

이제는, 딥러닝 개발환경도
Docker로 올려보자!!

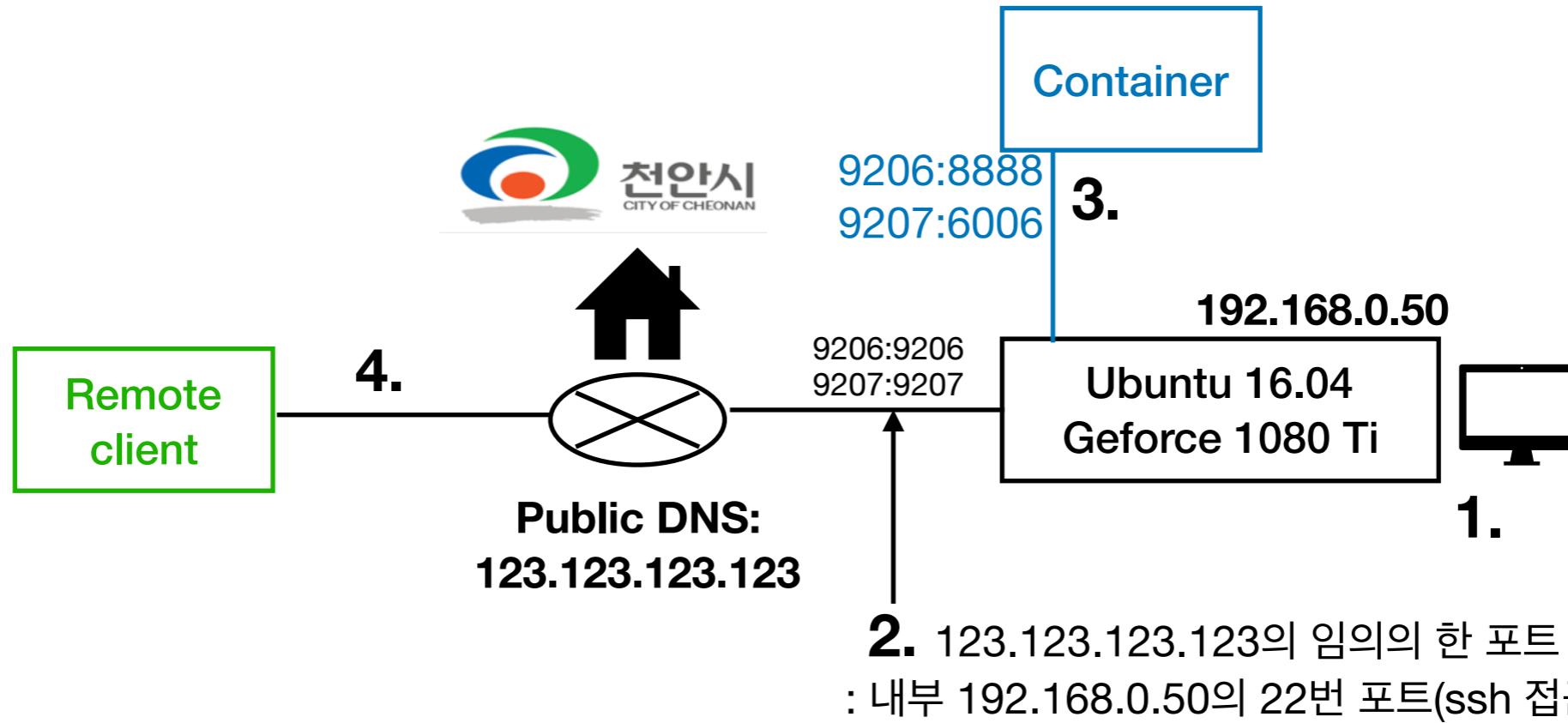
이상수

예상 독자 및 청자:

딥러닝은 배우고 싶고, 환경설정은 어렵기만 하고..

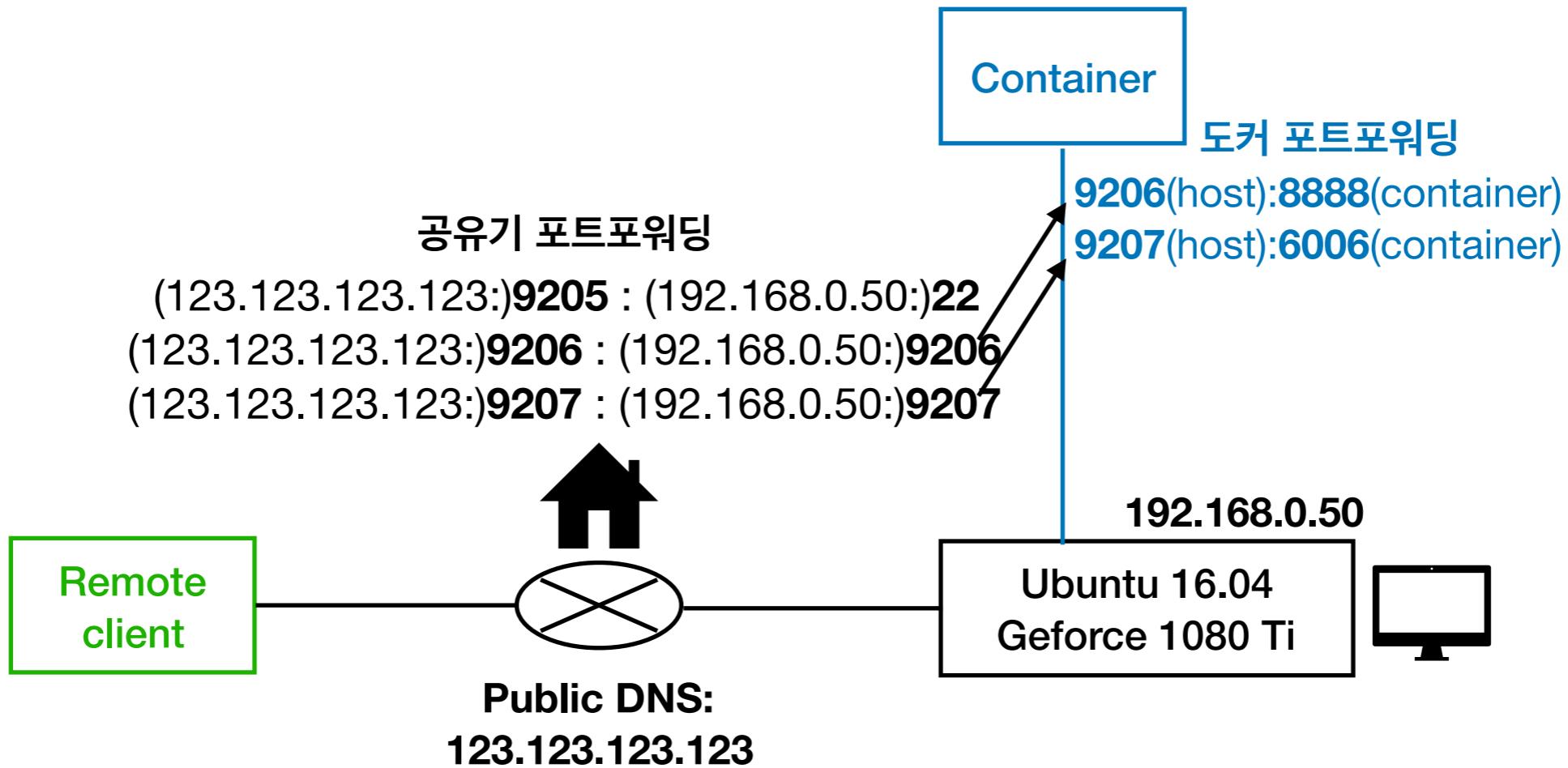
- Docker를 주변에서 많이 들어는 봤는데, 자세히는 모르겠다.
(+ Docker를 배워야 하는데 지금은 미루고 있다.)
- 로컬에 패키지 몇개만 깔면 되던데, conda 쓰면 되지 굳이 도커가 필요할까?
- pip, conda로 뭘 지우고 뭘 새로 깔아서 안되는건지,
어디서부터 꼬였는지 잘 기억도 안난다. 내 로컬을 깨끗하게 유지하고 싶다.
- GPU 셋업하는데 CUDA, cudnn, TF 버전때문에 우분투를 몇 번째 새로 갈 아엎고 있는건지 모르겠다.(★★★)
- 난 그냥 정말 순수하게 내 코드를 돌리고 싶었을 뿐인데,
코드 짜는 시간보다 의존성 에러 해결하는데 시간을 더 많이 쓰는 것 같다.

저는 이렇게 하고 있습니다. 😊😊



1. 집에 있는 GPU서버는 **우분투 16.04, VGA와 맞는 Nvidia 그래픽 카드 드라이버, 그리고 docker**만 설치되어 있습니다. 딥러닝과 관련된 패키지들을 포함해 파이썬 패키지는 단 한 개도 설치하지 않습니다.
2. 외부에서 ssh(or putty) 22번 포트를 포함해 다른 개발환경에 바로 접근하기 위해 공유기 어드민에서 몇개의 포트를 내부 서버와 포워딩해 놓았습니다
3. Jupyter를 포함한 프로세스들은 사실 서버 안에 도커 컨테이너로 올라가 있습니다. 컨테이너와 호스트 간 포트를 포워딩해 외부에서 접근하는 트래픽을 바로 컨테이너로 넘겨주도록 설정해 놓습니다.
4. Google Colab을 쓰듯이 내 원격 서버로 Jupyter notebook에 바로 접근할 수 있고, 데이터는 모두 호스트에 저장됩니다. 개발에 필요한 의존성들은 모두 컨테이너 안에서 관리됩니다. 도커를 사용하면 충돌이 일어나는 케이스가 많지 않고, 일어나더라도 빠르게 다시 셋업할 수 있습니다.

저는 이렇게 하고 있습니다. 😊😊



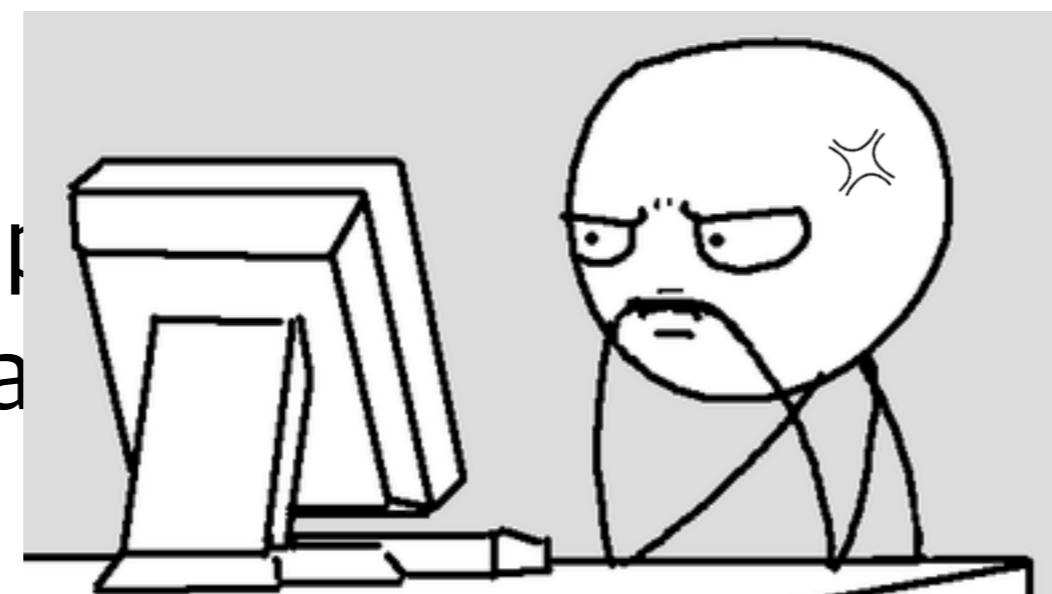
내용의 흐름

- 패키지 충돌: 내가 뭘 잘못했기에 이런 시련을...
- **docker** 소개: 왜, 언제 사용하는 툴일까요?
- Docker로 딥러닝 개발환경 셋업하기
- Virtual environment tools, Package managing tools **vs docker**

패키지 충돌: 내가 뭘 잘못했기에..

