

## 1章 クイックスタート

### 1.1. 手順の概要

必要な機材を確認します。必要環境は WindowsPC のみです。

オペレーティングシステム	Windows10 64bit (Home/Pro)
対応プロセッサ	仮想化支援機能「Intel VT/AMD-V」対応プロセッサ
メモリ	4GB 以上
HDD 空き	100GB 以上
Version	2004 以降
その他	インターネット接続環境

まず、仮想化支援機能の有効化を確認します。

続いて、必要なツールをインストールを行います。

- 1) WSL2 の設定 (Windows Subsystem For Linux Version2)
- 2) Docker Desktop for Windows のダウンロードとインストール
- 3) Visual Studio Code for Windows ダウンロードとインストール  
さらに拡張機能のインストール
- 4) Git for Windows と TortoiseGit のダウンロードとインストール

ここまで行えば、あとは

- 5) ソースコードを github から Clone
- 6) Visual Studio Code でソースを開く
- 7) ビルドと実行

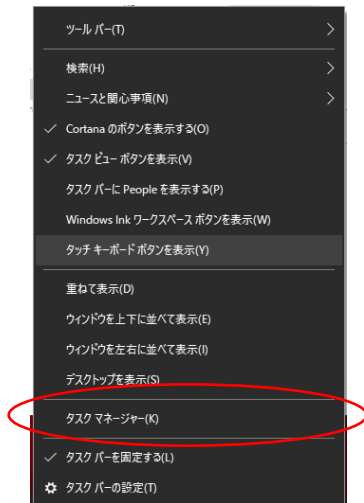
になります。

### 1.2. 仮想化支援機能の有効化

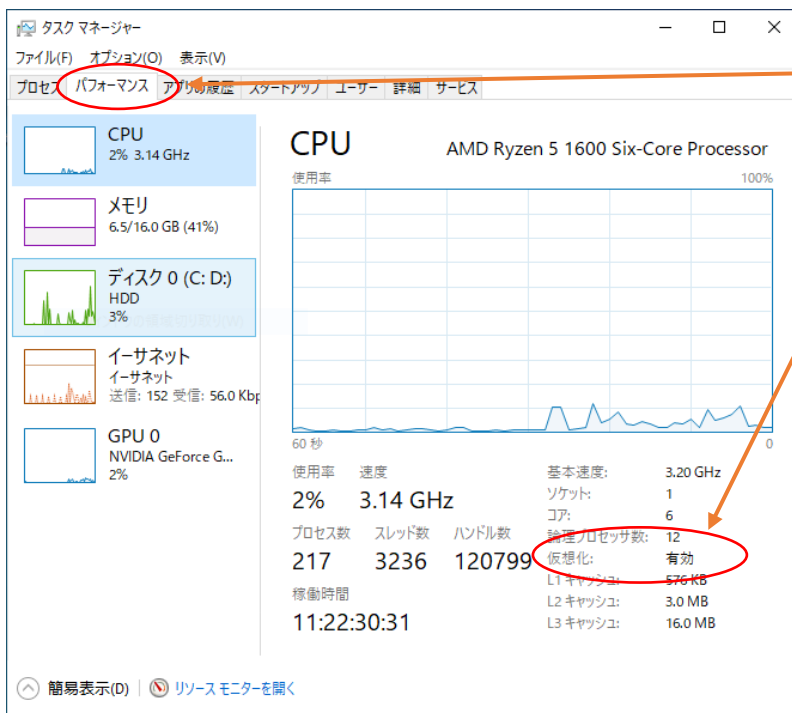
Intel VT AMD-V と呼ばれている機能です。通常は有効化されていますが、以下の方法で確認します。

デスクトップ画面下部にあるタスクバーの何も無いところを右クリックして

メニューを開き、「タスクマネージャー」をクリックして表示します。



タスクマネージャーを起動、「パフォーマンス」タブをクリック。メニューの「CPU」を選び、ボートグラフの下部の項目のうち「仮想化」が「有効」であれば有効化されています。



無効の場合は、有効化します。

有効化は、PC の BIOS 設定により設定します。Windows から操作できるマシン ThinkPadX240 の例を以下に説明します。

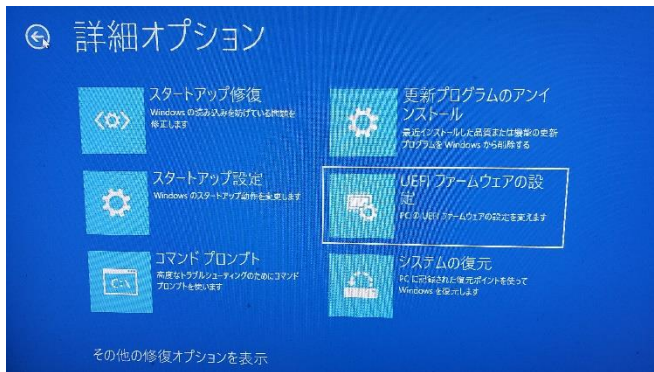
(注意) マシンによっては、「詳細オプション」に「UEFI ファームウェアの設定」が現れない場合があります。その場合は、お使いの PC のマニュアルに従って BIOS にて「Virtualization」を有効化してください。

「スタートメニュー」「設定」「Windows の設定」ウィンドウ「更新とセキュリティ」をクリック左側メニューの「回復」クリック、  
「PC の起動をカスタマイズする」「今すぐ再起動」をクリック。



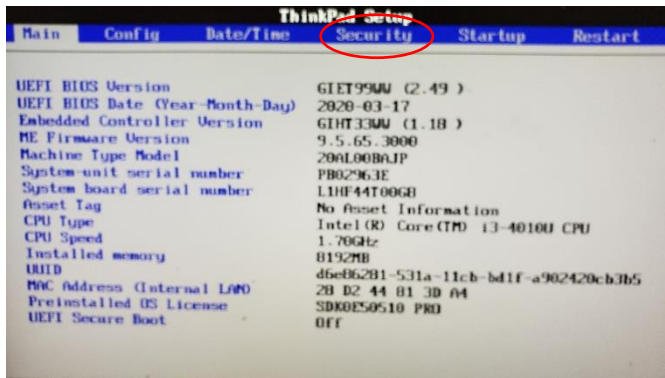
「オプションの選択」画面が表示されるので、「トラブルシューティング」をクリックします。



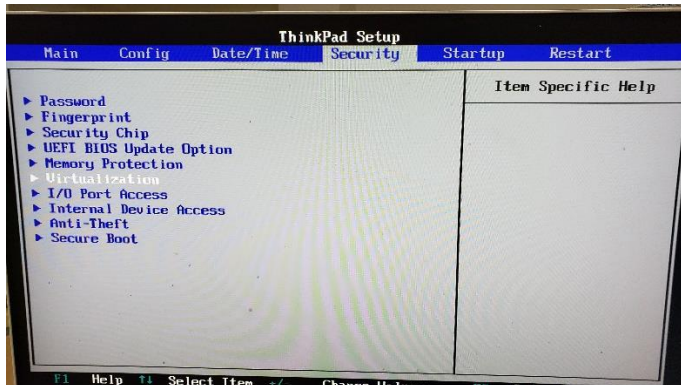


「詳細オプション」の項目一覧が表示されたら、「UEFI ファームウェアの設定」をクリックします。

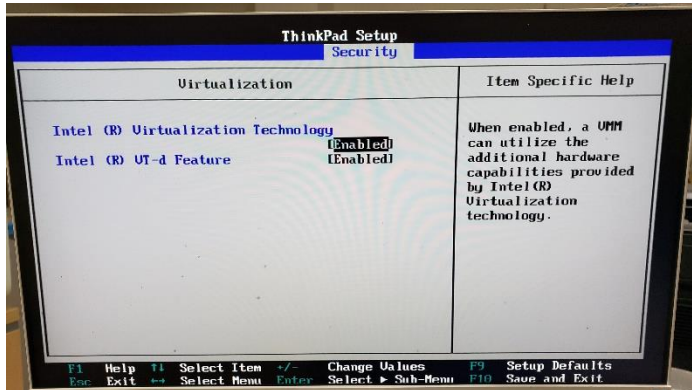
続いて「再起動」をクリックすることでパソコンの再起動が行われます。  
再起動によって PC の BIOS 画面が表示されます。  
以下 ThinkPadX240 BIOS 画面 起動時の画面から



