

機械学習のためのdocker

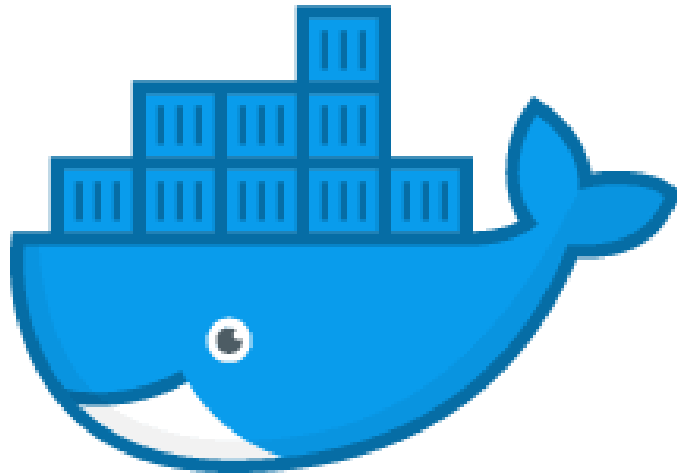
2020/10/14

木南 貴志

1.docker

docker

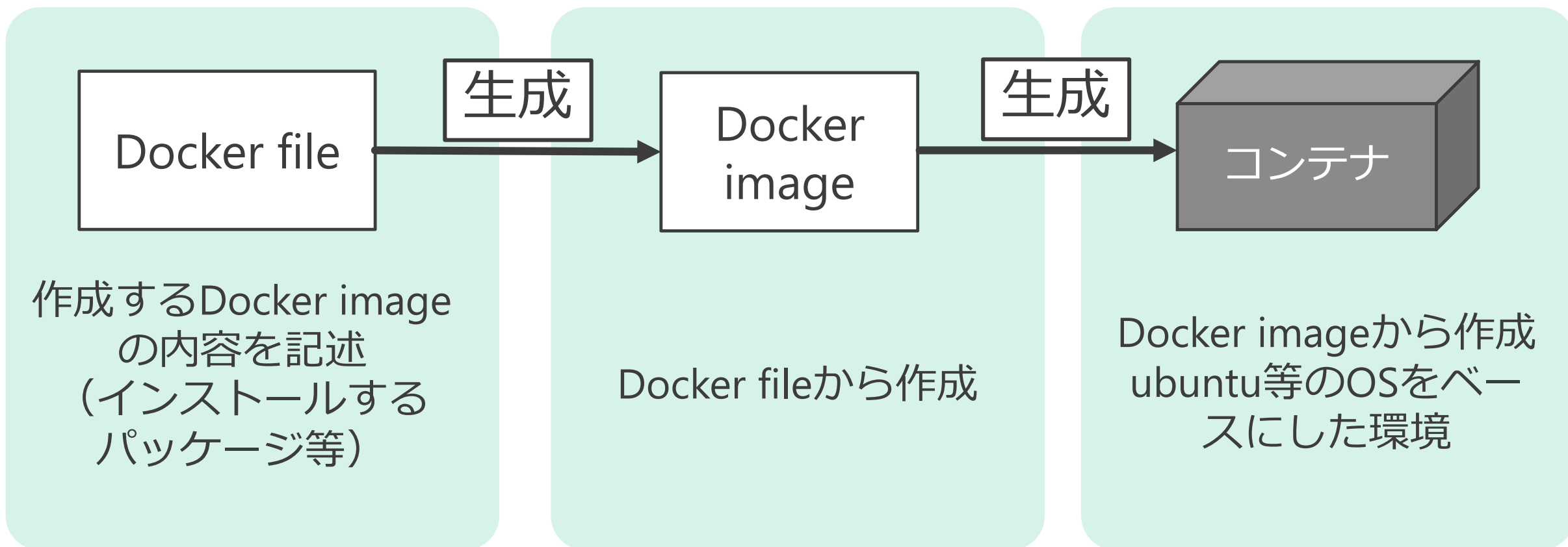
docker : 一つのPC内に仮想環境（のようなもの）を作成するツール



docker

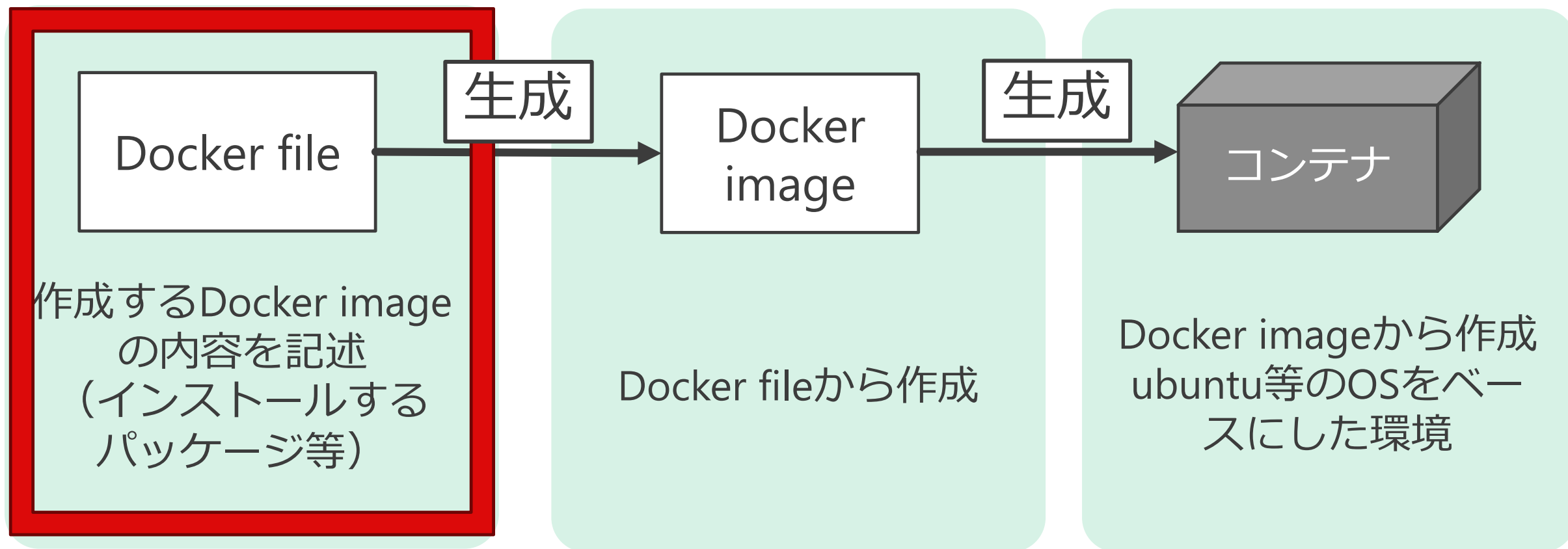
コンテナ作成 流れ

コンテナ作成のおおまかな流れ



コンテナ作成 流れ

コンテナ作成のおおまかな流れ



2.Docker file

Dockerfile

Docker file

環境構築のための
情報を記述したもの

**Docker fileの名前は
「Docker file」にすること**
※ buildの際「Docker file」と
いう名前のファイルを探して
buildするため

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
WORKDIR /opt
RUN wget
https://repo.anaconda.com/archive/Anaconda3-2020.02-
Linux-x86_64.sh && /
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p
/opt/anaconda3 && /
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
cupy-cuda90==5.3.0 /
    chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```

instruction

instruction : Docker fileに記述する際のコマンドのようなもの

FROM <base-Docker-image>

説明 : ベースのimage (主にubuntu等のOS) を指定

RUN <command>

説明 : FROMで指定したOSで使用できるコマンドを実行

ENV <path>

説明 : pathを通す (インストールしたパッケージ等の)

