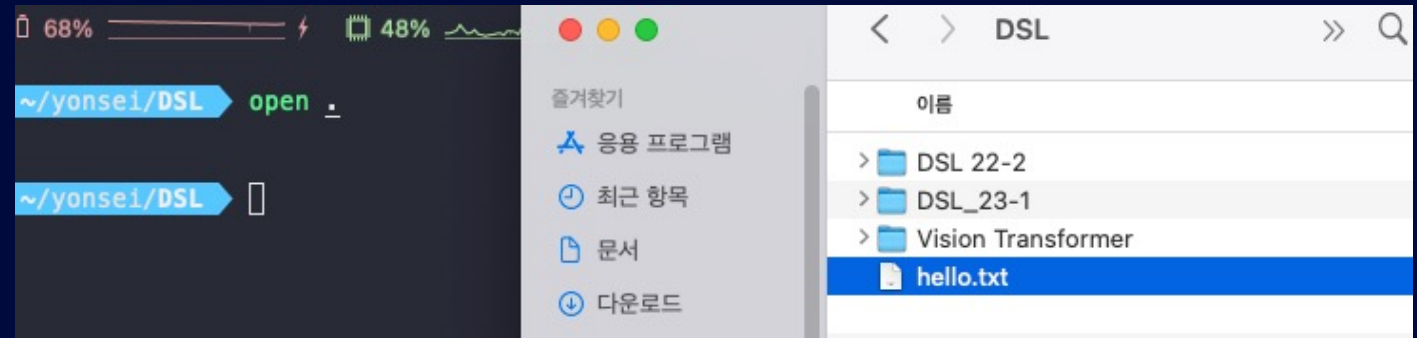
파일 시스템 탐색

```
~/yonsei/DSL ➔ cd DSL_23-1
~/yonsei/DSL ➔
~/yonsei/DSL/DSL_23-1 ➔
```

cd : 내부에 있는 디렉토리로 이동

```
~/yonsei/DSL/DSL_23-1 ➔ cd ..
~/yonsei/DSL ➔
```

cd .. : 바로 전 디렉토리로 이동



open . : 현재 디렉토리의 폴더창 열기

```
~/yonsei/DSL ➔ find . -type file -name "*.txt"
./DSL_23-1/시각적 패턴 인식 /2절 지수 평형 방법 (단순 지수 +이차식) 코딩.txt
./DSL_23-1/시각적 패턴 인식 /3절 계층적 지수 평형 방법 코딩.txt
./DSL_23-1/ML0ps/Udemy/dummy/test/test.txt
./DSL_23-1/ML0ps/azure-webapp/requirements.txt
./DSL_23-1/ML0ps/js-pipeline-project/app/requirements.txt
./DSL_23-1/ML0ps/github-actions-project-github-actions-cml/requirements.txt
./DSL_23-1/ML0ps/github-actions-project-github-actions-cml/metrics.txt
```

find . -type file -name "*.txt" : 디렉토리 내에서 이름 상관없이 .txt 로 끝나는 파일 찾아줘
(pws): get-childitem -Directory -Filter "*.txt" -Recurse

1. Linux

파일 시스템 탐색

```
~ ➤ which python  
python: aliased to /Library/Frameworks/Python.framework/Versions/3.10/bin/python3
```

which: 내가 실행하고자 하는 프로그램이 어디있는가

(pws) : get-command python

1. Linux

파일 시스템 탐색

```
~/yonsei/DSL ➤ mv hello.txt new-folder/
```

```
~/y/DSL/new-folder ➤ mv hello.txt ..
```

mv 파일명 폴더명 : 파일을 다른 폴더로 옮긴다

```
~/yonsei/DSL ➤ cp hello.txt new-folder/
```

cp 파일명 폴더명 : 파일을 다른 폴더로 복사한다

```
~/yonsei/DSL ➤ rm hello.txt
```

rm: 파일 삭제

1. Linux

파일 시스템 탐색

```
~/yonsei/DSL ➤ echo "hello world" > hello.txt
```

```
~/yonsei/DSL ➤ grep "world" *.txt  
hello world
```

```
~/yonsei/DSL ➤ grep -ni "world" *.txt  
1:hello world
```