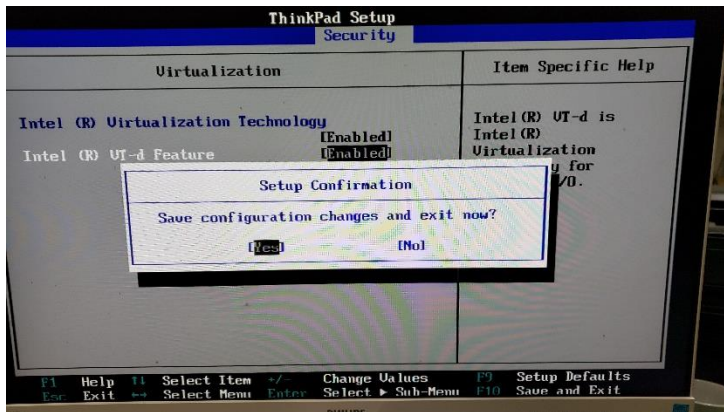
Security 画面に行き、Virtualization を選びます。



いずれも Enable にします。ここではすでに設定済みのものを確認しました。



セーブして再起動します。



以上で、仮想化支援機能が設定されているはずです。最初に確認した「タスクマネージャー」で仮想化が有効になっていることを確認します。

### 1.3.開発環境のセットアップ

#### 1) WSL2 のセットアップ

管理者モードで Windows Powershell を起動し。以下 コマンドを実行します。

```
> wsl --install
```



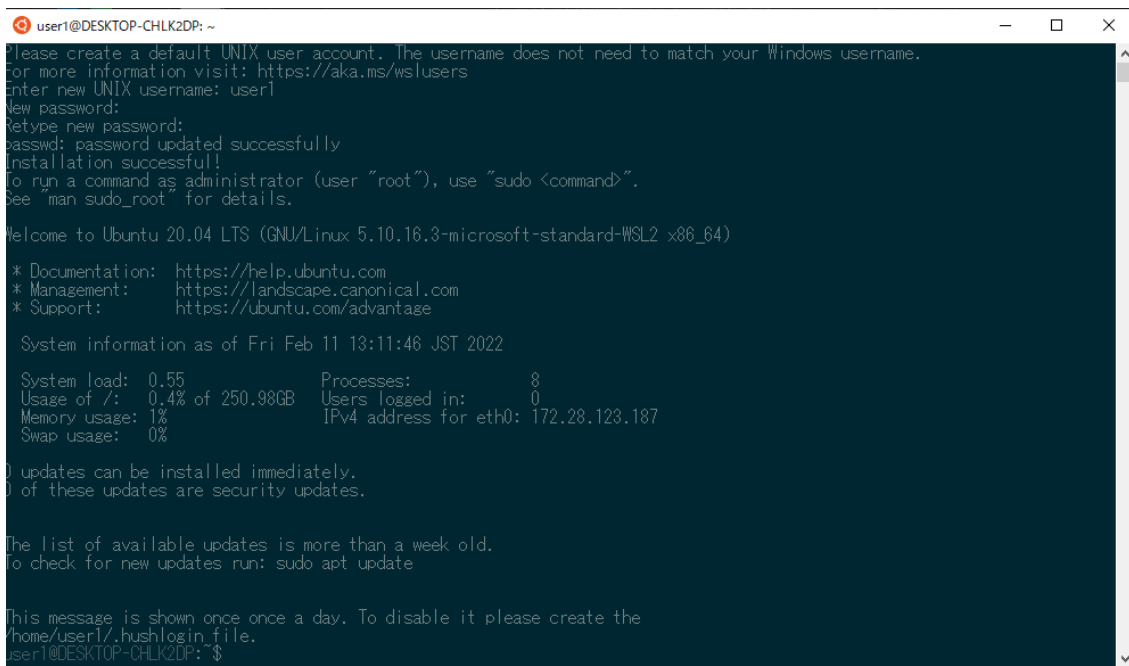
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

新しいクロスプラットフォームの PowerShell をお試しください https://aka.ms/pscore6

PS C:\Users\takahashi> wsl --install
インストール中: 仮想マシン プラットフォーム
仮想マシン プラットフォーム はインストールされました。
インストール中: Linux 用 Windows サブシステム
Linux 用 Windows サブシステム はインストールされました。
ダウンロード中: WSL カーネル
インストール中: WSL カーネル
WSL カーネル はインストールされました。
ダウンロード中: Ubuntu
要求された操作は正常に終了しました。変更を有効にするには、システムを再起動する必要があります。
PS C:\Users\takahashi>
```

再起動が要求されるので、PC を再起動します。

再起動すると自動的に ubuntu のコマンドが実行状態になります。



```
user1@DESKTOP-CHLK2DP: ~
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: user1
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Feb 11 13:11:46 JST 2022

System load:  0.55          Processes:      8
Usage of /:   0.4% of 250.98GB Users logged in: 0
Memory usage: 1%           IPv4 address for eth0: 172.28.123.187
Swap usage:   0%

0 updates can be installed immediately.
0 of these updates are security updates.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

This message is shown once once a day. To disable it please create the
/home/user1/.hushlogin file.
user1@DESKTOP-CHLK2DP: $
```

プロンプトでユーザー名とパスワードが求められるので入力します。

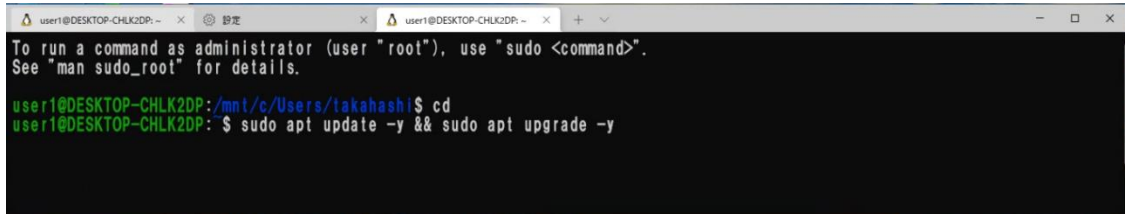
このユーザー名パスワードが WSL ubuntu のユーザー名パスワードなので忘れずに覚えておいてください。例では ユーザー名 user1 で設定しています。

終わると ubuntu のシェルの状態になります。

## 2) WSL のその後の設定

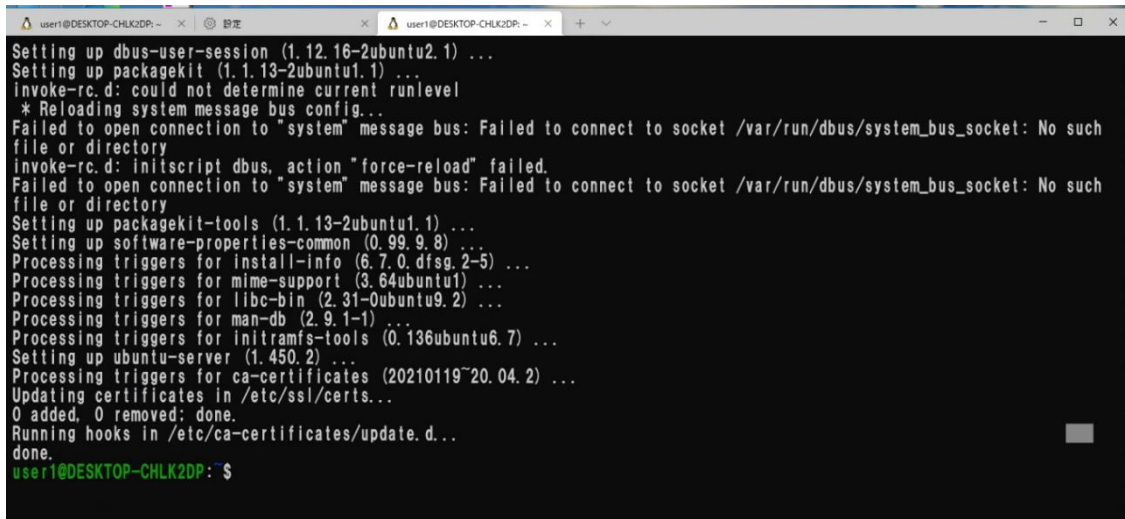
linux のパッケージのアップデートをします。

