Docker を使っていい感じに環境を作る

目次

はじめに

この資料について:

- 研究に使うための開発環境の構築に Docker container を使いたいときの参考資料の作成.
  - あまり時間をかけずに使えるように，やるべきことは必要最低限にとどめている.
- VSCode の拡張機能を使っている.

導入

Dockerとは...

コンテナ型 の仮想環境を作成，配布，実行するために用いられるプラットフォーム

- Docker の利点
  - 環境の再現性
  - 配布可能である

ホストOS のカーネルを利用している → 軽量，高速！

アプリケーションごとに隔離 → 各コンテナは独立

インストール

ここでは，Windows に Docker をインストールする場合について説明する.

手順

1. WSL2 の有効化とインストール
2. Docker Desktop のインストール

WSL2 の有効化

参考:
【Windows10】WSL2の有効化とUbuntuのインストール方法
以前のバージョンの WSL の手動インストール手順

コントロールパネル > プログラム > プログラムと機能
にある『Windowsの機能の有効化または無効化』で

- Linux 用 Windows サブシステム
- 仮想マシンプラットフォーム

にチェックを入れ，OK → 再起動.

PowerShell を管理者として開き，以下のコマンドを実行．

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all
```

このコマンドを実行することで，Virtual Machine Platform を有効にする．コマンド実行後，再起動する．

以前のバージョンの WSL の手動インストール手順 の手順4 にある『x64 マシン用 WSL2 Linux カーネル更新プログラム パッケージ』をダウンロードし，実行する．

PowerShell を管理者として開き，以下のコマンドを実行.

```
wsl --set-default-version 2
```

このコマンドを実行することで，WSL のデフォルトのバージョンを2にする.

その後，Microsoft Store で 好きなディストリビューションをインストールする.
Ubuntu 20 あたりがいいと思う.

Remark

最近は，PowerShell で管理者権限で

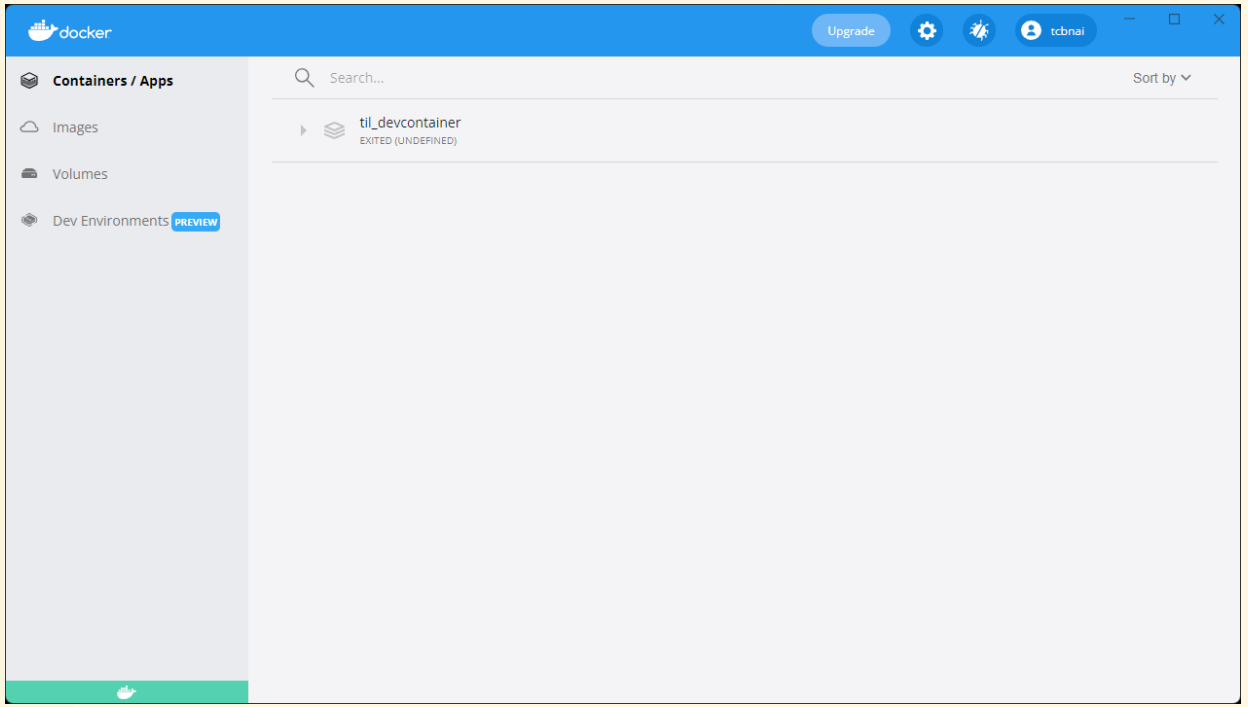
```
wsl --install
```

とすると，WSL を実行するために必要なすべてをインストールすることができるようになったらしい.

参考: WSL のインストール

Docker Desktop のインストール

公式サイト から exe ファイルをインストールして，実行.
指示に従ってインストールするだけなので，簡単.



設定

- WSL2 のメモリ食いすぎ問題
  - `C:\Users\[username]\` に `.wslconfig` ファイルを作る.

`.wslconfig` の書き方 (参考: Configure Linux distributions):