★★★ grep " 찾고 싶은 대상" 찾을 파일
(pws): select-string *.txt -pattern "world"

```
~/y/D/DSL/Mo/AuToeic ➤ main !3 !3 ?3 ➤ grep -ni "Create" *.py  
clip.py:5:from img2toeic import loadimages, CreateToeic  
clip.py:18:    text_prompts = CreateToeic()  
img2toeic.py:52:def CreateToeic():  
img2toeic.py:97:    answer , toeic_problem = CreateToeic()  
main.py:2:from img2toeic import CreateToeic  
main.py:24:CreateToeic()
```

```
~/y/D/DSL/Mo/AuToeic ➤ main !3 !3 ?3 ➤ grep -nir "toeic" .  
./img2toeic.py:46:    toeic_img = Image.open(path)  
./img2toeic.py:47:    images_list.append(toeic_img)  
./img2toeic.py:52:def CreateToeic():  
./img2toeic.py:93:    plt.savefig(f"Toeic problem_{timestamp}.jpg")  
./img2toeic.py:97:    answer , toeic_problem = CreateToeic()  
./img2toeic.py:99:    print(toeic_problem)
```

1. Linux

컨텐츠 생성과 제거

```
~/yonsei/DSL ➤ touch hello.txt  
  
~/yonsei/DSL ➤ ls  
DSL 22-2  
DSL_23-1  
Vision Transformer  
hello.txt
```

touch : 새로운 파일 만들기
(pws): new-item

```
~/yonsei/DSL ➤ file hello.txt  
hello.txt: empty
```

```
~/yonsei/DSL ➤ file hello.txt  
hello.txt: ASCII text
```

file: 그 컨텐츠에 대한 정보

```
~/yonsei/DSL ➤ cat hello.txt  
Hi I'm soeun
```

cat: 그 컨텐츠에 어떤 내용이 있는지 확인

```
~/yonsei/DSL ➤ mkdir new-folder  
  
~/yonsei/DSL ➤ ls  
DSL 22-2  
DSL_23-1  
Vision Transformer  
hello.txt  
new-folder
```

mkdir : 새로운 폴더 생성

```
~/yonsei/DSL ➤ rmdir new-folder  
  
~/yonsei/DSL ➤ ls  
DSL 22-2  
DSL_23-1  
Vision Transformer  
hello.txt
```

rmdir : 새로운 폴더 생성

컨텐츠 생성과 제거

```
~/yonsei/DSL ➤ cat hello.txt
Hi I'm soeun
>
~/yonsei/DSL ➤ echo "hello world"
hello world
>
~/yonsei/DSL ➤ echo "hello world" > hello.txt
>
~/yonsei/DSL ➤ cat hello.txt
hello world
```