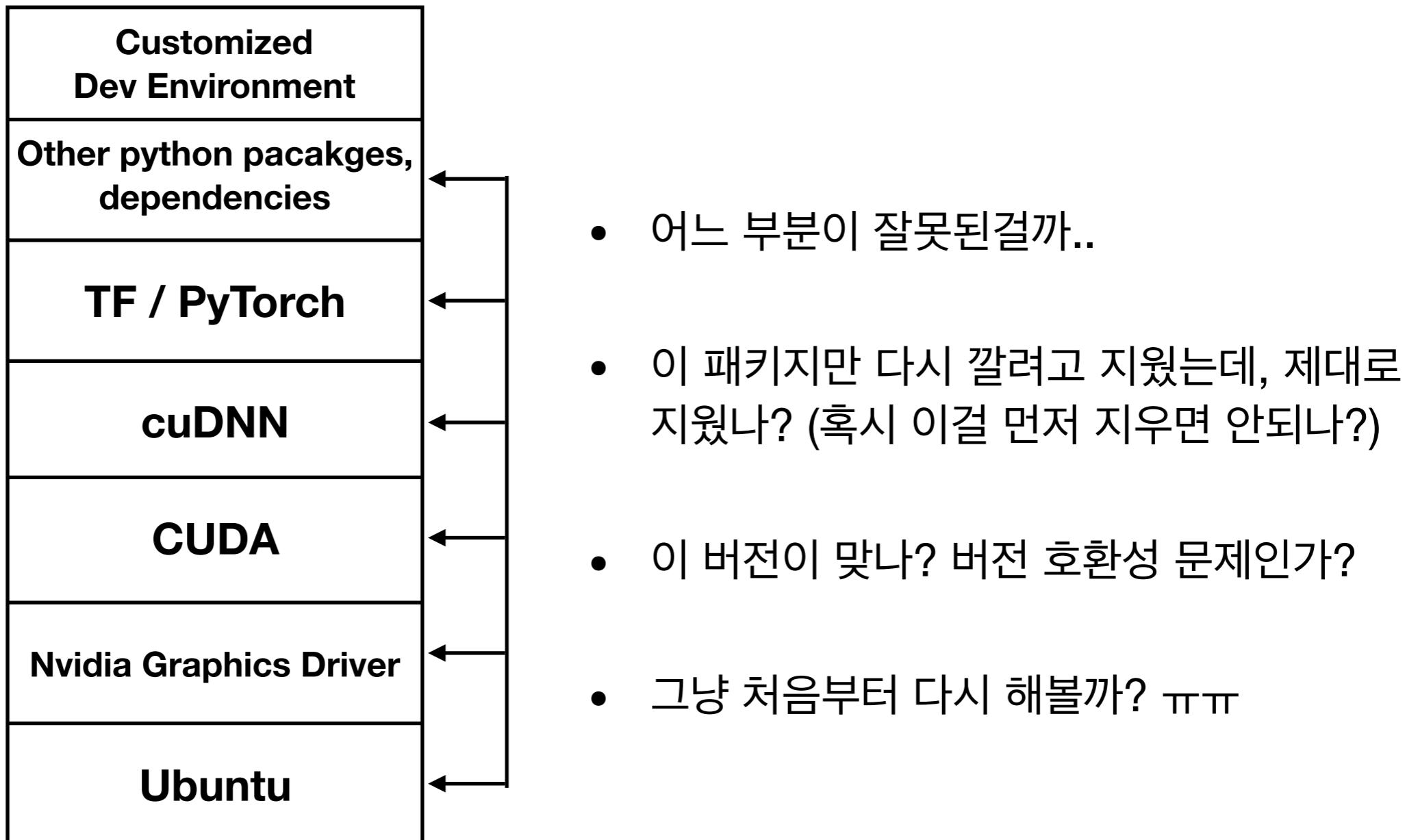
```
(tensorflow) C:\Users\sr>pip install pymc3
Requirement already satisfied: pymc3 in c:\anaconda3\envs\tensorflow\lib\site-packages (3.4.1)
Requirement already satisfied: theano>=1.0.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (1.0.1)
Requirement already satisfied: numpy>=1.13.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (1.14.0)
Requirement already satisfied: pandas>=0.18.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (0.22.0)
Requirement already satisfied: patsy>=0.4.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (0.5.0)
Requirement already satisfied: joblib>=0.9 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (0.11)
Requirement already satisfied: tqdm>=4.8.4 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (4.23.2)
Requirement already satisfied: six>=1.10.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (1.11.0)
Requirement already satisfied: h5py>=2.7.0 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pymc3) (2.7.1)
Requirement already satisfied: scipy>=0.14 in c:\anaconda3\envs\tensorflow\lib\site-packages (from theano>=1.0.0->pymc3) (1.0.0)
Requirement already satisfied: pytz>=2011k in c:\anaconda3\envs\tensorflow\lib\site-packages (from pandas>=0.18.0->pymc3) (2017.3)
Requirement already satisfied: python-dateutil>=2 in c:\anaconda3\envs\tensorflow\lib\site-packages (from pandas>=0.18.0->pymc3) (2.6.1)
tensorflow-tensorboard 1.5.0 has requirement bleach==1.5.0, but you'll have bleach 2.1.2 which is incompatible.
tensorflow-tensorboard 1.5.0 has requirement html5lib==0.9999999, but you'll have html5lib 1.0.1 which is incompatible.
```

pip, sudo pip
conda, venv a



conda, sudo
ip uninstall ...

사소한 듯 사소하지 않은 주제

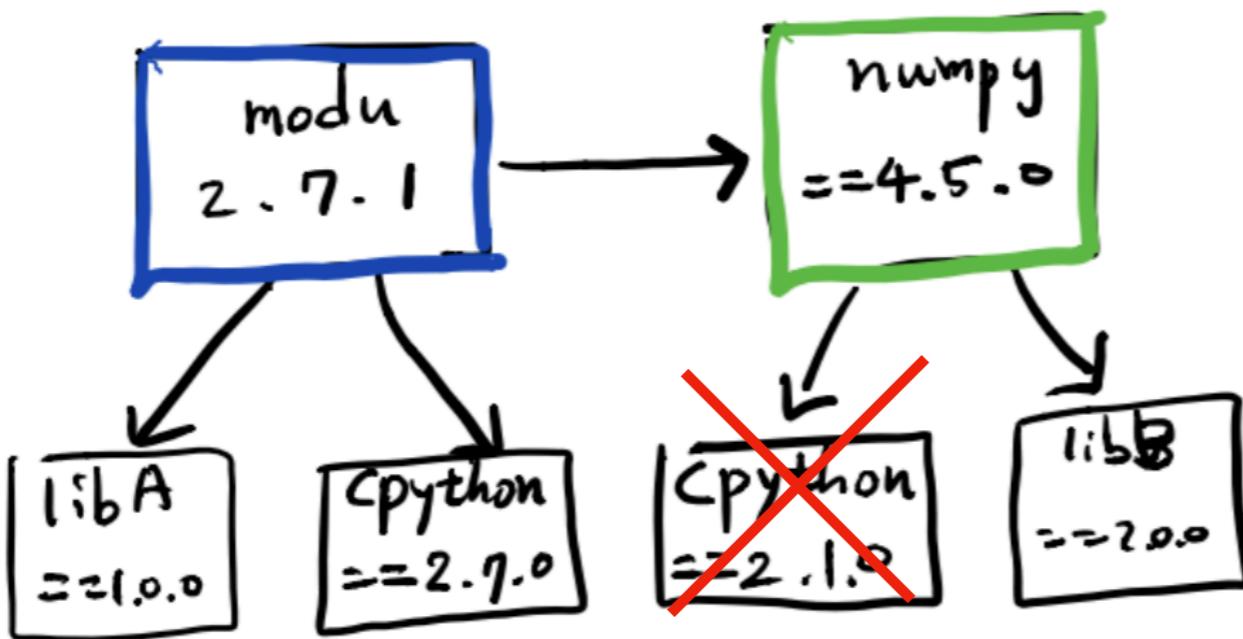


예상되는 행동 패턴

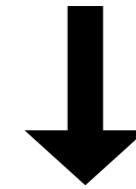
- 코드에는 이상이 없는데, 패키지 깔 때 잘못 깔아서 실행이 안 되나?
 1. 에러메시지를 구글링한다
 2. 나와 같은 상황에 처해있는 사람이 많음을 확인하고 일단은 안심한다.
(그럼 그렇지 ^^)
 3. 가장 신뢰도가 높은 Stackoverflow 솔루션 / 블로그를 따라한다.
 4. 그 부분은 다행히 해결! 그런데 뒤쪽에 비슷하지만 다른 에러가 난다.
 5. 이건 찾아봐도 나와 비슷한 상황의 사람들이 많이 없다.
 6. 문제가 발생한 패키지를 다시 설치해본다.
=> 그래도 안된다
 7. Pip(conda) uninstall -y tensorflow && pip install -y tensorflow
=> 여전히 안된다
 8. 프로젝트 코드를 백업하면서 usb에 우분투 iso 이미지를 받고, 부팅 디스크를 만든다.

왜 충돌이 날까?

```
$ pip install modu █  
Installing modu==2.7.1...  
██████████ 156%
```



- Cpython 2.1.0:
`def multiply(x, y):
 # codeblock`



- Cpython 2.7.0:
`def multi(x, y):
 # codeblock`

- Modu 패키지 설치하면 로컬의 CPython 2.1.0이 2.7.0로 업데이트! 하지만 numpy에서는 CPython의 바뀐 함수 multi() 대신 여전히 multiply() 호출. 없는 함수라고 나올 수 있는데, 분석이 어렵다.

너무나도 많은, 가능한 시나리오

- 로컬에 원래 설치되어 있던 A 패키지의 버전은 v1.0,
새로 설치하려는 B 패키지에서는 A패키지가 필요하고, v1.0 버전도 허용한다고 명시.
하지만 실제로는 B 패키지 로직 안에서 **v1.5에서 새로 추가된 A패키지의 함수를**
호출하고 있을 경우
 - > 새로 설치한 B패키지에서 A의 의존성 버전을 맞게 잡아주지 못한 탓 (내 탓 아님)
 - > **내 탓은 아니지만 어쨌든 내가 해결해야 합니다.** 에러 메시지는
'함수가 없습니다.' 몇줄. ㅠㅠ
- 의존성 패키지들끼리 서로 공통적으로 필요한 패키지의 버전이 다를 경우
- 최신 버전으로 업데이트된 패키지에서 삭제된 함수가 있을 경우, 제대로 돌아가던게 안 됨. (그러면 downgrade..?)
- 이것도 저것도 아닌 것 같은 경우 (원인을 알 수 없음)

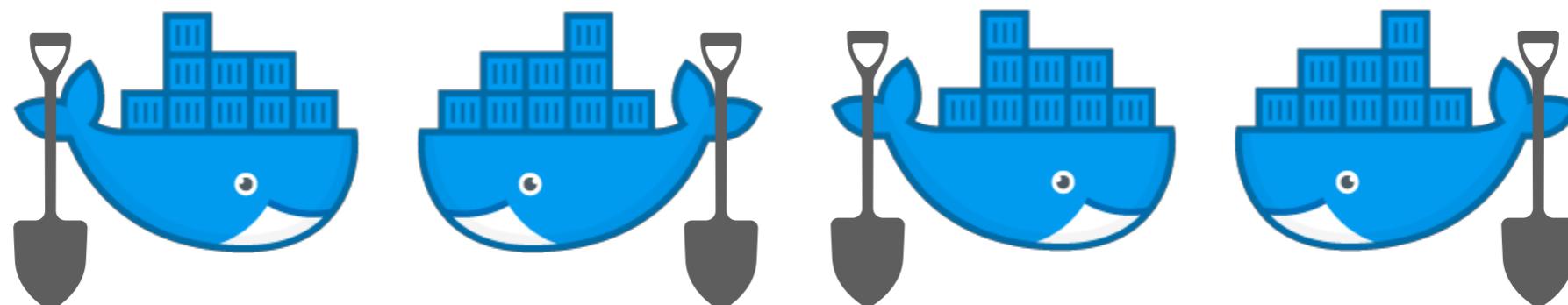
너무나도 많은, 가능한 시나리오

- 로컬에 원래 설치되어 있던 A 패키지의 버전은 v1.0, 새로 설치하려는 B 패키지에서는 A패키지가 필요하고, v1.0 버전도 허용한다고 명시. 하지만 실제로는 B 패키지 로직 안에서 **v1.5에서 새로 추가된 A패키지의 함수를 호출하고 있을 경우**
 - > 새로 설치한 B패키지에서 A의 의존성 버전을 맞게 잡아주지 못한 탓 (내 탓 아님)
 - > **내 탓은 아니지만 어쨌든 내가 해결해야 합니다.** 에러 메시지는 ‘함수가 없습니다.’ 몇줄. ㅠㅠ
- 의존성 패키지들끼리 서로 공통적으로 필요한 패키지의 버전이 다를 경우
- 최신 버전으로 업데이트된 패키지에서 삭제된 함수가 있을 경우, 제대로 돌아가던게 안 됨. (그러면 downgrade..?)
- 이것도 저것도 아닌 것 같은 경우 (원인을 알 수 없음)

