# 社会科学ためのDocker入門

*レプリケーション駆動な研究のために*

柳本和春

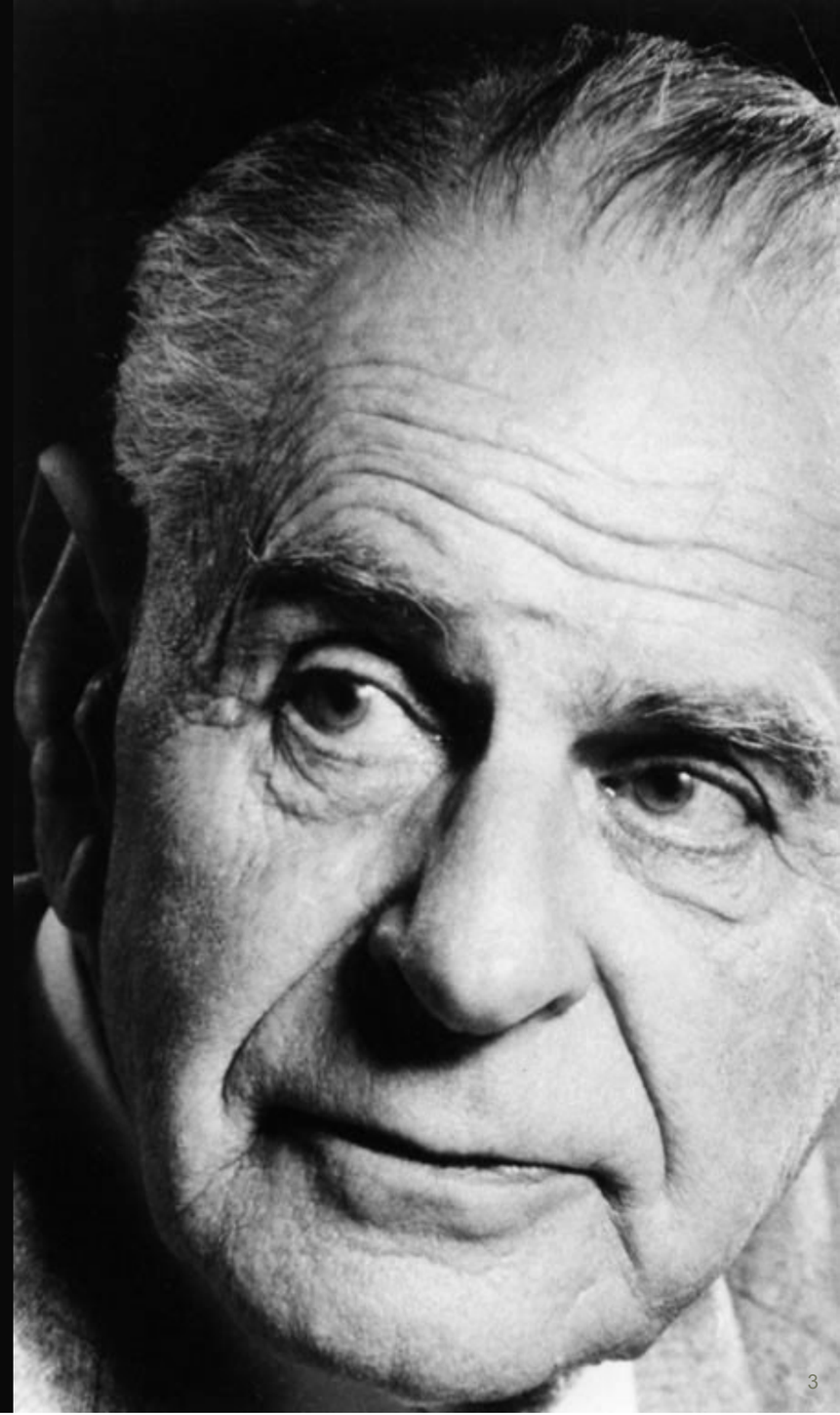
*kazuharu.yanagimoto@cemfi.edu.es*

*CEMFI*

August 31, 2023

# 検証可能性 *Testability*

> “I shall certainly admit a system as empirical or scientific only if it is capable of being tested by experience.”
>
> –Karl Popper, 1934