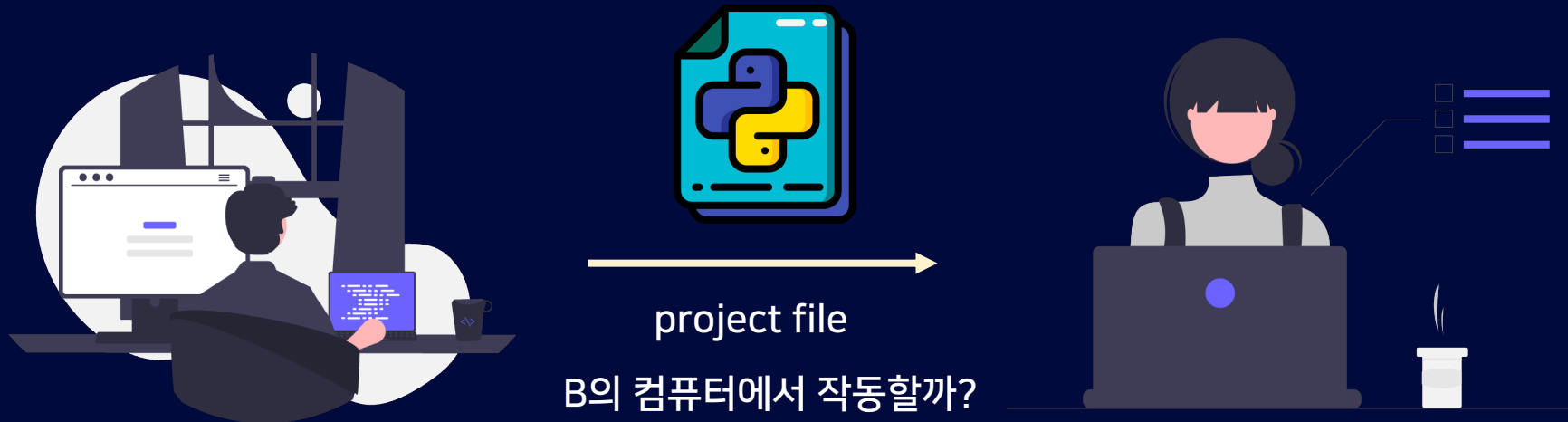
```
~/yonsei/DSL ➤ echo "bye world" >> hello.txt
>
~/yonsei/DSL ➤ cat hello.txt
hello world
bye world
```

echo "내용" > file명 : 그 파일안에 덮어쓰우고 저장

echo "내용" >> file명 : 원래 파일 아래 내용 추가

2. Docker

Docker 는 왜 필요할까?



팀원A

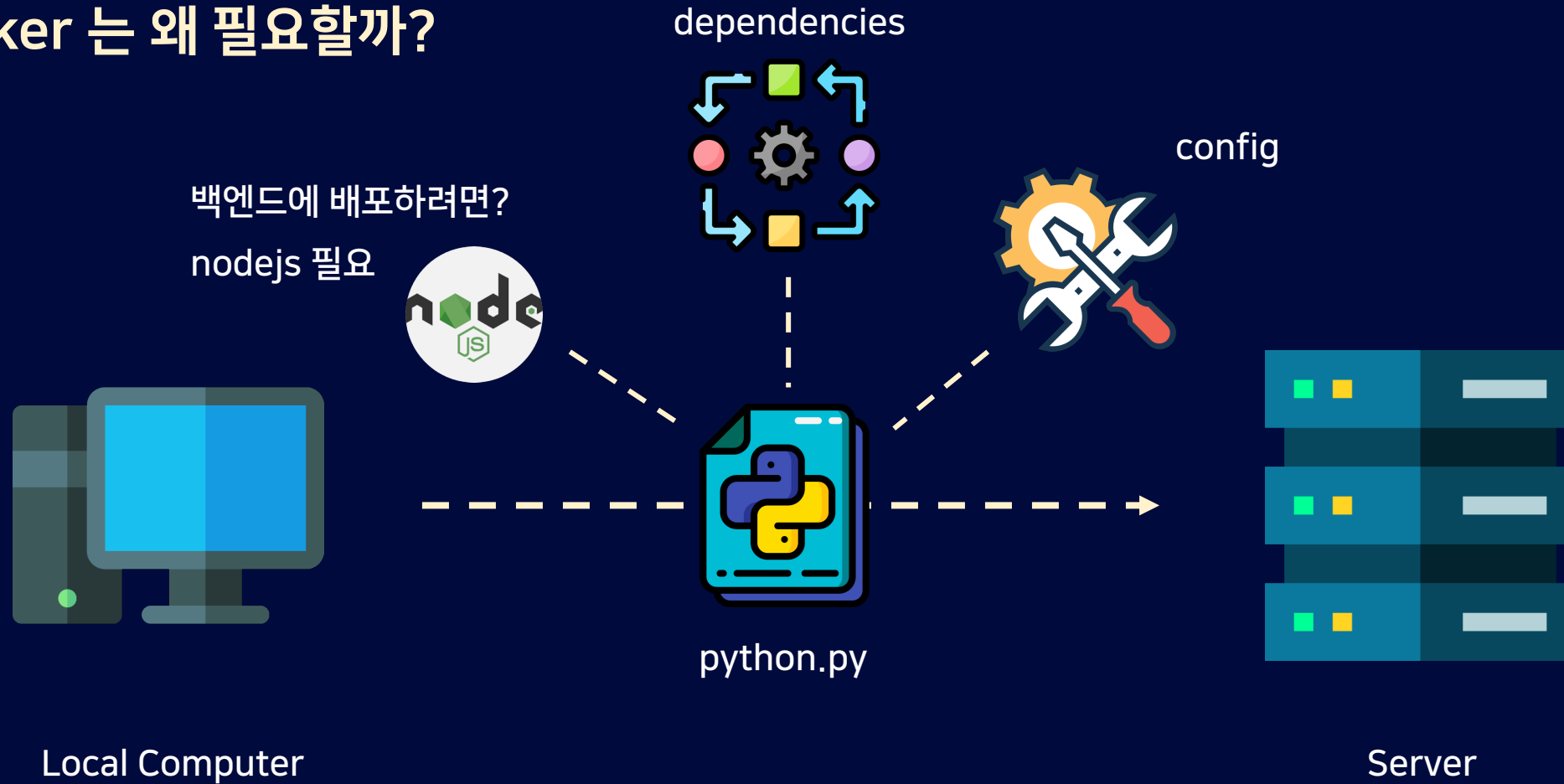
- python=3.8
- transformers=4.27.1
- torch=1.12.0