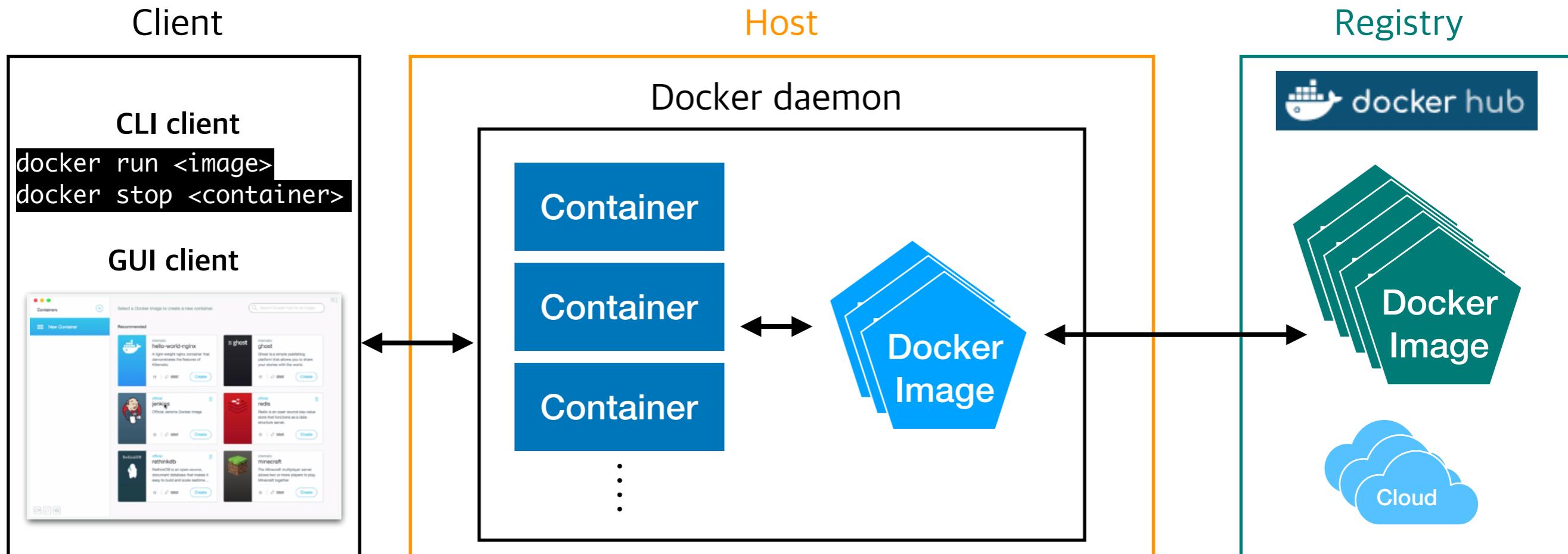
파이썬 뿐만 아니라 드라이버, 많은 툴들이
너무 얹혀있어 원인을 제대로 파악할 수 없는 경우가 가장 많습니다.

원인을 알 수 없는 충돌 문제는?

- 모든 문제는 원인 파악이 가장 중요하지만, 이 의존성 충돌의 경우 파악이 힘든 경우가 많아요.
- 지금 이 순간에도 프레임워크와 의존 라이브러리들의 버전은 계속 올라가고 충돌 가능성은 항상 여기저기에 널려 있습니다. 계속 그랬고 앞으로도!!
- 충돌을 피할 수 없고, 해결이 우선이라면 최대한 **빨리, 효율적으로** 삽질을 하는 것이 가장 바람직합니다!



Docker overview

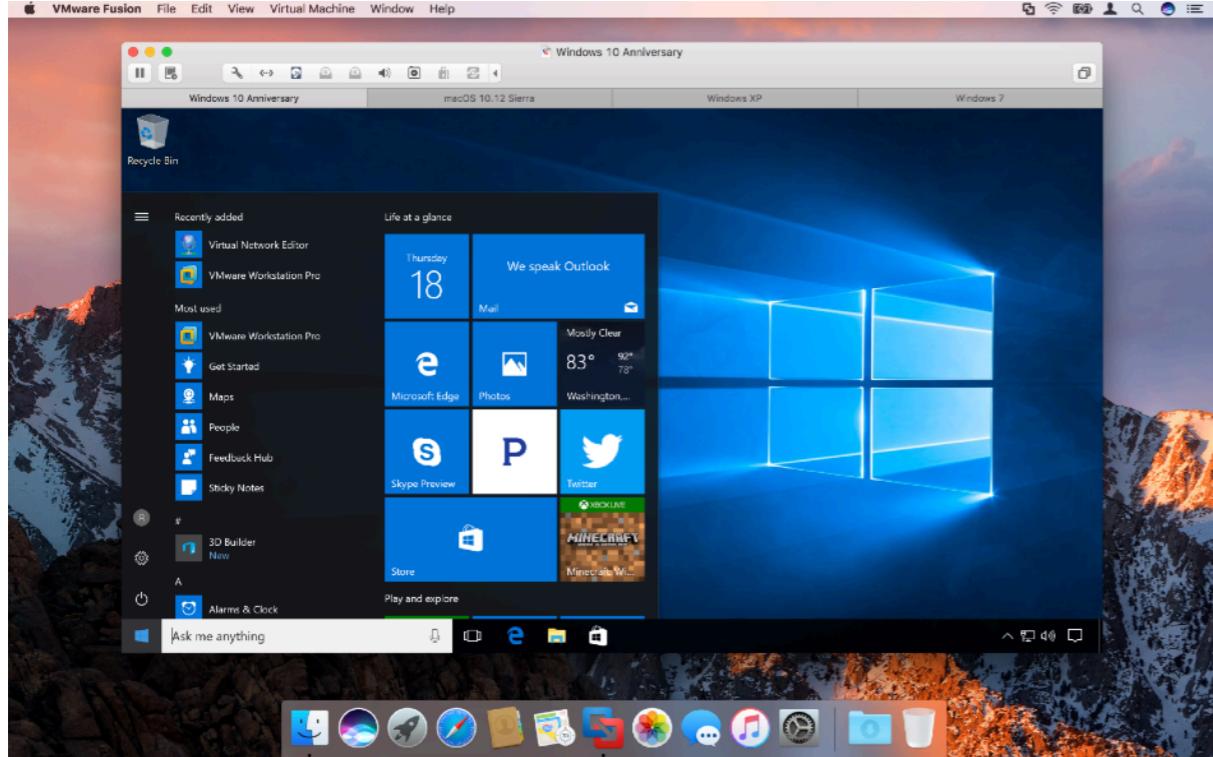


- Git은 코드를 관리하고 배포하지만, Docker는 OS 환경을 관리하고 배포합니다.
- **Docker image**에 OS, 환경설정을 명시하거나 셋업한 뒤 Host 위에 Container로 올려서 실행합니다.

docker 소개: 왜, 언제 사용할까요?

- Released in 2013, by Solomon Hykes
- Docker: OS-level 가상화 툴 (Containerization)
- [cf. VMware, VirtualBox: Hardware-level 가상화 툴]
 
- 하드웨어 환경을 가상화 vs OS 환경을 가상화
- Git은 코드를 관리하고 배포하지만, Docker는 환경을 관리하고 배포합니다.
- ‘제 로컬에서는 잘 되던데요? ㅋㅋ’ 와 같은 상황을 해결
- 특정한 OS, 의존성 패키지들이 미리 깔려있는 환경을 받아
그 위에서 작업할 수 있고, 내 환경도 쉽게 배포할 수 있습니다.

VMware, VirtualBox



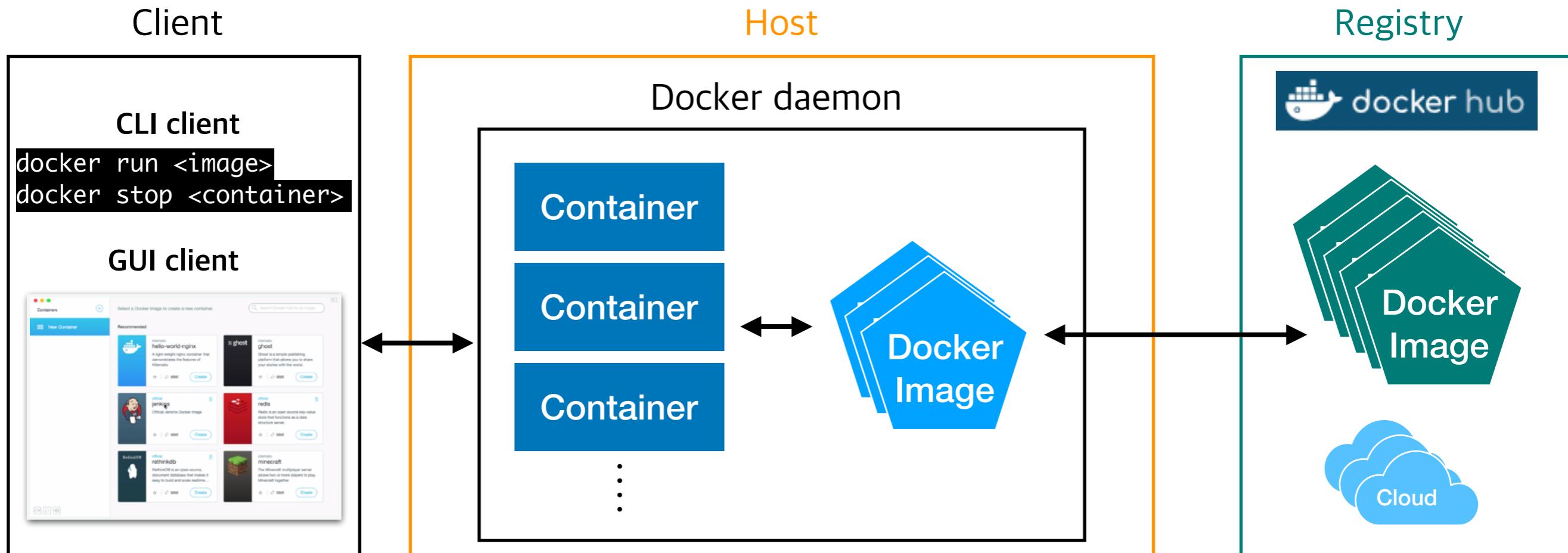
VMware Workstation
(MacOS 위의 windows)

- Docker와의 가장 큰 차이점은 GUI가 있고 없고도 중요하지만, 어떤 수준에서의 가상화를 제공하는가 (OS-level, Hardware-level)
- Docker는 가벼운 LXC/libcontainer를 사용하고, 비교적 무거운 machine/hardware 에뮬레이션은 제공하지 않음 : 아주 빠르게 원하는 Linux 환경을 셋업할 수 있지만, kernel 레벨의 작업은 제한적



VirtualBox
(Window 위의 linux)

Docker



- **Container**가 우리가 사용하려고 하는 OS와 툴, 의존성들이 담긴 상자.
우리의 최종 목표는 이 컨테이너 안에 개발 환경을 셋업하는 것!
- **Container**는 **Image**로부터 만들어지는데, 이는 마치
Class로부터 **Instance**가 생성되는 개념과 매우 흡사합니다.

Dockerhub

- Dockerhub는 **Docker Image** 저장소!
- 오픈소스 프로젝트마다 Dockerfile을 빌드해 이미지를 만들고, 해당 이미지를 사용자들이 편하게 **바로 컨테이너로 올릴 수 있도록** 업로드 해 놓는 경우가 굉장히 많습니다.
- 특정 Use-case에 특화된 이미지를 만들어서 직접 배포할 수도 있고, (예를 들어:
Deep learning stack: Jupyter + Tensorflow + Pytorch
NLP stack: Tensorflow + Gensim + nltk + konlpy
...)
- 이슈가 발생한 내 환경을 복제해서 다른 사람에게 보여줄 때도 Dockerhub에 업로드 후 링크만 알려주면 됩니다!