```
[wsl2]
kernel=C:\\temp\\myCustomKernel
memory=4GB # Limits VM memory in WSL 2 to 4 GB
processors=2 # Makes the WSL 2 VM use two virtual processors
```

用語と最低限のコマンド

- コンテナ
  - 実行環境を他のプロセスから隔離し，その中でアプリケーションを動作させる技術
- イメージ
  - Docker コンテナの実行に必要なパッケージをまとめたもの

Docker は `docker` コマンドで操作する.

```
docker [コマンド] [操作] [オプション]
```

最低限のコマンド

最低限覚えておくべき (だと私が考える) コマンドを挙げる.

コマンド	操作
`ps`	コンテナ一覧を参照する
`run`	イメージを取得し，コンテナを作成し，起動する
`stop`	コンテナを停止する
`rm`	コンテナを削除する
`image`	イメージに対する操作をする

`docker ps` コマンド

コンテナの一覧を表示する.

```
C:\Users\stare\Documents\GitHub\TIL>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

```
C:\Users\stare\Documents\GitHub\TIL>docker ps -a
CONTAINER ID   IMAGE                    COMMAND            CREATED       STATUS                          PORTS     NAMES
029be545abe1   til_devcontainer_python   "sleep infinity"   6 hours ago   Exited (137) 15 minutes ago               til_devcontainer_python_1
```

`-a` を付けると，停止しているコンテナも表示する.

`docker run` コマンド

```
docker run [オプション] イメージ シェル(e.g. /bin/bash)
```

- イメージを取得し (`pull`)，コンテナを作成し (`create`)，起動する (`start`).
  - これら3つのコマンドをまとめたもの.

`docker stop` コマンド

```
docker stop コンテナ名
# コンテナIDでも良い
```

よく使うオプション:

オプション	説明
`--name コンテナ名`	コンテナ名をつける
`-e 環境変数名=値`	コンテナに渡す環境変数を設定する
`-p ポート番号(ホスト):ポート番号(コンテナ)`	ポート番号をマッピングする
`-v ディレクトリ(ホスト):ディレクトリ(コンテナ)`	コンテナの特定のディレクトリにホストのディレクトリをマウントする
`-dit`	バックグラウンドで動かす
`--rm`	コンテナの終了時，自動的に削除する

`docker rm` コマンド

```
docker rm コンテナ名
# コンテナID でも良い
```

- 停止しているコンテナを削除する.

`docker image` コマンド

```
docker image 操作

操作の例:
ls: ホストOSにあるイメージファイルの一覧を表示する
rm: イメージファイルを削除する
```

R 環境

rocker/rstudio を使う.

