

コンテナ仮想化 (Docker互換 – Rancher Desktop)

目次

はじめに	1
1. Docker の構成と有償化部分	1
2. Docker の互換ソフトウェア	1
3. Rancher Desktop のセットアップ	2
3.1. ダウンロード	3
3.2. インストール	3
4. Rancher Desktop の起動と終了	5
5. カスタムイメージの作成	6
6. イメージの起動 (docker-compose コンテナ化)	7
7. 同一イメージから複数コンテナの生成	9
8. コンテナ上のサービス利用	9
9. コンテナの状態と停止・削除	10
10. コンテナへのターミナル接続	10
11. ホストとコンテナのネットワーク	11
11.1. ゲートウェイ	11
11.2. コンテナからホスト OS	12
11.3. ホスト OS からコンテナ	14
11.4. コンテナから Hyper-V 仮想マシン	16
12. WSL2 とコンテナの注意点	18
13. 使い方	20

コンテナ仮想化 (Docker互換 – Rancher Desktop)

はじめに

「オープンソース」のソフトウェアはソースを公開していますが、特許や著作権、商標等の知的財産権を放棄しているわけではありません。指定のライセンス条件下でのみ利用することができ、条件が変更されたりソフトウェアの一部（コアのみ）だけがオープンソースになっている場合もあります。

Docker は Docker 社が GitHub に公開しているオープンソース（リポジトリが複数存在します）で、2022 年 2 月 1 日から条件付きで有償になりました。これに先立ち、Red Hat 社は「RHEL 8 から Docker コンテナエンジンと、docker コマンドが削除されました。」と発表¹し、コンテナ運用管理の主力ソフトウェア Kubernetes は技術的な理由で v1.20 以降 Docker を非推奨にしました²。

ミドルウェアやアプリをインストール済のプラットフォーム（コンテナ）をひな形イメージから簡単に作ることができ、公開されている（Docker Hub³）イメージが多数あるので、コンテナを使い続けることができる Docker 代替ソフトウェア群の開発が活発に行われています。

1. Docker の構成と有償化部分

Docker は Linux 環境を対象⁴に開発され、主要な機能に以下のものがあります（一部）。

- ・ひな形イメージからカスタマイズしたイメージの作成（コマンド：buid）
- ・コンテナの起動（コマンド：start, run）
- ・コンテナ間の連動やネットワーク管理（コマンド：docker-compose）
- ・コンテナの監視や削除、シェルを含むコマンド実行（コマンド：ps, rm, exec）

Docker はサーバの Docker デーモン(dockerd)でコンテナを操作したり監視を行い、操作コマンドと操作の内容をサーバに送信するクライアント(docker)で構成されています。

クライアントはホスト OS 用の実行可能ファイルで、Linux や Windows、Mac 等ホスト OS 毎の実行環境（Windows であれば exe ファイル）が必要になります。Docker 社は自社サイトで公開している実行環境毎のコマンドや GUI の監視ツールを同梱した“Docker Desktop”を有償化しました。

2. Docker の互換ソフトウェア

Docker はコンテナランタイム、イメージのビルド、ネットワーク等の機能で構成されています。現在では各機能毎に類似・互換可能なコンポーネントがそれぞれ（オーナー組織／リポジトリ）で開発されていて(containerd/containerd ⇔ opencontainers/runc、moby/buildkit ⇔ containers/buildah、moby/libnetwork ⇔ containernetworking/cni、…)、色々な組合せで Docker を代替します。

¹ [Red Hat Customer Portal] 1.3. Docker を使用せずにコンテナを実行

https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/8/html/building_running_and_managing_containers/con_running_containers-without-docker_assembly_starting-with-containers

² Don't Panic: Kubernetes and Docker <https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/>

³ Docker Hub <https://hub.docker.com/search?q=>

⁴ マイクロソフト社は Docker Desktop で Windows が動作する Windows コンテナを公開しています <https://learn.microsoft.com/ja-jp/virtualization/windowscontainers/about/>

コンテナ仮想化 (Docker互換 – Rancher Desktop)

< Docker 代替候補 >

Podman⁵ : Docker が“対応外”となった RHEL 8 で推奨しています。コマンドが podman に変わりますが、ほぼ同一のサブコマンドが使えます。イメージの作成(buid)に使う Dockerfile は Docker のものを流用できますが、Docker-Compose の設定ファイルはコンテナを連動させる方法がポッド(Pod)に変わったため使えません。

nerdctl⁶ : コマンドが docker から nerdctl に変わる以外は Docker と互換・機能追加があります。Windows 版の実行ファイルもありますが、他に以下の追加インストールが必要です。
追加インストールが必要なコンポーネント… containerd、BuidKit、CNI

Rancher Desktop⁷ : nerdctl と Docker(moby)から選択でき、Docker(moby)を選択すると、docker コマンド (Docker-Compose を含む) がそのまま使えます。

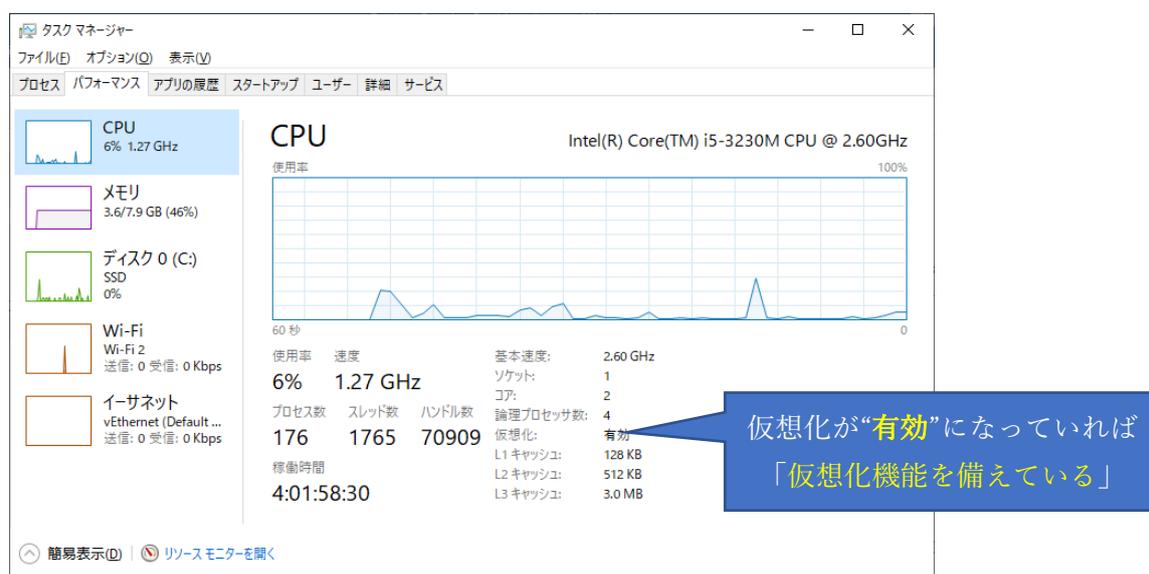
3. Rancher Desktop のセットアップ

以下、Rancher Desktop のバージョン 1.8.1 を Windows 10 Pro 22H2 の環境で試行します。

Rancher Desktop 1.8.1 は Apache-2.0 ライセンス (商用利用可) です。

<必要な環境>Windows では Rancher Desktop には次のものがが必要です⁸。

- ・ Windows 10 ビルド 1909 以降。Home エディションもサポート対象です
- ・ 仮想化機能を備えたマシン
- ・ 永続的なインターネット接続



⁵ Podman(ダウンロード Windows/Mac/Linux) <https://github.com/containers/podman-desktop>

⁶ nerdctl <https://github.com/containerd/nerdctl>

⁷ Rancher Desktop(ダウンロード) <https://rancherdesktop.io/>
(リポジトリ) <https://github.com/rancher-sandbox/rancher-desktop/>

⁸ (Windows 環境) <https://docs.rancherdesktop.io/getting-started/installation#windows>