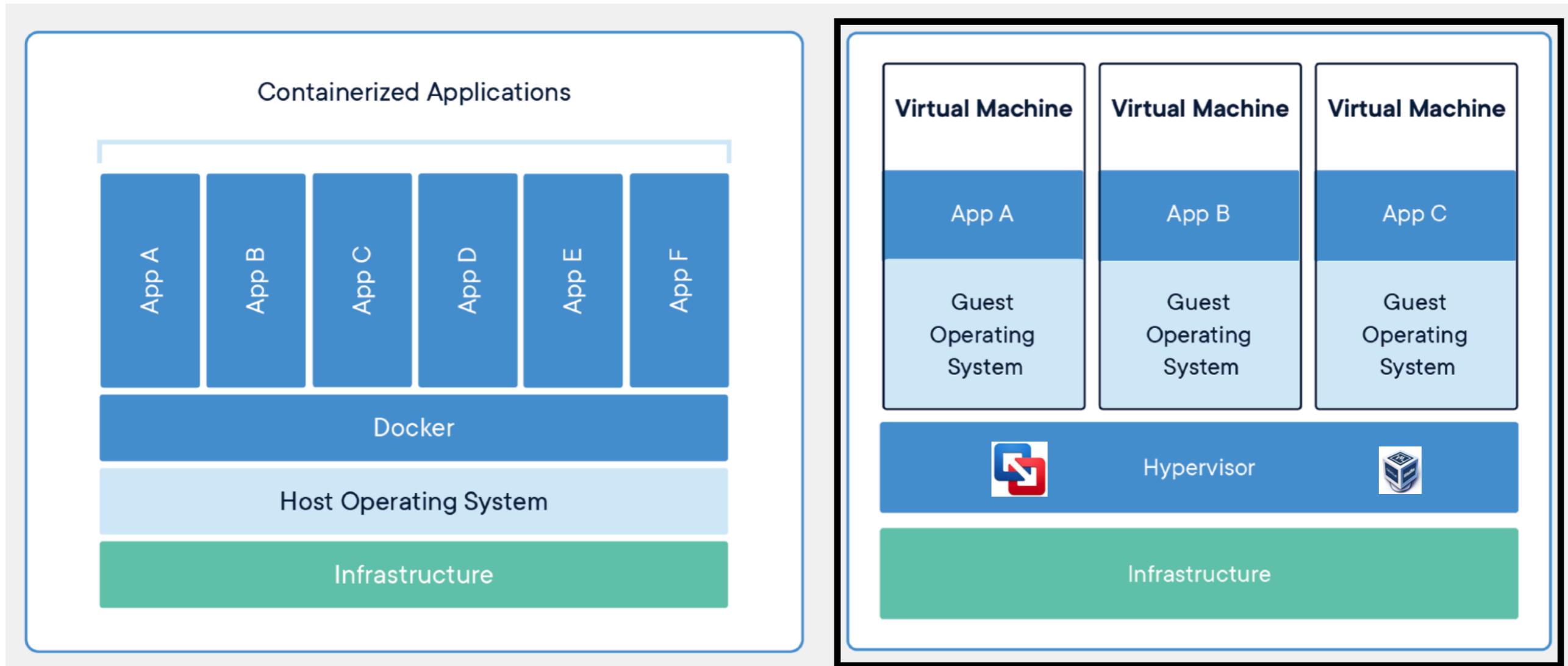
Dockerhub

The screenshot shows the Dockerhub homepage with a dark header bar containing a ship icon, a search bar, and navigation links for Explore, Help, Sign up, and Sign in. Below the header, a blue banner says "Explore Official Repositories". The main content area displays a list of official Docker repositories:

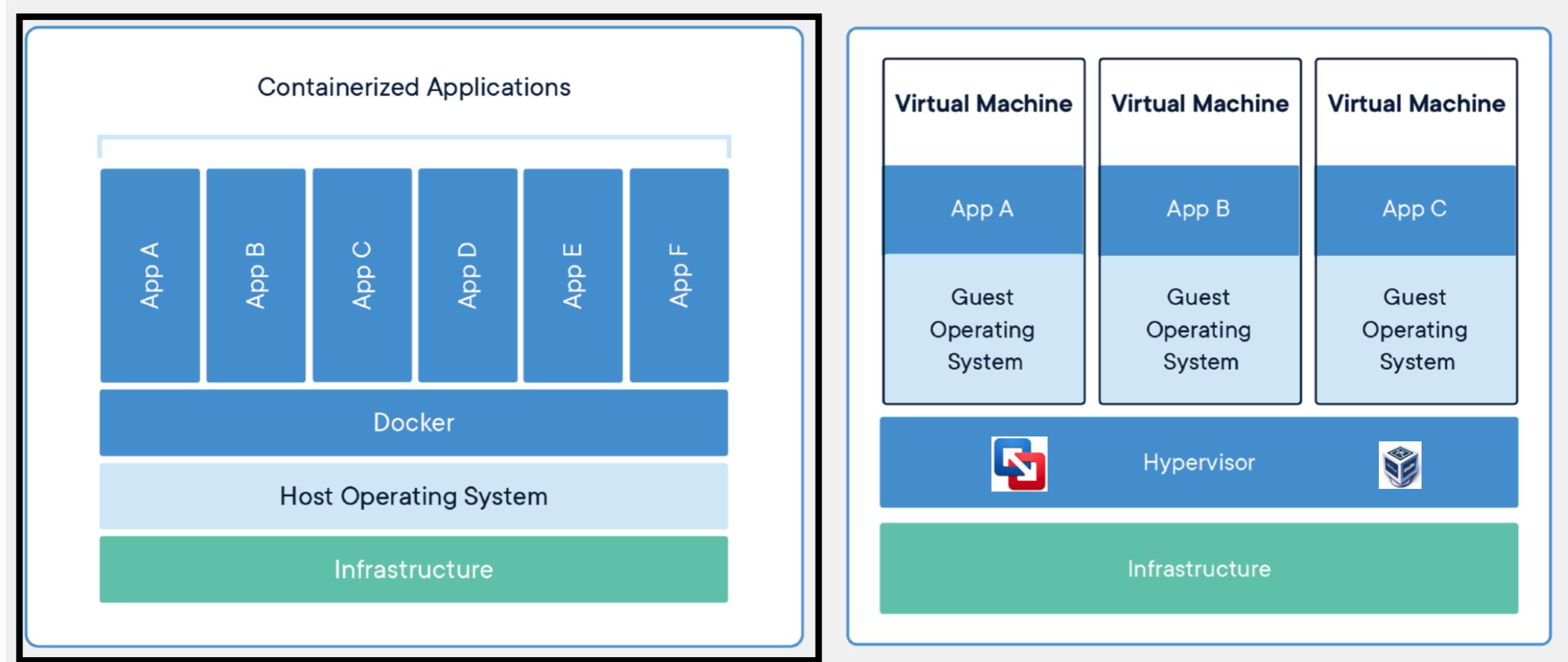
Repository	Owner	Description	Stars	Pulls	Actions
NGINX	nginx official		10.5K STARS	10M+ PULLS	DETAILS
ALPINE Linux (5MB)	alpine official		4.7K STARS	10M+ PULLS	DETAILS
Busybox	busybox official		1.4K STARS	10M+ PULLS	DETAILS
Redis	redis official		6.2K STARS	10M+ PULLS	DETAILS
mongo	mongo official	■ ■ ■	5.4K STARS	10M+ PULLS	DETAILS
httpd	httpd official		2.2K STARS	10M+ PULLS	DETAILS
ubuntu	ubuntu official		8.9K STARS	10M+ PULLS	DETAILS

Dockerhub - tensorflow/tensorflow

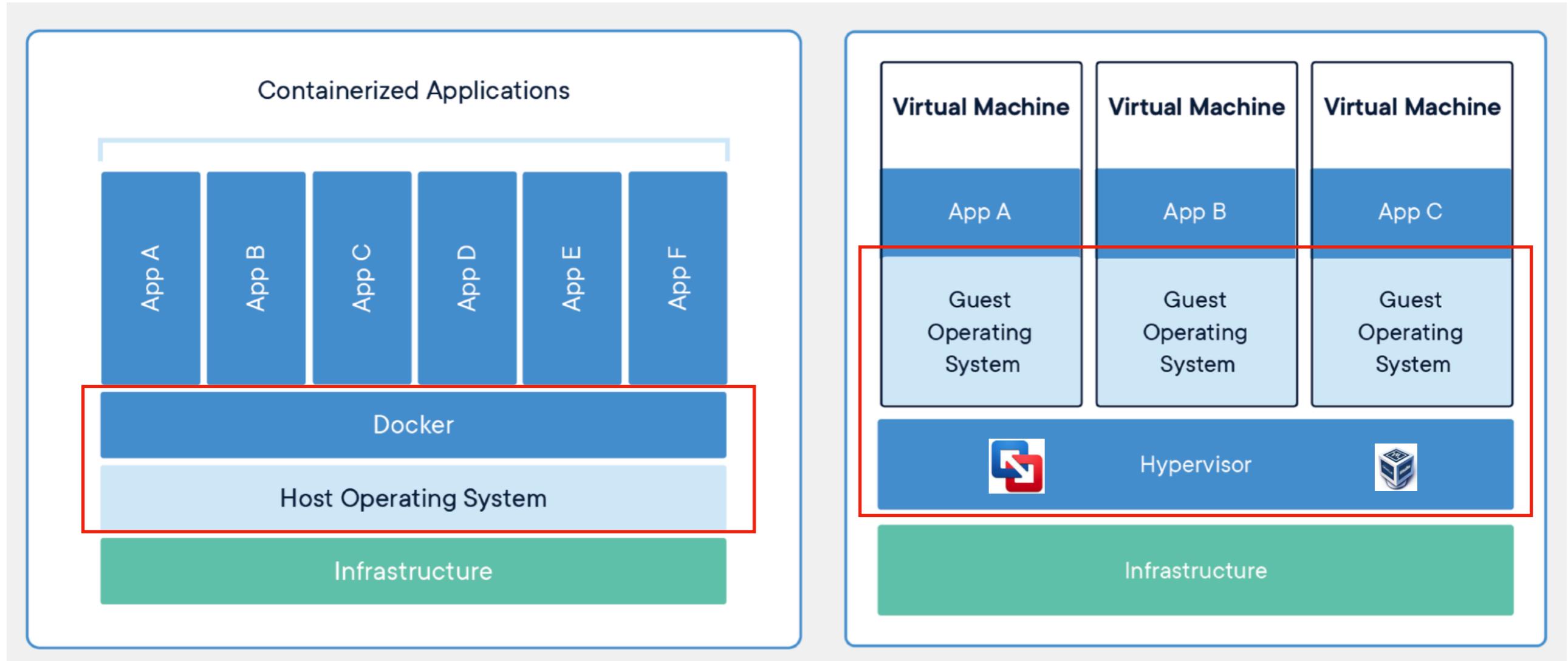
The screenshot shows the Dockerhub interface for the tensorflow/tensorflow repository. At the top, there's a search bar with a Docker icon and the text "tensorflow". Below the search bar, it says "PUBLIC REPOSITORY". The repository name "tensorflow/tensorflow" is displayed in blue, followed by a star icon indicating it's favorited. A timestamp "Last pushed: a day ago" is shown below the repository name. There are two tabs at the top of the main content area: "Repo Info" and "Tags", with "Repo Info" being the active tab. Under "Repo Info", there are two sections: "Short Description" and "Full Description". The "Short Description" section contains the text: "Official docker images for deep learning framework TensorFlow (<http://www.tensorflow.org>)". The "Full Description" section contains the text: "Start CPU only container" followed by a command line example: "\$ docker run -it -p 8888:8888 tensorflow/tensorflow" and a link "Go to your browser on <http://localhost:8888/>".



- **Hypervisor:** VMware, VirtualBox 등이 있으며, 하드웨어를 추상화하여 완전한 (Guest) OS를 포함하는 가상머신을 Host OS의 프로세스로써 올려주는 툴.



- **Docker:** Linux host 위에서 다른 linux들과 커널을 공유하는, 격리된 컨테이너를 올려주는 툴
- 컨테이너끼리 격리되어 있으며, linux 자원만 호스트와 공유합니다.



- Docker가 지금의 인기를 얻게 된 이유: 리소스, 속도, 표준화, 자동화, 격리
- 개발 환경 구축에 시간이 많이 드는 상황에서 충분히 고려해볼 수 있습니다.
- 다른 OS, 다른 버전의 의존성들의 조합에서 잘 호환되는지 테스트 용도로도 굉장히 많이 쓰입니다.

Docker로 딥러닝 개발환경 셋업

들어가기 전에

- 튜토리얼에 사용되는 도커 이미지는,
cpu 버전은 tensorflow/tensorflow:latest, (tab)
gpu 버전은 Nvidia 그래픽 카드를 가정할 때 tensorflow/tensorflow:latest-gpu
를 기준으로 합니다.

PyTorch는, official docker image에 Jupyter가 설치되어 있지 않으므로
TF 컨테이너 위에 pip으로 **PyTorch**를 설치하신 뒤 이미지에 변경사항을
커밋시키면 gpu까지도 잘 작동합니다.

- Nvidia-docker 프로젝트는 아직 윈도우를 지원하지 않습니다. 즉,
윈도우에서는 도커 컨테이너에 tensorflow-gpu는 설치할 수 없습니다. (cpu는 됩니다!)
- 도커를 한번 사용해 보시기로 결심하셨다면 호스트 OS(Ubuntu 혹은 다른 Linux)에는
 - gpu의 경우: {그래픽카드와 맞는 버전의 Nvidia-driver, docker, nvidia-docker2}
 - cpu만 사용하실 경우: {Docker}이렇게만 설치하신 뒤 그 외의 모든 파이썬 패키지는 **Docker** 이미지 안에서만 생각해 주시면 됩니다.

Docker CE 설치

- [Linux]: <https://docs.docker.com/install/#supported-platforms>에서 OS에 맞는 인스톨 가이드라인을 따라해 주세요!

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64	IBM Power (ppc64le)	IBM Z (s390x)
CentOS	✓		✓		
Debian	✓	✓	✓		
Fedor	✓				
Ubuntu	✓	✓	✓	✓	✓

링크 클릭!