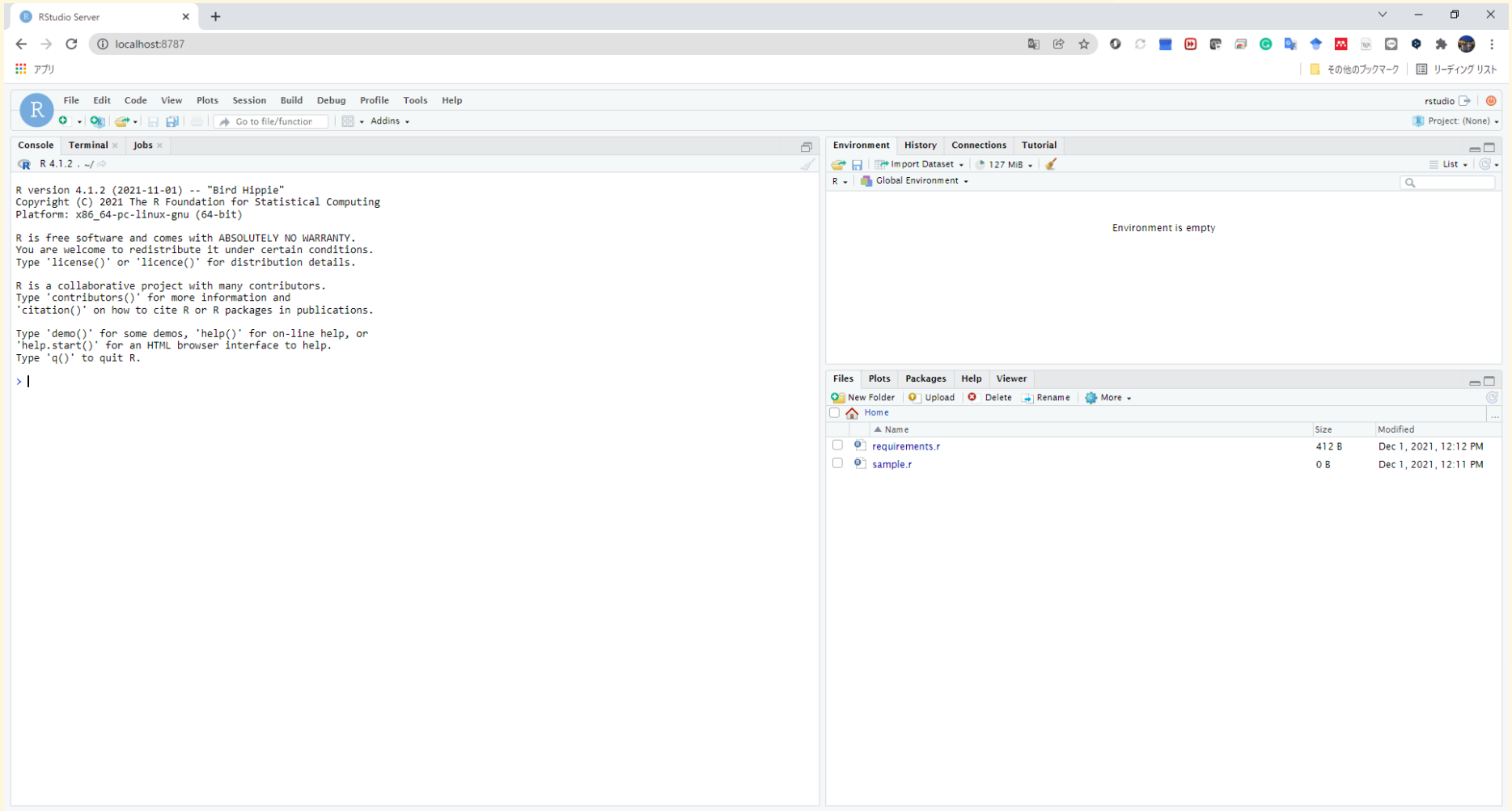
`mountdir` をマウントしてコンテナに入るには，以下のコマンドを実行する.

```
docker run --rm -d -p 8787:8787 -e PASSWORD=[yourpasswordhere] -v [mountdir]:/home/rstudio rocker/tidyverse

--rm: コンテナ停止後に削除する.
```

そして，http://localhost:8787 にアクセス．ユーザ名に rstudio, パスワードに自分が入力したパスワードを入れる.

ブラウザ上で R を実行できる.



Remark

- カレントディレクトリの指定について
  - Powershell の場合 `${pwd}`
  - コマンドプロンプトの場合 `%cd%`
  - Linux の場合 `$(pwd)`
- 終了時
  - `docker ps` コマンドで起動しているコンテナを調べる.
  - 該当するコンテナを `docker stop [container name]` で停止.
    - `--rm` オプションを付けているので，コンテナを停止したら削除される.

