はじめに

研修環境の一部に Docker を使用していますが、Docker 自体の説明をしていないのでこの資料に纏めます。Docker のサイトには以下の記述があります（2021 年 3 月 3 日現在）。

『Docker で開発する理由

今日のアプリの開発には、コードを書くだけでは不十分です。複数の言語、フレームワーク、アーキテクチャ、およびライフサイクルステージごとのツール間の不連続なインターフェイスにより、非常に複雑になります。Docker を使用すると、ワークフローが簡素化および高速化されると同時に、開発者は各プロジェクトのツール、アプリケーションスタック、およびデプロイメント環境を自由に選択してイノベーションを起こすことができます。』

※日本語ドキュメントは https://docs.docker.jp/index.html にありますが、最新化されていない部分が残っていたりするので、https://www.docker.com/get-started のサイトを GoogleTransrace で日本語化して見ることをお勧めします。

上記の理由、良く分からないと思いますがスルーして重要な点を以下に纏めます。

●Docker は以下の範囲での利用は Free FOR EVERYBODY（誰であれ無料）です

- 無制限の公開リポジトリ
- Docker Desktop は継続的に更新されます
- Docker Desktop は、Kubernetes およびその他のランタイムをサポートします
- 限られたコンテナ画像のリクエスト
- 二要素認証

＜以下の内容から有料になります $5/月 …2021/3/3 現在＞

- 無制限のプライベートリポジトリ
- 無制限のコンテナ画像リクエスト
- 2 つの並列ビルド
- 毎月 300 回のハブイメージ脆弱性スキャン
- デスクトップとハブのプレミアムカスタマーサポート

※必要なのは「公開リポジトリ」なのでユーザ登録が必要ない範囲で十分利用可能です

●Docker を使う理由（『Docker で開発する理由～』の理由）

① Docker は隔離した環境（コンテナ=ミドルウェア＋依存ライブラリ）を作ることができる。ミドルウェアをインストールする際、他のミドルウェアとの干渉（依存ライブラリのバージョン相違等）がない

② Docker の公開リポジトリから最新・設定済のミドルウェアを手軽にレジストリの汚染等を気にすることなく導入できる

●注意点

Docker はホスト（例えば Windows が動作している）環境上にネットワーク接続可能な隔離された Linux の環境を構築できますが、VMware 等とは異なり、可搬性のあるオールインワンのイメージファイルができるわけではありません。特に永続性が必要なデータボリュームを作る場合は想定している運用形態に合致するか検討が必要です。

1. Docker のインストール

以下の内容は Windows10(Home エディションも可)で、2021/03/01 までの Windows Update が全て摘要されていることを前提にしています。

1.1. 必要なツール／ライブラリ

（1） WSL2 (Windows Subsystem for Linux 2[1])