コンテナ仮想化（Docker互換－Rancher Desktop）

3.1. ダウンロード

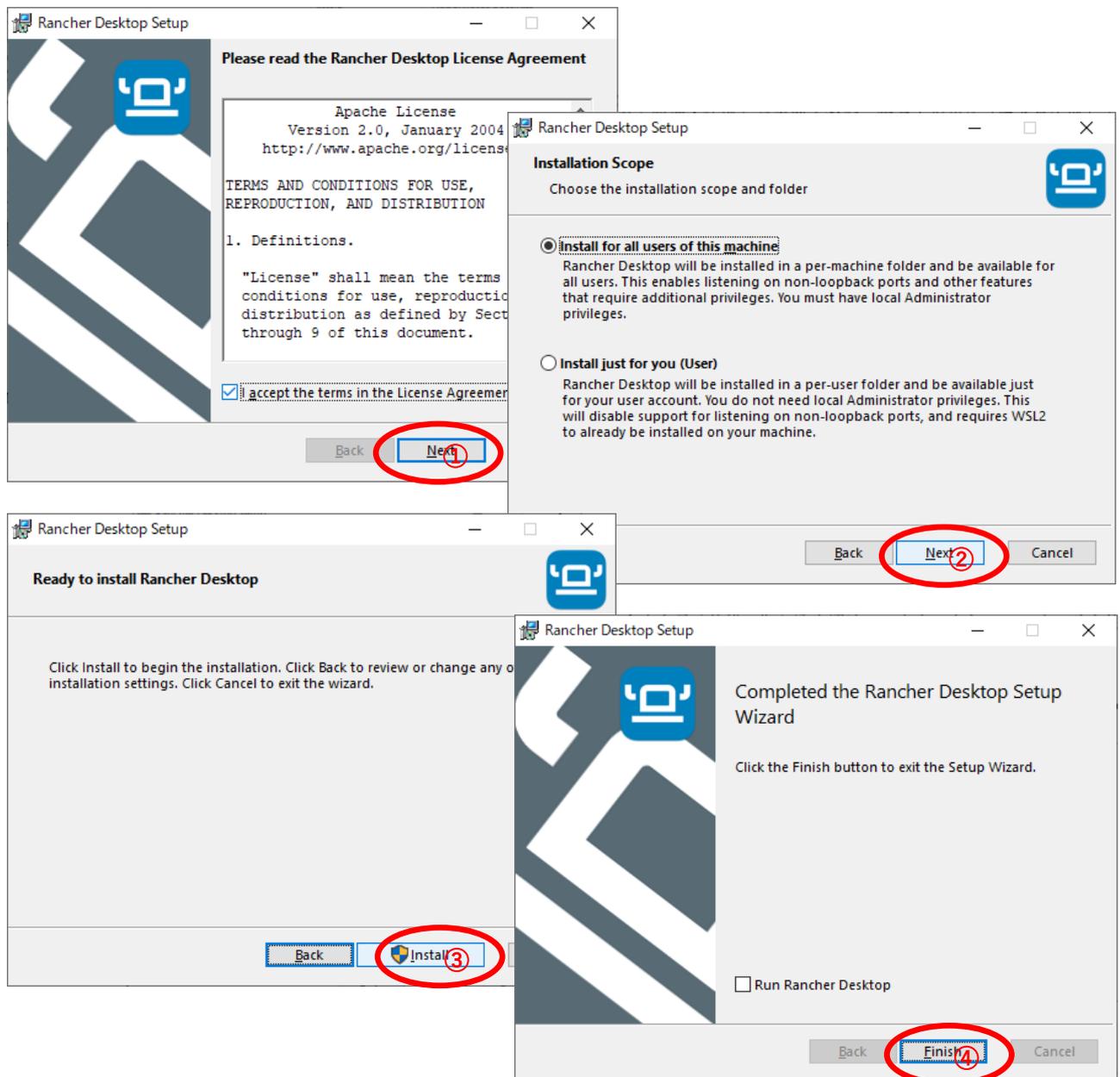
<https://rancherdesktop.io/>から Windows 用のインストーラをダウンロードします。

3.2. インストール

Rancher Desktop には Windows 上の Linux 用 Windows サブシステム（WSL2）⁹が必要で、これは Rancher Desktop セットアップの一部として自動的にインストールされます。

以下は、WSL2 を設定済の環境で Rancher Desktop のインストーラを実行した例です。

- (1) インストール時の選択肢は②の画面の利用者の範囲（全利用者／インストール者のみ）だけで、どちらを選んでも機能に違いはできません。



⁹ 手動でインストールする場合は管理者モードのコマンドプロンプトまたは PowerShell から `wsl --install` を実行します。 <https://learn.microsoft.com/ja-jp/windows/wsl/install>

コンテナ仮想化 (Docker互換 - Rancher Desktop)

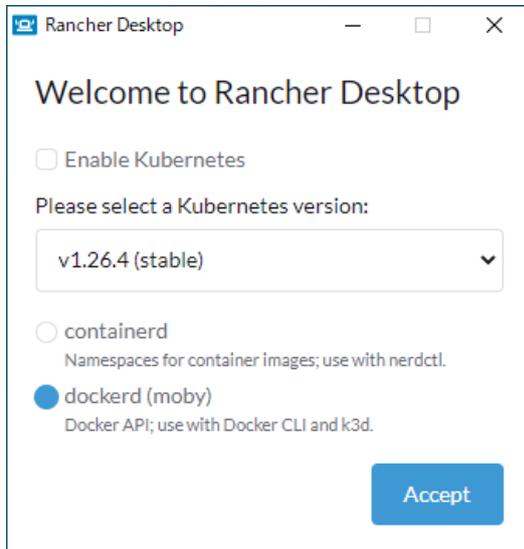
【インストールされるファイル…Windows 関連以外は省略しています】

```
C:\PROGRAM FILES\RANCHER DESKTOP
├── chrome_100_percent.pak
├── chrome_200_percent.pak
├── d3dcompiler_47.dll
├── ffmpeg.dll
├── icudtl.dat
├── libEGL.dll
├── libGLESv2.dll
├── LICENSE.electron.txt
├── LICENSES.chromium.html
├── Rancher Desktop.exe
├── resources.pak
├── snapshot_blob.bin
├── v8_context_snapshot.bin
├── vk_swiftshader.dll
├── vk_swiftshader_icd.json
├── vulkan-1.dll
├── locales
│   └── ja.pak (他略)
├── resources
│   └── resources
│       ├── darwin
│       ├── icons
│       ├── linux
│       ├── rancher-dashboard
│       └── win32
│           ├── distro-0.34.tar
│           ├── wsl-helper.exe
│           └── bin
│               ├── docker-buildx.exe
│               ├── docker-compose.exe
│               ├── docker-credential-ecr-login.exe
│               ├── docker-credential-none.exe
│               ├── docker-credential-wincred.exe
│               ├── docker.exe
│               ├── helm.exe
│               ├── kubectl.exe
│               ├── kuberlr.exe
│               ├── nerdctl.exe
│               ├── rdctl.exe
│           └── internal
│               ├── host-resolver.exe
│               ├── host-switch.exe
│               ├── privileged-service.exe
│               ├── steve.exe
│               └── vtunnel.exe
```

コンテナ仮想化（Docker互換－Rancher Desktop）

（2）初回起動の構成選択

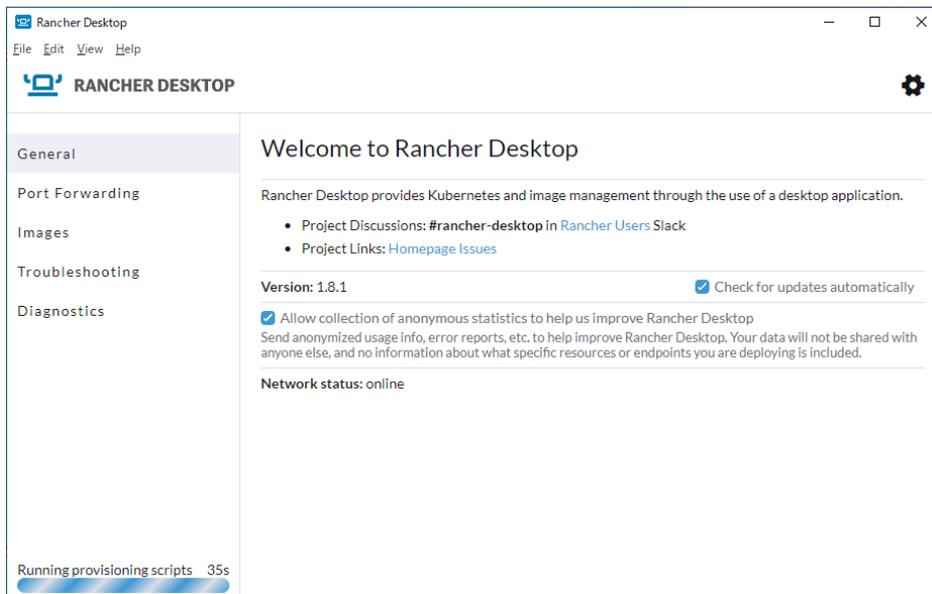
インストール後の初回起動時に containerd または docker(moby)のどちらを使うか選べます。docker(moby)を選択すると操作コマンド（CLI）は Docker と同様“docker”になります。（Enable Kubernetes のオプションは、Kubernetes を使う場合に選択…説明は割愛します）