Python 環境

- カレントディレクトリをコンテナにマウントして，コンテナ内でカレントディレクトリ内のプログラムを実行したい.
- 開発は手元のエディタでしたい.

→ VSCode の Remote Development を使うと楽！

```
|- ./devcontainer/              <- カレントディレクトリにこのディレクトリを作成
|    |- dockerfile              <- コンテナの定義
|    |- docker-compose.yml      <- docker-compose でコンテナのビルドや起動を容易に
|    |- devcontainer.json       <- VSCode 用のコンテナの設定ファイル
|    |- requirements.txt        <- pip install するパッケージ
```

dockerfile

まず，引っ張ってくる Docker Image を指定する．
例: Python

```
from python:3.8-buster  # from image:tag という形
```

処理を書いていく．

```
RUN apt-get update && \                 # RUN 以下にコンテナ内で実行したいコマンドを書く
    apt-get -y upgrade && \             # && \ で改行するのが一般的らしい
    apt-get install -y vim git && \     # とりあえず vim と git を入れておく
    rm -rf /var/lib/apt/lists*          # キャッシュを消す
```

```
ARG USERNAME=user    # ARG は構築時にユーザが渡せる変数
ARG GROUPNAME=user
ARG UID=1000
ARG GID=1000
RUN groupadd -g ${GID} ${GROUPNAME} && \     # グループとユーザを追加する
    useradd -m -s /bin/bash -u ${UID} -g ${GID} ${USERNAME}
USER ${USERNAME}     # ユーザを user にする
```

- デフォルトでは super user になっている.
- 新しく一般ユーザを作ることで，コンテナ内で作成されたファイルをホストOSの一般ユーザでも扱えるようにする.

```
RUN mkdir /home/${USERNAME}/code
WORKDIR /home/${USERNAME}/code                    # workdir の設定
ADD ./requirements.txt /home/${USERNAME}/code
# dockerfile と同階層にある requirements.txt を workdir に追加
```

```
RUN pip3 install -r requirements.txt     # python image を用いているので, pip3 を使える
ADD . /home/${USERNAME}/code/            # dockerfile と同階層のものを workdir に追加
```

この dockerfile によって，以下が実現される.