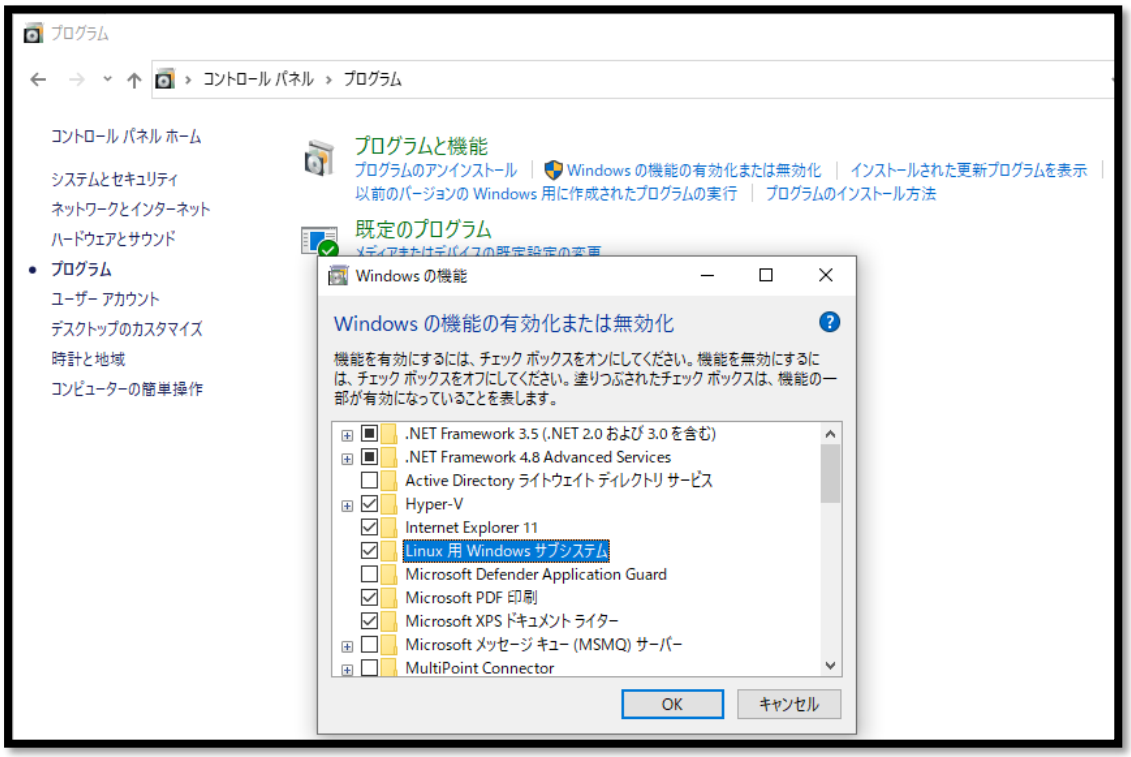
Windows10 Home エディションを使う場合は WSL2 が必要です。

Windows10 Pro の場合は WSL2 の使用は任意ですが、資料都合上 WSL2 を前提で進めます。

【手順】

① コントロールパネル ＞ プログラム ＞ 「Windows の機能の有効化または無効化」

「Linux 用 Windows サブシステム」に☑を確認 …チェックを入れたら OK 後再起動



② https://docs.microsoft.com/ja-jp/windows/wsl/wsl2-kernel

上記 URL の「手順 4 -Linux カーネル更新プログラムパッケージをダウンロード」を行い、更新プログラムをインストールする。

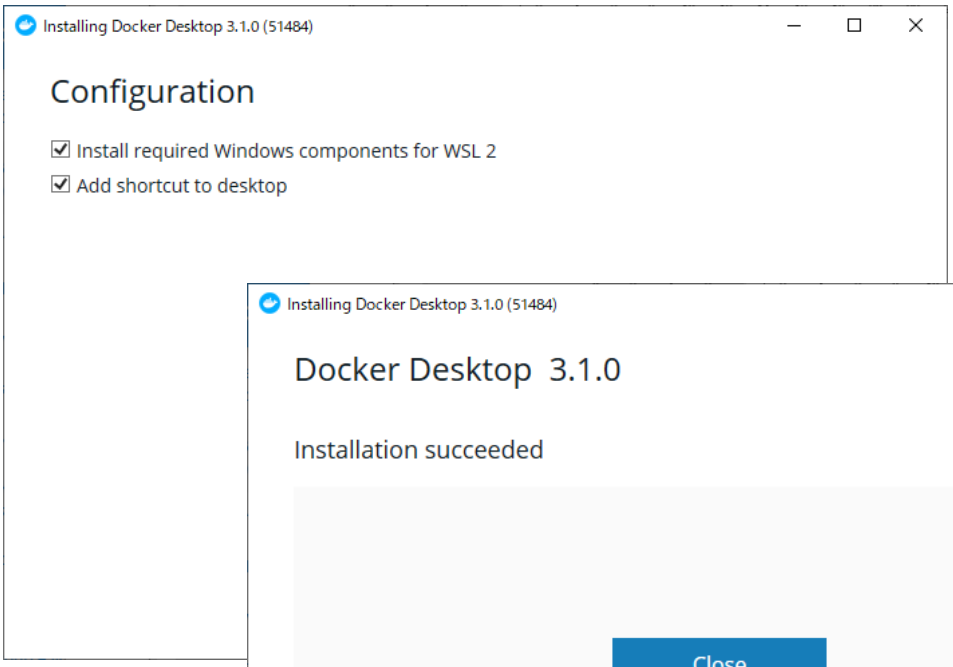
---

[1] Windows 用にカスタマイズした Linux カーネルを Hyper-V 上で動作させる仕組み。WSL2 をインストールすると Linux カーネルがインストールされ、この上で Docker を動作させる