```
$ sudo apt update -y && sudo apt upgrade -y
```



```
user1@DESKTOP-CHLK2DP:~$ cd
user1@DESKTOP-CHLK2DP:~$ sudo apt update -y && sudo apt upgrade -y
```

パスワードが要求されるので先ほど設定したパスワードを入力します。  
しばらく待ちます。



```
Setting up dbus-user-session (1.12.16-2ubuntu2.1) ...
Setting up packagekit (1.1.13-2ubuntu1.1) ...
invoke-rc.d: could not determine current runlevel
* Reloading system message bus config...
Failed to open connection to "system" message bus: Failed to connect to socket /var/run/dbus/system_bus_socket: No such file or directory
invoke-rc.d: initscript dbus, action "force-reload" failed.
Failed to open connection to "system" message bus: Failed to connect to socket /var/run/dbus/system_bus_socket: No such file or directory
Setting up packagekit-tools (1.1.13-2ubuntu1.1) ...
Setting up software-properties-common (0.99.9.8) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for initramfs-tools (0.136ubuntu6.7) ...
Setting up ubuntu-server (1.450.2) ...
Processing triggers for ca-certificates (20210119~20.04.2) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
user1@DESKTOP-CHLK2DP:~$
```

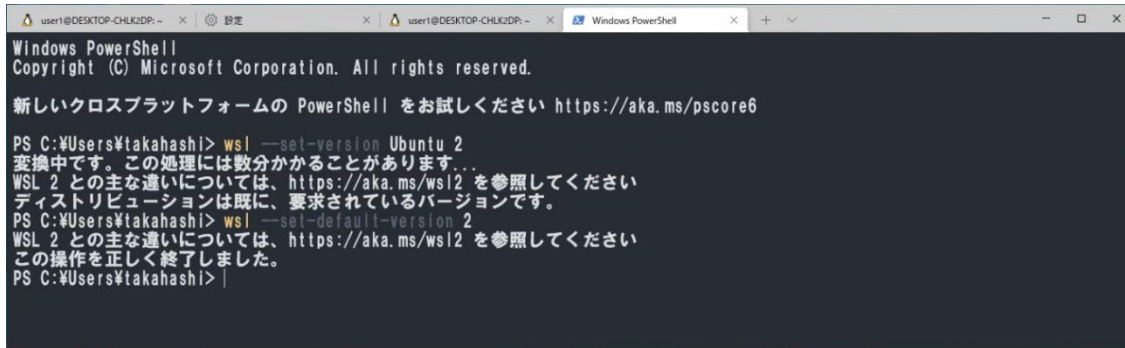
上記のように終了します。



Linux ディストリビューションの WSL のバージョンの規定をセットします。  
また WSL2 を規定のバージョンを 2 に設定します。

```
$wsl --set-version Ubuntu 2
```

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



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

新しいクロスプラットフォームの PowerShell をお試しください https://aka.ms/pscore6

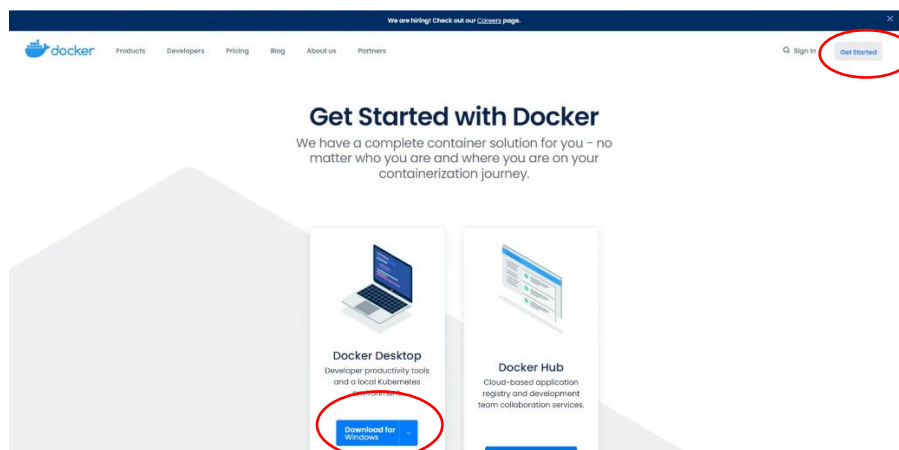
PS C:\Users\takahashi> wsl --set-version Ubuntu 2
変換中です。この処理には数分かかることがあります...
WSL 2 との主な違いについては、https://aka.ms/wsl2 を参照してください
ディストリビューションは既に、要求されているバージョンです。
PS C:\Users\takahashi> wsl --set-default-version 2
WSL 2 との主な違いについては、https://aka.ms/wsl2 を参照してください
この操作を正しく終了しました。
PS C:\Users\takahashi> |
```

### 3) Docker のインストール

以下のサイトから Docker Desktop Windows 64bit 版をダウンロードします。

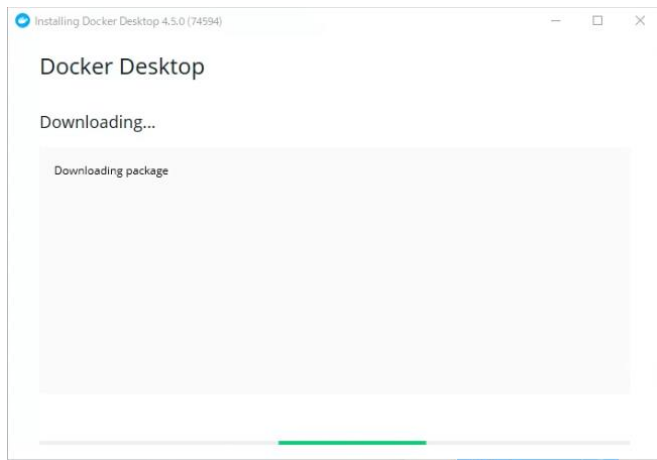
<https://www.docker.com/>

右上の「Get Started」をクリック、その後画面中央の Docker Desktop の「Download for Windows」をクリックしてダウンロードします。



