

1. VS Code で Docker にリモート接続する

1.1. 拡張機能を追加

以下の拡張機能を VS Code に追加する.

- Docker
- REmote-Containers

1.2. デフォルトの Dockerfile を作成



次の手順を踏むことによって、ワーキングディレクトリ上に、Docker コンテナを自動的に構築するためのディレクトリが作成される。

1. `><` ボタンを押す。
2. 表示された選択肢から、`Remote-Containers: Reopne in Container` を選択する。
3. ワーキングディレクトリ上に `.devcontainer` ディレクトリが作成される。

このディレクトリは、次のような構造を持つ。

```
.
|-- ./devcontainer
    |-- .devcontainer
        |-- devcontainer.json
    |-- library-scripts
        |-- common-debian.sh
        |-- docker-debian.sh
    |-- devcontainer.json
    |-- docker-compose.yml
    |-- Dockerfile
```

`.devcontainer` を含むワーキングディレクトリ上で、先ほど、同じ手順を踏むことで自動的に Docker コンテナを立ち上げ、接続することができる。

2. Dockerfile を修正していく

本 Dockerfile は、様々なサイトを参考に作成しているが、そこで使用されているコマンドのうち、わからなかったものについては随時調べて載せていくこととする。

まず、デフォルトで作成された Dockerfile を載せる

```
# Note: You can use any Debian/Ubuntu based image you want.
FROM mcr.microsoft.com/vscode/devcontainers/base:0-buster

# [Option] Install zsh
ARG INSTALL_ZSH="true"
# [Option] Upgrade OS packages to their latest versions
ARG UPGRADE_PACKAGES="false"
# [Option] Enable non-root Docker access in container
ARG ENABLE_NONROOT_DOCKER="true"
# [Option] Use the OSS Moby CLI instead of the licensed Docker CLI
ARG USE_MOBY="true"

# Install needed packages and setup non-root user. Use a separate RUN statement to add your
# own dependencies. A user of "automatic" attempts to reuse an user ID if one already exists.
ARG USERNAME=automatic
ARG USER_UID=1000
ARG USER_GID=$USER_UID
COPY library-scripts/*.sh /tmp/library-scripts/
RUN apt-get update \
    && /bin/bash /tmp/library-scripts/common-debian.sh "${INSTALL_ZSH}" "${USERNAME}" "${USER_UID}" "${USER_GID}" "${UPGRADE_PACKAGES}" \
    # Use Docker script from script library to set things up
    && /bin/bash /tmp/library-scripts/docker-debian.sh "${ENABLE_NONROOT_DOCKER}" "/var/run/docker-host.sock" "/var/run/docker.sock" "${USERNAME}" \
    # Clean up
    && apt-get autoremove -y && apt-get clean -y && rm -rf /var/lib/apt/lists/* /tmp/library-scripts/

# Setting the ENTRYPOINT to docker-init.sh will configure non-root access
# to the Docker socket. The script will also execute CMD as needed.
ENTRYPOINT [ "/usr/local/share/docker-init.sh" ]
CMD [ "sleep", "infinity" ]

# [Optional] Uncomment this section to install additional OS packages.
# RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
#     && apt-get -y install --no-install-recommends <your-package-list-here>
```

2.1. ubuntu を使用する

まず、構築する docker image を ubuntu に変更する。
そこで、FROM 部分を以下のように変更する。

```
# base image
ARG BASE_IMAGE=ubuntu:latest
FROM ${BASE_IMAGE}
```

2021 年 1 月 25 日追記

2.1.1. \$chown

chown コマンドは、ファイルやディレクトリの所有者を変更するためのコマンドである。

```
$ chown 所有者名 ファイル名またはディレクトリ名
```

2.1.2. \$ wget

`wget` コマンドは、ノンインタラクティブな（1方向のみの）ダウンローダ。

詳細は『【[wget](#)】コマンド——URLを指定してファイルをダウンロードする』を参照。

```
$ wget [オプション] URL
$ wget -r [オプション] URL
```

`wget` の主なオプションについては、『[wget](#)』を参照。

短	長	意味
-b	--background	バックグラウンドで実行する。 経過メッセージは「-o」オプションで指定したログファイルへ、指定がない場合は「wget-log」に出力する。
-o filename	--output- file=filename	経過メッセージを全て、指定したファイルに出力する。
-a filename	--append- output=filename	「-o」オプションのように、経過メッセージを指定ファイルに出力するが、指定したファイルが既にある場合は上書きではなく追加する。

短	長	意味
-d	--debug	動作内容を詳しく出力する.
-q	--quiet	経過のメッセージを出力しない.
-v	--verbose	経過のメッセージを出力する
-i filename	--input-file=filename	指定したファイルからダウンロードする URL を読み込む.
-t 回数	--timers=回数	リトライ回数(デフォルトは 20 回)
-T 秒数	--timeout=秒数	タイムアウトするまでの秒数を指定する. (デフォルトは 900 秒)
-W 秒数	--wait=秒数	リトライの待ち時間を指定する.
-nc	--no-clobber	ファイルを上書きしない.
-c	--continue	途中までダウンロードされていたら続きをダウンロードする.
-O filename	--output-document=filename	複数のテキストデータを一つに連結して, 指定のファイルへ書き込む. 指定の場合は, 標準出力に対して出力する.