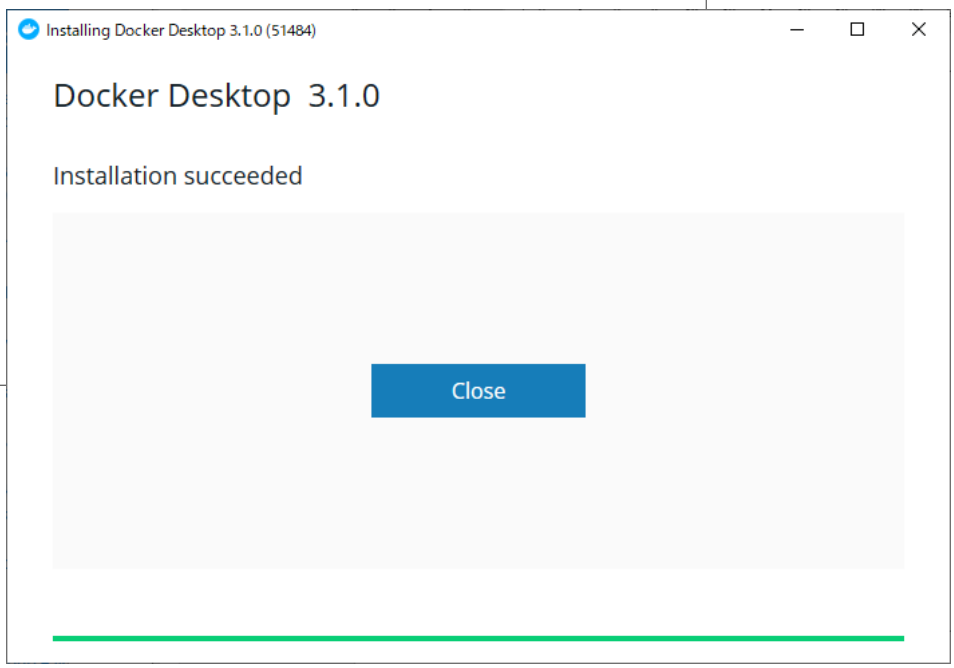
（２） Docker Desktop

https://hub.docker.com/editions/community/docker-ce-desktop-windows/

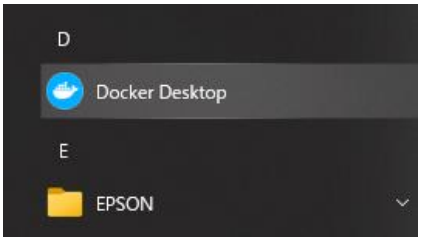
「Get Docker Desktop for Windows」よりインストーラをダウンロードし起動する。



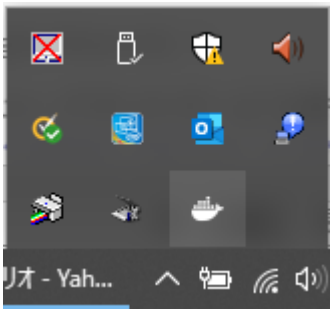
＜数分後＞



Close でインストーラ画面を閉じ、「スタート」ボタンを押下すると...



Docker Desktop を選択して起動する。
（自動起動に設定する。以後は明示的な起動は不要）



タスクトレイに Docker のアイコンがでたら稼働中

2. イメージ作成

Docker のミドルウェアインストール済のコンテナを作成します。

ここでは Alpine という Linux ディストリビューションに PostgreSQL の最新バージョンをセットアップしたイメージを作成します。

（1）Dockerfile ファイルの作成

Docker は Dockerfile に書かれた内容を実行します。

フォルダを作成（例：C:¥Tools¥postgres）し、以下の内容のテキストファイル（拡張子無）を格納します。

＜Dockerfile の例＞

```
FROM postgres:alpine
# 日本語環境に設定… image 作成ｺﾏﾝﾄﾞ [ docker build -t postgres-jp --rm=true . ]
RUN cp /usr/share/zoneinfo/Asia/Tokyo /etc/localtime
ENV LANG ja_JP.utf8
```