- Python 3.8 系の環境構築
- コンテナ内で一般ユーザとしてコマンドを実行すること

docker-compose.yml

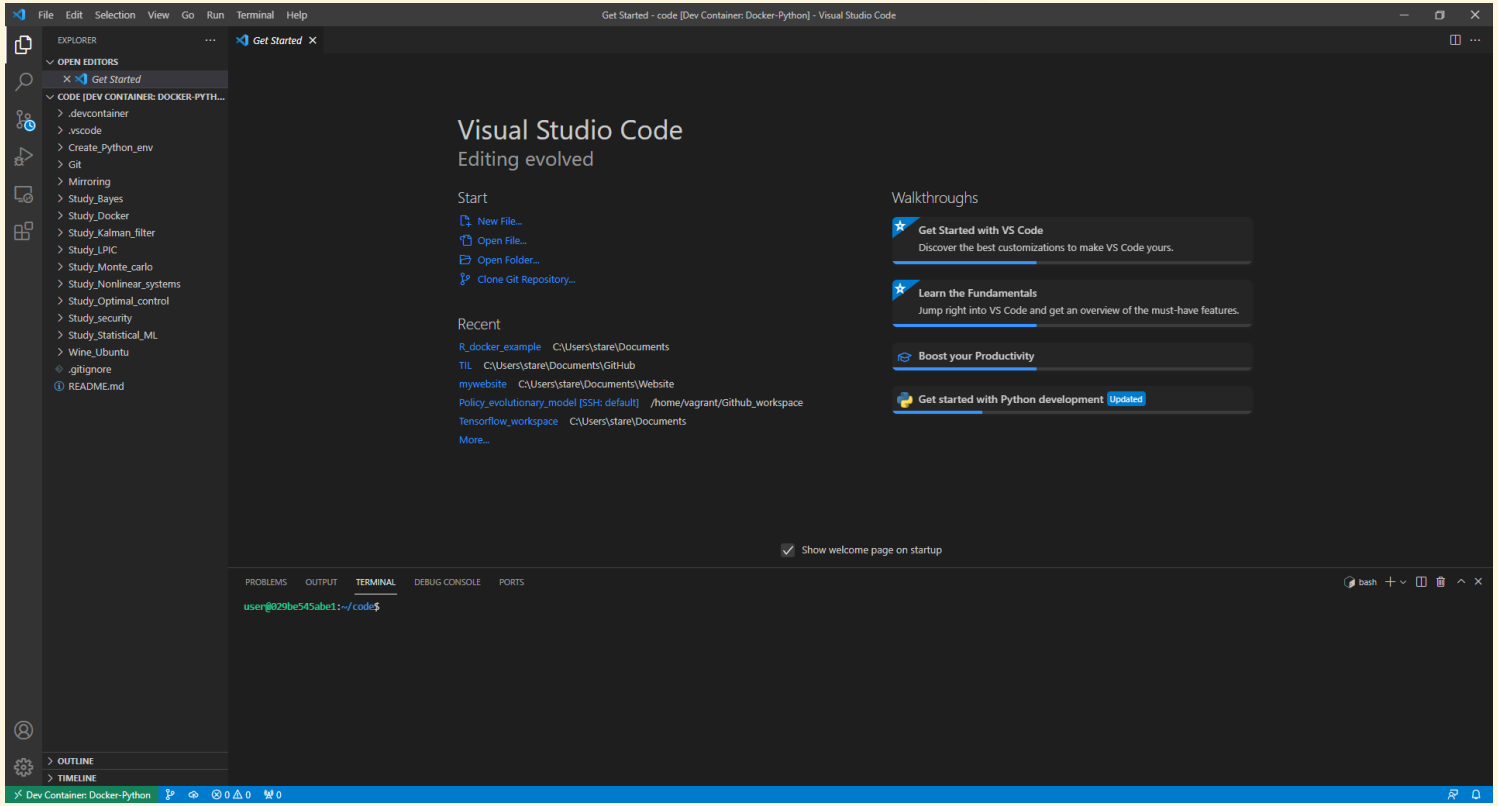
```yaml
version: "3"
services:
    python: # ここの名前とdevcontainer.jsonの"service"を一致させる
        build: .    # 同階層のdockerfileからビルドする
        command: sleep infinity
        volumes:
            - ../:/home/user/code  # 上階層のディレクトリをDocker Container上のworkdirにマウント
        environment:
            SHELL: /bin/bash
```

dockerfile の親ディレクトリ (つまりカレントディレクトリ) を，コンテナ内の `/home/user/code` にマウントすることで，カレントディレクトリ内のファイルをコンテナ内で扱うことができる！

devcontainer.json

```
{
        // 名前は任意
        "name": "Docker-Python",
        // dockercomposefileの場所 (同階層に置いている)
        "dockerComposeFile": "docker-compose.yml",
        // 使う拡張機能
        "extensions": [
                "ms-python.python"
        ],
        // ここに記載している"service"名とdocker-compose.ymlに記載している"service"を一致させる
        "service": "python",
        // コンテナ内に入ったときのworkdir
        "workspaceFolder": "/home/user/code",
        // VSCodeを閉じたときのアクション
        "shutdownAction": "stopCompose"
}
```

VSCode の画面の左下 (Open a remote window) をクリック
→ Reopen in Container をクリック
→ エラーがなければコンテナ内に入ることができる



requirements.txt

`pip install` するパッケージを記述したファイル．例えば...

```
pylint
numpy
scipy
sympy
matplotlib
statsmodels
sklearn
pandas
networkx
ipykernel
jupyter
```

補足 (GPU 環境で pytorch 等を使う場合, Ubuntu)

ここでは，Ubuntu 20 に Docker を入れ，コンテナで GPU を認識させる方法を簡単にまとめる.

1. NVIDIA Driver のインストール
2. Docker Engine のインストール
3. Nvidia Container Toolkit の設定
4. dockerfile の作成
5. docker-compose.yml の作成

NVIDIA Driver のインストール

たぶんここが一番ハマる箇所だと思う (やった記録を残すの忘れた).