Linux 인스톨 과정은 공통적으로 이렇게 세 단계입니다.

1. Package manager repository setup
2. 패키지 매니저 update
3. 패키지 매니저로 Docker install

Docker CE 설치

- [MacOS, Windows]:
.dmg 혹은 .exe 파일 다운로드 후 인스톨러로 Desktop version 설치(추천)

Docker for Developers

Building and deploying new applications is faster with containers. Docker containers wrap up software and its dependencies into a standardized unit for software development that includes everything it needs to run: code, runtime, system tools and libraries. This guarantees that your application will always run the same and makes collaboration as simple as sharing a container image.

Docker containers whether [Windows](#) or Linux are backed by Docker tools and APIs and help you build better software:

- Onboard faster and stop wasting hours trying to set up development environments, spin up new instances and

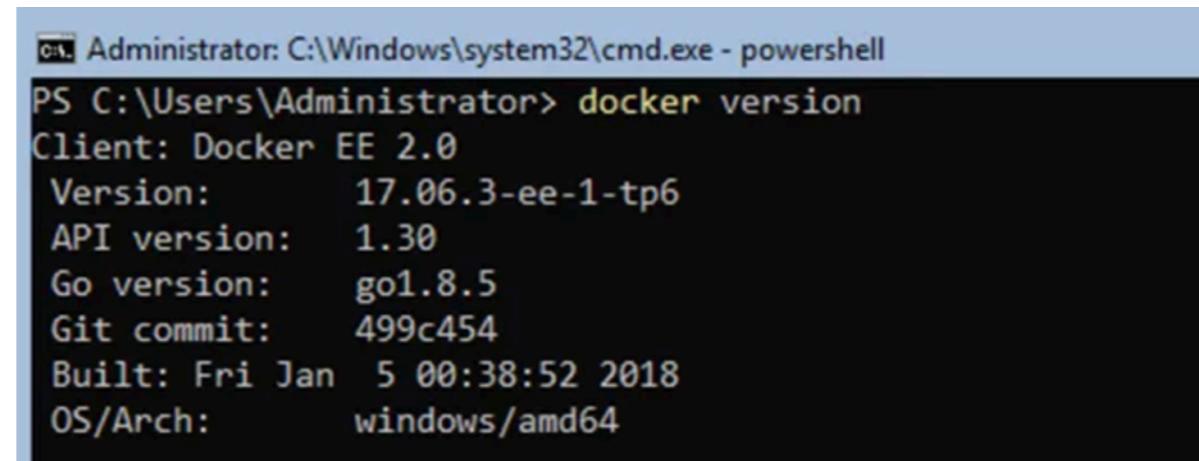
[Download for Mac](#)

[Download for Windows](#)

<https://www.docker.com/get-started>

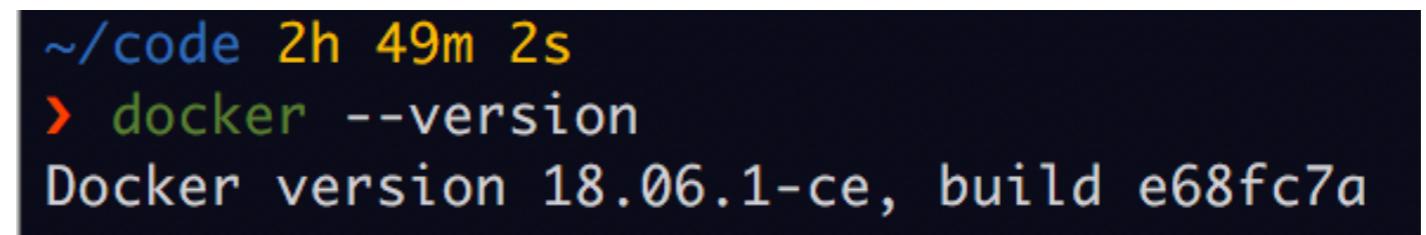
Docker CE 설치

- [MacOS, Windows, linux]:
이후 PowerShell 혹은 Terminal에서 ‘docker version’ 커マン드로 설치 확인



```
Administrator: C:\Windows\system32\cmd.exe - powershell
PS C:\Users\Administrator> docker version
Client: Docker EE 2.0
  Version: 17.06.3-ee-1-tp6
  API version: 1.30
  Go version: go1.8.5
  Git commit: 499c454
  Built: Fri Jan 5 00:38:52 2018
  OS/Arch: windows/amd64
```

- Windows:



```
~/code 2h 49m 2s
> docker --version
Docker version 18.06.1-ce, build e68fc7a
```

Windows image:

<https://www.deploycontainers.com/2018/01/10/update-docker-enterprise-preview-edition-windows-server-1709/>

Docker CE 설치

- [Windows]:

만약 PowerShell이 없다면, ‘Docker toolbox’를 함께 설치하면 Docker client CLI를 실행할 수 있습니다.

Install

Double-click `Docker for Windows Installer` to run the installer.



When the installation finishes, Docker starts automatically. The whale in the notification area indicates that Docker is running, and accessible from a terminal.

- Windows:

Run

Open a command-line terminal like PowerShell, and try out some Docker commands!

Run `docker version` to check the version.

Run `docker run hello-world` to verify that Docker can pull and run images.

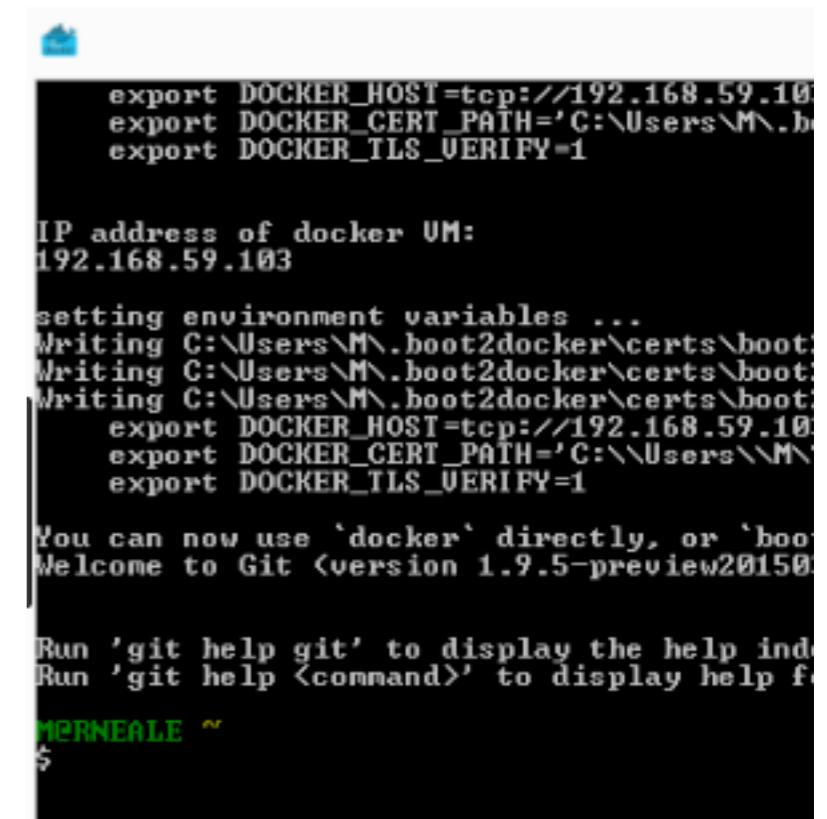
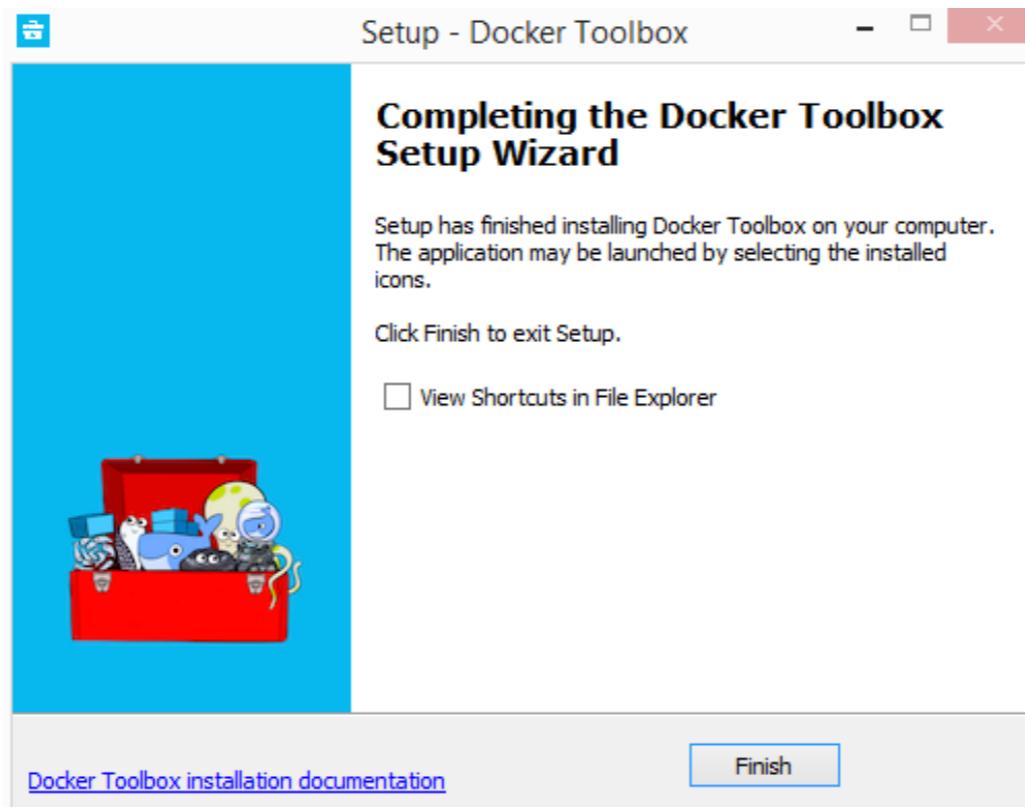
Install docker toolbox for Windows:

https://docs.docker.com/toolbox/toolbox_install_windows/#step-3-verify-your-installation

Docker CE 설치

- [Windows]:
만약 PowerShell이 없다면, ‘Docker toolbox’를 함께 설치하면
Docker client CLI를 실행할 수 있습니다.

- Windows:



Install docker toolbox for Windows:

https://docs.docker.com/toolbox/toolbox_install_windows/#step-3-verify-your-installation

Post-installation steps for linux

- Docker는 커맨드 실행 시 root 권한이 필요합니다.
- ‘**docker**’ 명령어 앞에 항상 sudo를 붙여줘야 하는 불편함을 덜기 위해 Linux의 경우 아래와 같이 인스톨 후에 도커에게 루트 권한을 줍시다.

To create the `docker` group and add your user:

1. Create the `docker` group.

```
$ sudo groupadd docker
```

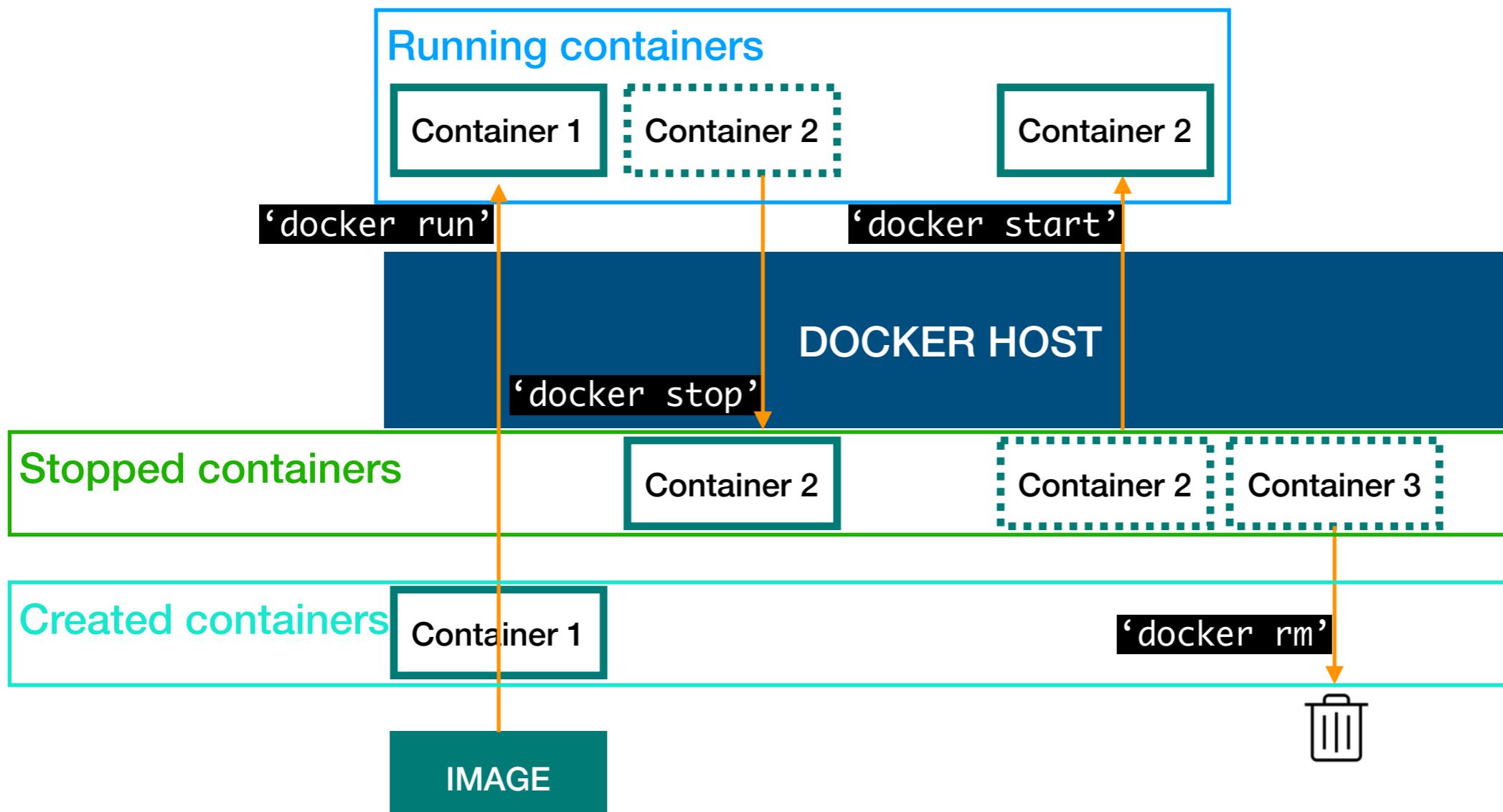
2. Add your user to the `docker` group.

```
$ sudo usermod -aG docker $USER
```