팀원B

- python=3.10
- transformers=4.30.0
- torch=1.16.0

2. Docker

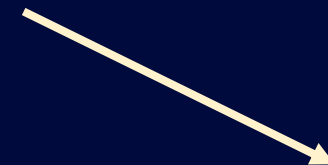
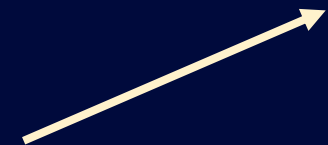
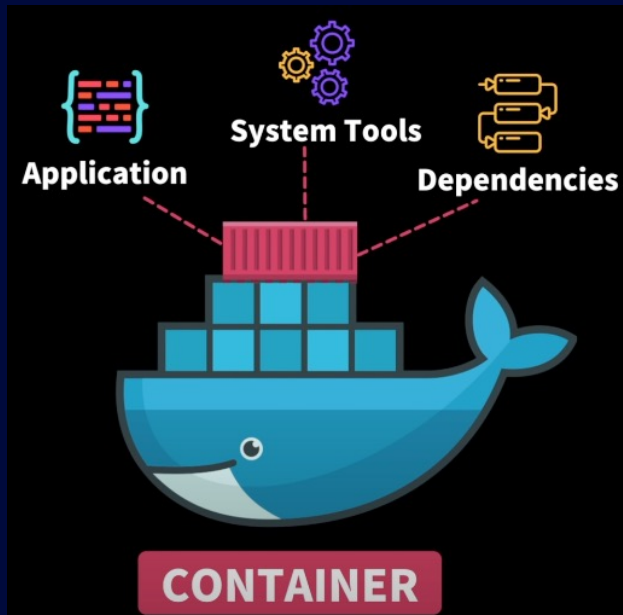
Docker 는 왜 필요할까?



2. Docker

Docker 개념

- Docker: 어플리케이션을 패키징 할 수 있는 툴 !
- Docker container 안에 하나의 프로젝트에 필요한 application, 환경설정, dependencies 를 모두 담는다
- 다른 사람 /서버에서 이것저것 다른 것들을 설치할 필요 없이 도커파일만 run 하면 된다 !