次のようにオプションを指定することで、任意の PATH、任意のファイル名で保存することができる。

詳細は次の記事を参照。

[wget コマンドで覚えておきたい使い方 16 個\(+1 個\)](#)

```
$ wget -O 出力先のPATH http://ファイルのURL
```

2.1.3. \$ md5sum

詳細は次の記事を参照。

- [md5sum コマンドについて詳しくまとめました](#)
- [【 md5sum 】 コマンド——ダウンロードファイルを検証する](#)

MD ハッシュ値とは

ハッシュ値とは、ファイルやデータを数学的に圧縮したもので、内容が同じであれば同じ値になる。MD アルゴリズムのハッシュ値は 32 個の 16 進数(128bit) で構成される文字列であり、ファイルの偽造防止などに使われる。また、ファイル名が異なる場合でも、内容が同じであればハッシュ値は同じになる。

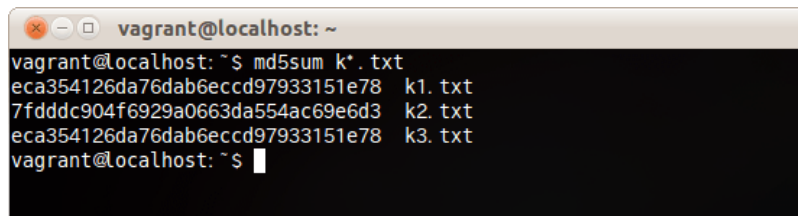
コマンドの基本操作

```
$ md5sum [オプション] 対象ファイル名 (算出時)  
$ md5sum -c ファイル名 [オプション] (照合時)
```

複数のファイルを指定する場合は、空白で列挙数 r こともできる。
さらに、ファイル名にワイルドカード `*` を指定することもできる。

例えば、`koo.txt` というファイル名全ての MD5 ハッシュ値を計算するには、

```
$ md5sum k*.txt
```



```
vagrant@localhost: ~  
vagrant@localhost: ~$ md5sum k*.txt  
eca354126da76dab6eccd97933151e78  k1.txt  
7fdddc904f6929a0663da554ac69e6d3  k2.txt  
eca354126da76dab6eccd97933151e78  k3.txt  
vagrant@localhost: ~$
```

オプション

短	長	意味
-b	--binary	ファイルをバイナリファイルとして扱う.
-t	--text	ファイルをテキストファイルとして扱う(デフォルト).
-c file	--check file	MD5 ハッシュ値とファイルが一致しているかをチェックする.
	--strict	無効な設定業がある場合には 0 以外で終了する.
	--quiet	一致しない場合のみ表示する.
	--status	何も出力しない(終了コードで結果を判定する).
-w	--warn	ファイルの書式が不正な場合に警告する.

2.1.4. \$ echo

`echo` は画面に文字列や数値, 変数を表示する Linux コマンド.
以下のように使用する.

```
$ echo 表示するもの
```

詳細は次の記事を参照.

[echo コマンドの詳細まとめました](#)

出力をファイルではなく画面に出力する場合

`echo` は画面ではなく, ファイルに出力することもできる.
これは, **リダイレクト**と呼ばれる Linux シェルの機能であり, 画面に表示されたものを別のところに挿入できる.

`echo` コマンドで新規作成したファイルに出力する

`echo` コマンドで文字列をファイルに出力する場合は, `>` で出力するファイル名を指定する.

```
$ echo 文字列 > ファイル名
```

2.1.5. miniconda のサイレントモードでのインストール

Miniconda を bash を用いてインストールする場合、次のオプションを指定することでサイレントインストールすることができる。

オプション	意味
-b	PATH を変更しないバッチモード。 使用許諾契約に同意することを前提としており、 <code>.bashrc</code> または <code>.bash_profile</code> を編集しない。
-p	インストールプレフィックスパス
-f	プレフィックスが既に存在する場合でも、強制的にインストール。