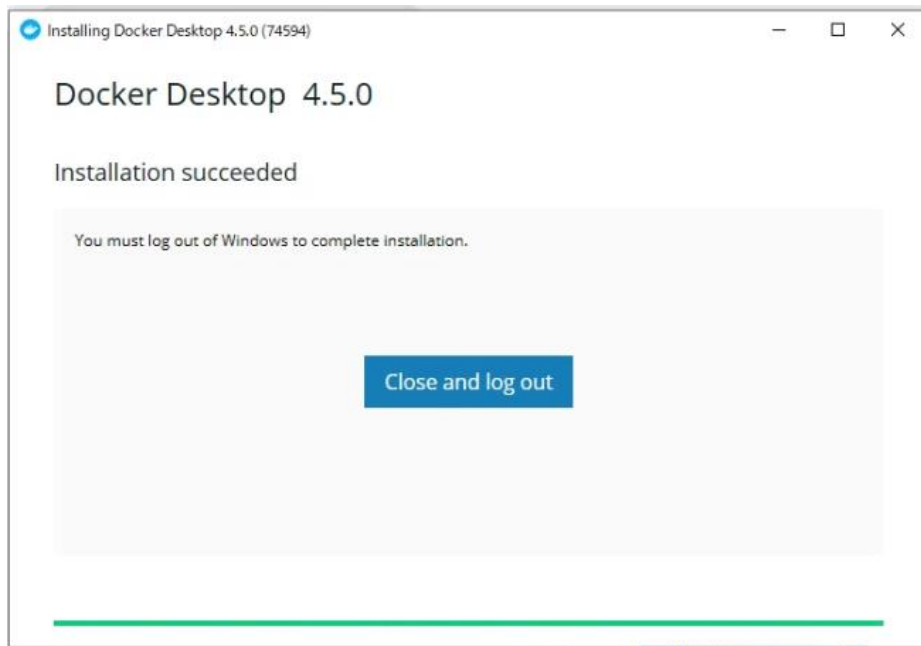


ダウンロードしたインストーラを起動します。



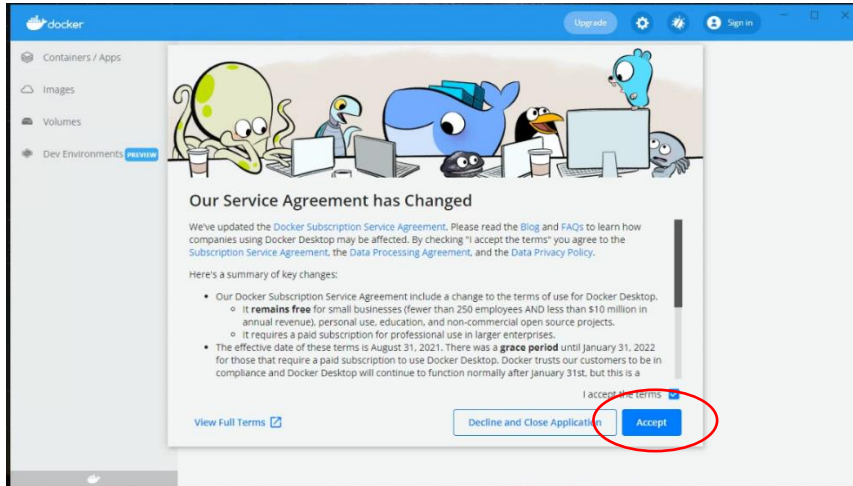
上記の確認画面が表示されます。 両方チェックを確認して、「Ok」をクリックします。

終了すると以下のようになり、「Close and logout」をクリックします。

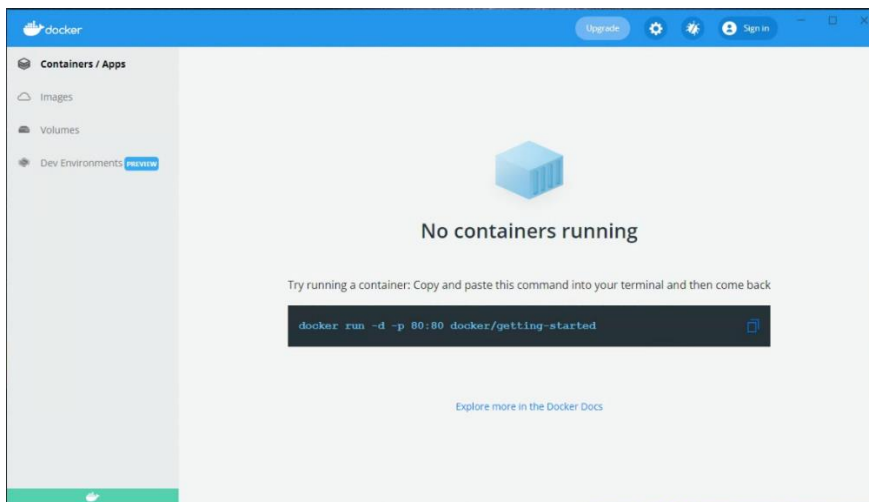


一旦 Windows からログオフになります。

再度 Windows ログオンすると DockerDesktop が立ち上がり、条件に関するアグリーメントがでけますので、確認し「Accept」をクリックします。