2. Docker

Docker Container 만드는 방법



Dockerfile 로 Image를 만든다

Image를 이용해 Container 를 만든다

(Image=설계도면, Container= 건물)

하나의 Image로 여러개의 Container 를 만들 수 있다

2. Docker

Dockerfile



Dockerfile



Copy files



Install dependencies



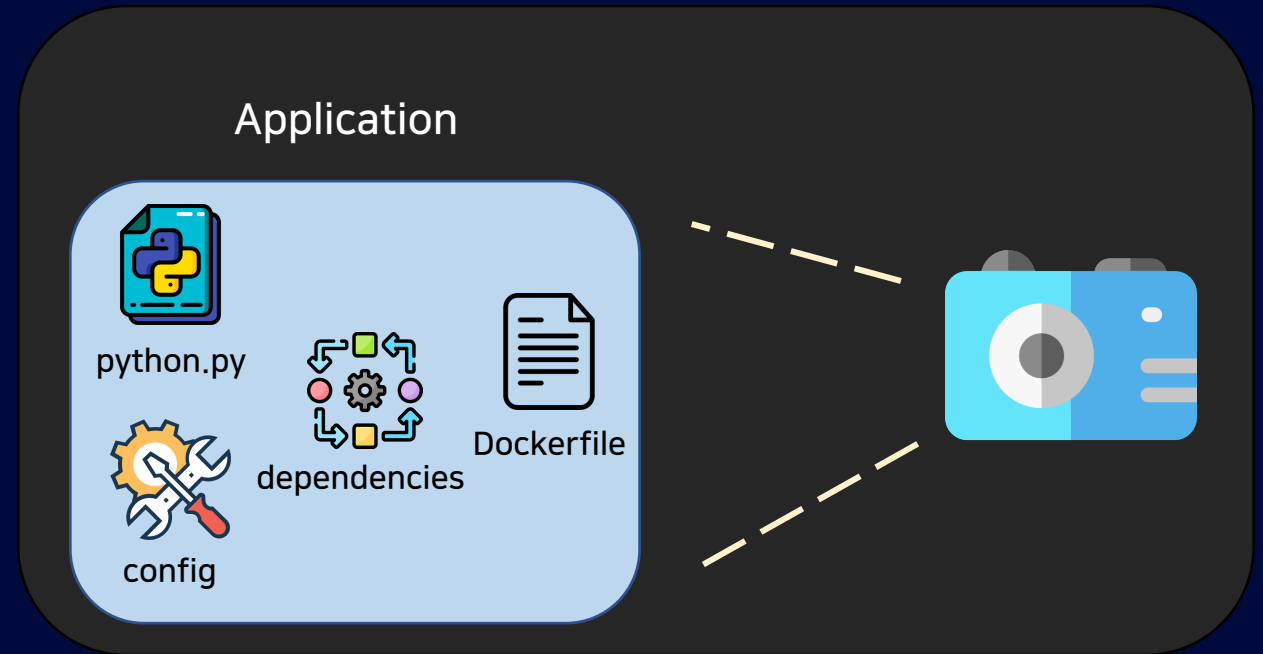
Set environment variables



Run setup scripts

2. Docker

Docker Image



현재 구동되고 있는 내 프로젝트 상태를
찍어서 이미지로 보관해둔다!
한번 빌드한 이미지는 변경 불가능하다

2. Docker