1 行目：alpine に postgres(PostgreSQL)が入った最新イメージをダウンロード

2 行目：コメント

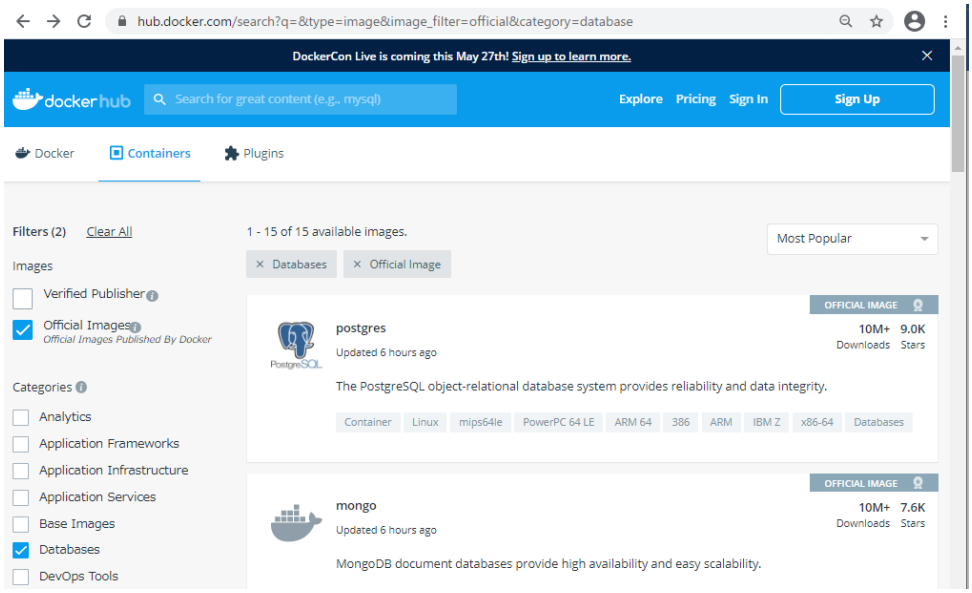
3 行目：イメージ/OS に日本時刻を設定（時刻設定ファイルのコピー）

4 行目：イメージ/OS の言語設定を日本語、文字コード UTF-8 に設定（環境変数のセット）

※1行目だけでイメージが作られます。2行目以降は、FROM で作られたイメージ(OS:Alpine)に渡されます

【Docker のイメージ】以下の公開リポジトリから選ぶことができます

https://hub.docker.com/search?q=&type=image



（２）docker コマンドの実行(イメージ作成)