4. Rancher Desktop の起動と終了

（1）起動

仮想マシン（コンテナ）の起動やイメージの操作の前にインストーラが作ったショートカットが Windows スタートボタンから C:\PROGRAM FILES\RANCHER DESKTOP\Rancher Desktop.exe を起動します。または Rancher Desktop に同梱の rdctl.exe を使い、rdctl start コマンドでも起動できます。



画面左下に表示されたプログレスバーが消えたら起動完了で、以降 docker コマンドが使えます。

（2）終了

画面の File メニューから Exit を選択すると Rancher Desktop.exe 終了します（閉じるボタンで画面を消しても終了しません）。またはコマンドプロンプト等から rdctl shutdown と打鍵します。

コンテナ仮想化 (Docker互換 – Rancher Desktop)

5. カスタムイメージの作成

操作方法は Docker と同一です。Docker Hub からひな形になるイメージをダウンロードして自分の環境にカスタマイズしたイメージを作るために、以下の作業を行います。

(1) Dockerfile の作成

Docker で使用したものがそのまま使えます。

以下、Docker Desktop のバージョン 3 で使用したファイルの例(フォルダと Dockerfile を作る)。

C:¥TOOLS¥POSTGRES

└─Dockerfile

----Dockerfile の内容----

FROM postgres:alpine

日本語環境に設定… image 作成[docker build -t postgres:jp --rm=true .]

RUN cp /usr/share/zoneinfo/Asia/Tokyo /etc/localtime

ENV LANG ja_JP.utf8

(2) docker build コマンドの実行

- ① wsl --list --verbose で rancher-desktop が Running になっているのを確認します
- ② docker build -t リポジトリ名:タグ (--rm=true で中間イメージ削除) Dockerfile のパスを実行するとイメージの取得が始まります
- ③ docker images コマンドで②で指定した リポジトリ名:タグ が表示されれば成功です

```

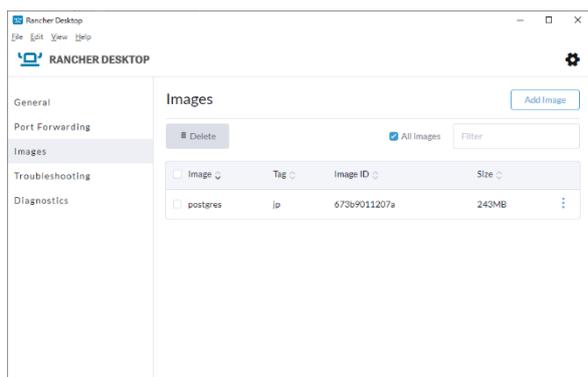
C:¥Tools¥postgres>wsl --list --verbose
NAME                STATE          VERSION
* rancher-desktop    Running        2
rancher-desktop-data Stopped        2

C:¥Tools¥postgres>docker build -t postgres:jp --rm=true .
[+] Building 10.5s (6/6) FINISHED
=> [internal] load build definition from Dockerfile                    3.6s
=> => transferring dockerfile: 32B                                    0.0s
=> [internal] load .dockerignore                                       2.3s
=> => transferring context: 2B                                         0.0s
=> [internal] load metadata for docker.io/library/postgres:alpine     4.2s
=> [1/2] FROM docker.io/library/postgres:alpine@sha256:d9c304353c031b21e9a7e33dc4781e272a9fa802a2ab9703fe4199d72ba1  0.0s
=> CACHED [2/2] RUN cp /usr/share/zoneinfo/Asia/Tokyo /etc/localtime  0.0s
=> exporting to image                                                  1.8s
=> => exporting layers                                                 0.0s
=> => writing image sha256:673b9011207ad431dc8bd3687797f30efd2c8d93aeca3f8f15e245bc3c909e  0.2s
=> => naming to docker.io/library/postgres:jp                         0.2s

C:¥Tools¥postgres>docker images
REPOSITORY TAG          IMAGE ID      CREATED          SIZE
postgres   jp           673b9011207a 19 minutes ago  243MB

C:¥Tools¥postgres>

```



作成されたイメージは Rancher Desktop の GUI から確認したり操作できます。

コンテナ仮想化 (Docker互換 - Rancher Desktop)

6. イメージの起動 (docker-compose コンテナ化)

Docker (ver3.1) で使用したものがそのまま使えます。

(1) docker-compose.yaml

起動環境を記述した docker-compose.yaml を作ります。

以下、Docker Desktop のバージョン 3 で使用したファイルの例

C:¥TOOLS¥POSTGRES

└docker-compose.yaml

└Dockerfile

---- docker-compose.yaml の内容 ----

```
version: '3'
```

```
services:
```

```
  postgres:
```

```
    image: postgres:jp
```

```
    ports:
```

```
      - "5432:5432"
```

```
    environment:
```

```
      POSTGRES_USER: admin
```

```
      POSTGRES_PASSWORD: admin
```

```
    volumes:
```

```
      - dbdata:/var/lib/postgresql/data
```

```
      - ./docker-entrpoint-initdb.d:/docker-entrpoint-initdb.d
```

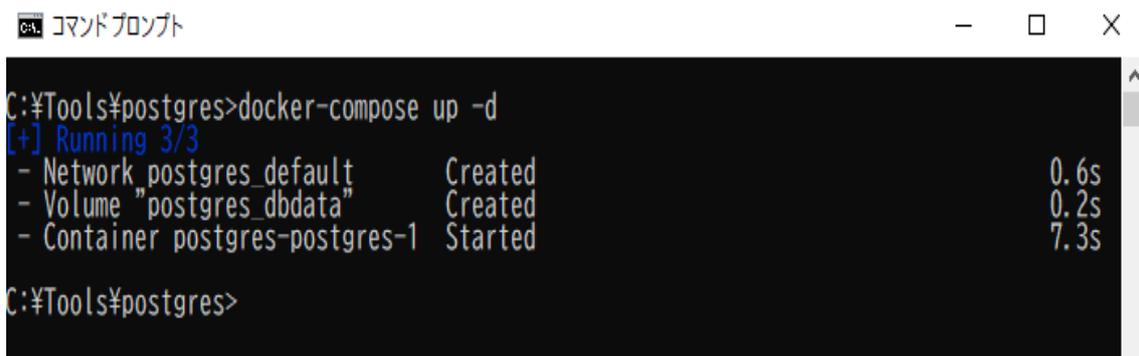
```
volumes:
```

```
  dbdata:
```

(2) docker-compose コマンドの実行

docker-compose.yaml ファイルのディレクトリに cd 後に以下のコマンドを実行します。

```
docker-compose up -d
```



```

C:¥Tools¥postgres>docker-compose up -d
[+] Running 3/3
 - Network postgres_default      Created           0.6s
 - Volume "postgres_dbdata"     Created           0.2s
 - Container postgres-postgres-1 Started            7.3s

C:¥Tools¥postgres>
  
```

docker-compose.yaml で参照しているフォルダ docker-entrpoint-initdb.d は自動生成されます



ホスト OS とディレクトリやファイルが共用でき更新もできますが、利用者権限や inode に関する機能は違いが出るのでコンテナ側からは更新しない方が無難です。

コンテナ仮想化 (Docker互換 – Rancher Desktop)