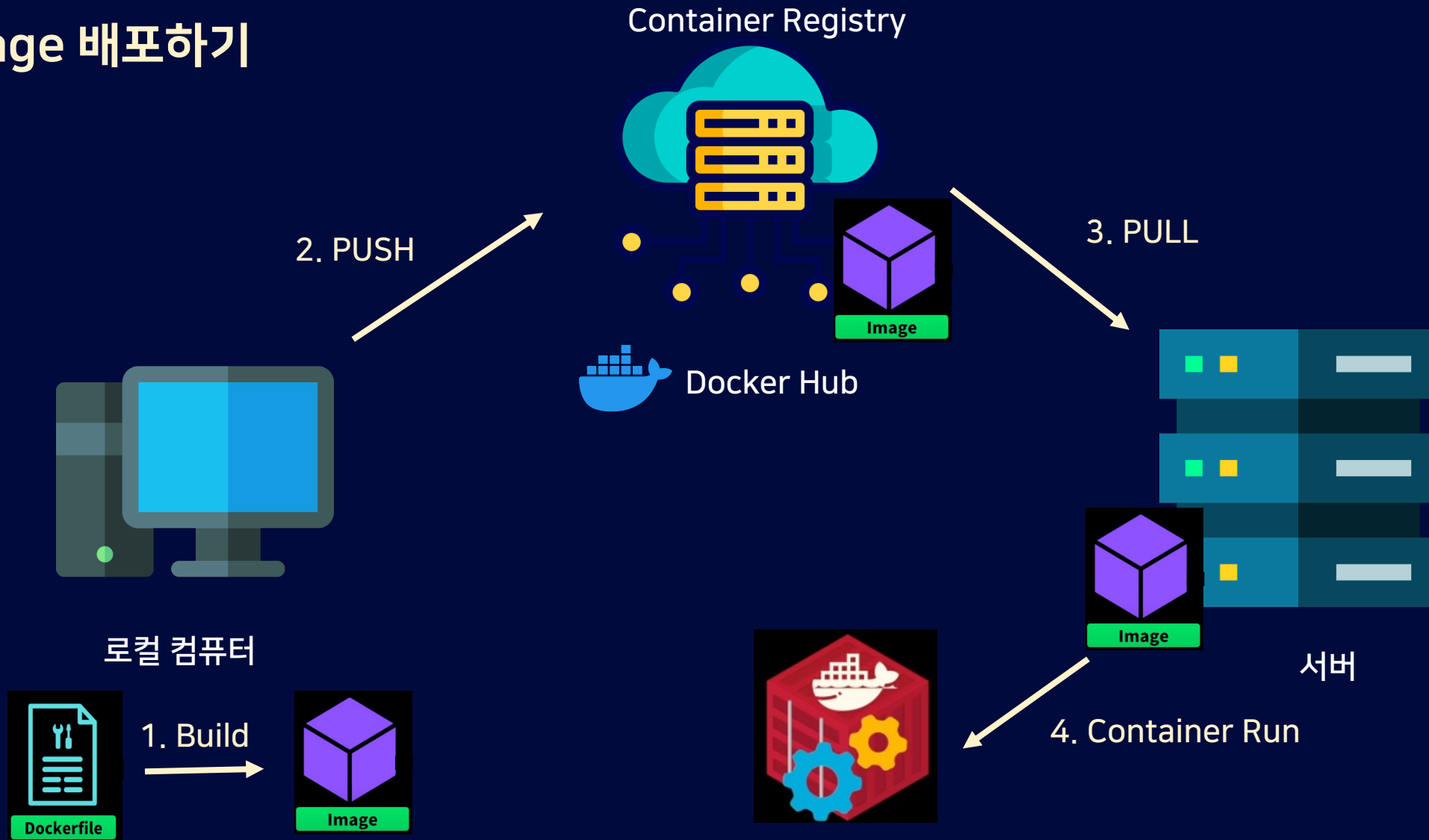
Docker Container



Image 를 토대로 run 하는 것이 docker containers
(객체지향 개념으로 생각해보면, Image = Class, Container = Instance)

2. Docker

Image 배포하기



2. Docker

Dockerfile 만들어보기

```
You, 3주 전 | 1 author (You)  
FROM python:3.8  
  
WORKDIR /usr/src/app  
  
ADD main.py .  
COPY requirements.txt ./  
  
RUN pip install -r requirements.txt  
  
COPY . .  
  
CMD [ "python", "./main.py" ]
```