CMD <command>

説明 : コンテナ生成時に実行されるコマンドを指定

FROM

FROM <Docker image> : ベースとするDocker imageを記述（だいたいOS）

➡ このOSの上にパッケージインストール等して環境を構築

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
```



Docker hub（Docker imageを管理するサイト）
からベースimageを持ってくる

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
WORKDIR /opt
RUN wget
https://repo.anaconda.com/archive/Anaconda3-
2020.02-Linux-x86_64.sh && /
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p
/opt/anaconda3 && /
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
    cupy-cuda90==5.3.0 /
    chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```

ベースimageを探す

docker hubで使いたいDocker imageを検索

The screenshot shows the Docker Hub search results for the query 'nvidia/cuda'. The search bar at the top left contains the text 'nvidia/cuda'. Below the search bar, a dropdown menu shows the search results, with 'nvidia/cuda' selected. The search results are displayed in a table with columns for the repository name, status, stars, downloads, and visibility. The repository 'nvidia/cuda' is listed with a status of 'Not Scanned', 0 stars, 5 downloads, and is public. To the right of the search results, there is a 'Create Repository' button. Below the search results, there is a tip: 'Tip: Not finding your repository? Try switching namespace via the top left dropdown.' On the right side of the page, there are two promotional banners. The top banner is titled 'Create an Organization' and 'Manage Docker Hub repositories with your team'. The bottom banner is titled 'New Subscription Plans Available!' and 'Starting at just \$5/month, get'.

docker hub

Search: nvidia/cuda

Community (4)

kinamitakashi

kinamitakashi / my-f
Updated 5 days ago

nvidia/cuda

nvidia/cudagl

nvidia/cuda-ppc64le

nvidia/cuda-arm64

Create Repository

Not Scanned ☆ 0 ↓ 5 Public

Tip: Not finding your repository? Try switching namespace via the top left dropdown.

Create an Organization
Manage Docker Hub repositories
with your team


New Subscription Plans
Available!
Starting at just \$5/month, get

ベースimageを探す

Tagsから使用したいimageのバージョンを確認

Explore > [nvidia/cuda](#) >

Using 0 of 1 private repositories. [Get more](#)



nvidia/cuda ☆

By [nvidia](#) • Updated an hour ago

CUDA and cuDNN images from [gitlab.com/nvidia/cuda](#)

Container

↓ Pulls **10M+**

Overview **Tags**

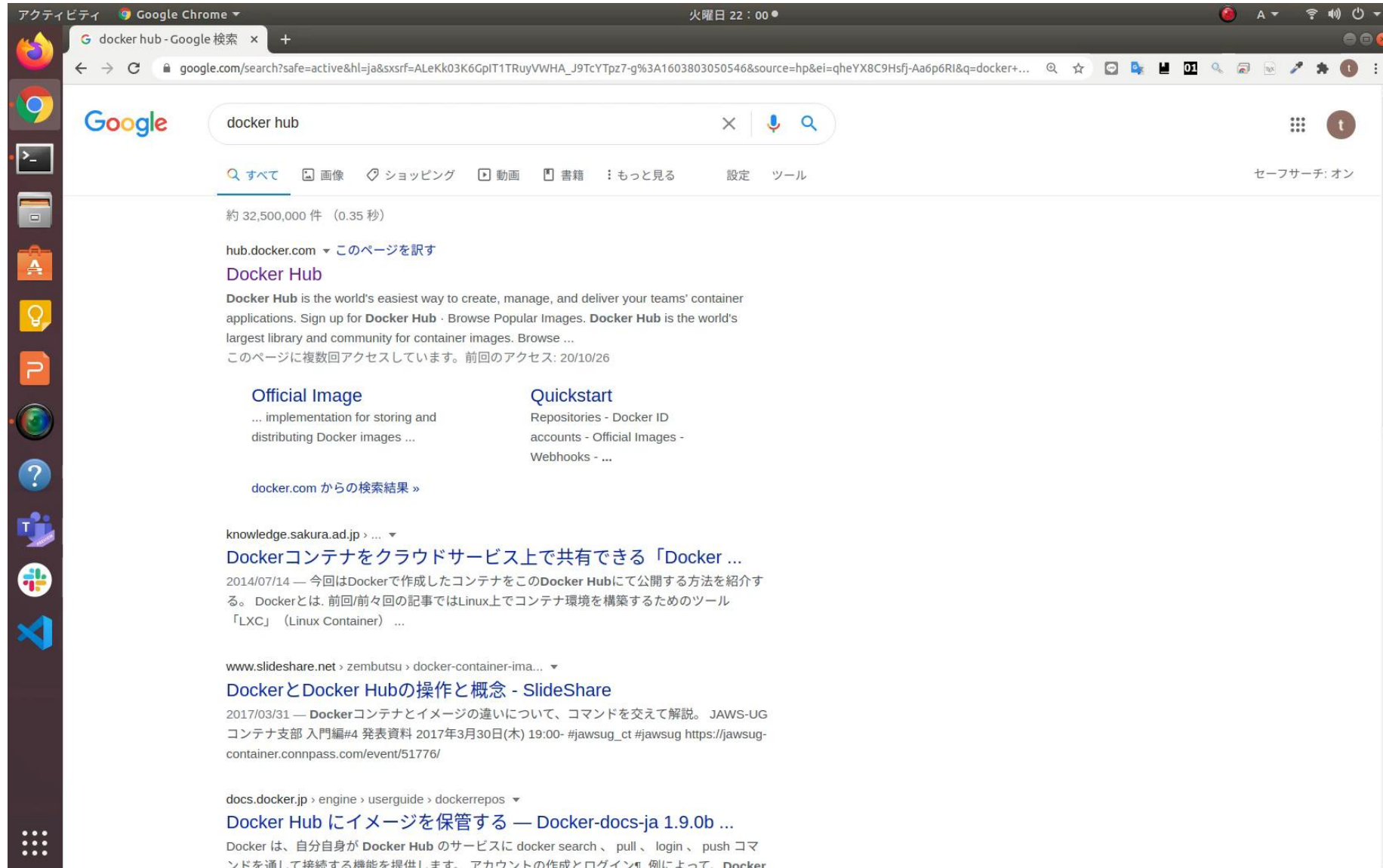
×

TAG	DIGEST	OS/ARCH	COMPRESSED SIZE ⓘ
9.0-cudnn7-devel-ubuntu16.04	08a0aa469494	linux/amd64	1.32 GB

この「docker pull」以降を DockerfileのFROMに記述

`docker pull nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04`

FROM



Dockerfileの作成, 記述



RUN

RUN <command> : ベースOSで利用できるコマンドを記述

```
RUN apt-get update && apt-get install -y /  
    sudo /  
    wget /  
    vim
```

ベースOSで利用できるコマンド
(linux系ならcdやls等) を記述

必要となるパッケージ等をインストールする等
して環境を設定

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04  
RUN apt-get update && apt-get install -y /  
    sudo /  
    wget /  
    vim  
WORKDIR /opt  
RUN wget https://repo.anaconda.com/archive/Anaconda3-  
2020.02-Linux-x86_64.sh && /  
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p  
/opt/anaconda3 && /  
    rm -f Anaconda3-2020.02-Linux-x86_64.sh  
ENV PATH /opt/anaconda3/bin:$PATH  
RUN pip install --upgrade pip && pip install /  
cupy-cuda90==5.3.0 /  
    chainer==5.3.0  
WORKDIR /  
CMD ["/bin/bash"]
```

WORKDIR

WORKDIR <path> : 作業ディレクトリを変更する

WORKDIR /opt

RUN等でコマンドを実行する場所を変更

注意

下記のような場合「hoge」は「/opt」ではなく「/」に作成される

```
WORKDIR /  
RUN cd /opt  
RUN mkdir hoge
```

RUN cd /opt && mkdir hoge で一度に実行すれば「/opt」に作成される

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04  
RUN apt-get update && apt-get install -y /  
    sudo /  
    wget /  
    vim  
WORKDIR /opt  
RUN wget https://repo.anaconda.com/archive/Anaconda3-  
2020.02-Linux-x86_64.sh && /  
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p  
/opt/anaconda3 && /  
    rm -f Anaconda3-2020.02-Linux-x86_64.sh  
ENV PATH /opt/anaconda3/bin:$PATH  
RUN pip install --upgrade pip && pip install /  
cupy-cuda90==5.3.0 /  
    chainer==5.3.0  
WORKDIR /  
CMD ["/bin/bash"]
```

ENV

ENV <path> : パスを通す

ENV PATH /opt/anaconda3/bin:\$PATH

パッケージ等がすぐに使用できるようにパスを通しておく

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
WORKDIR /opt
RUN wget https://repo.anaconda.com/archive/Anaconda3-
2020.02-Linux-x86_64.sh && /
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p
/opt/anaconda3 && /
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
cupy-cuda90==5.3.0 /
    chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```


CMD

CMD ["command"] : コンテナ作成時に実行するコマンドを指定

CMD ["/bin/bash"]

コンテナを作成する際に実行. 原則最後の行に記述.

/bin/bash の場合, シェルを操作できるようになる

※ 特に理由がない場合, /bin/bashを設定することが多い

このコマンドは下記の様にコンテナ作成時に上書き可能

```
$docker run --gpus all cuda9.0cnn7chainer5.3.0 nvidia-smi
```

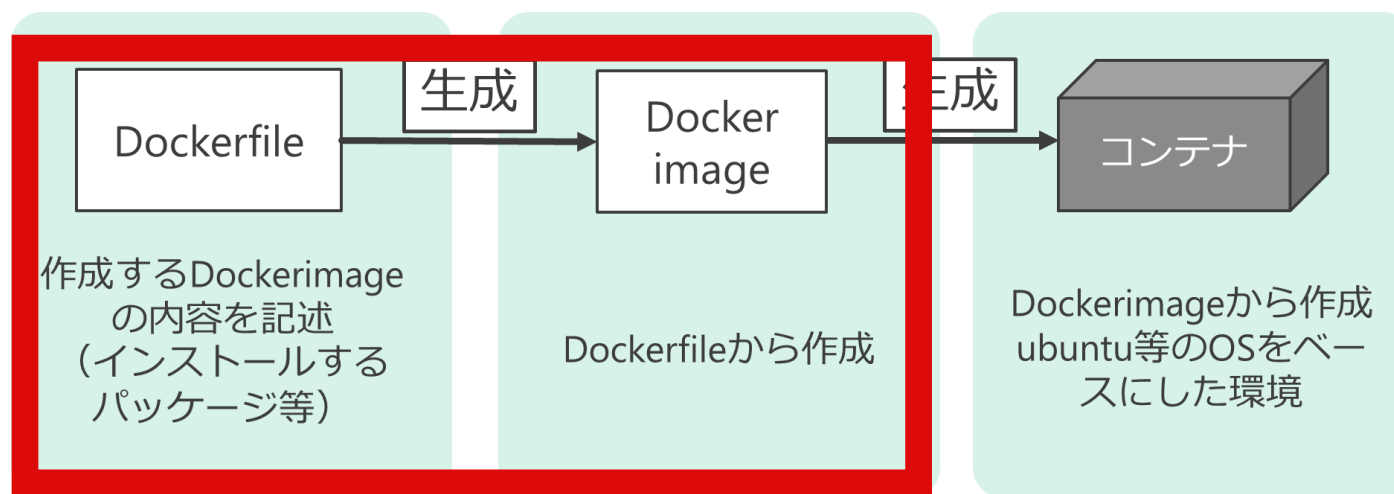
※ **/bin/bash** の代わりに **nvidia-smi** が実行される

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
  sudo /
  wget /
  vim
WORKDIR /opt
RUN wget https://repo.anaconda.com/archive/Anaconda3-
2020.02-Linux-x86_64.sh && /
  sh Anaconda3-2020.02-Linux-x86_64.sh -b -p
/opt/anaconda3 && /
  rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
cupy-cuda90==5.3.0 /
  chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```

docker build

Docker file ➡ Docker imageを生成

```
$docker build -t <Docker imageの名称（任意）> <Docker fileのある場所（相対パス）>
```



例) Docker fileがある場所が現在地 (.) , 生成するDocker imageの名前をcuda9.0chainer5.3.0とする場合

```
$docker build -t cuda9.0cudnn7chainer5.3.0 .
```

docker build 1回目

```
gizmo@gizmo-MS-7B98: ~/doc
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
gizmo@gizmo-MS-7B98:~/docker$
```

```
ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H)
🐳 Dockerfile ×
home > gizmo > docker > 🐳 Dockerfile > ...
1 FROM ubuntu:16.04
2
3 CMD ["/bin/bash"]
```

docker build 2回目 (キャッシュ使用)

```
gizmo@gizmo-MS-7B98: ~/dockl  
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)  
gizmo@gizmo-MS-7B98:~/docker$ docker build -t ubuntu16.04 .  
  
ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H)  
Dockerfile ×  
home > gizmo > docker > Dockerfile > ...  
1 FROM ubuntu:16.04  
2  
3 CMD ["/bin/bash"]
```

キャッシュを使用するので
一瞬でbuild完了

Docker image

Docker imageはDocker layerを重ねたもの(1layer=1instruction)

Docker image

Docker-layer4

Docker-layer3

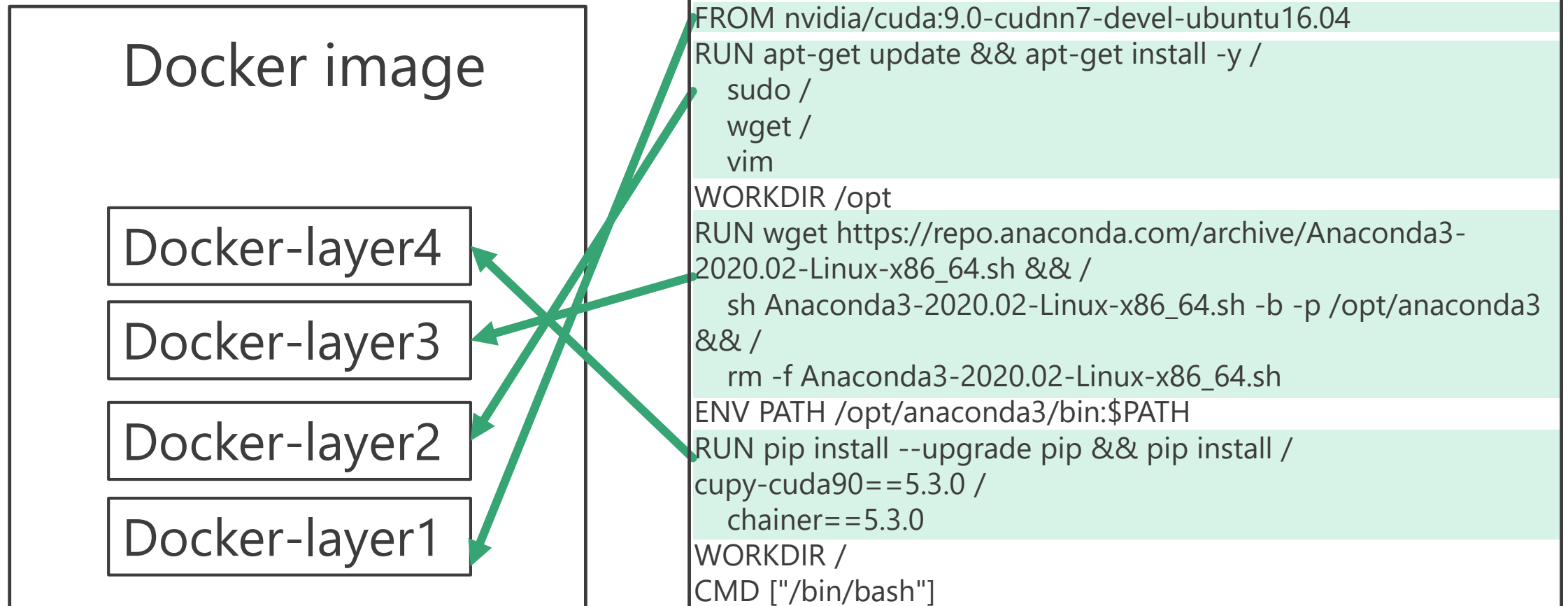
Docker-layer2

Docker-layer1

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
WORKDIR /opt
RUN wget https://repo.anaconda.com/archive/Anaconda3-
2020.02-Linux-x86_64.sh && /
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p /opt/anaconda3
&& /
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
cupy-cuda90==5.3.0 /
    chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```

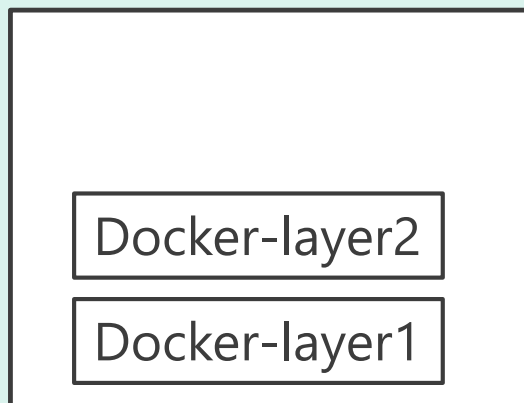
Docker image

Docker imageはDocker layerを重ねたもの(1layer=1instruction)



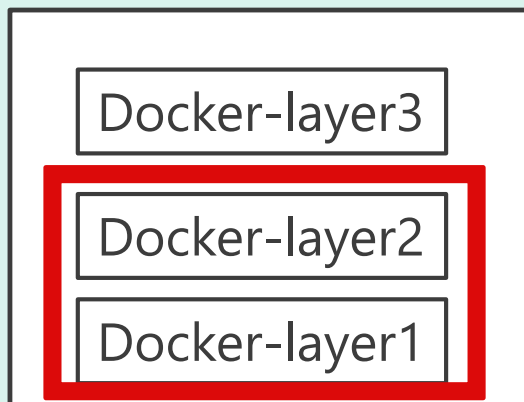
Docker layerを増やすメリット

layer毎にbuildされるため、build時間が短縮



1回目

```
FROM ubuntu:16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
```



2回目

```
FROM ubuntu:16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
```

build済

```
RUN apt-get install -y /
    python
```

layer1, 2はbuild済のためlayer3のみbuild

Docker layerを増やすデメリット

instructionが増えるとdocker imageの容量も増加

```
RUN apt-get update && apt-get install -y /  
sudo /  
wget /  
vim
```

build後のimage file

➡ 容量小

```
RUN apt-get update  
RUN apt-get install -y sudo  
RUN apt-get install -y wget  
RUN apt-get install -y vim
```

build後のimage file

➡ 容量大

同じ処理内容だがbuild後のimage fileの容量は下例の方が大

環境構築の進め方

環境構築中は、instructionを1行ずつ追加（buildの時間削減）

➡ 構築し終わったらDocker imageの容量を小さくするため、instructionをまとめる

```
RUN apt-get update  
RUN apt-get install -y sudo  
RUN apt-get install -y wget  
RUN apt-get install -y vim
```

環境構築中

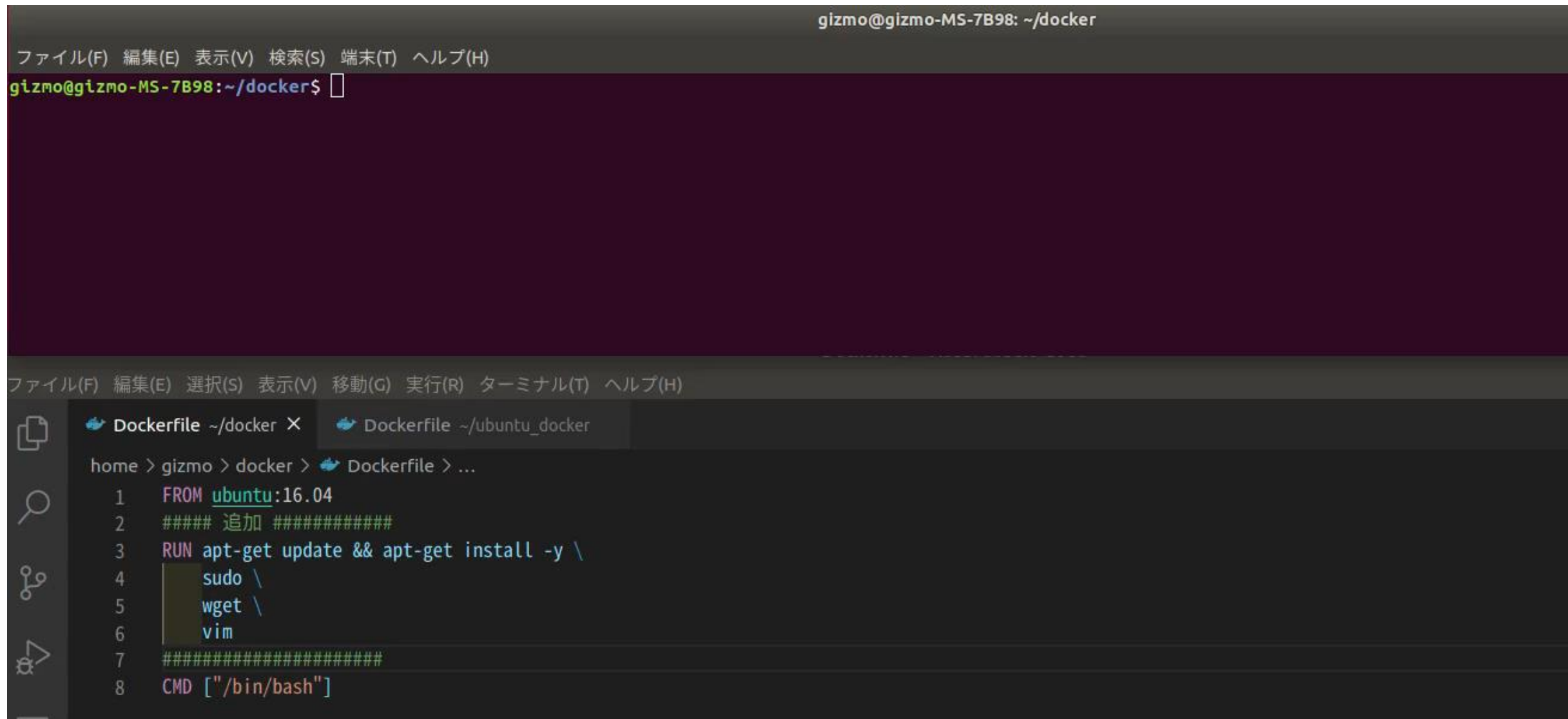
➡ instructionを多用

```
RUN apt-get update && apt-get install -y /  
    sudo /  
    wget /  
    vim
```

環境構築終了後

➡ instructionを最小限にまとめる

RUN



The screenshot displays a Docker IDE interface. At the top, a terminal window titled 'gizmo@gizmo-MS-7B98: ~/docker' shows a menu bar with 'ファイル(F)', '編集(E)', '表示(V)', '検索(S)', '端末(T)', and 'ヘルプ(H)'. Below the menu, the prompt 'gizmo@gizmo-MS-7B98:~/docker\$' is followed by a cursor. The bottom section of the IDE contains a file explorer on the left and a code editor on the right. The file explorer shows two tabs: 'Dockerfile ~/docker' (active) and 'Dockerfile ~/ubuntu_docker'. The code editor displays the content of the active Dockerfile, which includes instructions for building a container from Ubuntu 16.04, installing packages, and setting the entry point.