その他の点も Docker と同様に動作し、docker-compose.yaml に指定された資材が作られます。

① 初回の docker-compose によるコンテナ生成で名前付ボリュームが作られ、以下のコマンドで確認できます。

- docker volume ls
- docker volume inspect ボリューム名

```

C:\Tools\postgres>docker volume ls
DRIVER      VOLUME NAME
local      postgres_dbdata

C:\Tools\postgres>docker volume inspect postgres_dbdata
[
  {
    "CreatedAt": "2023-05-23T00:35:08Z",
    "Driver": "local",
    "Labels": {
      "com.docker.compose.project": "postgres",
      "com.docker.compose.version": "2.16.0",
      "com.docker.compose.volume": "dbdata"
    },
    "Mountpoint": "/var/lib/docker/volumes/postgres_dbdata/_data",
    "Name": "postgres_dbdata",
    "Options": null,
    "Scope": "local"
  }
]

```

② ports で指定したポートフォワーディングが有効になっているのは、docker ps コマンドで表示される「PORTS」で確認できます

```

C:\Tools\postgres>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                               NAMES
26ce741ae938  postgres:jp   "docker-entrypoint.s..." 12 minutes ago  Up 12 minutes  0.0.0.0:5432->5432/tcp, :::5432->5432/tcp  postgres-postgres-1

```

③ PostgreSQL のコンテナは docker-compose.yaml に指定した POSTGRES_USER のデータベースが作られ、POSTGRES_PASSWORD に指定したパスワードでデータベースに接続できます。

```

docker exec -it postgres:jp bash
1977f9792451:/# psql admin admin
psql (15.2)
Type "help" for help.

admin=# \l

          List of databases
  Name  | Owner  | Encoding | Collate  | Ctype    | ICU Locale | Locale Provider | Access privileges
-----+-----+-----+-----+-----+-----+-----+-----
 admin  | admin  | UTF8     | ja_JP.utf8 | ja_JP.utf8 |             | libc              |
 postgres  | admin  | UTF8     | ja_JP.utf8 | ja_JP.utf8 |             | libc              |
 template0  | admin  | UTF8     | ja_JP.utf8 | ja_JP.utf8 |             | libc              | =c/admin,+
 template1  | admin  | UTF8     | ja_JP.utf8 | ja_JP.utf8 |             | libc              | =c/admin,+
(4 rows)

admin=#

```

コンテナ仮想化 (Docker互換 - Rancher Desktop)

7. 同一イメージから複数コンテナの生成

コンテナは docker-compose.yaml を別のフォルダにコピーし、ports のホスト側 (左) を重複しない値に書き替えるだけでコンテナを生成させることができます。

```
ports:
  - "nnnn:5432"
```

<ホストのポート> : <コンテナ・アプリのポート>

```

C:\> コマンドプロンプト
C:\Tools\postgres-c2> docker-compose up -d
[+] Running 2/2
 - Network postgres-c2_default      Created           0.5s
 - Container postgres-c2-postgres-1 Started           6.2s

C:\Tools\postgres-c2> docker volume ls
DRIVER      VOLUME NAME
local       postgres-c2_dbdata
local       postgres_dbdata
  
```

同一イメージから生成したコンテナ名

コンテナの名前はデフォルトでは、docker-compose.yaml の格納フォルダ名が先頭に付きます。

postgres-c2 という名前前のフォルダに格納した場合は、"postgres-c2-" がイメージ名 (タグ除く) の先頭に付加されたコンテナ名になります。

コマンドプロンプト

```

C:\Tools\postgres-c2> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
7c505658ccb9   postgres:jp "docker-entrypoint.s..." 2 minutes ago Up 2 minutes
c2a2b59f2487   postgres:jp "docker-entrypoint.s..." 3 hours ago   Up 3 hours
  
```

```

STATUS        PORTS
Up 2 minutes  0.0.0.0:5433->5432/tcp, :::5433->5432/tcp
Up 3 hours    0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
NAMES
postgres-c2-postgres-1
postgres-postgres-1
  
```

8. コンテナ上のサービス利用

コンテナ上のサービスは docker-compose.yaml の ports で指定したポート経由で利用します。

※この例では port パラメータを省略してホストの 5432 ポート (Postgres のデフォルト) から接続

```

C:\Tools\postgres> docker-compose up -d
[+] Running 2/2
 - Network postgres_default      Created           0.5s
 - Container postgres-postgres-1 Started           5.6s

C:\Tools\postgres> psql -q admin admin
ユーザ admin のパスワード:
admin=#
  
```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

9. コンテナの状態と停止・削除

(1) docker-compose

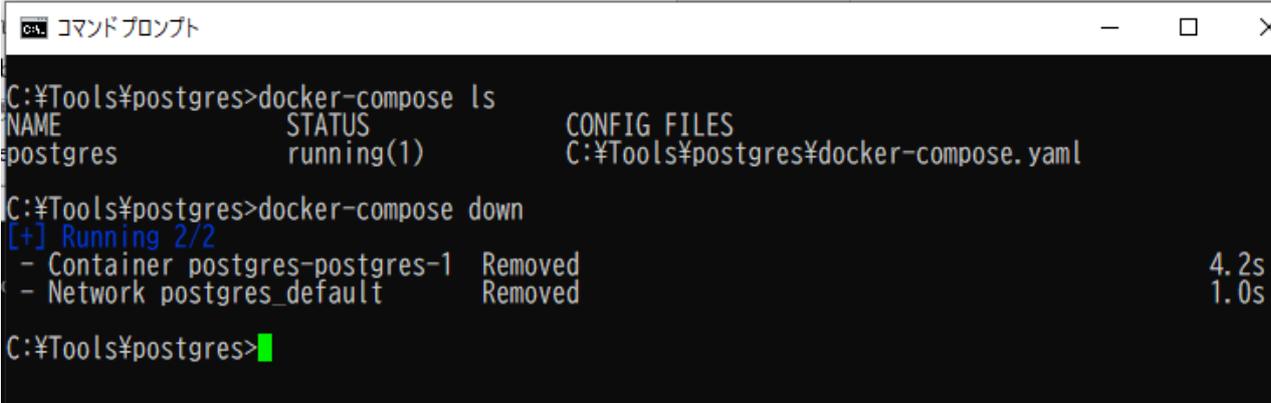
docker-compose up で生成・開始したコンテナは docker-compose のサブコマンドで監視・操作します。

リスト表示：docker-compose ls

停止・削除：docker-compose down

※down をオプションなしで実行すると削除されるのはコンテナとネットワークです。

docker-compose up で生成されたコンテナは docker-compose down で削除されます



```

C:\Tools\postgres>docker-compose ls
NAME                STATUS              CONFIG FILES
postgres            running(1)          C:\Tools\postgres\docker-compose.yaml

C:\Tools\postgres>docker-compose down
[+] Running 2/2
 - Container postgres-postgres-1   Removed          4.2s
 - Network postgres_default        Removed          1.0s

C:\Tools\postgres>

```

(2) docker ps -a/stop/rm

docker run コマンドで生成・起動したコンテナは、docker stop で停止します。

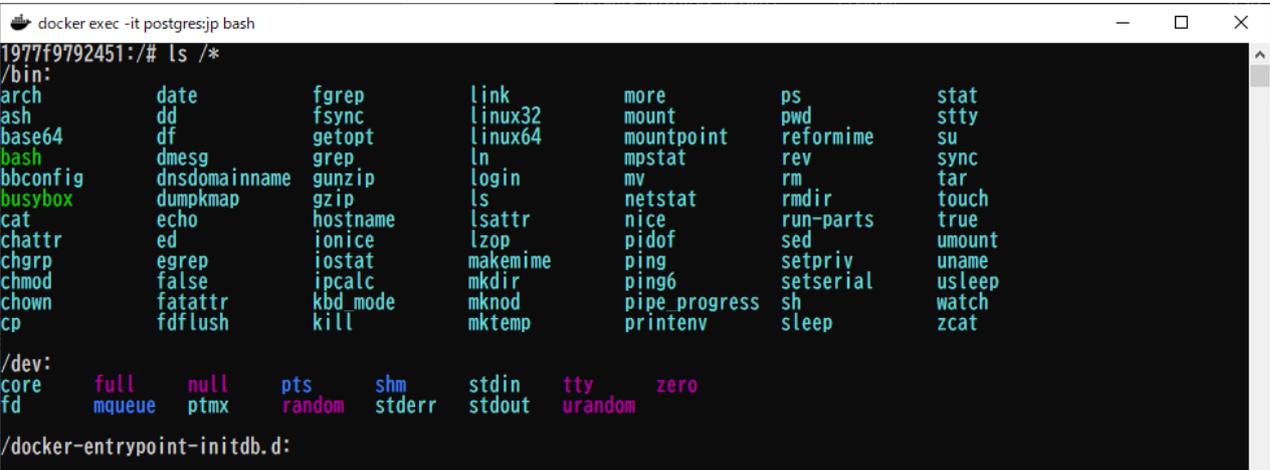
docker rm ID or NAMES でコンテナを消去すると docker ps -a で表示されなくなります