# レプリケーション駆動な研究

*研究が社会**科学**足るために再現性はその要件である*

- 研究が検証可能であるためには, 研究者はその再現性を保証しなくてはいけない
- AER, Data and Code Availability Policy

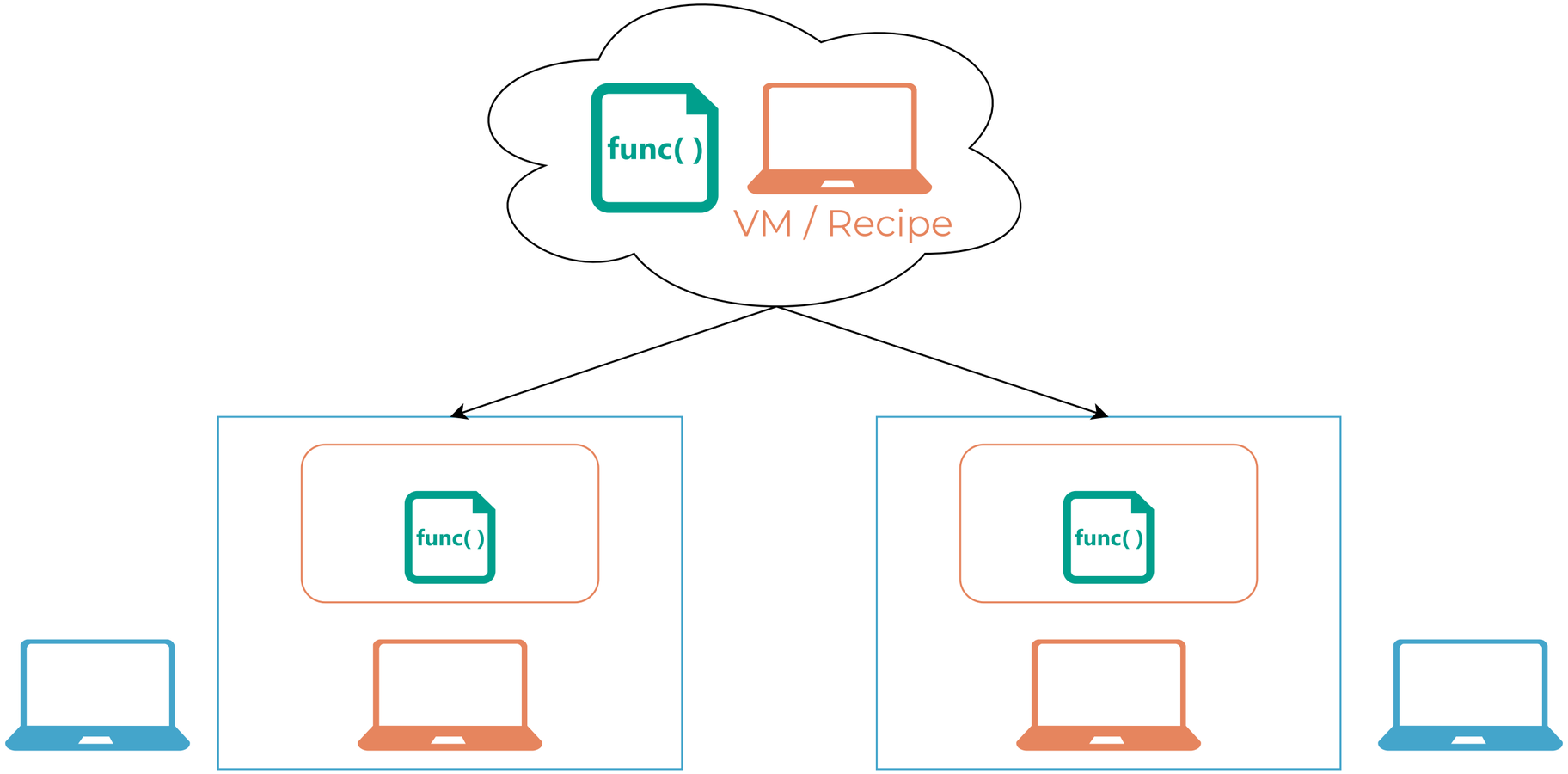
*レプリケーションは研究の**前提**であり, **過程**であり, **ゴール**である*