```
home > gizmo > docker > Dockerfile > ...  
1 FROM ubuntu:16.04  
2 ##### 追加 #####  
3 RUN apt-get update && apt-get install -y \  
4     sudo \  
5     wget \  
6     vim  
7 #####  
8 CMD ["/bin/bash"]
```

RUNを追加して*build*

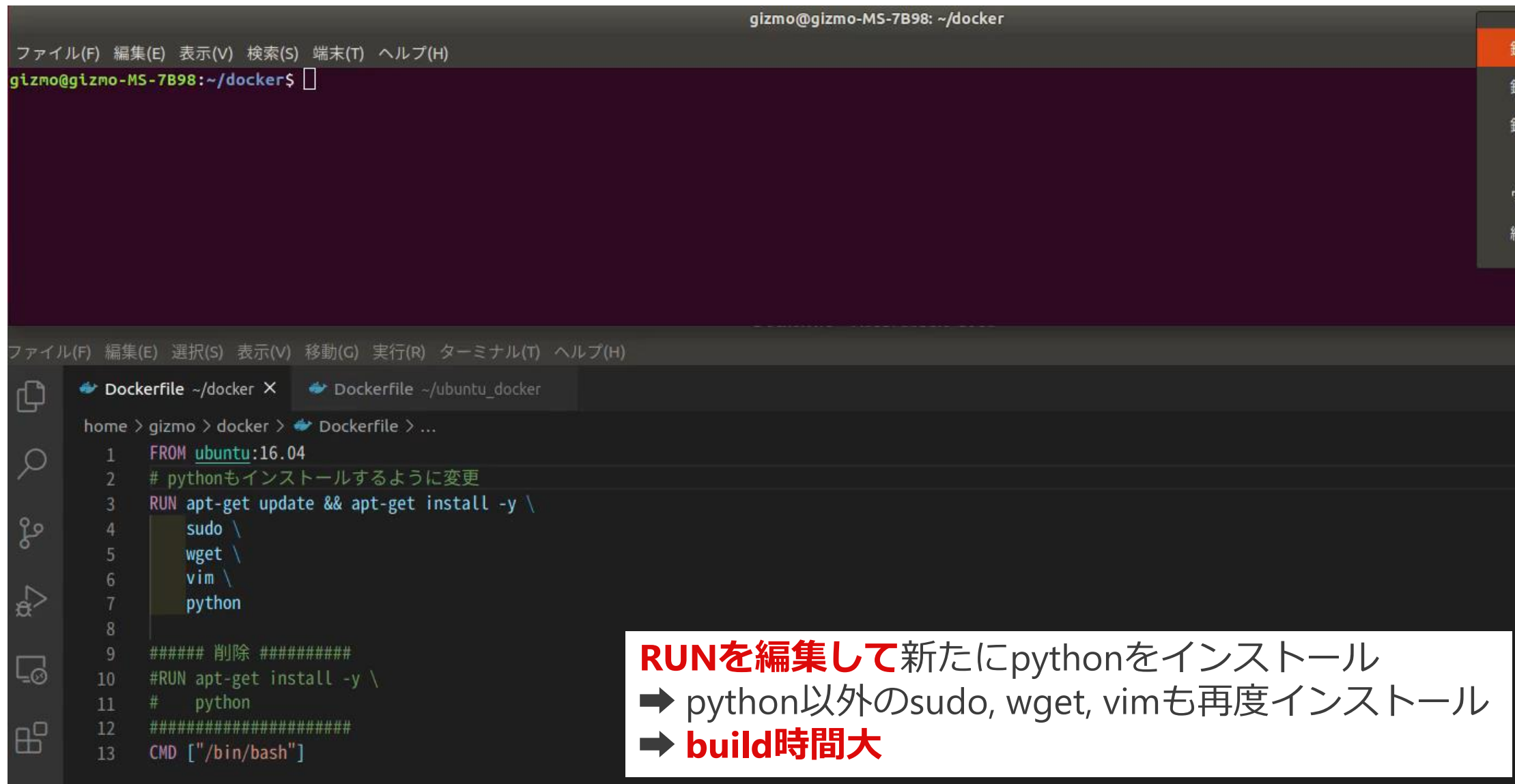
```
gizmo@gizmo-MS-7B98: ~/docker
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
gizmo@gizmo-MS-7B98:~/docker$
```

```
Dockerfile - Visual Studio Code
ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H)
```

```
Dockerfile ~/docker X Dockerfile ~/ubuntu_docker
home > gizmo > docker > Dockerfile > ...
1 FROM ubuntu:16.04
2 RUN apt-get update && apt-get install -y \
3     sudo \
4     wget \
5     vim
6 ##### 追加 #####
7 RUN apt-get install -y \
8     python
9 #####
10 CMD ["/bin/bash"]
```

RUNを追加して新たにpythonをインストール
➡ python以外はキャッシュを使用
➡ **build時間短縮**

*RUN*を編集して*build*



The screenshot shows a Docker IDE interface. The top terminal window displays the prompt `gizmo@gizmo-MS-7B98: ~/docker$`. The bottom editor window shows a Dockerfile with the following content:

```
1 FROM ubuntu:16.04
2 # pythonもインストールするように変更
3 RUN apt-get update && apt-get install -y \
4     sudo \
5     wget \
6     vim \
7     python
8
9 ##### 削除 #####
10 #RUN apt-get install -y \
11 # python
12 #####
13 CMD ["/bin/bash"]
```

On the right side of the editor, a red box contains the following text:

RUNを編集して新たにpythonをインストール
➡ python以外のsudo, wget, vimも再度インストール
➡ **build時間大**

image fileの比較

add-run : RUNを追加してbuildしたimage file ➡ 容量大

not-add-run : RUNを編集してbuildした image file ➡ 容量小

```
gizmo@gizmo-M  
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)  
gizmo@gizmo-MS-7B98:~/docker$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
not-add-run         latest             52691c2ed585       20 seconds ago    245MB  
add-run             latest             acb20431ce92       3 minutes ago     246MB  
apt-get-install     latest             9d2140c436a9       6 minutes ago     222MB  
ubuntu16.04         latest             c4425c725cc7       13 minutes ago    131MB  
ubuntu              16.04             dfeff22e96ae       3 days ago        131MB
```

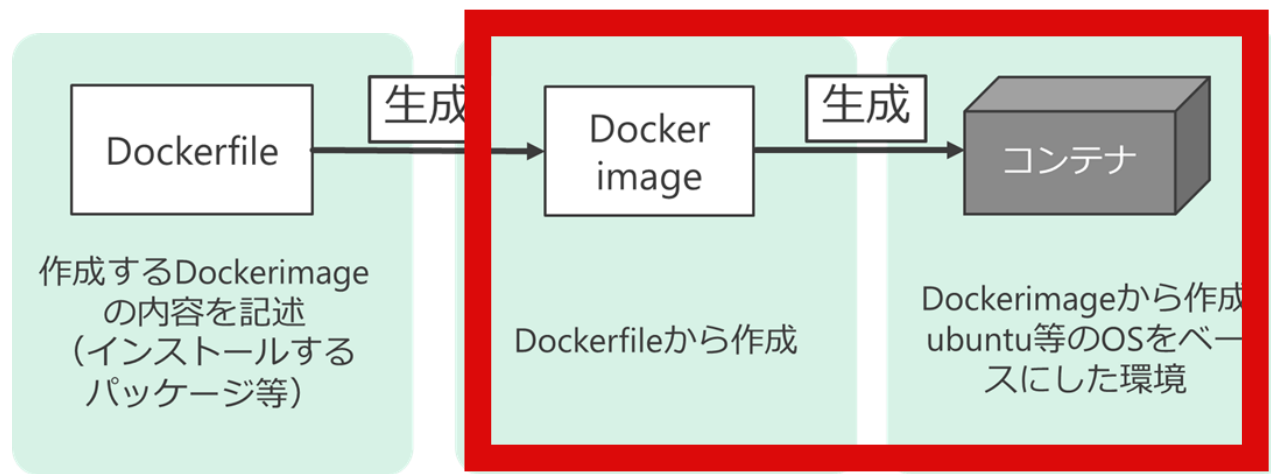
RUNを多用するとimage fileの容量も増大化

(今回は追加でpythonをインストールしただけなので1MBしか変わらないが)

docker run

Docker image ➡ containerを生成 (+コンテナ内に入る)

```
$docker run -it --name=<コンテナの名称 (任意)> <Docker image> <コンテナ起動時に実行したいコマンド>
```



例) Docker image 「cuda9.0cnn7chainer5.3.0」 からコンテナ 「cuda90cudnn7chainer530」 を生成