10. コンテナへのターミナル接続

docker-compose up の実行時に表示された Container 名を指定して docker exec -it コマンドで接続できます。

`start "docker exec -it postgres:jp bash"` `docker exec -it postgres-postgres-1 bash`

注) コマンド例の前半[`start "docker exec -it postgres:jp bash"`]は新しい画面を開くためのもので、docker exec コマンドとは直接関係はありません



```

docker exec -it postgres:jp bash
1977f9792451:/# ls /*
/bin:
arch          date          fgrep         link          more          ps            stat
ash           dd            fsync        linux32       mount         pwd           stty
base64        df            getopt       linux64       mountpoint    reformime     su
bash          dmesg         grep          ln            mpstat        rev           sync
bbconfig     dnsdomainname gunzip        login         mv            rm            tar
busybox       dumpkmap      gzip          ls            netstat       rmdir        touch
cat           echo          hostname     lsattr        nice          run-parts    true
chattr        ed            ionice        lzop          pidof         sed           umount
chgrp         egrep         iostat       makemime     ping          setpriv      uname
chmod         false         ipcalc       mkdir         ping6         setserial    usleep
chown         fatattr       kbd_mode     mknod        pipe_progress sh            watch
cp            fdflush       kill          mktemp        printenv     sleep         zcat

/dev:
core          full          null          pts           shm           stdin        tty           zero
fd            mqueue       ptmx         random        stderr        stdout       urandom

/docker-entrypoint-initdb.d:

```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

11. ホストとコンテナのネットワーク

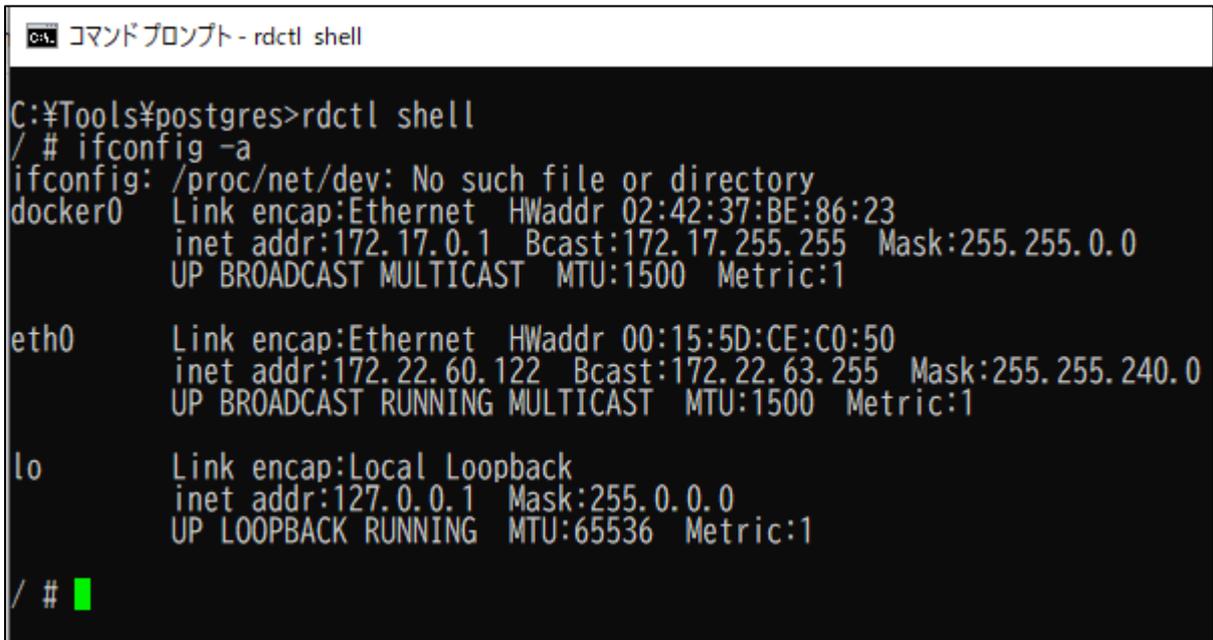
ネットワークに関連する機能は Windows OS も Docker、Rancher コンテナ側も更改を続けている¹⁰¹¹¹²のでこの資料ではデフォルト設定に絞って記述しますが、Rancher Desktop 1.8.1 を Windows 10 Pro 22H2 以降の環境では変わっている可能性があります。

デフォルトでは WSL2 が作成した“イーサネット アダプター vEthernet (WSL)” (仮想スイッチ) を通してコンテナのネットワークとホスト側のネットワークを接続する構成になります。

11.1. ゲートウェイ

コンテナはゲートウェイを経由して仮想スイッチと繋がります。コンテナは固有の IP アドレスとゲートウェイの IP アドレスを持っています。ゲートウェイは `rdctl shell` コマンド (Rancher Desktop に同梱) で Rancher Desktop の vm に接続すると確認できます。

起動直後 (コンテナが起動していない) のネットワーク・インタフェースは以下になっています。



```

C:\Tools\postgres>rdctl shell
/ # ifconfig -a
ifconfig: /proc/net/dev: No such file or directory
docker0  Link encap:Ethernet  HWaddr 02:42:37:BE:86:23
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1

eth0     Link encap:Ethernet  HWaddr 00:15:5D:CE:C0:50
          inet addr:172.22.60.122  Bcast:172.22.63.255  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo       Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1

/ #
  
```

¹⁰ Linux Containers on Windows [https://learn.microsoft.com/ja-](https://learn.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/advanced#linux-containers-on-windows)

[jp/virtualization/windowscontainers/container-networking/advanced#linux-containers-on-windows](https://learn.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/advanced#linux-containers-on-windows)

¹¹ Moby Linux VM では、Docker for Windows (Docker CE の製品) と共に DockerNAT スイッチを使用しています <https://learn.microsoft.com/ja-jp/virtualization/windowscontainers/container-networking/advanced#moby-linux-vm-use-dockernat-switch-with-docker-for-windows-a-product-of-docker-ce>

¹² Windows Containers side by side with Rancher Desktop - for windows containers implementation ideas #3999 <https://github.com/rancher-sandbox/rancher-desktop/issues/3999>

Rancher Desktop issues (ネットワーク関連の作業中の課題) <https://github.com/rancher-sandbox/rancher-desktop/issues?q=is%3Aissue+is%3Aopen+network>

コンテナ仮想化 (Docker互換 – Rancher Desktop)

docker-compose で起動すると、172.18.1, 172.19.0.1, … と順次にインタフェースが追加され、✓

```

C:\Tools\postgres>rdctl shell
/ # ifconfig -a
ifconfig: /proc/net/dev: No such file or directory
br-90a56fcab337 Link encap:Ethernet HWaddr 02:42:72:E5:5F:56
    inet addr:172.18.0.1 Bcast:172.18.255.255 Mask:255.255.0.0
    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

br-e85192c6da49 Link encap:Ethernet HWaddr 02:42:60:6C:AF:16
    inet addr:172.19.0.1 Bcast:172.19.255.255 Mask:255.255.0.0
    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

docker0 Link encap:Ethernet HWaddr 02:42:37:BE:86:23
    inet addr:172.17.0.1 Bcast:172.17.255.255 Mask:255.255.0.0
    UP BROADCAST MULTICAST MTU:1500 Metric:1

eth0 Link encap:Ethernet HWaddr 00:15:5D:CE:C0:50
    inet addr:172.22.60.122 Bcast:172.22.63.255 Mask:255.255.240.0
    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

lo Link encap:Local Loopback
    inet addr:127.0.0.1 Mask:255.0.0.0
    UP LOOPBACK RUNNING MTU:65536 Metric:1

/ #
  
```

コンテナからの下り通信はデフォルトゲートウェイ 172.22.48.1 [vEthernet(WSL)] に送られます。

```

C:\Tools\postgres>rdctl shell
default via 172.22.48.1 dev eth0
172.17.0.0/16 dev docker0 scope link src 172.17.0.1
172.18.0.0/16 dev br-90a56fcab337 scope link src 172.18.0.1
172.19.0.0/16 dev br-e85192c6da49 scope link src 172.19.0.1
172.22.48.0/20 dev eth0 scope link src 172.22.60.122

/ #
  