- 研究の過程においてレプリケーション可能な操作のみを行うべき
- そして研究者間のコミュニケーションコストを下げ, 研究を促進する

# なぜ彼のコードは私のPCで動かないのか?

- **パスが異なる**
  → 常にパスを`here::here()`で対応可能. Pythonの場合 `pyprojroot.here()`
- **パッケージのバージョンが異なる**
  → `renv`を用いてバージョンを記録する. Pythonの場合 `venv`, `poetry` など
- **Rのバージョンが異なる**
  → `renv`はRのバージョンを切り替えることができないので, 手動で切り替えるか常に最新版を用いるなどで対応
  → Pythonの場合は各種パッケージ管理ツールで対応可能
- **OSが異なる**
  → 特定のOSに特有のバグなどが発生しうる

# バーチャルマシンとレプリケーション

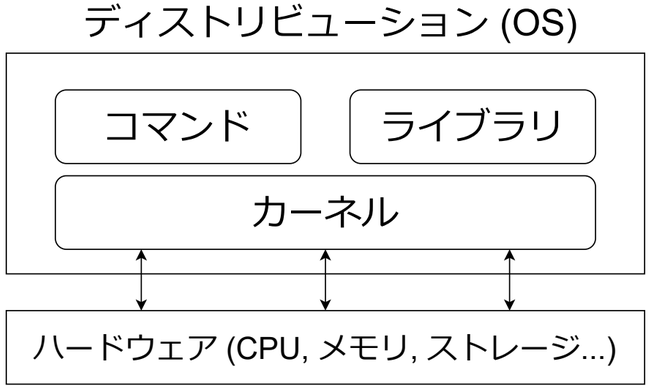



- コードと同時にVMを配布することで, 環境の違いを解消できる
- しかしVMはファイルサイズが大きい, 実行速度が遅いなどの問題があった

# LinuxとDockerの基礎

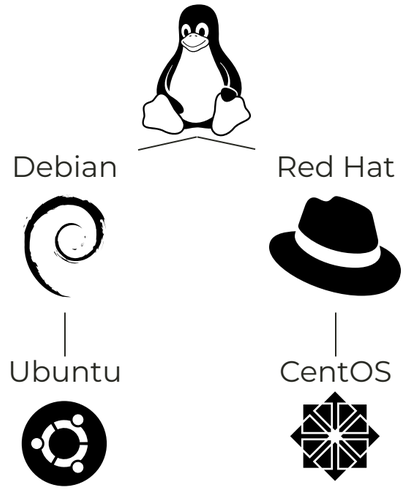
# Linuxとは

- **狭義**: Linuxカーネルのこと
  → OSの基礎的な機能を提供するライブラリ群
- **広義**: Linuxカーネルを採用したOS
  → コマンドやライブラリなどの違いによってディストリビューションに分かれる




## *Linuxディストリビューション*

- 主にDebian系, RedHat系, その他独立系に分かれる
  → **Debian**: Ubuntu, Raspberry Pi
  → **RedHat**: CentOS, Amazon Linux
  → **独立系**: Chrome, Android, Arch, Alpine
- Docker用としてはUbuntuがおすすめ