しばらくして、以下のように RUN 状態になれば OK です。

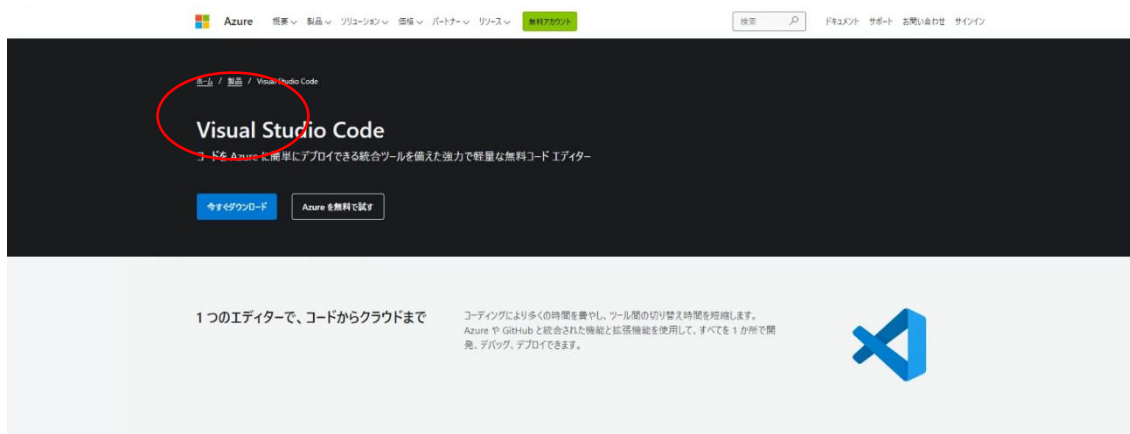


最小化しておきましょう。

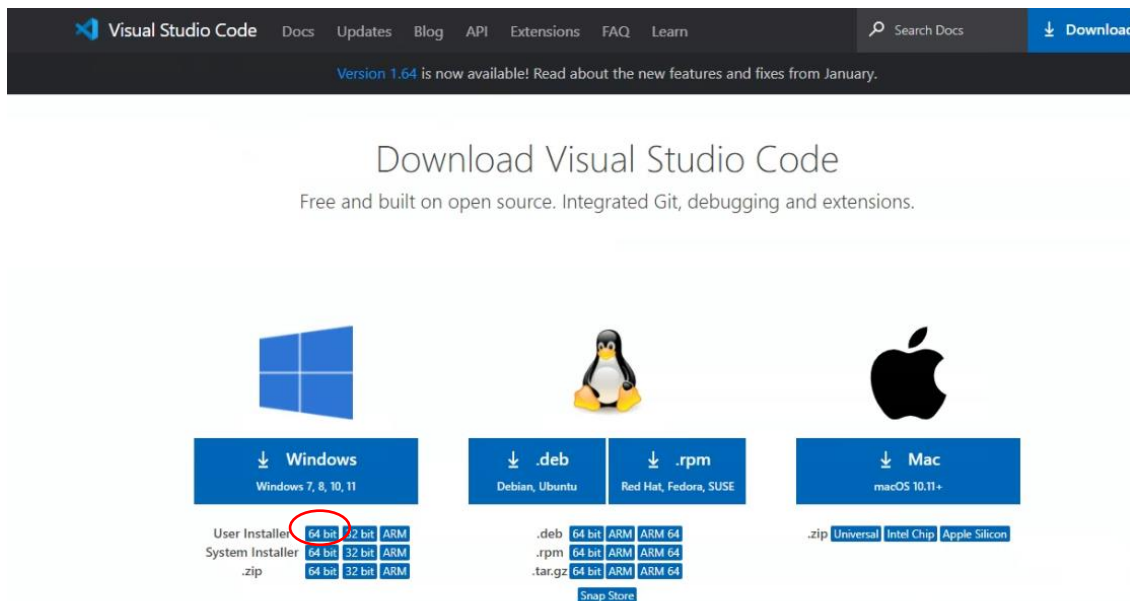
#### 4) Visual Studio Code のインストール

以下のサイトから Visual Studio Code Windows 64bit をダウンロードします。

<https://azure.microsoft.com/ja-jp/products/visual-studio-code/>



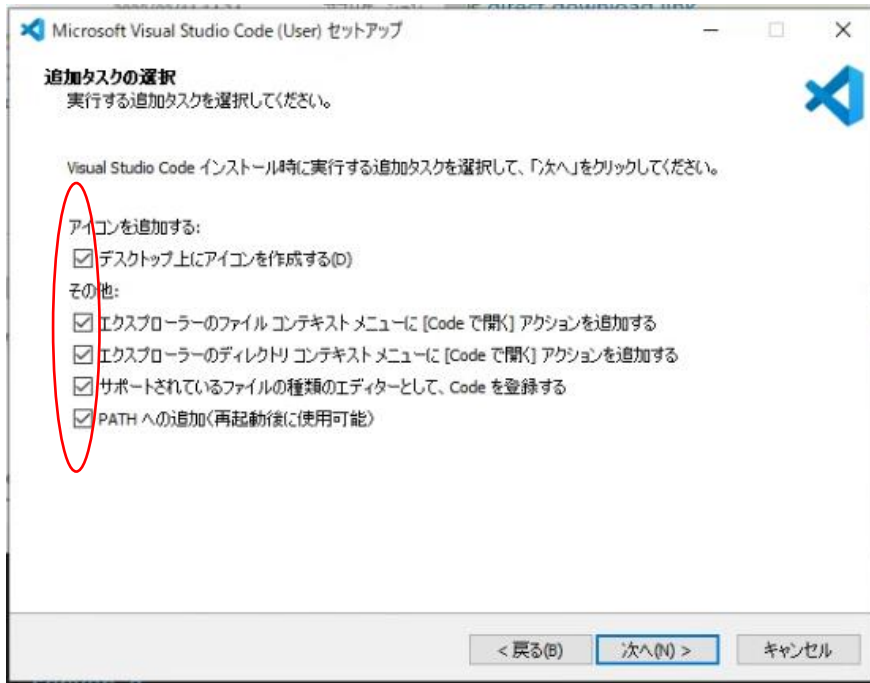
「今すぐダウンロード」をクリックしてダウンロードします。



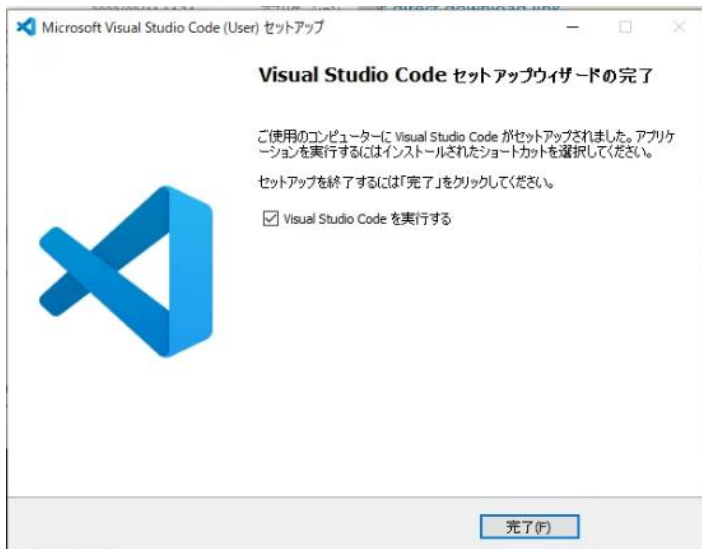
Windows インストーラ 64bit をクリックします。

ダウンロードが終了したら、そのインストーラを起動します。

使用許諾に同意するなど、そのまま進めてインストールします。  
特に必須ではありませんが、「VSCODE で開く」などをエクスプローラから指定できるようにするのがお勧めです。



インストールが終わるとすぐ実行するボタンのダイアログがでます。  
インストールに引き続き、VSCODE の初期設定に続きます。

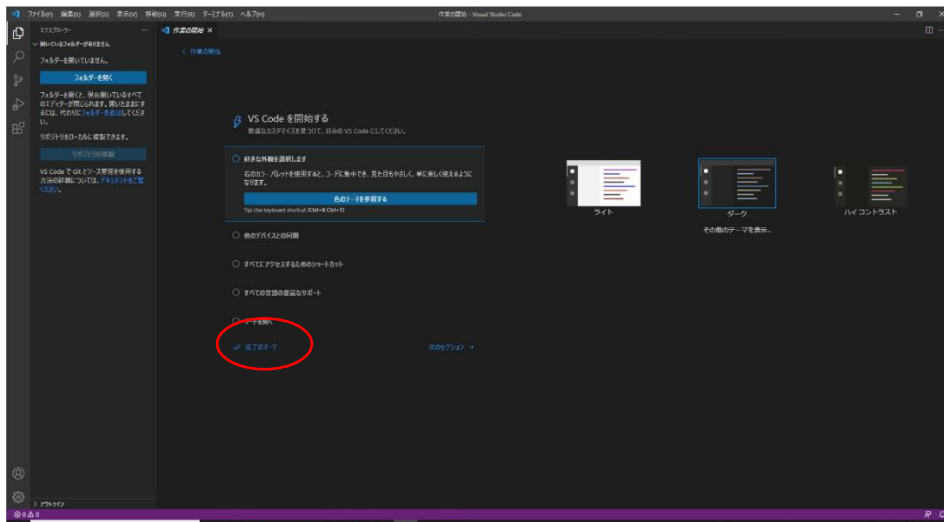


## 5) Visual Studio Code の初期設定

設定と拡張プラグインのインストールになります。

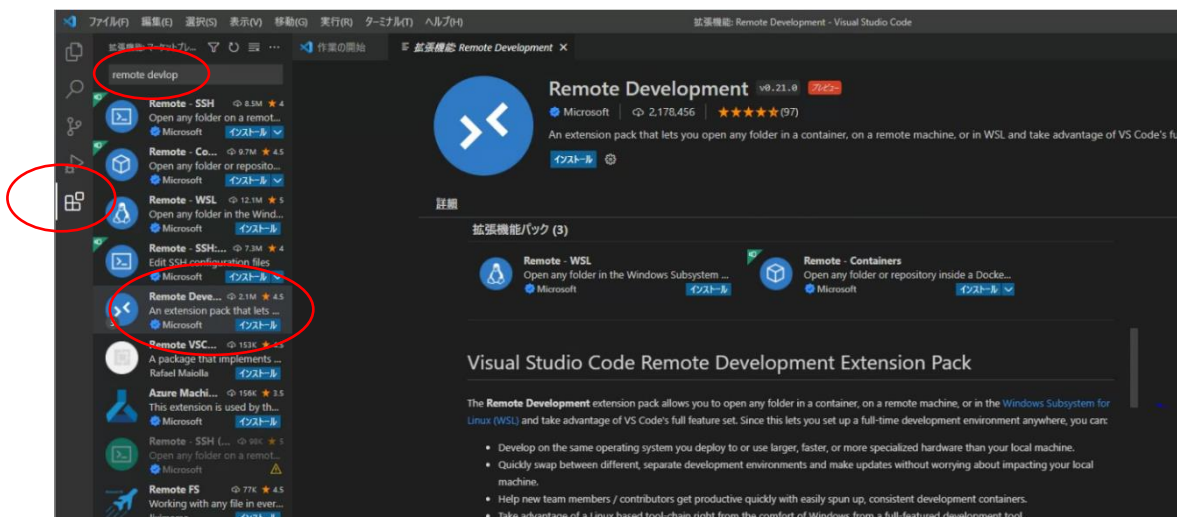
まず、日本語プラグインのインストールが求められるので、インストールします。

設定系のウィザードがでますが、デフォルトのままでよければ 「完了のマーク」をクリックします。



必須の拡張プラグイン Remote Development をインストールします。

拡張ボタンをクリック 、検索「Remote Development」を入れて Remote Development をクリックします。



あとはお好みで拡張プラグインを入れてみます。

お勧めは、

- C/C++ Extension Pack
- Git Extension Pack
- Git Blame

環境設定は以上になります。

#### 1.4. 実行してみます。

##### 1) 何を実行するのか？

PC のシミュレーションで Raspberry-pi2 を動作させます。

シミュレーションは QEMU というエミュレータを利用します。

環境は、DockerHUB にある ubuntu イメージをダウンロードして実行することになります。

実行は Visual studio code で実行されます。実行時に dbg というデバッグインターフェースを使ってソースレベルデバッグ状態にてソースステップ実行を確認できます。