```

11.2. コンテナからホスト OS

コンテナ側からホストにアクセスするには、以下の2つのホスト名を使うことができます。

host.rancher-desktop.internal
host.docker.internal

コンテナ内で上記ホスト名を照会 (nslookup) すると下図のように表示されます。

Server : 127.0.0.11 は Docker が固定で使っている DNS サーバのアドレスで、DNS が返してきている Address : 172.22.48.1 は WSL 仮想スイッチ [vEthernet (WSL)] の IP アドレスで再起動により変動します。※nslookup のパラメータを host.docker.internal に変えても同一の値が返ってきます

```

選択 docker exec -it postgres:jp bash
755661b16b21:/# nslookup host.rancher-desktop.internal
Server:      127.0.0.11
Address:     127.0.0.11:53

Non-authoritative answer:
Name:   host.rancher-desktop.internal
Address: 172.22.48.1
  
```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

ホスト OS のコマンドプロンプトで powershell;Get-NetIPConfiguration -InterfaceAlias "*WSL*" を実行すると、WSL2 が構成した仮想スイッチが確認できます。コンテナはここを經由して外部ネットと通信します。(Hyper-V マネージャーから仮想スイッチマネージャーを起動しても表示されます)

```

C:\Users\User>powershell;Get-NetIPConfiguration -InterfaceAlias "*WSL*"

InterfaceAlias      : vEthernet (WSL)
InterfaceIndex      : 48
InterfaceDescription : Hyper-V Virtual Ethernet Adapter #2
IPv4Address         : 172.22.48.1
IPv6DefaultGateway  :
IPv4DefaultGateway  :
DNSServer           : fec0:0:0:ffff::1
                   : fec0:0:0:ffff::2
                   : fec0:0:0:ffff::3
  
```

※ DNSServer に IPv6 で初期値 (実在しません) が入っていますが、問題ありません (以降に例)

docker-compose up コマンドで生成された (Network ネットワーク名 Created) ネットワークは docker network ls で表示でき設定値は docker network inspect **ネットワーク名** で確認できます。

下図の例では、コンテナの IP アドレス :172.18.0.2(IPv4 のみ設定)、

デフォルトゲートウェー:172.18.0.1 に設定されていることが分かります。

```

C:\Tools\postgres>docker network inspect postgres default
[
  {
    "Name": "postgres_default",
    "Id": "270da7b19e19504dacbbaab50c0210a246108e4c3f6847cd15cf2682d1ad35c7",
    "Created": "2023-05-18T00:00:01.4942311Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "ac7730baee4fa43d79bdecd9c87f23b80a6d565d34b7c8834c2173530cc5dc9e": {
        "Name": "postgres-postgres-1",
        "EndpointID": "f0cbf57d7e65204571586e7e3cfff7bcbe11fc1ed127547dcc5b7dc7a75513dec",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "default",
      "com.docker.compose.project": "postgres",
      "com.docker.compose.version": "2.16.0"
    }
  }
]
  
```

コンテナ仮想化 (Docker互換 - Rancher Desktop)

Rancher Desktop はデフォルトで、コンテナのプライベートネットワークを作り Host 側とは仮想スイッチの“イーサネット アダプター vEthernet (WSL)”経由で接続します。

vEthernet (WSL) は NAT¹³の機能を持っているので意識せずに外部ネットワーク / インターネット上のサーバに接続し、名前解決や http アクセス等ができます。

下図の例は、インターネット上のサーバ focs.co.jp に接続する経路を見たときの出力です。

172.18.0.1 (デフォルトゲートウェイ) ⇒ 172.22.48.1 (仮想スイッチ)

⇒ 192.168.11.1 Host OS が接続しているネットワーク

```
docker exec -it postgres:jp bash
755661b16b21:/#
755661b16b21:/# traceroute focs.co.jp
traceroute to focs.co.jp (59.106.171.42), 30 hops max, 46 byte packets
 1 172.18.0.1 (172.18.0.1)  0.032 ms  0.019 ms  0.012 ms
 2 172.22.48.1 (172.22.48.1)  0.696 ms  0.416 ms  0.460 ms
 3 192.168.11.1 (192.168.11.1)  5.836 ms  2.223 ms  2.097 ms
```

11.3. Host OS からコンテナ

Host OS からコンテナのサービスに接続する仕組みとしては以下のものがあります。

- ① ポートフォワーディングでコンテナのポートと Host OS のポートを接続する
- ② `docker exec コンテナ名 コマンド` でコマンドの結果 (stdout/err) をコマンドプロンプトへ
- ③ `docker exec -it コンテナ名` で `bash` 等に接続する (ターミナルで連続した操作ができます)

【②の実用例】 コンテナ内で実行した `top` コマンドの結果を Host OS 上に表示しています

```
cmd コマンドプロンプト - docker exec postgres-postgres-1 top
Mem: 680368K used, 5711348K free, 14280K shrd, 7492K buff, 433532K cached
CPU:  0% usr  0% sys  0% nic 99% idle  0% io  0% irq  0% sirq
Load average: 0.00 0.00 0.00 1/187 47
  PID  PPID  USER    STAT  VSZ %VSZ  CPU %CPU  COMMAND
   27   1 postgres S      169m  3%    1    0% postgres: autovacuum launcher
   28   1 postgres S      169m  3%    2    0% postgres: logical replication laun
   23   1 postgres S      167m  3%    2    0% postgres: checkpointer
   24   1 postgres S      167m  3%    2    0% postgres: background writer
   26   1 postgres S      167m  3%    3    0% postgres: walwriter
    1   0 postgres S      167m  3%    3    0% postgres
   41   0 root    R      1596  0%    1    0% top

C:\Tools\postgres>docker exec postgres-postgres-1 ps
PID  USER  TIME  COMMAND
  1  postgres  0:00 postgres
 23  postgres  0:00 postgres: checkpointer
 24  postgres  0:00 postgres: background writer
 26  postgres  0:00 postgres: walwriter
 27  postgres  0:00 postgres: autovacuum launcher
 28  postgres  0:00 postgres: logical replication launcher
 41  root     0:00 top
 48  root     0:00 ps

C:\Tools\postgres>docker exec postgres-postgres-1 kill 41

C:\Tools\postgres>docker exec postgres-postgres-1 ps
PID  USER  TIME  COMMAND
  1  postgres  0:00 postgres
 23  postgres  0:00 postgres: checkpointer
 24  postgres  0:00 postgres: background writer
 26  postgres  0:00 postgres: walwriter
 27  postgres  0:00 postgres: autovacuum launcher
 28  postgres  0:00 postgres: logical replication launcher
 60  root     0:00 ps
```

【注意】

top コマンドのように停止操作 (q 打鍵等) が必要なコマンドは `ctrl+c` で切断後、`ps` コマンドで PID を確認して `kill pid` コマンドで終了させてください。プロセスが生き続けて資源の消費が続きます。

¹³ Hyper-V ホストの管理 - NAT ネットワークの設定 <https://learn.microsoft.com/ja-jp/virtualization/hyper-v-on-windows/user-guide/setup-nat-network>

コンテナ仮想化 (Docker互換 - Rancher Desktop)

④ IP アドレスを使った接続 (注意)

コンテナで起動しているサービスはポートフォワーディングで使うのが一番効果的 (手軽で安全) ですが、IP アドレスを使ってサービスにリクエストを送ることもできます。

```

C:\Tools\postgres-c2>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
297b198f0e34   postgres:jp   "docker-entrypoint.s..." 10 minutes ago Up 9 minutes  0.0.0.0:5433->5432/t
2b37f7a723a5   postgres:jp   "docker-entrypoint.s..." 10 minutes ago Up 10 minutes  0.0.0.0:5432->5432/t

C:\Tools\postgres-c2>psql -p 5433 -q admin admin
ユーザ admin のパスワード:
admin=# select inet_server_addr(), inet_server_port();
 inet_server_addr | inet_server_port
-----
172.19.0.2        | 5432
(1 行)
  
```

— コンテナの実 IP アドレスとポート番号

- docker ps でホストのポート 5433 に接続している postgres のコンテナがあることが分かります
- psql でホストの 5433 ポートでデータベースを開きます
- データベースにサーバの IP とポート番号を問い合わせる (select inet_server_addr(), inet_server_port();) と DB サーバは IP アドレス 172.19.0.2、5432 ポートで稼働していることが分かります