3. Log out and log back in so that your group membership is re-evaluated.

<https://docs.docker.com/install/linux/linux-postinstall/>

Basic docker commands



Basic docker commands

- 명령어의 옵션도 같이 알아두시면 더 좋습니다. ^^
- **docker ps**: 현재 실행중인 컨테이너들의 상태를 출력
- **docker attach <container-name>** : 컨테이너 안에서 프로세스가 실행되고 있는 그 상태의 쉘로 접속
- **docker commit <container-name> <image-id>** : 컨테이너 안의 변경사항들을 이미지에 반영(커밋).
- **docker images**: 현재 로컬에 저장된 이미지들의 목록
- **docker push/pull**: 내 로컬에 있는 이미지를 원격 레포지토리에 push하거나 원격 레포지토리의 이미지를 로컬로 당겨옴

Dockerhub - tensorflow/tensorflow

- 이제 Tensorflow 공식 이미지를 가져와서 올려봅시다!
- Usage: docker run [OPTIONS] IMAGE [COMMAND]
- **docker run -it -p 9206:8888 tensorflow/tensorflow [command??]**
- -i: interactive, -t: tty
keep stdin open / allocate a terminal
(옵션을 주지 않으면 컨테이너 안에서의 터미널을 볼 수 없어요.)
- -p: port 9206:8888
호스트의 포트 : Docker 컨테이너의 포트
- 이미지 이름의 형식은 <repo>/<image_name>:<tag_name> 입니다.
tag name이 주어지지 않을 경우 default는 ‘latest’
- [COMMAND]가 주어지지 않을 경우 tensorflow 이미지의 디폴트 실행 프로그램은 Jupyter notebook!

Dockerhub - tensorflow/tensorflow

- run 명령어 최초 실행 시 저장되어 있는 이미지가 없으므로, tensorflow/tensorflow 이름을 가진 이미지가 dockerhub에 있는지 찾고, 있다면 로컬로 가져옵니다.
- 즉 ‘docker pull’ 명령어로 이미지를 먼저 받아온 뒤, run을 한 것과 같습니다.

```
> docker run -it -p 8888:8888 tensorflow/tensorflow
Unable to find image 'tensorflow/tensorflow:latest' locally
latest: Pulling from tensorflow/tensorflow
18d680d61657: Pull complete
0addb6fece63: Pull complete
78e58219b215: Pull complete
eb6959a66df2: Pull complete
4263945a5710: Extracting [=====] 36.21MB/140MB
282d99d903f2: Download complete
dd620fc3ae28: Download complete
aa8e48babf4b: Download complete
27c088fb1ccc: Downloading [=====] 81.08MB/100.9MB
ca88730530b5: Download complete
5f530f9c7fc4: Download complete
8ea52f77069e: Download complete
```

Dockerhub - tensorflow/tensorflow

```
[I 13:17:59.413 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 13:17:59.445 NotebookApp] Serving notebooks from local directory: /notebooks
[I 13:17:59.445 NotebookApp] The Jupyter Notebook is running at:
[I 13:17:59.445 NotebookApp] http://(20ff290d68a7 or 127.0.0.1):8888/?token=e6215842df5c594b21e25cb7eff2c8c976403bc2362e13b6
[I 13:17:59.446 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation)
[C 13:17:59.449 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://(20ff290d68a7 or 127.0.0.1):8888/?token=e6215842df5c594b21e25cb7eff2c8c976403bc2362e13b6
```

- tensorflow/tensorflow:latest 이미지를 컨테이너로 올리면, 컨테이너에서 Jupyter notebook이 컨테이너의 8888번 포트에서 디폴트로 실행됩니다.
- 현재 터미널의 상태는 컨테이너에 attach 되어 있는 상태입니다. Ctrl + c로 Jupyter 서버를 중지시키면 자기 할 일을 마친 컨테이너는 바로 정지되기 때문에, attach 상태에서 호스트 쉘로 서버를 중지하지 않고 돌아가려면 Ctrl + p, Ctrl + q를 차례대로 입력하면 됩니다.(Escape sequence)
- 컨테이너의 8888 포트와 로컬의 9206 포트를 이전에 -p 옵션으로 포워딩 해 주었으니, 로컬의 브라우저에서 9206 포트로 접속해 봅시다!

Password or token:

|

Log in

Token authentication is enabled

If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you [enable a password](#).

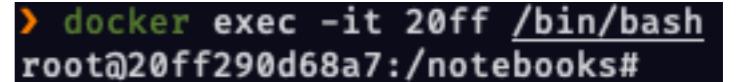
The command:

```
jupyter notebook list
```

will show you the URLs of running servers with their tokens, which you can copy and paste into your browser. For example:

```
Currently running servers:  
http://localhost:8888/?token=c8de56fa... :: /Users/you/notebooks
```

or you can paste just the token value into the password field on this page.

- 콘솔에 보이는 token=[**abcdabcd** …] 문자열을 복사해 붙여줍니다.
최초 1회만 확인하며, 외부에서 접속하는 ip가 바뀌면 다시 입력해 주어야 합니다. 비밀번호를 설정할 수도 있습니다.
- 호스트 쉘에서는 ‘**docker exec -it <container_name> /bin/bash**’ 명령어로 컨테이너 bash를 새로 실행시킨 뒤, `jupyter notebook list`를 입력하면 다시 볼 수 있습니다.


```
> docker exec -it 20ff /bin/bash  
root@20ff290d68a7:/notebooks#
```
- ‘**docker attach <container_name>**’ 명령어는 컨테이너 생성 시 실행하고 있던 프로세스의 상태 그대로 접속됩니다.
즉 jupyter server를 실행시켰을 때의 콘솔 상태로 attach되며, ctrl-c를 눌러 보면 확인할 수 있습니다.

잘 됩니다! 그런데 문제는..

The screenshot shows a Jupyter Notebook interface. At the top, there's a browser-style header with back/forward buttons, a search bar containing 'localhost:8888/tree?', and various icons. Below that is the Jupyter logo and 'Logout' buttons. A navigation bar has tabs for 'Files' (selected), 'Running', and 'Clusters'. A message says 'Select items to perform actions on them.' Below is a file list table with columns for selection, name, last modified, and file size. The table contains five entries: '1_hello_tensorflow.ipynb', '2_getting_started.ipynb', '3_mnist_from_scratch.ipynb', 'BUILD', and 'LICENSE'. At the bottom, a code cell shows the command 'import tensorflow as tf' and its output 'Out[1]: '1.12.0''.

	Name ↓	Last Modified	File size
<input type="checkbox"/> 0	/		
<input type="checkbox"/> 1_hello_tensorflow.ipynb	a month ago	25 kB	
<input type="checkbox"/> 2_getting_started.ipynb	a month ago	165 kB	
<input type="checkbox"/> 3_mnist_from_scratch.ipynb	a month ago	210 kB	
<input type="checkbox"/> BUILD	a month ago	119 B	
<input type="checkbox"/> LICENSE	a month ago	586 B	

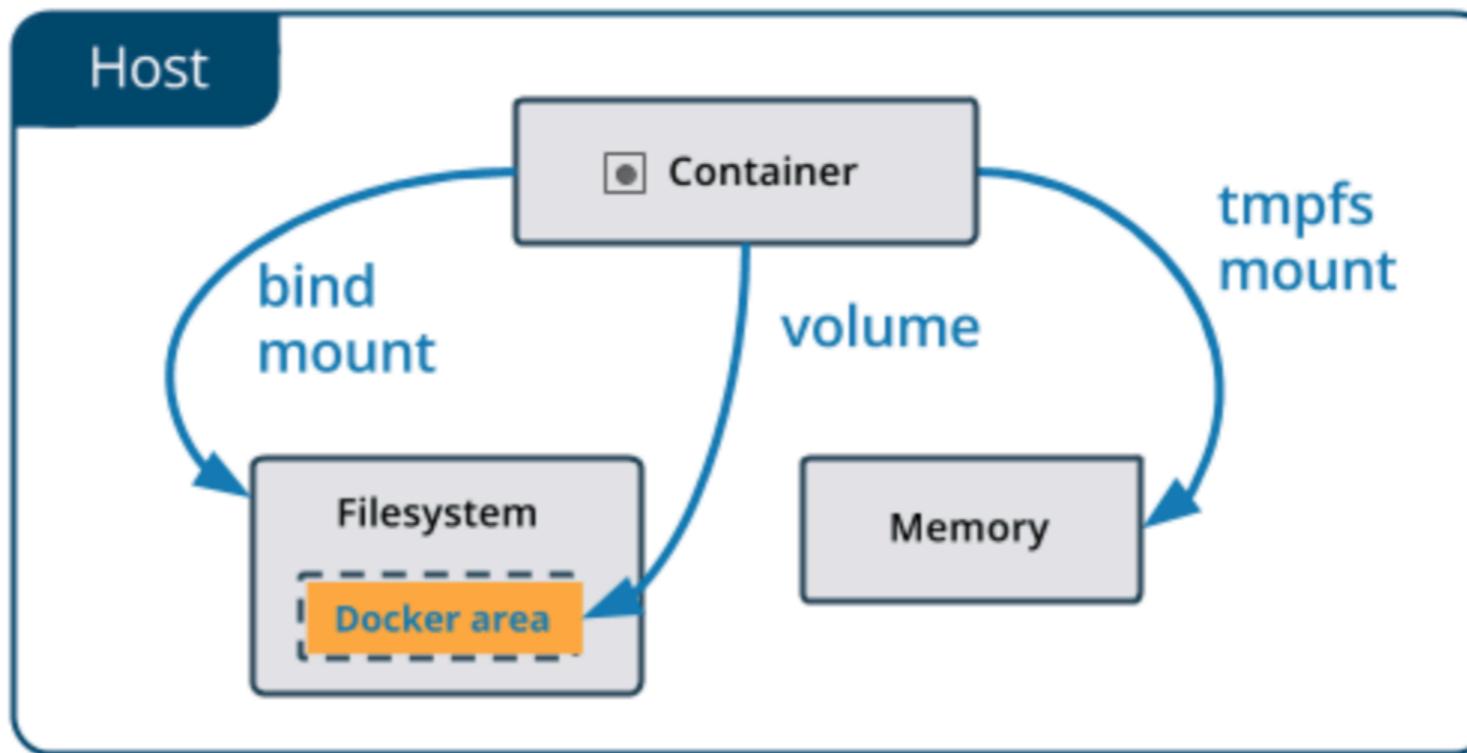
```
In [1]: import tensorflow as tf  
       tf.__version__
```

Out[1]: '1.12.0'

컨테이너는 휘발성!