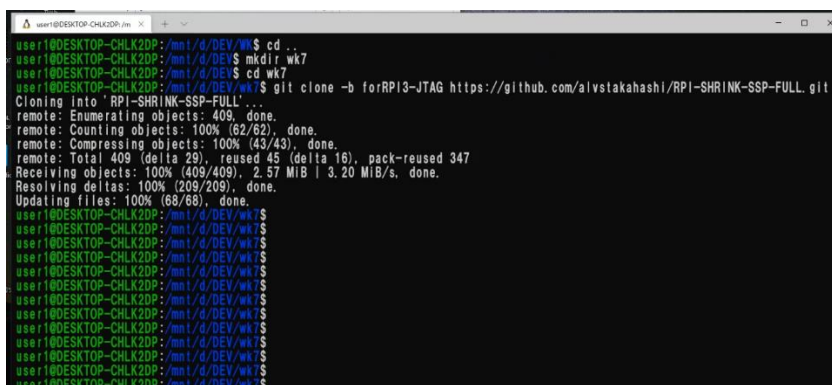
##### 2) 実行の大まかな手順

- ①Github からソースコード一式を Clone します。
  - ②Visual Studio Code から、Clone したソースコードを load します。
  - ③実行手順に従って実行します。
- 実行手順は追って説明します。

##### 3) 実施手順

- ①ubuntu シェルを起動し、以下のコマンドで Github から Clone します。

\$ git clone -b forRPI3-JTAG https://github.com/alvstakahashi/RPI-SHRINK-SSP-FULL.git



```

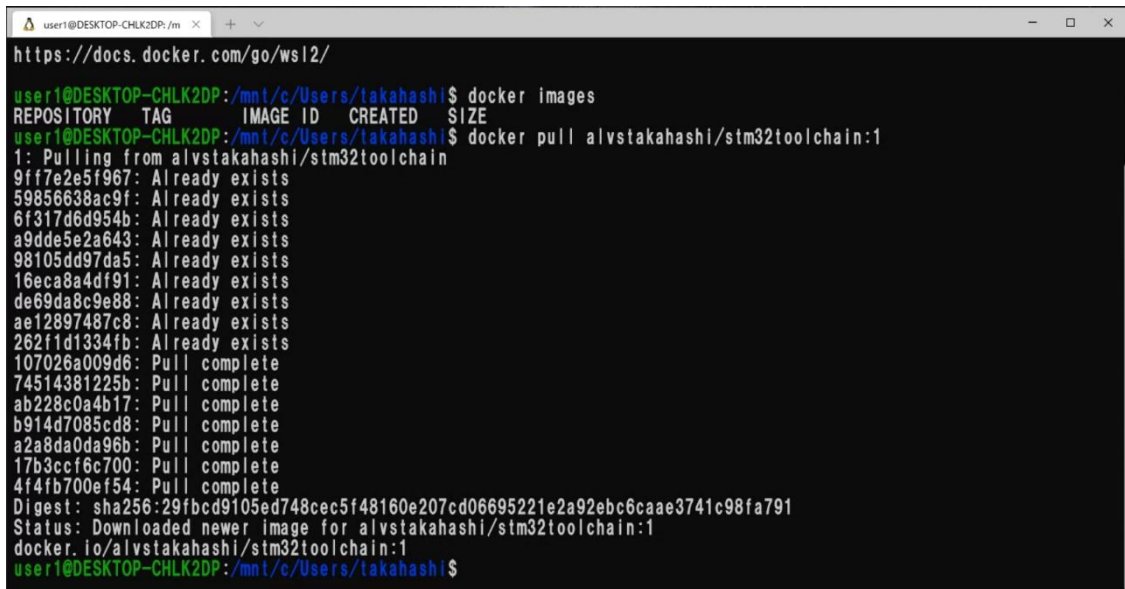
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk$ cd ..
user1@DESKTOP-CHLK2DP: /mnt/d/DEV$ mkdir wk7
user1@DESKTOP-CHLK2DP: /mnt/d/DEV$ cd wk7
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$ git clone -b forRPI3-JTAG https://github.com/alvstakahashi/RPI-SHRINK-SSP-FULL.git
Cloning into 'RPI-SHRINK-SSP-FULL'...
remote: Enumerating objects: 409, done.
remote: Counting objects: 100% (62/62), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 409 (delta 29), reused 45 (delta 16), pack-reused 347
Receiving objects: 100% (409/409), 2.57 MiB | 3.20 MiB/s, done.
Resolving deltas: 100% (209/209), done.
Updating files: 100% (68/68), done.
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$
user1@DESKTOP-CHLK2DP: /mnt/d/DEV/wk7$

```

②VSCode から、Clone したソースコードを load します。

そのまま読み込んでも可能なのですが、データ量が大きいので失敗するケースがあるので docker イメージをあらかじめローカル PC に取り込んでおきます。

```
$ docker pull alvstakahashi/stm32toolchain:1
```

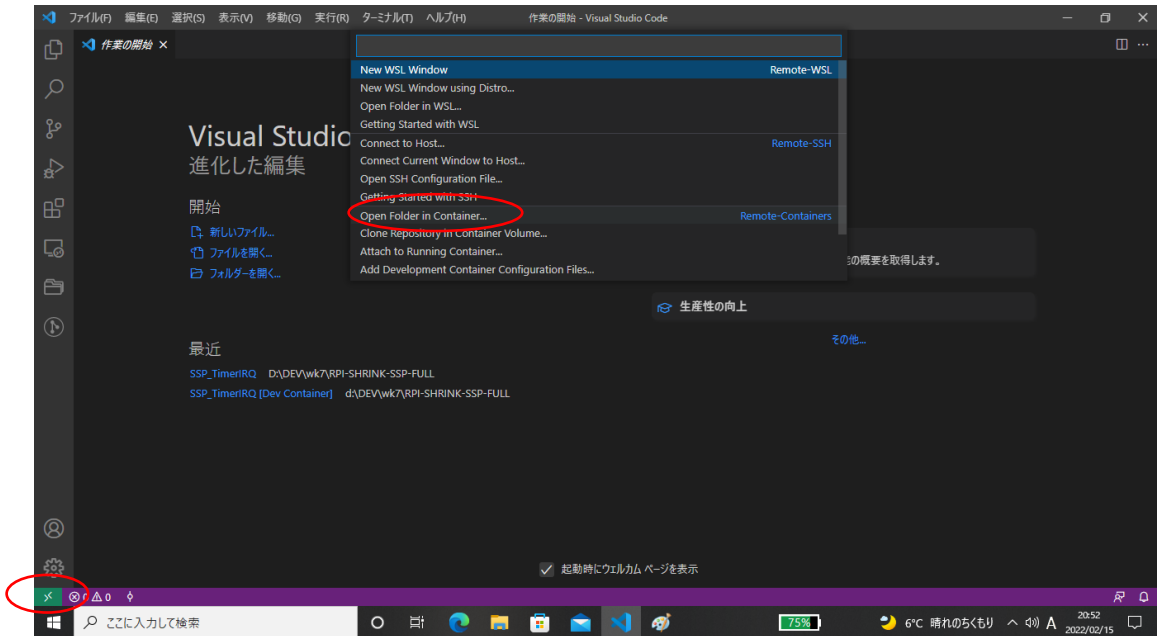
A terminal window screenshot showing the execution of a Docker pull command. The terminal title is 'user1@DESKTOP-CHLK2DP: /mnt/c/Users/takahashi'. The command entered is 'docker pull alvstakahashi/stm32toolchain:1'. The output shows a list of existing images and the successful pull of the specified image. The terminal text is as follows:

```
https://docs.docker.com/go/ws12/
user1@DESKTOP-CHLK2DP:/mnt/c/Users/takahashi$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED        SIZE
user1@DESKTOP-CHLK2DP:/mnt/c/Users/takahashi$ docker pull alvstakahashi/stm32toolchain:1
1: Pulling from alvstakahashi/stm32toolchain
9ff7e2e5f967: Already exists
59856638ac9f: Already exists
6f317d6d954b: Already exists
a9dde5e2a643: Already exists
98105dd97da5: Already exists
16eca8a4df91: Already exists
de69da8c9e88: Already exists
ae12897487c8: Already exists
262f1d1334fb: Already exists
107026a009d6: Pull complete
74514381225b: Pull complete
ab228c0a4b17: Pull complete
b914d7085cd8: Pull complete
a2a8da0da96b: Pull complete
17b3ccf6c700: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:29fbc9105ed748cec5f48160e207cd06695221e2a92ebc6cae3741c98fa791
Status: Downloaded newer image for alvstakahashi/stm32toolchain:1
docker.io/alvstakahashi/stm32toolchain:1
user1@DESKTOP-CHLK2DP:/mnt/c/Users/takahashi$
```