参考:

- Ubuntu 20.04セットアップ
- ubuntuにCUDA、nvidiaドライバをインストールするメモ

(1) 現状入っている CUDA, NVIDIA Driver の確認:

```
dpkg -l | grep nvidia
dpkg -l | grep cuda
```

(2) 現状入っている CUDA, NVIDIA Driver の削除:

```
sudo apt-get --purge remove nvidia-*
sudo apt-get --purge remove cuda-*
```

(3) NVIDIA Driver のインストール:

```
ubuntu-drivers devices  # 推奨ドライバの確認
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt update
sudo apt install nvidia-driver-470  # e.g. 460をインストールする場合
sudo reboot
```

(4) 確認: `nvidia-smi` コマンドを実行する.

```
tcbn@tcbn-V530-15ICR:~$ nvidia-smi
Wed Dec  1 18:33:54 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.00    Driver Version: 470.82.00    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0  On |                  N/A |
| 30%   43C    P0    N/A /  75W |    319MiB /  4031MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       985      G   /usr/lib/xorg/Xorg                 35MiB |
|    0   N/A  N/A      3792      G   /usr/lib/xorg/Xorg                122MiB |
|    0   N/A  N/A      3931      G   /usr/bin/gnome-shell               32MiB |
|    0   N/A  N/A      6247      G   ...AAAAAAAAA= --shared-files       39MiB |
|    0   N/A  N/A      6942      G   ...AAAAAAAAA= --shared-files       78MiB |
+-----------------------------------------------------------------------------+
```

Docker Engine のインストール

以下のシェルスクリプトを作成した．これを実行する．

```
#!/bin/bash
# Require password
printf "password: "
read -s password

# Update and Upgrade
echo "$password" | sudo -S apt update && sudo -S apt -y upgrade

# Install Docker
## Delete old version (if exists)
sudo -S apt -y remove docker docker-engine docker.io containerd docker-ce docker-ce-cli
sudo -S apt -y autoremove
```

```
## Install required software
sudo -S apt update
sudo -S apt -y install apt-transport-https ca-certificates curl software-properties-common
sudo -S apt -y install linux-image-generic
## Set docker repository
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
apt-key fingerprint 0EBFCD88
sudo -S add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -sc) \
    stable"
sudo -S apt update
## Install docker.io
sudo -S apt -y install docker.io containerd docker-compose
## Add authority
sudo -S usermod -aG docker $USER
## Set autostart
sudo -S systemctl unmask docker.service
sudo -S systemctl enable docker
sudo -S systemctl is-enabled docker
## let user ubuntu use docker
sudo gpasswd -a $USER docker
```

Nvidia Container Toolkit の設定

参考: [Setting up NVIDIA Container Toolkit]

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
   && 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
sudo apt-get install -y nvidia-docker2
sudo systemctl restart docker
```

リポジトリと GPG キーの設定をして，nvidia-docker2 をインストールする.

動くかどうかテストする.

```
sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

実行して次ページのような出力が得られれば問題なく動く.

```
Wed Dec  1 09:34:41 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.00    Driver Version: 470.82.00    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0  On |                  N/A |
| 30%   42C    P0    N/A /  75W |    316MiB /  4031MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

dockerfile の作成

例えば，イメージファイルを，

```
FROM nvidia/cuda:11.0.3-devel-ubuntu20.04
```

にする．あとは 22 - 24 ページと同じ．

これでビルドして，GPU を使えるか確認する．

```
user@5ee05258584f:~/code$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0
```

```
user@5ee05258584f:~/code$ python3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.cuda.is_available())
True
```

docker-compose.yml の作成

```yaml
version: "3"
services:
    python: # ここの名前とdevcontainer.jsonの"service"を一致させる
        build: .    # 同階層のdockerfileからビルドする
        command: >
            sh -c "nvidia-smi && sleep infinity"
        volumes:
            - ../:/home/user/code  # 上階層のディレクトリをDocker Container上のworkdirにマウント
        environment:
            SHELL: /bin/bash
        deploy:
            resources:
                reservations:
                    devices:
                        - driver: nvidia
                          capabilities: [gpu]
```