2.2. Reopen Container

既存の Dockerfile の一部を以下のように変更した。

```
# Note: You can use any Debian/Ubuntu based image you want.
ARG BASE_IMAGE=ubuntu:latest
FROM ${BASE_IMAGE}

# [Option] Install zsh
ARG INSTALL_ZSH="true"
# [Option] Upgrade OS packages to their latest versions
ARG UPGRADE_PACKAGES="false"
# [Option] Enable non-root Docker access in container
ARG ENABLE_NONROOT_DOCKER="true"
# [Option] Use the OSS Moby CLI instead of the licensed Docker CLI
ARG USE_MOBY="true"

# Install needed packages and setup non-root user. Use a separate RUN statement to add your
# own dependencies. A user of "automatic" attempts to reuse an user ID if one already exists.
ARG USER_NAME=automatic
ARG USER_UID=1000
ARG USER_GID=$USER_UID
COPY library-scripts/*.sh /tmp/library-scripts/
RUN apt-get update \
    && /bin/bash /tmp/library-scripts/common-debian.sh "${INSTALL_ZSH}" "${USER_NAME}" "${USER_UID}" "${USER_GID}" "${UPGRADE_PACKAGES}" \
    # Use Docker script from script library to set things up
    && /bin/bash /tmp/library-scripts/docker-debian.sh "${ENABLE_NONROOT_DOCKER}" "/var/run/docker-host.sock" "/var/run/docker.sock" "${USER_NAME}" \
    # Clean up
    && apt-get autoremove -y && apt-get clean -y && rm -rf /var/lib/apt/lists/* /tmp/library-scripts/

# Setting the ENTRYPOINT to docker-init.sh will configure non-root access to the Docker socket. The script will also execute CMD as needed.
ENTRYPOINT [ "/usr/local/share/docker-init.sh" ]
#CMD [ "sleep", "infinity" ]

# [Optional] Uncomment this section to install additional OS packages.
# RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
#     && apt-get -y install --no-install-recommends <your-package-list-here>

ENV CONDA_DIR = /opt/conda
ENV CONDA_TMP_DIR = /tmp/conda

#RUN mkdir -p ${CONDA_DIR} && \
#    mkdir -p ${CONDA_TMP_DIR}
# ファイルの所有者を変更
#RUN chown ${USER_NAME}:${USER_UID} ${CONDA_DIR} && \
#    chown ${USER_NAME}:${USER_UID} ${CONDA_TMP_DIR}

# miniconda
ARG CONDA_VERSION=py38_4.9.2
ARG CONDA_MD5=122c8c9beb51e124ab32a0fa6426c656

ENV PATH=${CONDA_DIR}/bin:$PATH

# tmpにディレクトリを移動
RUN cd /tmp && \
    # miniconad.shをダウンロード
    wget --quiet https://repo.anaconda.com/miniconda/Miniconda3-${CONDA_VERSION}-Linux-x86_64.sh -O miniconda.sh && \
    # MD5 ハッシュ値の確認
    echo "${CONDA_MD5} miniconda.sh" > miniconda.md5 && \
    if ! md5sum --status -c miniconda.md5; then exit 1; fi && \
    # miniconda のインストール
    /bin/bash miniconda.sh -b -p /tmp/conda

#RUN rm miniconda.sh miniconda.md5
RUN echo ". /opt/conda/etc/profile.d/conda.sh" >> ~/.bashrc && \
    echo "conda activate base" >> ~/.bashrc && \
    find /tmp/conda -follow -type f -name '*.a' -delete && \
    find /tmp/conda -follow -type f -name '*.js.map' -delete && \
    /tmp/conda/bin/conda clean -afy
```

上記の `Dockerfile` は、以下のサイトを参考にした。
[yaml で保存してある conda 仮想環境を Docker 上で構築](#)

2.2.1. 起動時の Bash について

前述の `Dockerfile` でコンテナを構築すると、以下のように立ち上げ時に SHELL が立ち上げ時の log を表示したままとなるため、コマンドを入力することができない。

```
[370709 ms] [01:38:46] Installation completed. vscode.yaml
[370709 ms] [01:38:46] Extensions installed successfully: vscode.yaml
[370709 ms] [01:38:46] Extensions installed successfully: ms-azuretools.vscode-docker
[370709 ms] Extension 'ms-azuretools.vscode-docker' v1.9.1 was successfully installed.
[370919 ms] Start; Run: docker exec -i -u vscode -e VSCODE_REMOTE_CONTAINERS_SESSION=9
4f0d096-8f53-44cd-8ea5-2ae6481f93e41611624755134 c272f109f0837939ffdad08e41328797cce47
9ac2e55e2ff7a9a03691ccddd40 /home/vscode/.vscode-server/bin/ea3859d4ba2f3e577a159bc91e
3074c5d85c0523/node -e
```

これは、`start up docker` コマンドの `log` ファイルを開かないようにすることで解決した。
しかし、`DeBug` にはこの `log` を見ておきたいので、後々にはコマンドラインを切り替えられるようにしたい。

2.2.2. conda コマンドを実行できない問題について

PATH の変更

しか、この `DockerImage` では `conda` コマンドを実行することができなかった。

```
vscode → /workspace $ conda
bash: conda: command not found
vscode → /workspace $ |
```

この原因は、最後の `RUN` の PATH の一部が間違っていたためと考えられる。そこで、PATH を正しいものに変更した。

```
RUN echo ". /tmp/conda/etc/profile.d/conda.sh" >> ~/.bashrc && \
echo "conda activate base" >> ~/.bashrc && \
find /tmp/conda -follow -type f -name '*.a' -delete && \
find /tmp/conda -follow -type f -name '*.js.map' -delete && \
/tmp/conda/bin/conda clean -afy
```

しかし、この対処ではコンテナを立ち上げた際に、`conda` コマンドを実行することができなかった。

3. 結論

本日も、自力で `miniconda` を搭載した `Docker container` を作成使用と試みたが、構築することはできなかった。