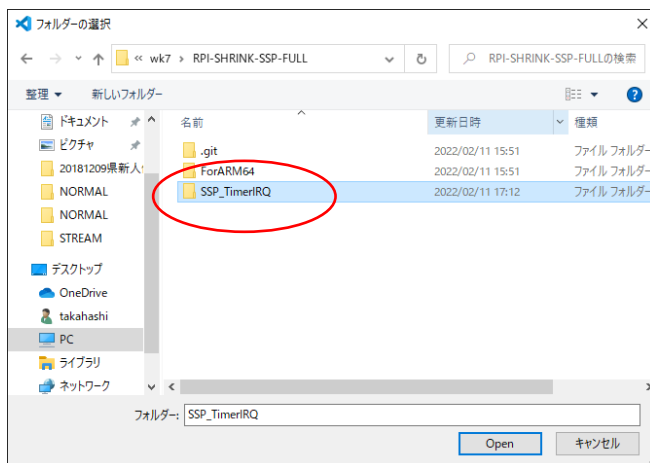


次に Visual Studio Code で Load します。

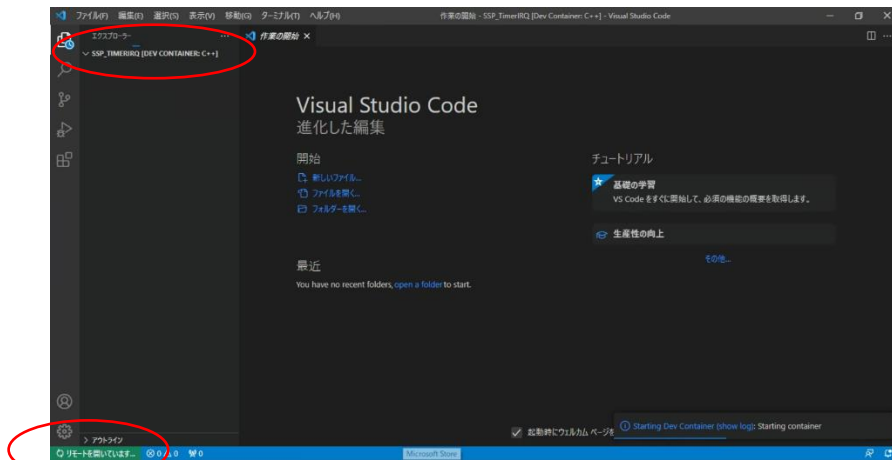
Visual Studio Code を起動後 左下の緑の部分をクリック プルダウンが出てくるので、「Open Folder in Container...」を選びます。