- 컨테이너는 실행되면 자신의 할 일을 끝내면 다시 내려가고, 따로 설정을 해 주지 않으면 **기본적으로 자신의 자취를 Host OS에 남기지 않습니다.**
- 즉, Jupyter notebook 안에서 작업하고 저장한 파일들은 호스트에서 직접 접근할 수 없고, 컨테이너가 내려간 상태에서 삭제되면 **같이 삭제됩니다.**
- 불의의 사고를 방지하기 위해, **Host OS <=> 컨테이너** 간 **공유 디렉토리를** 두고 컨테이너에서 작업하고 저장한 것이 바로 host OS의 파일시스템에 써질 수 있게 하는것이 좋겠죠?
- 최초 컨테이너 run 시에,
 - volume option인 -v를 사용하거나,
 - 마운트 옵션인 —mount, —bind 등을 사용해 해결합니다!

docker run -v



- docker volume은, **호스트의 파일시스템과 격리된 도커 컨테이너의 파일시스템을** 이어주는 역할을 합니다. docker run 커맨드 실행시 -v 옵션으로 임시로 생성하거나, 혹은 volume만 따로 생성하고 run 시에 붙여서 실행시키는 것도 가능합니다.
- –mount, –bind 또한 비슷한 개념의 옵션입니다.

docker run -v

- 이제 커맨드는:
Host port Container port Host path Container path
docker run -it -p 9206:8888 -v \${HOME}/code:/notebooks \
tensorflow/tensorflow
- 잘 되나 한번 같이 볼까요?

중간 정리: Commands

- tensorflow/tensorflow 도커 이미지로부터 컨테이너 생성, 실행:
`docker run -it -p 9206:8888 -v ${HOME}/code:/notebooks \ tensorflow/tensorflow`
- Container 내부에 새로운 쉘을 띄우고 접속 (attach는 실행되고 있는 프로세스의 쉘로 접속):
`docker exec -it <container_id> /bin/bash`
- 컨테이너 변경사항을 이미지에 커밋:
`docker commit <container_id> <image_id>:<tag>`
- 컨테이너 안에서 패키지를 깔거나, 바인딩 된 볼륨(/notebooks) 이외의 컨테이너 경로에 파일이 추가되면, 이미지에 커밋을 해 주어야만 다음 ‘run’ 시에 변경사항이 적용된 이미지를 본딴 컨테이너가 생성됩니다.

중간 정리: Miscellaneous

- 많이 쓰는 docker 명령어들은 alias, 혹은 스크립트를 짜서 로컬에 놓으면 조금 더 편하게 쓸 수 있습니다.
- 이후에 Tensorboard 실행을 염두해 처음에 컨테이너의 포트를 하나 더 열어주시는 것도 좋습니다. (docker run -it -p 9206:8888 -p 9207:6006 tf/tf:latest)

노출될 포트 설정을 포함한 ‘run’의 옵션들은 이미지로부터 컨테이너가 생성될 때만 설정할 수 있고, 컨테이너로 올린 이후에 변경은 힘듭니다.

(패키지를 다 깔았는데 포트 오픈을 깜빡했다면? -> 변경사항을 이미지에 커밋 후 다시 run!)

Tensorflow-gpu

- Overview
-[Nvidia Graphics Driver] - [Docker] - [Nvidia-docker2] - DONE!
- Nvidia GPU, 그리고 Ubuntu OS를 가정합니다. 처음 해야 할 일은 ~~apt-get update~~ 내 하드웨어에 맞는 그래픽 드라이버를 설치하는 것!
=> ‘ubuntu-drivers devices’로 내 gpu에 맞는 드라이버를 확인하실 수 있습니다.

```
sangsuLee@sangsuLee-desktop:~$ ubuntu-drivers devices
== /sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0 ==
modalias : pci:v000010DEd00001B06sv000010DEsd0000120Fbc03sc00i00
vendor   : NVIDIA Corporation
model    : GP102 [GeForce GTX 1080 Ti]
driver    : nvidia-driver-415 - third-party free recommended
driver    : nvidia-driver-390 - third-party free
driver    : nvidia-driver-410 - third-party free
driver    : nvidia-driver-396 - third-party free
driver    : xserver-xorg-video-nouveau - distro free builtin

sangsuLee@sangsuLee-desktop:~$ █
```

Tensorflow-gpu

- ‘sudo ubuntu-drivers autoinstall’로 recommended driver를 바로 설치하셔도 되고,
- ‘**sudo add-apt-repository ppa:graphics-drivers/ppa**’로 ppa를 추가하신 뒤,
‘**sudo apt update**’ -> ‘**sudo apt install nvidia-415**’
- 이제 재부팅하시고 ‘nvidia-smi’로 확인!

```
sangsulee@sangsulee-desktop:~/code$ nvidia-smi
Fri Dec  7 19:11:22 2018
+-----+
| NVIDIA-SMI 415.13      Driver Version: 415.13      CUDA Version: 10.0 |
|-----+
| GPU  Name     Persistence-MI Bus-Id     Disp.A  Volatile Uncorr. ECC |
| Fan  Temp   Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|-----+
|  0  GeForce GTX 108... Off  00000000:01:00.0 Off   N/A |
|  0%   38C     P8    21W / 260W |      32MiB / 11177MiB |     0%       Default |
+-----+
+
| Processes:                               GPU Memory |
| GPU  PID  Type  Process name          Usage        |
|-----+
|  0    988   G   /usr/lib/xorg/Xorg            22MiB |
|  0   1146   G   /usr/bin/gnome-shell         7MiB  |
+-----+
sangsulee@sangsulee-desktop:~/code$
```

Tensorflow-gpu

- 그 다음으로 Docker는 설치했으니 이제 nvidia-docker2만 설치하면 끝!

Ubuntu 14.04/16.04/18.04, Debian Jessie/Stretch

```
# If you have nvidia-docker 1.0 installed: we need to remove it and all existing GPU containers
docker volume ls -q -f driver=nvidia-docker | xargs -r -I{} -n1 docker ps -q -a -f volume={} | xargs -r dock
sudo apt-get purge -y nvidia-docker
```

```
# Add the package repositories      apt-key에 package repository를 추가하고 apt-get update!
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
  sudo apt-key add -
distribution=$( . /etc/os-release;echo $ID$VERSION_ID )
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update
```

```
# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2      Nvidia-docker2 설치
sudo pkill -SIGHUP dockerd
```

```
# Test nvidia-smi with the latest official CUDA image
docker run --runtime=nvidia --rm nvidia/cuda:9.0-base nvidia-smi
```

잘 설치되었는지 ‘nvidia/cuda:9.0-base’ 이미지를 받아와
‘nvidia-smi’ 명령어를 컨테이너 안에서 실행시키고, 결과를 확인

Tensorflow-gpu

- 이렇게 Docker 명령어에서 ‘`--runtime=nvidia`’ 옵션을 주면 Host의 Nvidia 드라이버가 컨테이너에도 잘 적용될 수 있음을 직접 확인해 주세요!

```
sangsuLee@sangsuLee-desktop:~/code$ docker run --runtime=nvidia --rm nvidia/cuda:9.0-base nvidia-smi  
Unable to find image 'nvidia/cuda:9.0-base' locally  
9.0-base: Pulling from nvidia/cuda  
18d680d61657: Already exists  
0addb6fece63: Already exists  
78e58219b215: Already exists  
eb6959a66df2: Already exists  
6ef1ff668c93: Pull complete  
f5f8f0544aa2: Pull complete  
3d28d96eb352: Pull complete  
Digest: sha256:764039ce9ff2cfb44d646fde6930099493334bb743e5b4f089d820de023c5d9a  
Status: Downloaded newer image for nvidia/cuda:9.0-base  
Sun Dec  9 06:54:20 2018  
+-----+  
| NVIDIA-SMI 415.13      Driver Version: 415.13      CUDA Version: 10.0      |  
+-----+  
| GPU  Name      Persistence-MI Bus-Id      Disp.A  Volatile Uncorr. ECC  |  
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. | | | | | |
|====|=====|=====|=====|=====|=====|=====|=====|  
| 0  GeForce GTX 108... Off  | 00000000:01:00.0 Off |          N/A |  
| 0%   38C    P8    21W / 260W |      32MiB / 11177MiB |     1%   Default |  
+-----+  
  
+-----+  
| Processes:                               GPU Memory |  
| GPU  PID  Type  Process name        Usage  |  
|====|====|====|====|====|====|  
+-----+
```

컨테이너 안에서
'nvidia-smi'
의 실행 결과

Tensorflow-gpu

- 마지막으로, Dockerhub에 있는 official tensorflow-gpu 컨테이너 실행 가이드를 따르면 끝입니다!

The screenshot shows the Dockerhub interface for the tensorflow repository. On the left, there's a sidebar with a search bar and a 'PUBLIC REPOSITORY' section for tensorflow/tensorflow. Below it are tabs for 'Repo Info' and 'Tags'. A red arrow points from the 'Tags' tab towards the list of tags on the right. On the right, a list of tags is displayed with their sizes:

Tag	Size
latest-devel-py3	813 MB
latest-gpu-py3	2 GB
latest-py3	488 MB
latest-devel-gpu	2 GB

Start GPU (CUDA) container

Install nvidia-docker and run

```
$ docker run --runtime=nvidia -it -p 8888:8888 tensorflow/tensorflow:latest-gpu
```

Go to your browser on <http://localhost:8888/>

<https://hub.docker.com/r/tensorflow/tensorflow/>

Tensorflow-gpu

- 그리고 실행시킨 gpu 컨테이너의 jupyter notebook 안에서 gpu를 잘 인식하는지 확인해 보시면 됩니다!

```
In [1]: from tensorflow.python.client import device_lib
def get_available_gpus():
    return [x.name for x in device_lib.list_local_devices()]

get_available_gpus()

Out[1]: ['/device:CPU:0', '/device:XLA_GPU:0', '/device:XLA_CPU:0', '/device:GPU:0']
```

<tensorflow>

- PyTorch의 경우, 바로 전 단계까지 확인하시고 노트북 cell에서 아래처럼 ‘! pip install torch torchvision’로 파이토치를 설치하셔도 되고,
- ‘docker exec -it <container_name> /bin/bash’로 컨테이너 쉘에 새로 접속하신 후 ‘pip install’로 설치하신 후 아래와 같이 확인하시면 됩니다!
- 이 때, 태그가 py3로 명시되어있는 이미지의 경우에도 pip3가 아닌 pip으로 설치하시면 됩니다. (py2가 설치되어있지 않고, python3, pip3가 기본 python, pip으로 설정되어 있습니다.)

```
In [2]: !pip install torch torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.5/dist-pac
kages (1.0.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.5/di
st-packages (0.2.1)
Requirement already satisfied: six in /usr/local/lib/python3.5/dist-packa
ges (from torchvision) (1.11.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.5/
dist-packages (from torchvision) (5.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.5/dist-pac
kages (from torchvision) (1.15.4)
```

```
In [3]: import torch
torch.cuda.get_device_name(0)
```

```
Out[3]: 'GeForce GTX 1080 Ti'
```

<pytorch>

개발 중 컨테이너 안에서 충돌 발생 시

- Tensorflow 컨테이너는 root 권한으로 모든 파이썬 패키지가 설치되어 있습니다.
docker exec 명령어로 컨테이너 쉘에 접속한 뒤, **pip**으로만 패키지를 설치/삭제하시고, 그래도 버전 충돌 / 의존성 충돌이 일어난다면,
- 해당 컨테이너를 내리고 삭제한 뒤 기존에 커밋했던 다른 이미지로 다시 컨테이너를 띄우시거나,
- tensorflow 버전이 문제라고 생각되시면, dockerhub에 있는 tensorflow 이미지의 다른 tag를 받아와 시도해 보시면 됩니다.
- Uninstall 과정 때문인가?(다른 걸 지웠나?), sudo 권한때문에 충돌하는 거 아냐?라는 생각은 안 하게 됩니다.
컨테이너도 새로 금방 올릴 수 있고, 컨테이너 안 패키지 설치 경로도 단순합니다.

Virtual environment tools, Package managing tools vs docker

- Standardization: official image는 99.999% 깨끗하게 잘 돌아갑니다.
(특히 tensorflow-gpu의 경우, Nvidia Driver의 버전만 생각해주면 됩니다)
~~cudnn 버전 어떻게 확인하지? 잘 지워졌나? 안지워져서 충돌 나는거 아냐? cuda가 문제인가? tf랑 호환이 안되는 버전이라고?~~
- 이 위에 필요한 패키지를 깔다 충돌이 나면, 충돌의 원인을 분석하기 더 쉽습니다.
(일단 파이썬과 tensorflow는 깨끗하니까요!)
- OS을 백업하고 다시 설치하기까지는 몇 시간이 걸리지만,
원본 도커 이미지를 다시 올리는데에는 몇 초면 충분합니다(!!)
- commit 명령어로 셋업 중간 지점마다 이미지 스크린샷을 남겨 놓는것도
물론 가능합니다.
- 로컬에는 파이썬조차 설치되어 있지 않아도 됩니다. 항상 아주 깨끗한 상태!