— ネットワーク経路(route)の確認 / 存在しない場合に追加

- コンテナの IP アドレスへのルートが登録されていないとポートフォワーディング以外では接続できません。IP アドレスで接続する場合は、管理者権限による root add が必要です

route add サブネットアドレス mask 255.255.0.0 仮想スイッチ(WSL)のアドレス

※下線部は固定、mask 値は docker network inspect ネットワーク名 の Subnet 末尾 "/16" より

```

C:\WINDOWS\system32>route add 172.19.0.0 mask 255.255.0.0 172.22.48.1
OK!

C:\WINDOWS\system32>route print
(中略)

IPv4 ルート テーブル
=====
アクティブ ルート:
ネットワーク宛先      ネットマスク      ゲートウェイ      インターフェイス  メトリック
-----
0.0.0.0                0.0.0.0            192.168.11.1      192.168.11.2      55
127.0.0.0              255.0.0.0          リンク上          127.0.0.1          331
127.0.0.1              255.255.255.255   リンク上          127.0.0.1          331
127.255.255.255       255.255.255.255   リンク上          127.0.0.1          331
172.18.0.0             255.255.0.0       リンク上          172.22.48.1       5001
172.18.255.255        255.255.255.255   リンク上          172.22.48.1       5256
172.19.0.0             255.255.0.0       リンク上          172.22.48.1       5001
172.19.255.255        255.255.255.255   リンク上          172.22.48.1       5256
  
```

【注意】 使用したバージョン (Rancher Desktop 1.8.1 / WSL2 プレイインストール版) では、IP アドレスを使った接続は以下のように **不整合な環境になります** (172.19.0.1 で 172.18.0.2 に接続する)

```

C:\Users\User>psql -h 172.19.0.1 -p 5432 -q admin admin
ユーザ admin のパスワード:
admin=# select inet_server_addr(), inet_server_port();
 inet_server_addr | inet_server_port
-----
172.18.0.2        | 5432
(1 行)
  
```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

11.4. コンテナから Hyper-V 仮想マシン

コンテナと Hyper-V 仮想マシンは各々別の仮想スイッチに繋がってるので、この間で通信を行う場合はパケットが通過できるように設定する必要があります。これは以下の手順で行います。

- ① WSL2 と Hyper-V マネージャーが使っている仮想スイッチの一覧を探します。仮想スイッチは”vEthernet”という文字列を含んでいるので、以下のコマンドで抽出します。

```
powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*'"
```

```

C:\Users\User>powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*'"
ifIndex InterfaceAlias AddressFamily NIMtu(Bytes) InterfaceMetric Dhcp ConnectionState PolicyStore
-----
48 vEthernet (WSL) IPv6 1500 5000 Enabled Connected ActiveStore
29 vEthernet (Default Switch) IPv6 1500 5000 Enabled Connected ActiveStore
48 vEthernet (WSL) IPv4 1500 5000 Disabled Connected ActiveStore
29 vEthernet (Default Switch) IPv4 1500 5000 Disabled Connected ActiveStore
  
```

※ IPv6 と IPv4 のペアになっていますが、IPv6 のアドレスはデフォルトでは設定されていません。

Hyper-V 仮想マシンの eth0 の下から 2 行目に inet6 fe80::~ scope link と IPv6 アドレスが表示されていますが scope link はローカルリンクアドレスでスイッチやルータを越えられません

```

adminuser@ubuntu2004: ~
adminuser@ubuntu2004:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
   link/ether 00:15:5d:0b:05:07 brd ff:ff:ff:ff:ff:ff
   inet 172.24.23.121/20 brd 172.24.31.255 scope global dynamic eth0
       valid_lft 82902sec preferred_lft 82902sec
   inet6 fe80::215:5dff:fe0b:507/64 scope link
       valid_lft forever preferred_lft forever
adminuser@ubuntu2004:~$
  
```

したがって、コンテナから ping コマンドを実行しても届きません。

(下図の ubuntu2004 は Hyper-V 仮想マシン。名前解決は成功しても ping は届いていません)

```

C:\Tools\postgres>docker exec postgres-postgres-1 ping -c 3 ubuntu2004
PING ubuntu2004 (172.24.23.121): 56 data bytes

--- ubuntu2004 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
  
```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

② 仮想スイッチ (vEthernet) の Forwarding を Enabled にします

管理者権限で Powershell を起動し、名前が vEthernet で始まるネットワークインタフェースを選んで Set-NetIPInterface コマンドレットを使って変更します。

```
powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*' |Set-NetIPInterface -Forwarding Enabled"
```



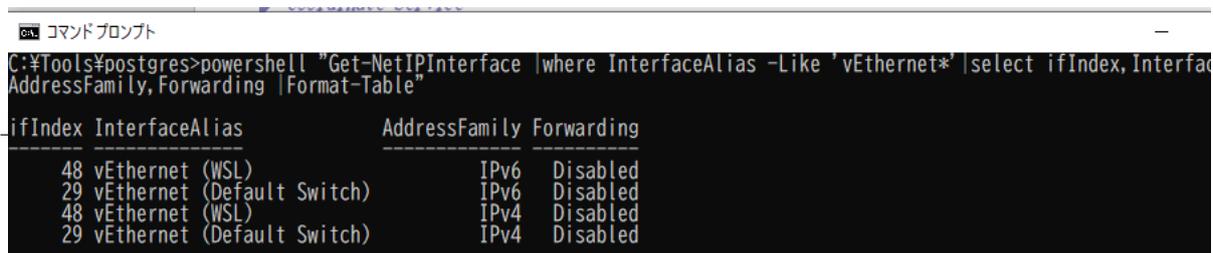
```
管理: コマンドプロンプト
C:\WINDOWS\system32>powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*' |Set-NetIPInterface -Forwarding Enabled"
```

【Set-NetIPInterface -Forwarding Enabled の実行前後の状態確認】

以下のコマンドで Forwarding の値を確認します

```
powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*' |select ifIndex, InterfaceAlias, AddressFamily, Forwarding |Format-Table"
```

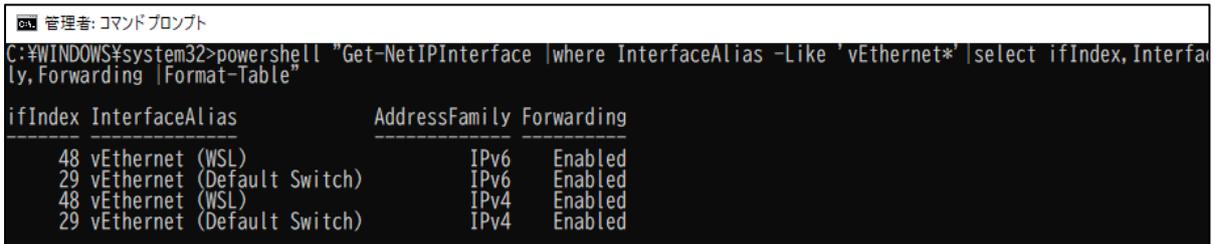
<初期状態>



```
管理: コマンドプロンプト
C:\Tools\postgres>powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*' |select ifIndex, InterfaceAlias, AddressFamily, Forwarding |Format-Table"

ifIndex InterfaceAlias                AddressFamily Forwarding
-----
48 vEthernet (WSL)                    IPv6         Disabled
29 vEthernet (Default Switch)        IPv6         Disabled
48 vEthernet (WSL)                    IPv4         Disabled
29 vEthernet (Default Switch)        IPv4         Disabled
```