```
$docker run -it --name=cuda90cudnn7chainer530 cuda9.0cudnn7chainer5.3.0 bash
```

docker run

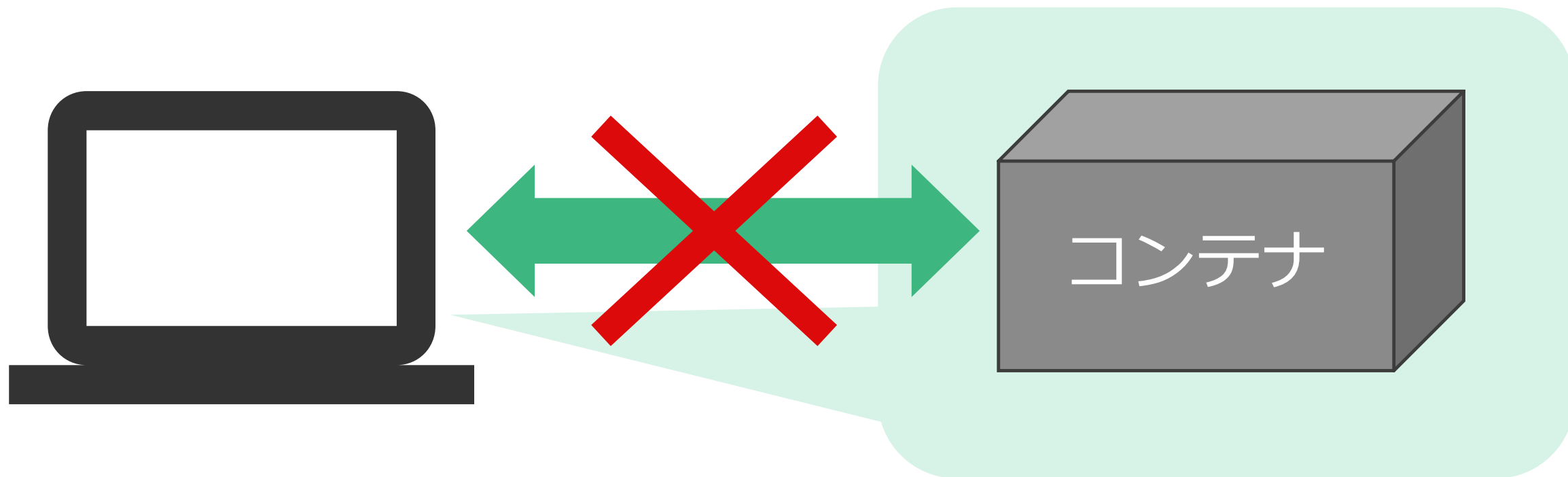


The screenshot shows a terminal window with a dark background. The title bar at the top reads "gizmo@gizmo-MS-7B98: ~/docker". Below the title bar, a menu bar is visible with the following items: "ファイル(F)", "編集(E)", "表示(V)", "検索(S)", "端末(T)", and "ヘルプ(H)". The main area of the terminal displays the prompt "gizmo@gizmo-MS-7B98:~/docker\$" followed by a cursor. On the right side of the terminal window, there is a vertical sidebar with several buttons: "録画を開始" (Start Recording), "録画をキャンセル" (Cancel Recording), "録画したものを保存" (Save Recorded Content), "ウィンドウ" (Window), and "終了" (Exit).

```
gizmo@gizmo-MS-7B98: ~/docker
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
gizmo@gizmo-MS-7B98:~/docker$
```

ホスト↔コンテナでファイル共有

通常、ホスト（自分のPC）↔コンテナの間でファイルの共有は不可

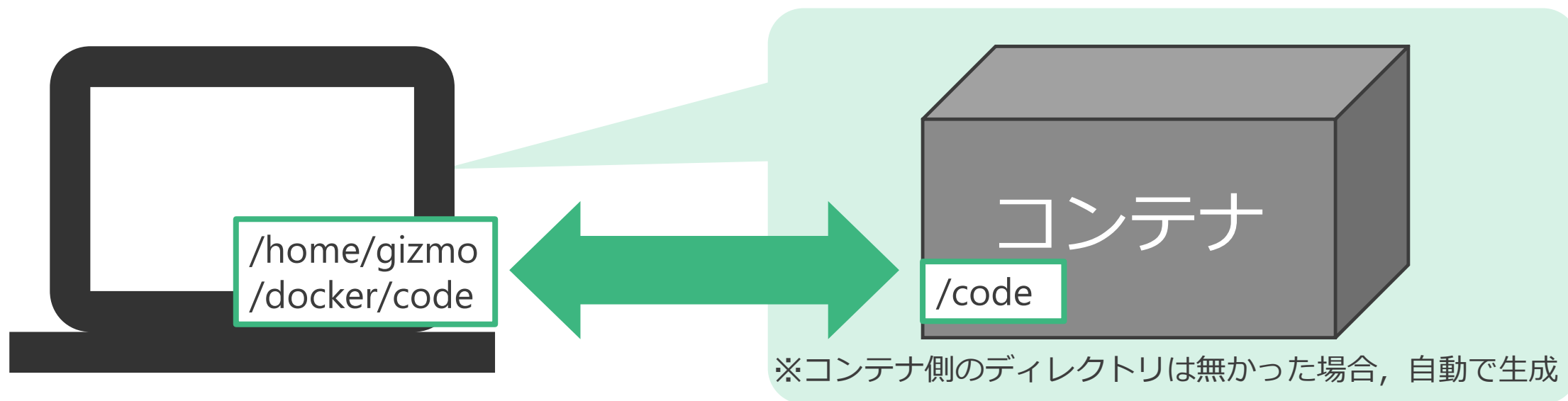


ホスト↔コンテナでファイル共有

-vオプションでディレクトリのマウントが可能

```
$docker run -it -v /home/gizmo/docker/code:/code cuda9.0cudnn7chainer5.3.0
```

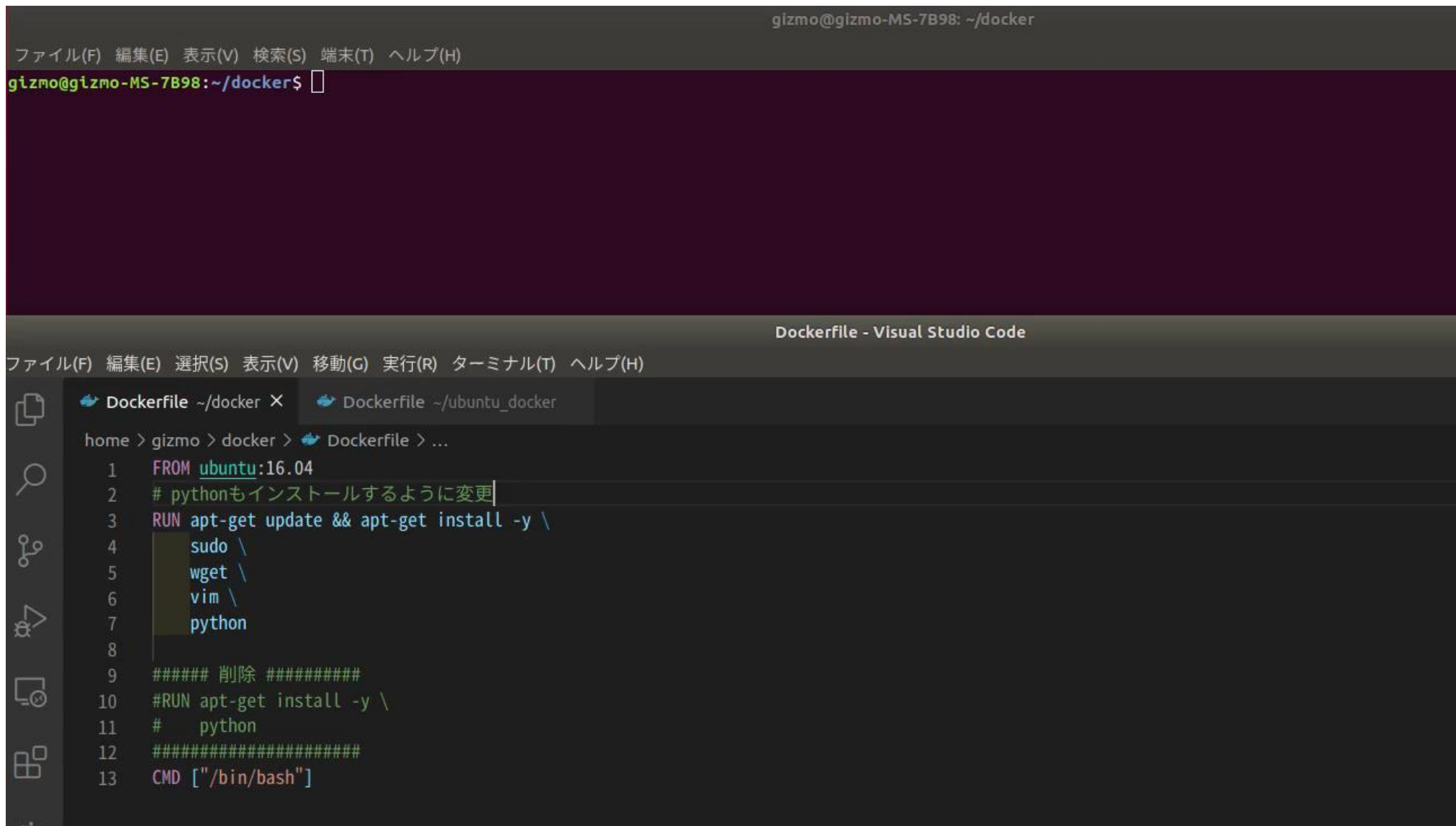
-v <ホスト側のマウントしたいディレクトリまでのパス>:<コンテナ側のパス>



ホストの「`/home/gizmo/docker/code`」内のファイルがコンテナの「`/code`」に反映

※参照しているだけなのでホスト側でファイルの追加・編集するとコンテナ側にも反映（逆も）

ホスト↔コンテナでファイル共有



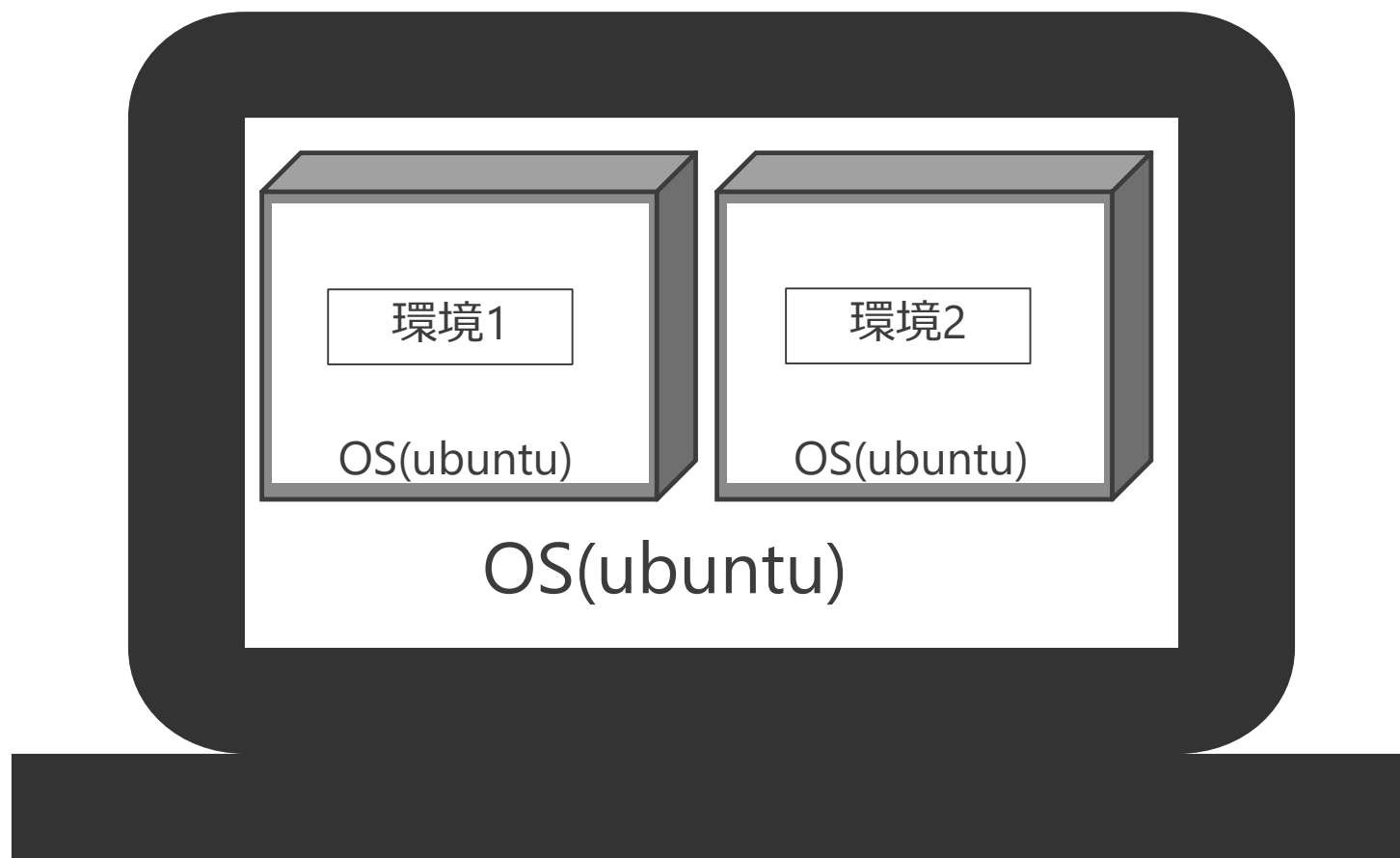
The image shows a terminal window at the top and a Visual Studio Code editor window below it. The terminal window has a title bar 'gizmo@gizmo-MS-7B98: ~/docker' and a menu bar 'ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)'. The prompt is 'gizmo@gizmo-MS-7B98:~/docker\$' followed by a cursor. The Visual Studio Code editor window has a title bar 'Dockerfile - Visual Studio Code' and a menu bar 'ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H)'. It shows two tabs: 'Dockerfile ~/docker' (active) and 'Dockerfile ~/ubuntu_docker'. The active tab displays the following Dockerfile content:

```
home > gizmo > docker > Dockerfile > ...
1 FROM ubuntu:16.04
2 # pythonもインストールするように変更
3 RUN apt-get update && apt-get install -y \
4     sudo \
5     wget \
6     vim \
7     python
8
9 ##### 削除 #####
10 #RUN apt-get install -y \
11 #     python
12 #####
13 CMD ["/bin/bash"]
```

2.機械学習を行うための docker

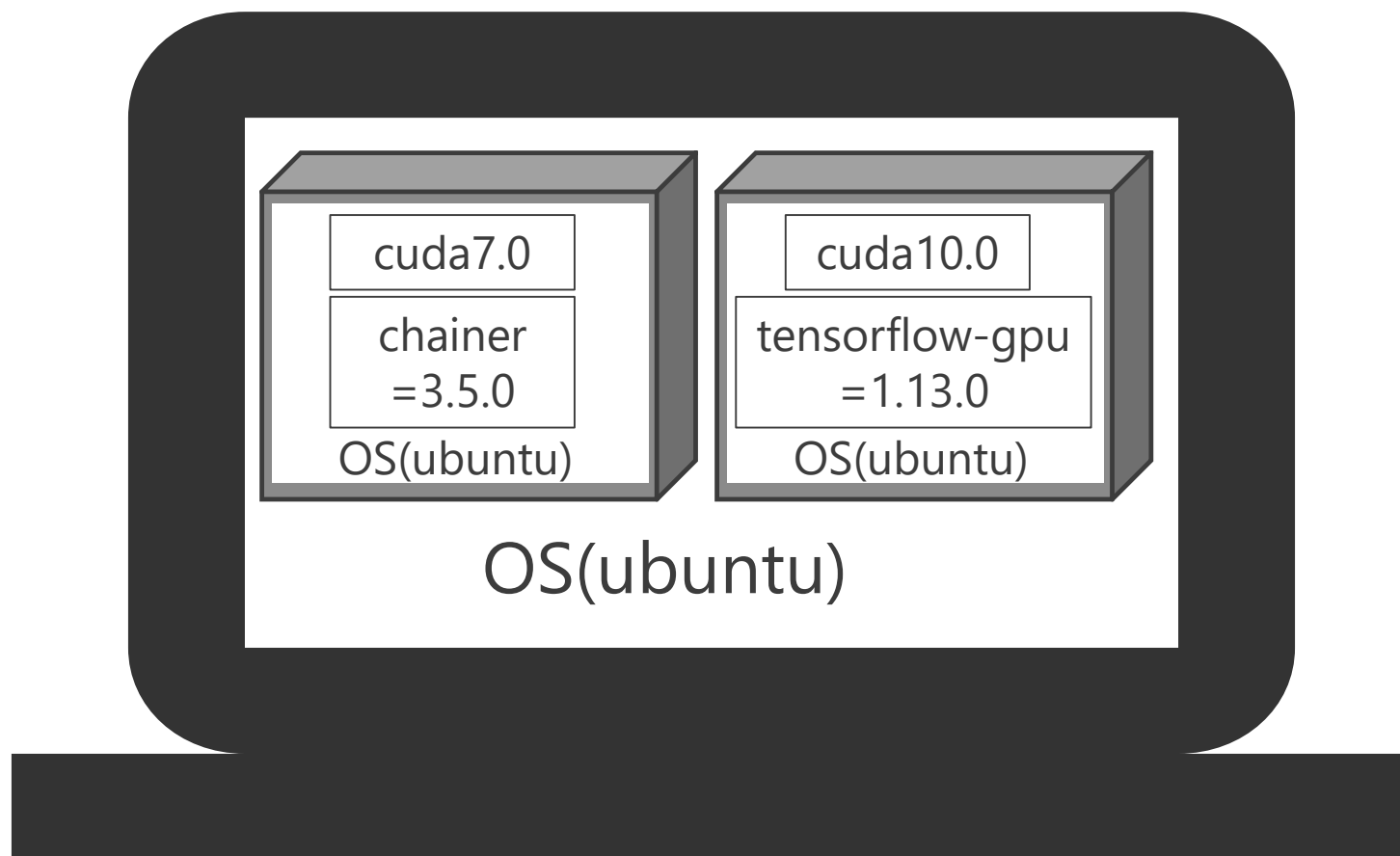
機械学習でコンテナを使う利点①

各コンテナで異なる環境を構築可能



機械学習でコンテナを使う利点①

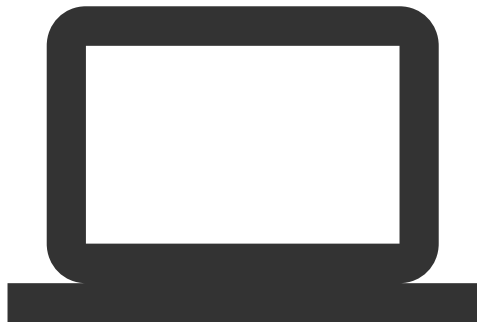
例) コンテナ1にcuda7.0, コンテナ2にcuda10.0の環境を構築



機械学習でコンテナを使う利点②

間違った環境構築をした場合 ➡ コンテナから抜けるだけでOK

PC本体に構築した場合



- パッケージのアンインストール
- 競合発生（ROSとAnaconda等）
 - ➡ 競合解消のため色々試行錯誤
- どうしようもなかったら最悪OSから入れ直し

コンテナに構築した場合

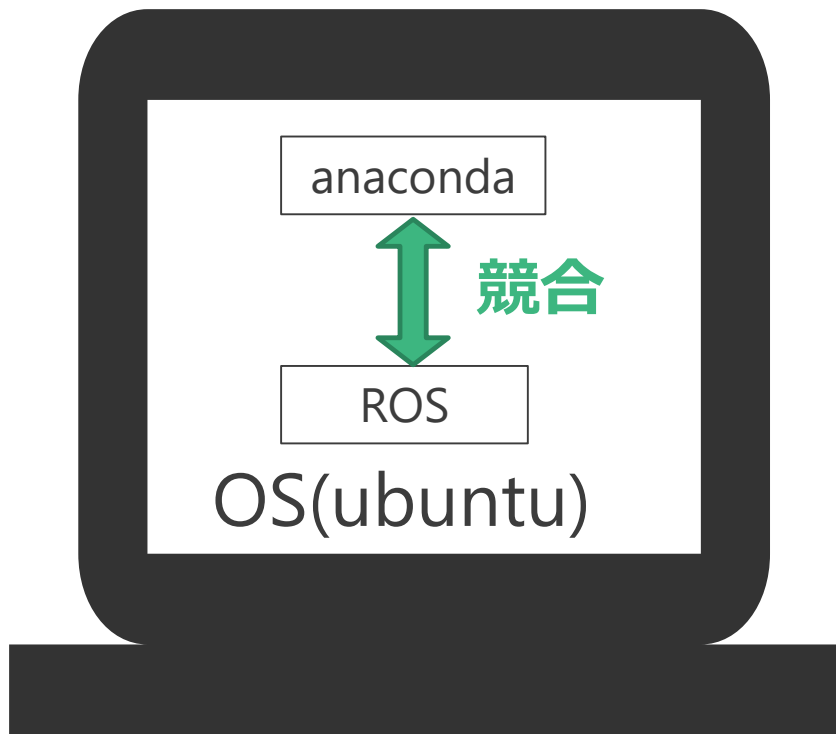


コンテナから抜ける. 以上

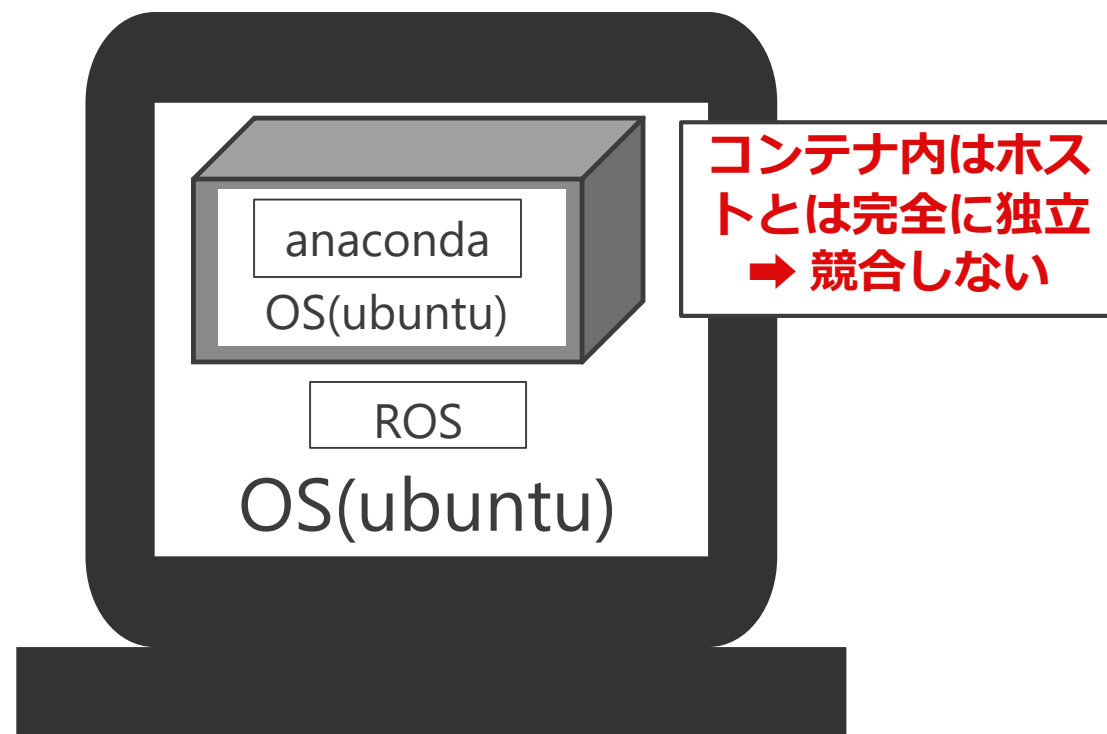
機械学習でコンテナを使う利点③

コンテナ内は完全に独立した環境 ➡ 競合の回避