-> FROM : Base Image (가장 기본이 되는 프로그램)

-> WORKDIR : 컨테이너 안에서 어떤 경로에서 실행할 것인지

-> ADD, COPY : 현재 호스트의 main.py 와 requirements.txt 파일을 복사해라

-> RUN: 필요한 dependencies 를 설치해라

-> COPY : 현재 디렉토리와 모든 파일들을 컨테이너의 작업 디렉토리로 복사

-> CMD : 컨테이너가 실행될때 이 명령어로 실행. (main.py 실행)

2. Docker

Dockerfile 명령어

You, 3주 전 · 1 author (You)

FROM `python:3.8`

Set the `baseImage` to use for subsequent instructions. `FROM` must be the first instruction in a `Dockerfile`.

FROM `baseImage`

FROM `baseImage:tag`

FROM `baseImage@digest`

[Online documentation](#)

WORKDIR `/usr/src/app`

You, 3주 전 · 0703 update

Set the working directory for any subsequent `ADD`, `COPY`, `CMD`, `ENTRYPOINT`, or `RUN` instructions that follow it in the `Dockerfile`.

WORKDIR `/path/to/workdir`

WORKDIR `relative/path`

[Online documentation](#)

2. Docker

Dockerfile 명령어

Copy files, folders, or remote URLs from **source** to the **dest** path in the image's filesystem.

```
ADD hello.txt /absolute/path
```

```
ADD hello.txt relative/to/workdir
```

[Online documentation](#)

```
ADD main.py .
```

You, 3주 전 • 0703 update

Copy files or folders from **source** to the **dest** path in the image's filesystem.

```
COPY hello.txt /absolute/path
```

```
COPY hello.txt relative/to/workdir
```

[Online documentation](#)

```
COPY requirements.txt ./
```

You, 3주 전 • 0703 update

```
WORKDIR /usr/src/app
```

Execute any commands on top of the current image as a new layer and commit the results.

```
RUN apt-get update && apt-get install -y curl
```

[Online documentation](#)

```
RUN pip install -r requirements.txt
```

[Dockerfile reference](#)

2. Docker

Dockerfile Layer

```
You, 3주 전 | 1 author (You)
FROM python:3.8

WORKDIR /usr/src/app

ADD main.py .
COPY requirements.txt ./

RUN pip install -r requirements.txt

COPY . .

CMD [ "python", "./main.py" ]
```

<- Layer 0



<- Layer 6

Update Frequency

Layer 0 이 바뀌면 그 위에 있는 모든 layer 를 다시 빌드 해야 함

Layer4만 바뀌면 Layer0,1,2,3은 캐시 파일 그대로 사용

가장 많이 바뀌는 것(소스파일)을 맨 아래 Layer 에 두기

2. Docker

Docker Image Build

```
~/yonsei/DSL/DSL 23-1/naver/AutoDQ_TeamA main*  
~/y/D/DSL_/naver/AutoDQ_TeamA main !2 docker build -t python-nf .
```

맨뒤에 . 까먹지 말기 !! (현재 디렉토리 의미)


`docker build -t <Image 이름> .`

2. Docker


Docker Hub 에 올리기


```
~/yonsei/DSL/DSL_23-1/naver/AutoDQ_TeamA main*  
~/y/D/DSL_/naver/AutoDQ_TeamA main !2 docker push soeunuhm/autodq:latest
```

docker push <Docker 사용자 이름>/<image이름>:<tag이름>

 soeunuhm / autodq

Description

Autodq_TeamA docker image 

 Last pushed: 23 days ago

Docker commands



To push a new tag to this repository,

```
docker push soeunuhm/autodq:tagname
```

[Public View](#)

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 latest		Image	6 minutes ago	23 days ago

[See all](#) [Go to Advanced Image Management](#)

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

[Upgrade](#)