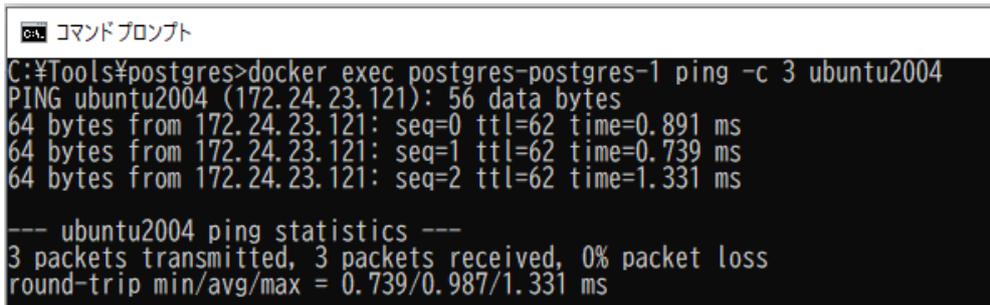
<変更後>



```
管理: コマンドプロンプト
C:\WINDOWS\system32>powershell "Get-NetIPInterface |where InterfaceAlias -Like 'vEthernet*' |select ifIndex, InterfaceAlias, AddressFamily, Forwarding |Format-Table"

ifIndex InterfaceAlias                AddressFamily Forwarding
-----
48 vEthernet (WSL)                    IPv6         Enabled
29 vEthernet (Default Switch)        IPv6         Enabled
48 vEthernet (WSL)                    IPv4         Enabled
29 vEthernet (Default Switch)        IPv4         Enabled
```

Forwarding Enabled 後に ① と同じように ping を実行してみると、今回は成功します



```
管理: コマンドプロンプト
C:\Tools\postgres>docker exec postgres-postgres-1 ping -c 3 ubuntu2004
PING ubuntu2004 (172.24.23.121): 56 data bytes
64 bytes from 172.24.23.121: seq=0 ttl=62 time=0.891 ms
64 bytes from 172.24.23.121: seq=1 ttl=62 time=0.739 ms
64 bytes from 172.24.23.121: seq=2 ttl=62 time=1.331 ms

--- ubuntu2004 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.739/0.987/1.331 ms
```

※ 但し、仮想スイッチの状態変更は再起動により変更前に戻ります。

コンテナ仮想化 (Docker互換 – Rancher Desktop)

12. WSL2 とコンテナの注意点

WSL2 は 2023 年 5 月末時点で、9 世代のプレリリース版が並列管理され GitHub で公開され¹⁴ていて 2000 件近い作業中の管理項目(issues)が存在します。Docker (や代替え製品) を Windows で使う場合、コンテナ間やホスト、他 VM との通信環境として WSL を使いますが issues の数を見ても先々仕様が変わっていくことが予測されます。

特に、NAT や名前解決を実現する仮想スイッチは、以前から Windows に備わっていた ICS(Internet Connection Sharing)や、Hyper-V が利用する HNS(Host Networking Service)を組み合わせで実現している¹⁵ようですが、環境作成時に ICS の設定が壊れることがあります。

【HNS のネットワーク情報】 ※情報参照には管理者権限で hnsdiag list networks を実行
Type:ICS と表示され、ICS を参照していることが確認できる

```

管理: コマンドプロンプト
C:\WINDOWS\system32>hnsdiag list networks
Network : C08CB7B8-9B3C-408E-8E30-5E16A3AEB444
  Name      : Default Switch
  Type      : ICS
  Subnet Address : 172.24.16.0/20
  Gateway   : 172.24.16.1

Network : B95D0C5E-57D4-412B-B571-18A81A16E005
  Name      : WSL
  Type      : ICS
  Subnet Address : 172.22.48.0/20
  Gateway   : 172.22.48.1
  
```

【hosts.ics】

ICS が名前解決に使っている C:\Windows\System32\drivers\etc\hosts.ics が仮想スイッチの登録時に IP アドレス(172.24.16.1 と 172.22.48.1)とホスト名の書き込みでフォーマットが壊れました

```

hosts.ics - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
# Copyright (c) 1993-2001 Microsoft Corp.
##
## This file has been automatically generated for use by Microsoft Internet
## Connection Sharing. It contains the mappings of IP addresses to host names
## for the home network. Please do not make changes to the HOSTS.ICS file.
## Any changes may result in a loss of connectivity between machines on the
## local network.
##
172.24.16.1 WIN-██████████.mshome.net #
48.1 WIN-██████████.mshome.net #
9
.16.1 WIN-██████████.mshome.net # 2027 4 3 21 3 4 32 346
  
```

IPv4 の形式が壊れ、ホスト名が全て同じ値になっている

¹⁴ microsoft/WSL <https://github.com/microsoft/WSL/releases>

¹⁵ ICS が無効になっている場合の WSL 2 エラー <https://learn.microsoft.com/ja-jp/windows/wsl/troubleshooting#wsl-2-errors-when-ics-is-disabled>

コンテナ仮想化 (Docker互換 - Rancher Desktop)

hosts.ics が壊れた場合 Hyper-V の vm で名前解決が正常にできなくなる可能性があります、今回使った環境は vm に Samba を入れてホスト名を公開していたので、Docker コンテナからも vm ホスト名の解決ができています。

次に、Rancher Desktop のネットワークに関しては、docker run で生成したものと docker-compose で生成したものではデフォルトで使われるネットワークが異なる (bridge と コンテナ名_default) ようですが、どちらもブリッジとサービスのペアになります (例 172.26.0.1 と 172.26.0.2)。

```

C:\> コマンドプロンプト

C:\> C:\Tools\postgres-c2>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
6742493ac92f        bridge              bridge              local
bb7c41f36432        host                host                local
439dc2f09586        none                null                local
fc0ac008584c        postgres-c2_default bridge              local
69490e0c9b17        postgres_default   bridge              local
  
```

<サービスの IP アドレスでアクセスできません>

docker exec -it コンテナ名 bash で接続する IP アドレスは n.n.n.2 ですが、このアドレスはホストから見つけることはできず、n.n.n.1 を指定することで接続できます。

```

C:\> コマンドプロンプト - docker exec -it postgres-c2-postgres-1 bash

C:\> C:\Tools\postgres-c2>docker exec -it postgres-c2-postgres-1 bash
dbc2-server:/# ip route
default via 172.26.0.1 dev eth0
172.26.0.0/16 dev eth0 scope link src 172.26.0.2
dbc2-server:/#
  
```

<ホスト OS 上の IP ルートは以下のコマンドで確認できます>

powershell "Get-NetRoute |where InterfaceAlias -like 'vEthernet*' |where AddressFamily -eq 'IPv4' |Sort InterfaceAlias |select ifIndex,InterfaceAlias,AddressFamily,DestinationPrefix"

```

C:\> コマンドプロンプト

C:\> C:\Tools\postgres-c2>powershell "Get-NetRoute |where InterfaceAlias -like 'vEthernet*' |where AddressFamily -eq 'IPv4' |Sort InterfaceAlias |select ifIndex,InterfaceAlias,AddressFamily,DestinationPrefix"

ifIndex InterfaceAlias                AddressFamily DestinationPrefix
-----
10 vEthernet (Default Switch)          IPv4 224.0.0.0/4
10 vEthernet (Default Switch)          IPv4 172.17.176.1/32
10 vEthernet (Default Switch)          IPv4 172.17.191.255/32
10 vEthernet (Default Switch)          IPv4 255.255.255.255/32
10 vEthernet (Default Switch)          IPv4 172.17.176.0/20
48 vEthernet (WSL)                     IPv4 172.18.255.255/32
48 vEthernet (WSL)                     IPv4 172.18.0.0/16
48 vEthernet (WSL)                     IPv4 255.255.255.255/32
48 vEthernet (WSL)                     IPv4 172.19.0.0/16
48 vEthernet (WSL)                     IPv4 172.22.111.255/32
48 vEthernet (WSL)                     IPv4 224.0.0.0/4
48 vEthernet (WSL)                     IPv4 172.22.96.1/32
48 vEthernet (WSL)                     IPv4 172.19.255.255/32
48 vEthernet (WSL)                     IPv4 172.22.96.0/20
  
```

コンテナ仮想化 (Docker互換 – Rancher Desktop)

13. 使い方

WSL 2 と Rancher Desktop がデフォルトで作るネットワークが不可解だったので、同様に疑問を感じた人が自分で確認できるようにコマンド例をあげておきました。ネットワーク関係はコード内に直書きされている部分があったり、今後の改善や機能追加でこの資料とは変わっていく可能性があるの
で動きが変わったと感じたら、コマンドで環境を確認してみてください。

ただ、ネットワークの構成は解り辛くともデフォルトで動作する環境が作られますし、基本的な使
い方の範囲 (項番 9 迄) では多量のコマンドを覚えなくても十分に利用できます。インターネットに公
開されているコンテナのイメージを使えば一定の動作確認が済んだ、モノによっては商用製品レベル
の環境を簡単に作ることができるので、是非試してください。

以上