コマンドプロンプトより、Docker ファイルを格納したフォルダに移りイメージ作成のコマンドを実行します。

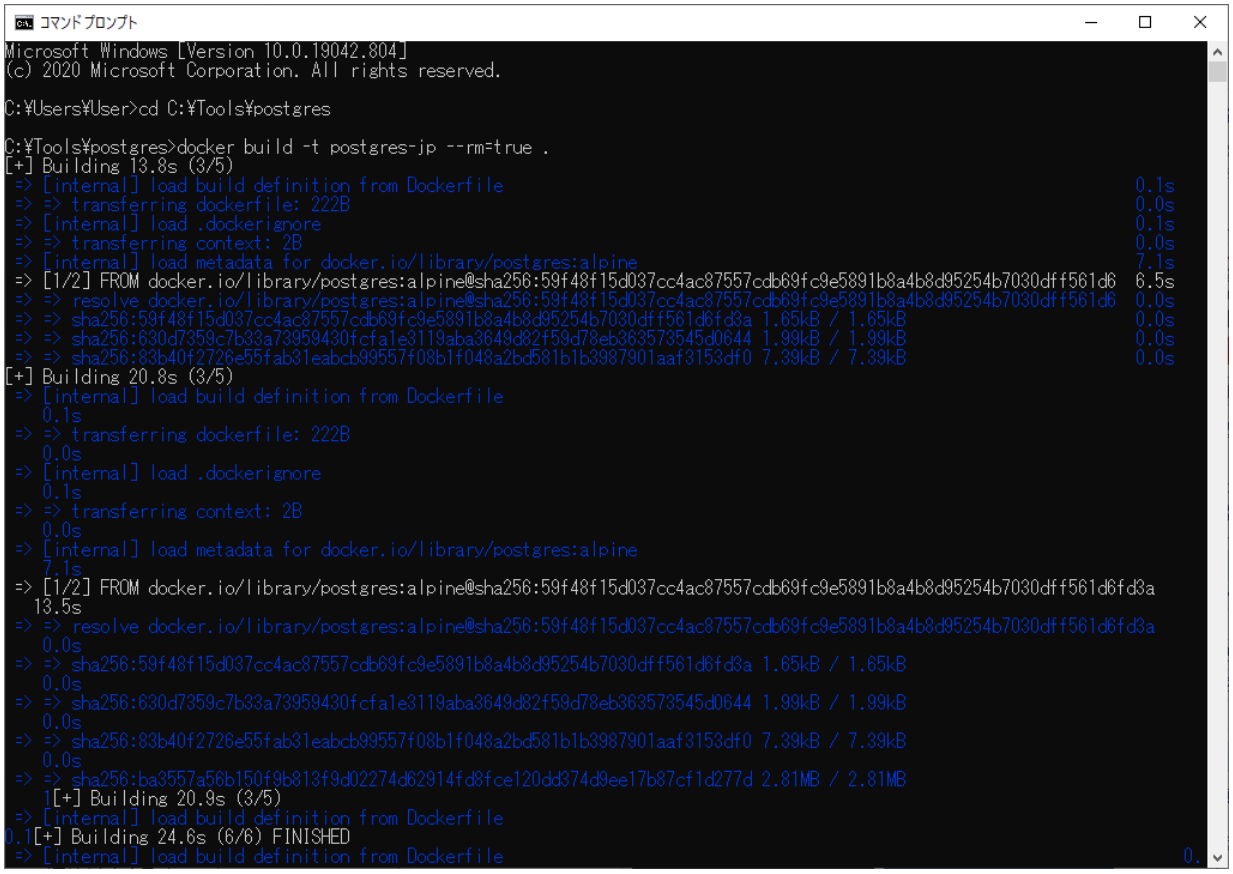
＜実行例＞

ｉ cd C:¥Tools¥postgres　…このフォルダに前項(1)の Docker ファイルが存在する前提

ii docker build -t **postgres-jp** --rm=true .

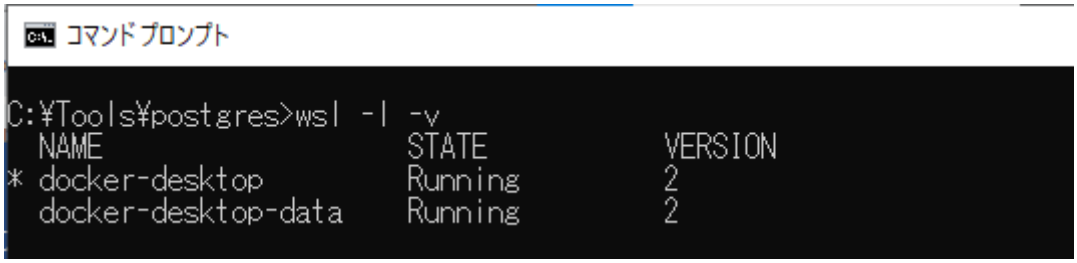
postgres-jp というタグ（これでイメージを識別します）付きのイメージを作成します。

最後の“.”（ピリオド）で Docker ファイルの探し場所をカレントディレクトリとしています



--- 24.6 秒後（回線速度によります）に build が完了----

●コマンド wsl -l -v で“docker-desktop”，“docker-desktop-data”の 2 つの wsl2 上で動作していることが分かります



●コマンド docker images で上記で作成したイメージ“postgres-jp”が確認できます

●Docker のリソース及びイメージは下記フォルダに格納されます。

- C:¥Program Files¥Docker¥Docker¥resources
- ¥¥wsl$¥docker-desktop-data¥version-pack-data¥community¥docker¥volumes
- ¥¥wsl$¥docker-desktop¥

（３）イメージの実行環境定義(docker-compose による起動)

作成したイメージは「docker container start ~」でも起動できますが、リスンポートやデータボリュームのマウント等をパラメータで指定する必要があります。

docker-compose コマンドを使うと複数コンテナの連動やポート、データボリューム等の構成の定義を **docker-compose.yaml** ファイルで行うことができます。