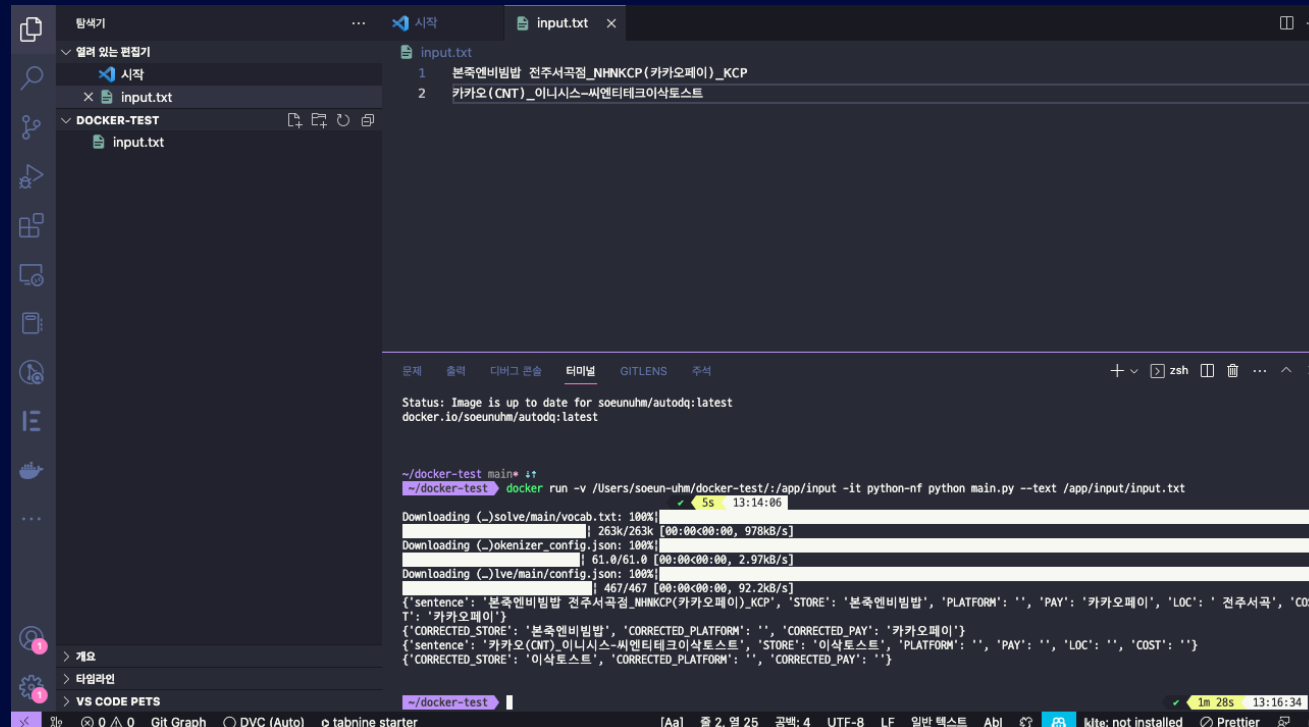
2. Docker

Docker Container 실행

(2023-1 AutoDQ Project)

[AutoDQ Docker Image](#)

```
~/docker-test$ docker pull soeunuhm/autodq:latest
latest: Pulling from soeunuhm/autodq
Digest: sha256:ebcce9082063d54f1f9cf53e6933c95357550b3326e1bac8ff4e80be1800912e
Status: Image is up to date for soeunuhm/autodq:latest
docker.io/soeunuhm/autodq:latest
```



실행에 필요한 input.txt 파일만 있는 폴더인데도
docker 가 실행되었다!

`docker run -v <현재 내 디렉토리>/:/app/input -it
soeunuhm/autodq:latest python main.py --
text /app/input/input.txt`

3. Reference

- 드림코딩 도커 정리, <https://www.youtube.com/watch?v=LXJhA3VWXFA>
- Udemy , Docker&Kubernetes
- Udemy, Linux guide