Virtual environment tools, Package managing tools vs docker

- virtualenv(venv activate), conda activate은:
OS를 새로 깔지 않아도, 가상환경에서 깨끗하게 시작할 수 있게 해 줍니다.
하지만 낮은곳부터 직접 설치하며 만들어 주어야 하고,
Host OS 어딘가에는 설치됩니다. ㅎㅎ (저번에 깔아서 그런가 의심..)
- Docker는:
유사하게 사용할 수 있으며, '**여기까지는 문제 없다고 보장된**'
Official image로부터 시작할 수 있고, 로컬을 가장 깨끗하게 유지할 수 있습니다.
하지만 툴에 대해서는 조금 배우셔야 사용하기에 편합니다.

마치면서

로컬을 정말로 깨끗하게 유지하고 싶으시다면,
Docker도 정말 괜찮습니다.

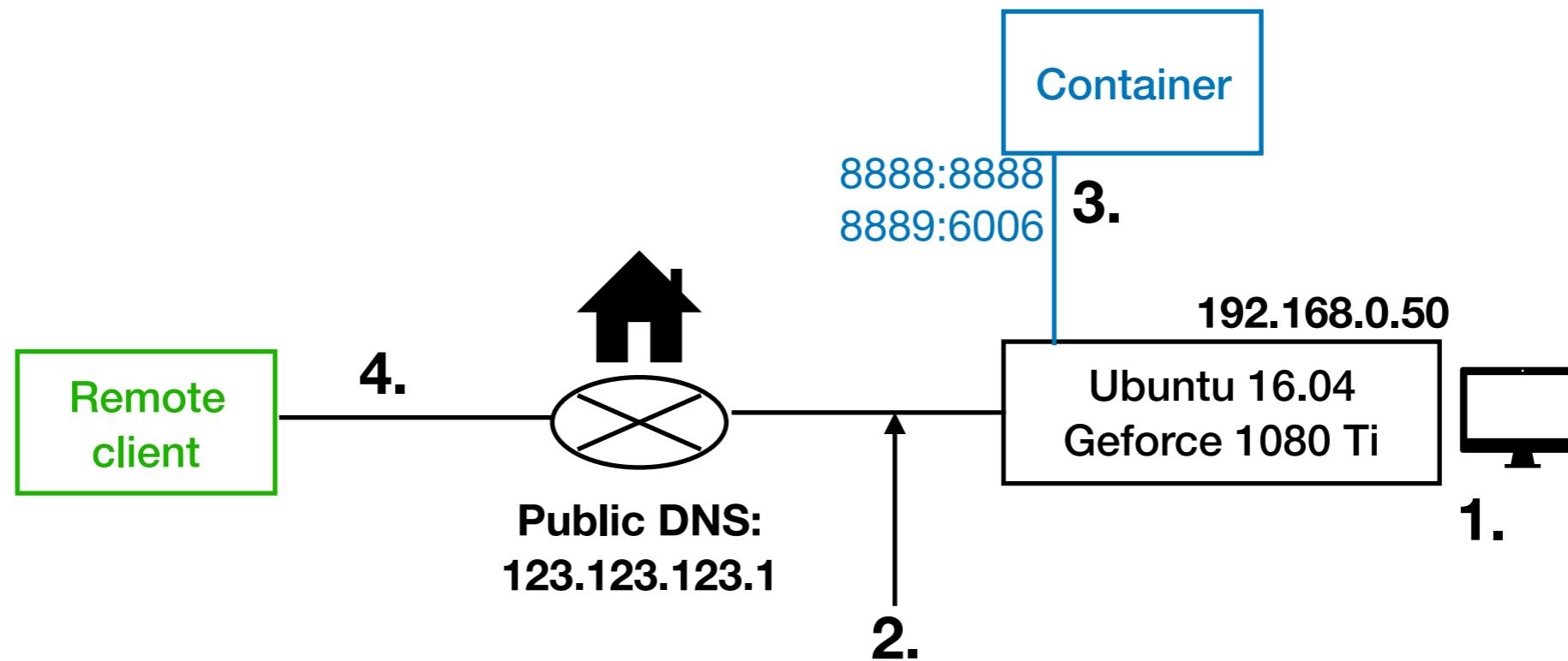
하지만 이제 도커라는 툴 때문에 일어나는 문제들이 새로 생길 수 있는데요,
그래도 딥러닝 이외에 평소에 알아놓으시면 정말 좋은 도구이기 때문에
써보시지 않으셨다면 한번쯤 시도해 보셨으면 좋겠습니다. ^^*

Q & A

부족한 발표이지만 끝까지 들어 주셔서 정말 감사합니다.

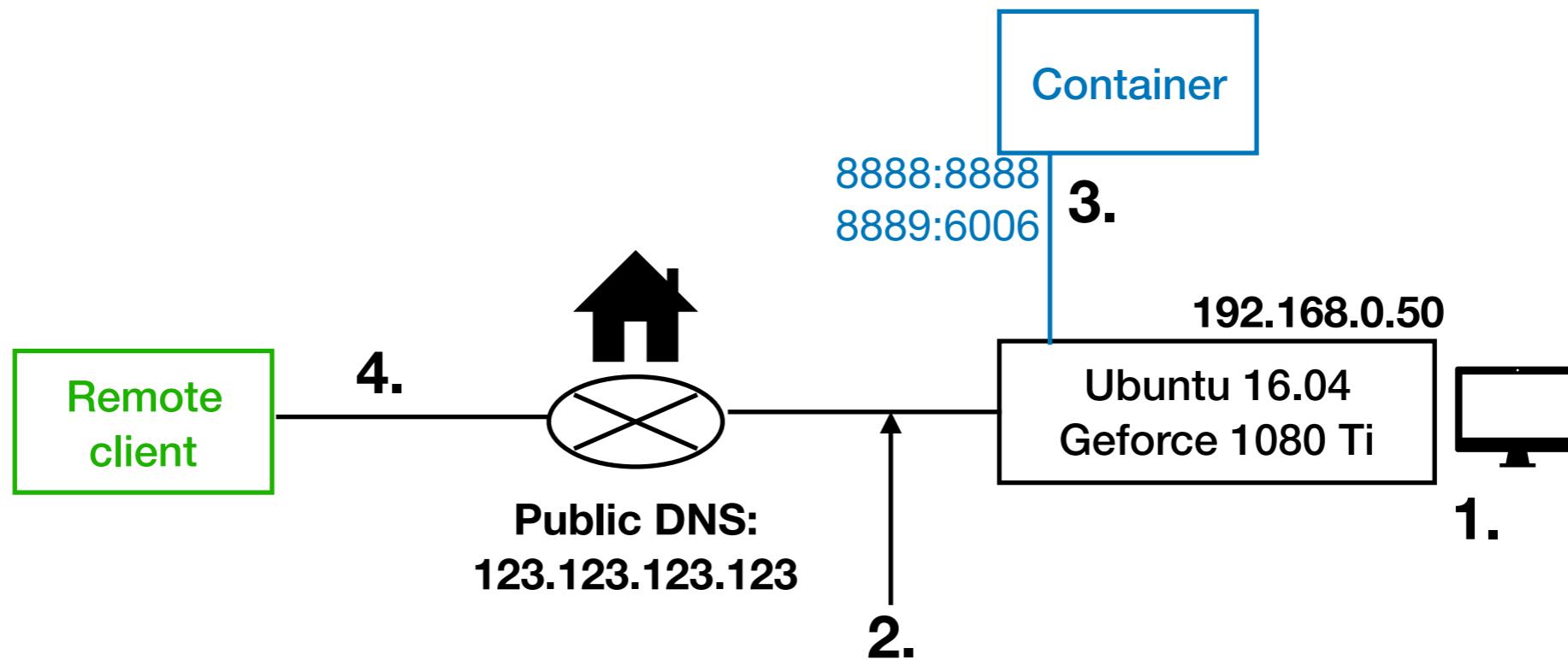
질문의 수준이 높을 시에는 발표자가 답변하지 못할 수 있습니다(..)

부록 1: 원격 개발 환경 예시



6. End

부록 1: 원격 개발 환경 예시



4. 이제 마지막으로 외부에서 브라우저로 <public DNS>:8888로 접속해, 컨테이너의 jupyter notebook이 잘 보이는지 확인합니다. 도커 커맨드들은 서버인 192.168.0.5로 접속해서 실행해야 하는데, 192.168.0.50의 22번 포트로 연결된 <public DNS>:8887로 접근합니다. 그 전에 서버에 ‘openssh-server’ 패키지가 잘 설치되었는지, 공유기 망 내부에서도 정상적으로 서버에 ssh(혹은 putty)를 사용해 접속은 가능한지 확인하신 뒤 외부에서 접속이 가능한지 테스트하시면 됩니다.

Ssh 커맨드는 다음과 같습니다. ‘ssh <public DNS> -p 8887’

부록 2: aliasing

- 자주 사용하는 쉘 커マン드의 경우, `~/.bashrc`, `~/.bash_profile` 등의 rc 파일에 `alias dps='docker ps'` 처럼 aliasing을 명시해 놓으면 편하게 쓸 수 있습니다.
- 본 자료에서처럼 고정된 여러 옵션들을 포함한 스크립트를 작성하려면, `run.sh`(예) 파일을 새로 생성한 뒤 안에 완전한 커マン드를 작성하시고, '`chmod +x run.sh`'로 실행권한을 준 뒤, `./run.sh`로 실행시키면 됩니다.

```
#!/bin/bash
docker run -d -it --name tfcon --runtime=nvidia \
-v ${HOME}/code/myvol:/notebooks \
-p 9206:8888 -p 9207:6006 -p 9208:9208 \
tensorflow/tensorflow:latest-gpu-py3
```

<run.sh (예시)>

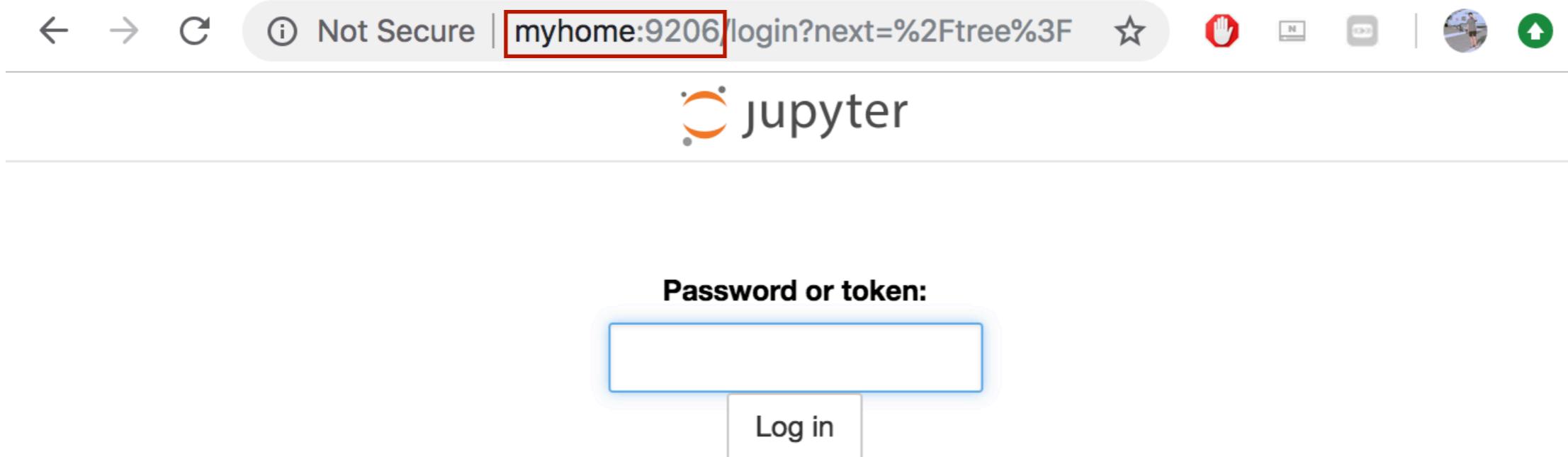
부록 3: host aliasing(1)

- 자동완성이 있지만, 브라우저에 public DNS를 항상 입력하기 귀찮다면?(123.123.123.123)
- sudo vi /etc/hosts 등 접속하려는 로컬의 에디터를 사용해 아래와 같이 ‘myhome’ 엔트리를 IP와 함께 추가하고 저장하면, 브라우저에서도 별명으로 바로 접근이 가능합니다.
- localhost 등 다른 엔트리들을 건드리거나 지우지 않도록 주의해 주세요!

```
127.0.0.1    localhost
123.123.123.1(예시)   myhome
255.255.255.255 broadcasthost
::1           localhost
```

부록 3: host aliasing(2)

- 자동완성이 있지만, 브라우저에 public DNS를 항상 입력하기 귀찮다면?(123.123.123.123)
- sudo vi /etc/hosts 등 접속하려는 로컬의 에디터를 사용해 아래와 같이 ‘myhome’ 엔트리를 IP와 함께 추가하고 저장하면, 브라우저에서도 별명으로 바로 접근이 가능합니다.



Token authentication is enabled

If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you [enable a password](#).

The command:

6. End