＜docker-compose.yaml の例＞

```
version: '3'
services:
  postgres:
    image: postgres-jp
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
    volumes:
      - dbdata:/var/lib/postgresql/data
      - ./docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d

volumes:
  dbdata:
```

①compose ファイルの様式
サービスの定義
　サービスを識別する名前
　使用するイメージ（タグ）名
　サービスが使用するポート
　　②コンテナ⇔ホストポート
　環境変数の宣言
　　PostgreSQL 利用 id
　　上記のパスワード
　③ボリューム関係付け
　　名前付ボリュームdbdata
　　ホストボリューム

名前付ボリュームの定義
　③dbdata ボリューム作成

①compose ファイルの様式はバージョンが変わると使えるキーワードが変わります
②コンテナは独立してポート番号が管理されます。コンテナの外（Docker が関係しないアプリ）コンテナ内のアプリと通信を行うためには、ホスト側のポートにブリッジする必要があります。この例では、コンテナの 5432 ポート（PostgreSQL のデフォルトポート番号）をホストの 5432 ポートにブリッジして PostgreSQL がホストにあるのと同様に見せています
③ボリュームは、Docker の用語集には **host volume**（ホストボリューム）、**named volume**（名前付ボリューム）、**anonymous volume**（匿名ボリューム）があります。コンテナ内のアプリケーションからデータを更新する場合は、名前付ボリュームにしてください。

【更新系ボリュームが named-volume である必要性】

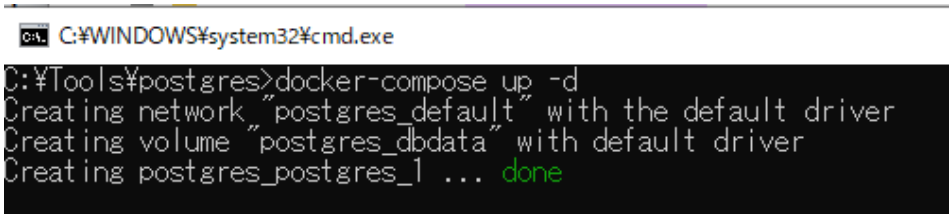
Windows をホストとして Docker のコンテナ(Linux)を起動する場合、アクセス権の整合性を合わせるのが困難になります。コンテナの中は Linux のユーザ管理とパーミッションによるアクセス制御が働き（ユーザ=“postgres”だけが読み、書き、実行の権限を必要とする）、ホスト側の Windows アカウントによるアクセス権を取得できずに更新できません。

（４）docker-compose によるコンテナの起動

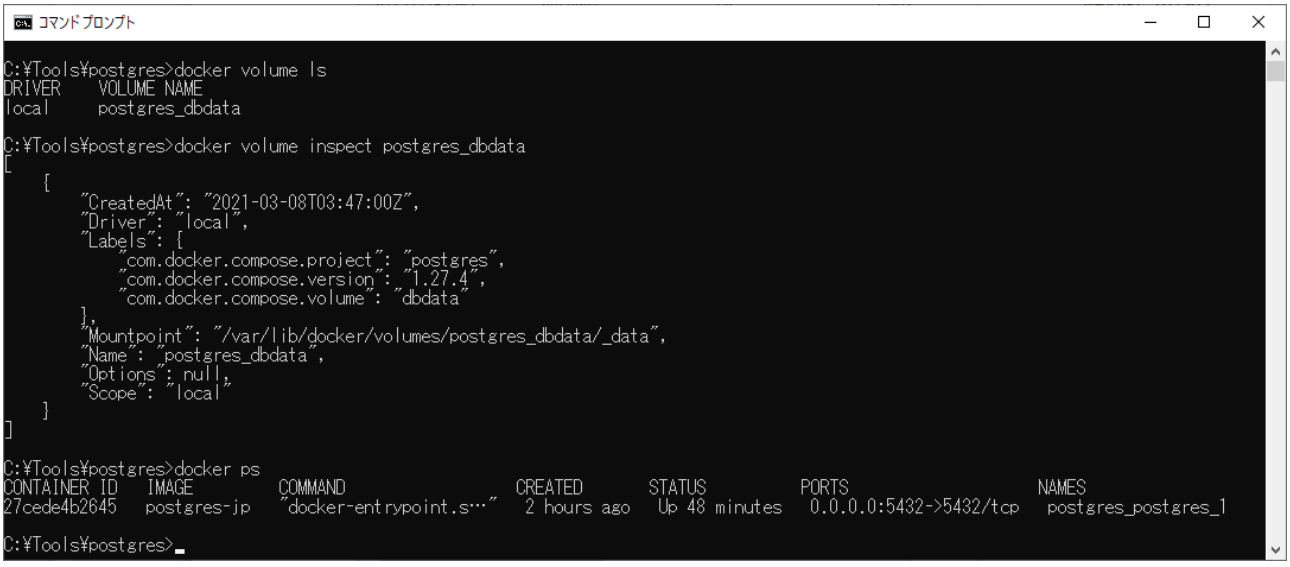
前項で作った docker-compose.yaml ファイルの格納先をカレントディレクトリとして