ファイラーがポップアップするので先ほど Clone した一つ下のフォルダ {Timer\_IRQ} を選ぶ



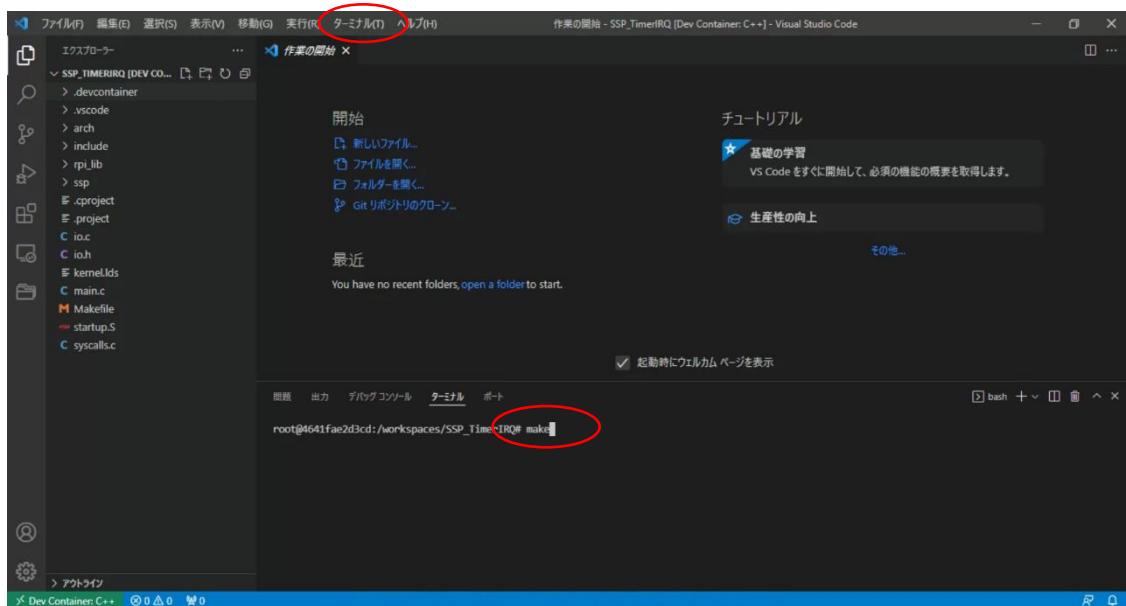
以下のような読み込み状態になります。



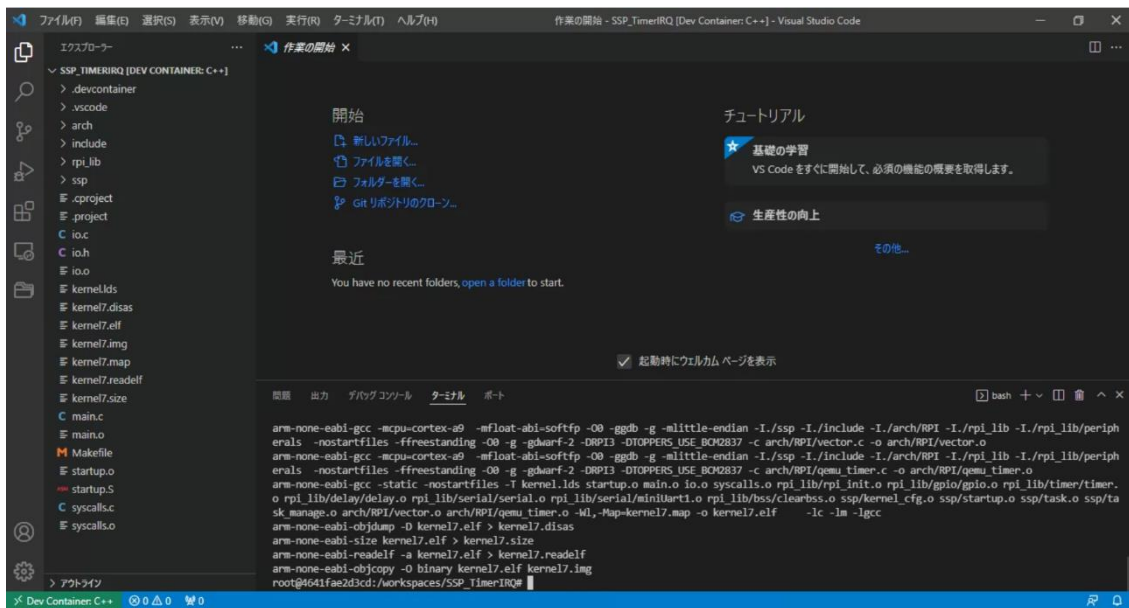
③実行手順に従って実行します。

メニュー「ターミナル」「新しいターミナル」をクリックし、ターミナルから make コマンドでビルドします。

# make



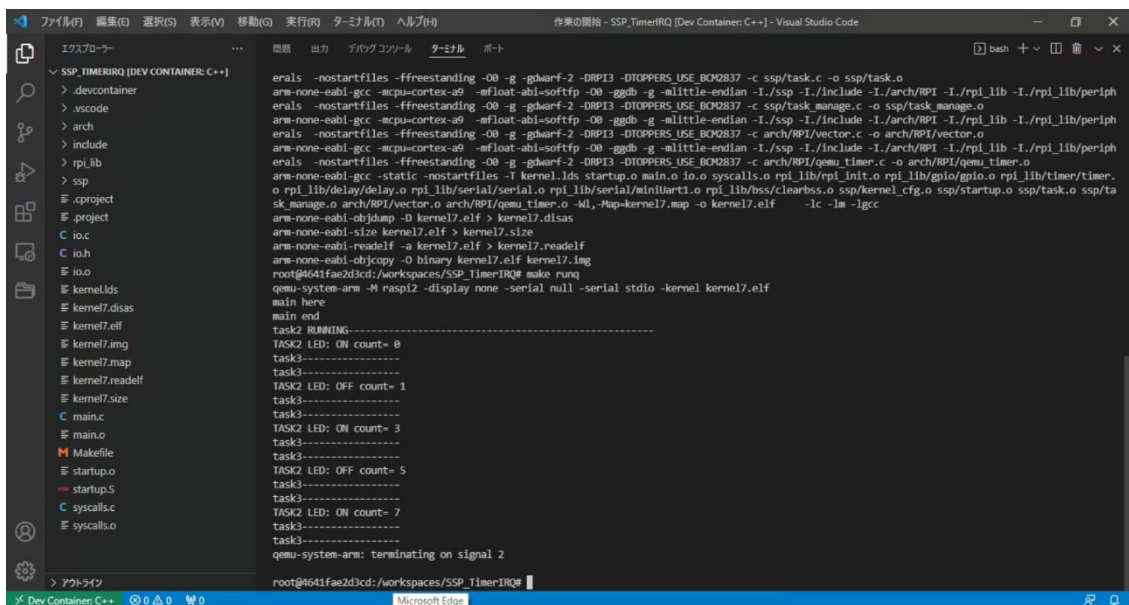
ビルドが完了します。



コンソールで実行してみましょう。

# make runq

2つのタスクが交互に動作している様子が確認できます。



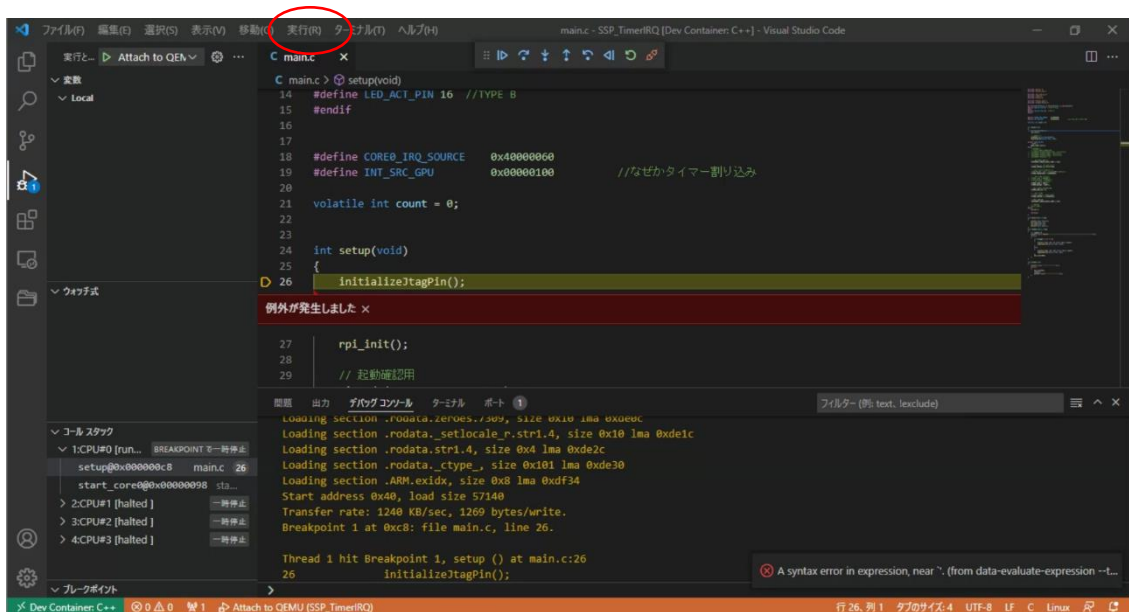
CTRL-C で停止できます。

ソースレベルデバッグを行います。まず デーモンを起動します。

```
#make runqd
```

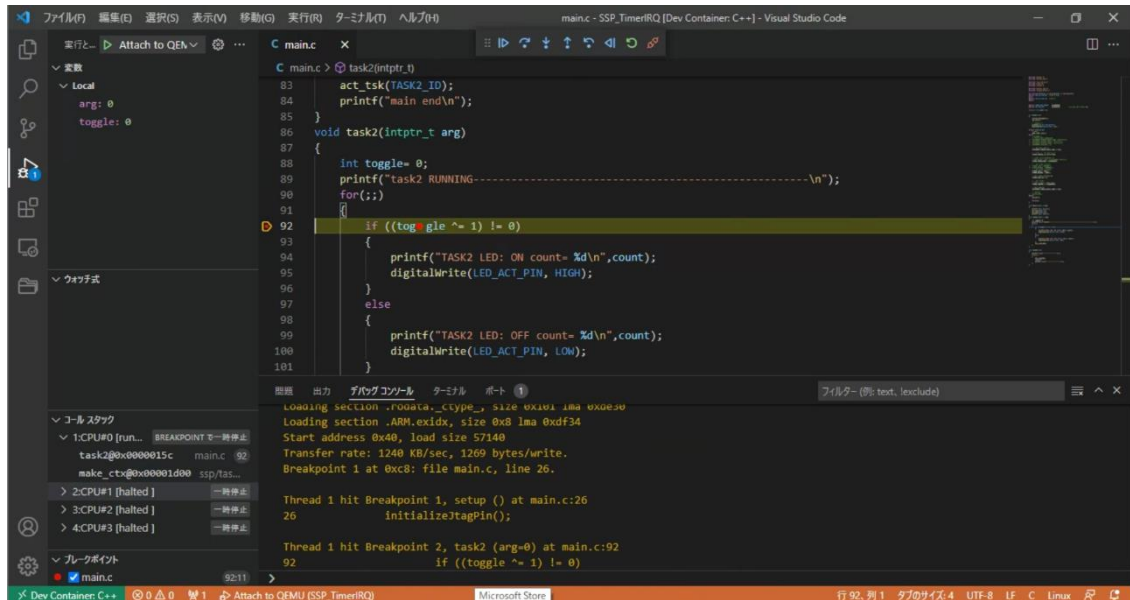
```
task3-----
qemu-system-arm: terminating on signal 2
root@4641fae2d3cd:/workspaces/SSP_TimerIRQ# make runqd
qemu-system-arm -M raspi2 -serial null -serial mon:stdio -S -gdb tcp::1234 -nographic
```

メニュー「実行」「デバッグ開始」で、setup() まで実行してブレークポイントで停止します。



「例外が発生しました」と表示されますが、問題ないです。  
 その後、GDB の操作で、実行が可能です。  
 ステップ実行 [F10] ステップイン[F11]など  
 またブレークポイントを設定して、実行[F5]などで動作確認できます。  
 カーネルの動きを確認可能です。

task2 のループにブレークポイントを置いて、続行すると以下のようになります。



## 停止方法

メニュー 「実行」「デバッグ停止」

その後 デバッグデーモンのターミナルが実行状態となっています。これを停止するには CTRL-A と X を入力します。

お疲れ様でした。 以上で動作確認できました。