(ただし, 設定すればホスト↔コンテナ間, コンテナ↔コンテナ間で共有することも可)



競合の可能性あり

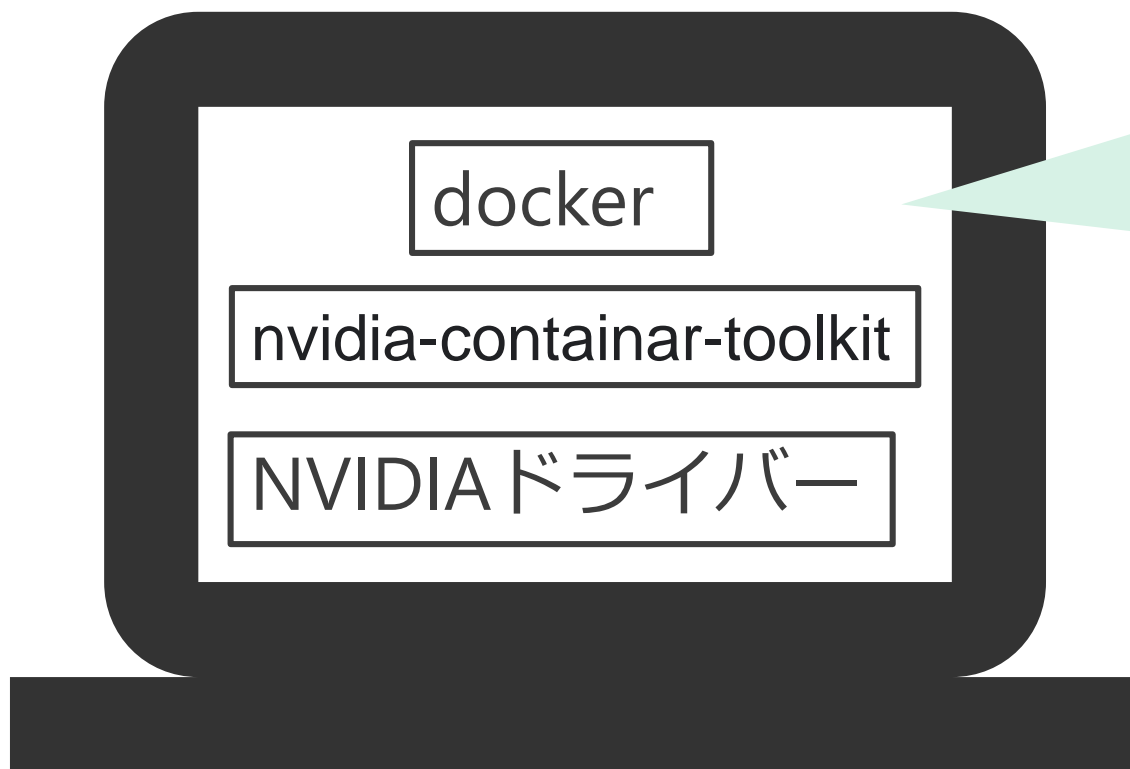


競合しない

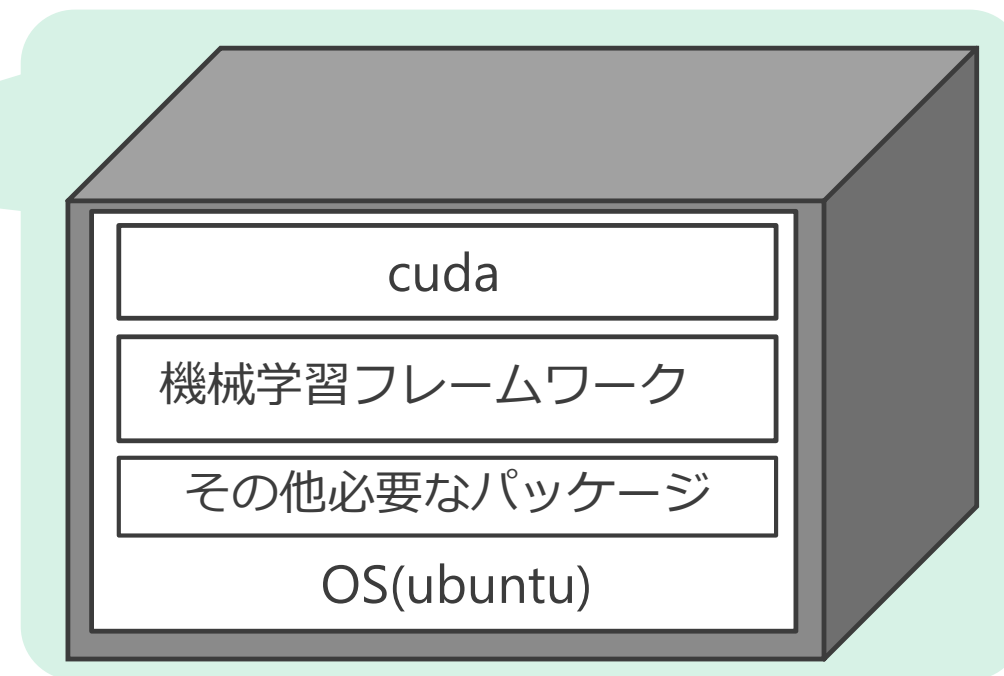
ホストとコンテナの関係性

機械学習の場合のイメージ

docker, nvidia-container-toolkitのみ
ホスト側にインストール



それ以外 (cuda, 機械学習フレームワーク, python等) 全てをコンテナ内のOS上にインストール



手順

1. Dockerをインストール
 2. 自分のPCに搭載されているGPUの, NVIDIA-driver（基本最新でOK）をインストール
 3. nvidia-container-toolkitをインストール（1.～3.は初回のみ）
 4. Dockerfileを記述
 5. Dockerfile ➡ image-fileを作成（ビルド）
 6. image-file ➡ containerを作成（container = 実際に作業する環境）
- ※ 環境構築が完了するまで, 4.～6.を繰り返す
7. 環境構築が完了したら, container内で作業（プログラム実行等）

Dockerをインストール

UbuntuにDockerEngineをインストールする

<https://docs.docker.com/engine/install/ubuntu/>

**※Dockerは頻繁に更新されるため、上記の公式のページを参考にして
インストールした方が無難**

dockerをsudoなしでログイン

<https://qiita.com/DQNEO/items/da5df074c48b012152ee>

NVIDIA driver 確認