コマンド：docker-conpose up -d （-d は任意)を実行します。

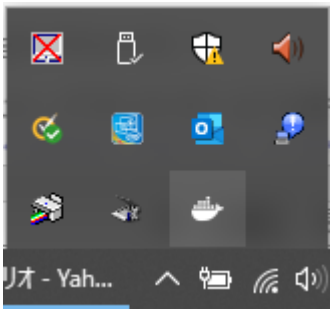
●初回起動時は、Creating network ～、Creating volume ～と表示されます。



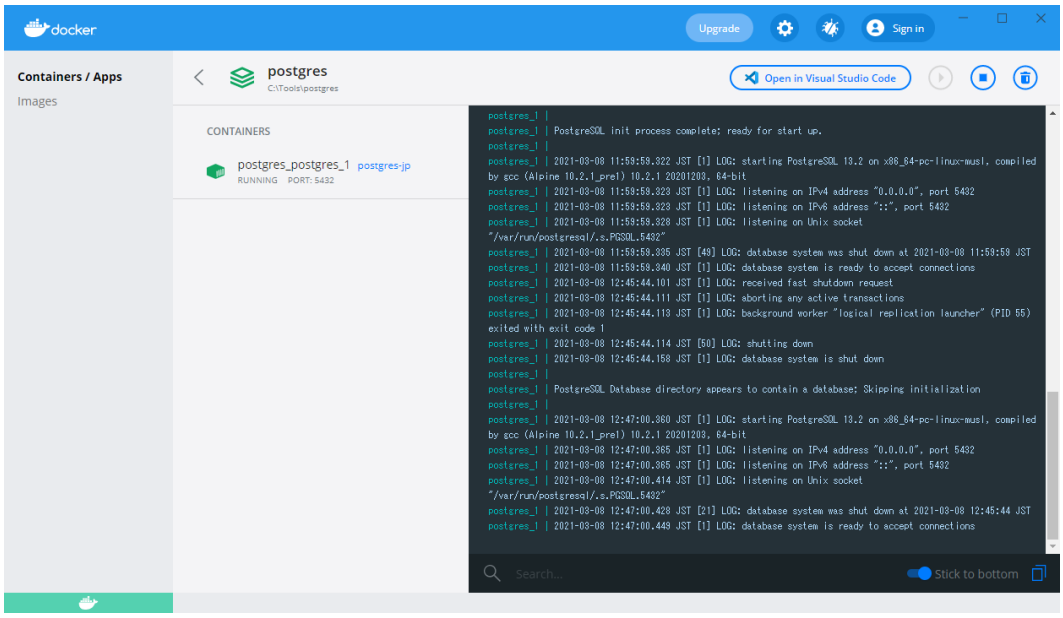
●docker-compose の実行により docker-compose.yaml に書いた名前付ボリュームが作成され、docker volume ls で確認できます。また、docker ps コマンドでコンテナの状態や名前を表示することができます。



●Docker Desktop の方からも状態確認ができます



タスクトレイのアイコンをクリックすると、コンテナの起動状態が表示される。



●コンテナで稼働している OS にログインすることもできる。

start "docker exec -it postgres-jp bash" docker exec -it **postgres_postgres_1** bash

start “*見出し*” で wind を開き、docker exec -it コマンドでコンテナ内で bash 実行