```
$ lspci | grep -i nvidia
```

自分のPCに搭載のGPUを調べる

- 調べたGPUに対応するNVIDIA driverの最新verを確認
<https://www.nvidia.co.jp/Download/index.aspx?lang=jp>

- NVIDIA driverとCUDAのverの対応表を確認
<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>

調べたNVIDIA driverの最新verが、CUDAのどのverまでサポートしているか確認

※使用したいCUDAにGPUが対応していない場合、GPUを新しい物に買い換える必要あり
例)CUDA9.0を使用したい→(linuxOSの場合)NVIDIA driverの最新verが384.81以上ならばOK

参考サイト <https://qiita.com/konzo/items/a6f2e8818e5e8fcdb896>

NVIDIA driver インストール

- 調べたGPUに対応するNVIDIA driverの最新verのインストールファイルをダウンロード
<https://www.nvidia.co.jp/Download/index.aspx?lang=jp>

```
$ chmod +x <ダウンロードしたファイル>
```

実行できるように

```
$ sudo ./<ダウンロードしたファイル>
```

ドライバインストール

```
$ reboot
```

再起動

```
$ nvidia-smi
```

ドライバがインストールされているか確認

NVIDIA driver インストール

※ nvidia-dmが使用中というエラーが出たら以降を実行

```
$ systemctl isolate multi-user.target
```

CUIモードに変更

CUIモードになったら alt+F1 → ユーザ名、パスワードを入力でログイン

```
$ modprobe -r nvidia-drm
```

無効化

```
$ modprobe -r nvidia-modeset
```

無効化

```
$ sudo ./<ダウンロードしたファイル>
```

ドライバインストール

```
$ reboot
```

再起動

```
$ nvidia-smi
```

ドライバがインストールされているか確認

*nvidia-container-toolkit*をインストール

nvidia-containar-toolkitをインストール

<https://github.com/NVIDIA/nvidia-docker>

**※Dockerは頻繁に更新されるため、上記の公式のページを参考にして
インストールした方が無難**

機械学習の際の*Docker file*の主な構成

```
FROM nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04
RUN apt-get update && apt-get install -y /
    sudo /
    wget /
    vim
WORKDIR /opt
RUN wget
https://repo.anaconda.com/archive/Anaconda3-2020.02-
Linux-x86_64.sh && /
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p
/opt/anaconda3 && /
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
ENV PATH /opt/anaconda3/bin:$PATH
RUN pip install --upgrade pip && pip install /
cupy-cuda90==5.3.0 /
    chainer==5.3.0
WORKDIR /
CMD ["/bin/bash"]
```

使用したいubuntu, cuda, cudnnのverを指定

anacondaのインストール

<https://repo.anaconda.com/archive>
で検索して任意のverのanacondaをダウンロード&
インストールが終わったらshファイルは削除
これで, python関連 (numpy, matplotlib等) は
ほとんどインストールされる

使用したい機械学習のフレームワークと
verを指定してインストール

Docker file作成の際の注意点

docker buildの際は、キーボードによる入力不可

➡ キーボード入力をしなくていいような工夫が必要

```
RUN apt-get update && apt-get install -y /  
    sudo /  
    wget /  
    vim
```

-yをオプションに指定 ➡ キーボード入力を要求されたら全て「y」で返答

```
RUN wget https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh && /  
    sh Anaconda3-2020.02-Linux-x86_64.sh -b -p /opt/anaconda3 && /  
    rm -f Anaconda3-2020.02-Linux-x86_64.sh
```

-bをオプションに指定 ➡ バッチモード（キーボード入力不要モード）でインストールを実行

※ -bオプションはshファイル共有のものではないため注意

機械学習におけるコンテナ作成の流れ

基本的に、コンテナ作成 ➡ 破壊を繰り返すことが多い

Dockerfile編集

Docker file ➡ Docker image生成

```
$docker build -t <Docker imageの名称 (任意) > <Docker fileのある場所 (相対パス) >
```

Dockerimage ➡ コンテナ生成(runの際に--rmをオプションにつけるとexit時にコンテナが自動で削除)

```
$docker run --gpus all -it --rm --name=<コンテナの名称 (任意) > -v <ホスト側のマウントしたいディレクトリまでの絶対パス>:<コンテナ側のディレクトリまでの絶対パス> <Dockerimage> <コンテナ起動時に実行したいコマンド>
```

環境構築途中

環境構築済

環境構築

作業開始 (プログラム実行など)

コンテナから抜ける

exit

コンテナから抜ける

exit

上手く環境構築できなかった
または、環境構築に必要な
パッケージ等判明
➡ Dockerfileに内容を反映

作業終了
(マウントしておけば、
追加・編集したファイル
等はホスト側に残る)

機械学習におけるコンテナ作成の流れ (例)

基本的に、コンテナ作成 ➡ 破壊を繰り返すことが多い

Dockerfile編集

Docker file ➡ Docker image生成

```
$docker build -t cuda9.0cudnn7 .
```

Docker image ➡ コンテナ生成(runの際に--rmをオプションにつけるとexit時にコンテナが自動で削除)

```
$docker run --gpus all -it --rm --name=cuda90cudnn7 -v /home/gizmo/docker/files:/files  
cuda9.0cudnn7 bash
```

環境構築途中

環境構築済

環境構築

作業開始 (プログラム実行など)

コンテナから抜ける

exit

コンテナから抜ける

exit

上手く環境構築できなかった
または、環境構築に必要な
パッケージ等判明
➡ Dockerfileに内容を反映

作業終了
(マウントしておけば、
追加・編集したファイル
等はホスト側に残る)

補足 機械学習のdocker run オプション

docker runで使用するオプションを説明

```
$docker run --gpus all -it --rm --name=cuda90cudnn7 -v /home/gizmo/docker/files:/files  
cuda9.0cudnn7 bash
```

- it : きれいに表示する. このコマンドがないと上手く表示されない (詳細は省略)
- v <host>:<container> : ホストのディレクトリとコンテナのディレクトリをマウントする.
※ コンテナ内に, 指定したディレクトリが存在しない場合は自動でディレクトリが生成される
- gpus : ホスト側のGPUをコンテナ内で使用. allを指定すると (ホストに複数のGPUが搭載されている場合) 全てのGPUをコンテナで使用
- rm : exitでコンテナから抜けた後に, コンテナが自動で削除される

コンテナ内でのGPUの認識

```
$docker run --gpus all (以下略)
```

でコンテナを起動したときにエラーが発生した場合, NVIDIAドライバ, nvidia-container-toolkitが正しくインストールされていない可能性あり

ホストとコンテナ内の両方で

```
$nvidia-smi
```

を実行した時, ホストとコンテナ内で内容が異なっていた場合 (CUDAのバージョンにERRと表示されている等) も同様にドライバのインストールができていない可能性あり

3.dockerのコマンド

コマンド一覧

dockerで使用することの多いコマンド

(Dockerimage, containerを指定する場合, IDか名前のどちらかを指定)

```
$ docker build <path_to_Dockerfile>
```

Dockerfile → Dockerimageを生成

options: -t : Dockerimageに名前をつける

-f <Dockerfilename> : Dockerfileに「Dockerfile」以外の名前をつけている場合

```
$ docker run <Dockerimage> <command>
```

Dockerimageからコンテナを生成

※ <command>を指定しない場合DockerfileのCMDで指定したコマンドを実行

exitしたらコンテナが停止する

options: -v <host>:<path> : 指定したディレクトリをマウント

--name <container_name> : コンテナに名前をつける

-it : きれいに表示する (詳細は省略)

--gpus <number> : ホストのGPUをコンテナ内で使用 (allでホストにある全てのGPUをコンテナで使用)

--rm : exitでコンテナから抜けた後にコンテナを削除 (指定しない場合, 停止はするが削除はされない)

-p <host_port>:<container_port> : ホストのポートとコンテナのポートを接続

-u <user_id><group_id> : ユーザIDとグループIDを指定 (通常はrootでログインする)

※ -u \$(id -u) \$(id -g) と指定すれば直接IDを入力しなくて良い

コマンド一覧

dockerで使用することの多いコマンド

(Dockerimage, containerを指定する場合, IDか名前のどちらかを指定)

```
$ docker exec <container> <command>
```

コンテナに再度入る

※ <command>を指定しない場合DockerfileのCMDで指定したコマンドを実行

options: -it : きれいに表示する (詳細は省略)

exitした場合もコンテナが停止しない.

```
$ docker attach <container> <command>
```

コンテナに再度入る

※ <command>を指定しない場合DockerfileのCMDで指定したコマンドを実行

options: -it : きれいに表示する (詳細は省略)

exitしたらコンテナが停止する

```
$ docker restart <container>
```

停止したコンテナを再アクティブ化

※ 停止している場合exec, attachができないため, 停止したコンテナに入りたい場合はこれを実行してから

コマンド一覧

dockerで使用することの多いコマンド

(Dockerimage, containerを指定する場合, IDか名前のどちらかを指定)

\$ docker commit <container> <new_Dockerimage> **コンテナ ➡ Dockerimageを生成**

\$ docker tag <befor_Dockerimage> <after_Dockerimage> **Dockerimageの名前変更**

Dockerimageの名前は「リポジトリ名:タグ名」

\$ docker push <Dockerimage> **Dockerimageをdocker hubにpush**

pushする際は, Dockerimageの名前は「pushする先のリポジトリ名:タグ名」にすること

(docker hubにpushする際にDockerimageを参照して, そのリポジトリにpushするため)

コマンド一覧

dockerで使用することの多いコマンド

(Dockerimage, containerを指定する場合, IDか名前のどちらかを指定)

```
$ docker rm <container>
```

コンテナを削除

停止していないコンテナは削除不可

```
$ docker stop <container>
```

コンテナを停止

```
$ docker system prune 停止しているコンテナ, 名前のついていないdockerimageを全て削除
```

```
$ docker rmi <Dockerimage>
```

Dockerimageを削除

コマンド一覧

コンテナ内で使用できるコマンド

\$ exit

コンテナから抜ける（抜けた後コンテナ停止）

\$ detach

コンテナから抜ける（抜けた後もコンテナ動作継続）

参考資料

参考サイト (**超おすすめ**)

https://datawokagaku.com/docker_lecture/

↑セールは行わないとのことなので、期間限定の割引クーポンを使用して講座を購入することをおすすめします