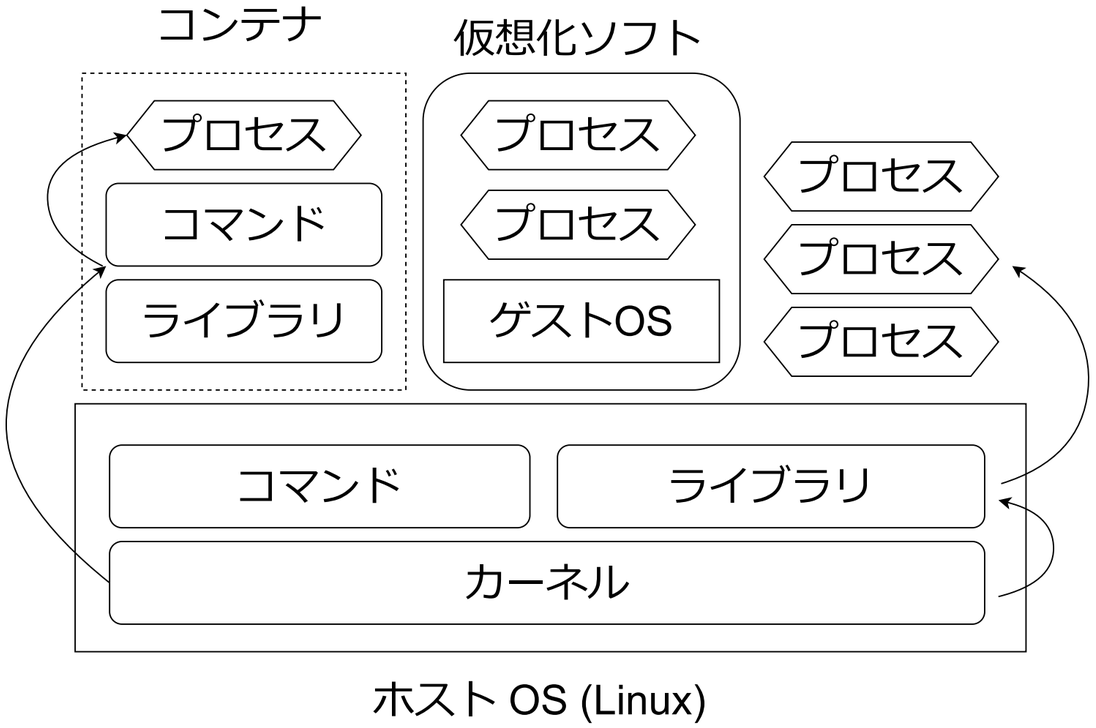
# Dockerの仕組み

## *Docker コンテナ*

- 1つのマシンのように振る舞う仮想環境
- ホストOSとカーネルをシェアする
- ホストもコンテナもLinux OS

## *仮想化ソフトとの違い*

- 仮想化ソフトはホスト上のソフトウェア
- ゲストOSをエミュレートして動作する
- ホストもゲストもLinuxの必要はない
- 仮想化ソフト上のマシンは遅くなる



# イメージとレジストリ

## *Docker イメージ*

- コンテナの元となるテンプレート
  - → OS単体 (ubuntu, alpine)
  - → OS + アプリケーション (rocker/rstudio)
- `USER/IMAGE:TAG` で指定する. `TAG`を省略すると`latest`が指定される

## *Docker レジストリ*

- イメージを保存する場所
- 基本的にはDocker Hubを用いる.
  - → Officialイメージ (ubuntu, rocker, node, jupyter, etc.)
  - → 自作イメージをアップロードもできる
- その他, GitHub Container Registry, Amazon Elastic Container Registry など

# VSCode & Docker

## *なぜVSCodeを使うのか?*

- VSCodeは軽量かつ高機能なエディタ. エディタの現王者といって過言ではない
- Remote Containersの登場によってDocker環境の導入が革命的に簡単になった

## *Handson*

0. (Windows) VSCode上で Ctrl-Shift-P を押して`WSL: Connect to WSL`を選択
1. ハンズオン用のフォルダ (`handson1`) を作成する
2. VSCodeでフォルダを開く. 左端の のアイコンから
3. Docker Desktopが起動していることを確認 ( のマーク)
4. VSCodeでCtrl-Shift-Pを押して`Open Folder in Container`を選択. 作成したフォルダを選び, `Ubuntu`を選択. 残りの選択肢はデフォルトのままでOK
5. `.devcontainer/devcontainer.json`という設定ファイルが作成される

# Dockerfile

- 先程のHandsonは既存のUbuntuイメージを利用した
- 通常は既存のイメージに必要なソフトやパッケージを追加する
- Dockerfileはその設計図. 拡張子がないことに注意

```
1 FROM ubuntu
2
3 ENV DEBIAN_FRONTEND noninteractive
4
5 RUN apt update && apt install -y curl
```

- `FROM`: ベースとなるイメージを指定
- `ENV`: 環境変数を設定. Ubuntuの場合は`DEBIAN_FRONTEND`はこの設定が必要
- `RUN`: シェルコマンドを実行
  - → `apt update`: パッケージリストを更新
  - → `apt install`: パッケージインストール (`-y` はすべてに自動でyesを返す)

# Handson: Remote containers with Dockerfile

1. VSCodeでCtrl-Shift-Pを押して`Reopen Folder in WSL/Locally`を選択
2. `.devcontainer/devcontainer.json`を一度削除する
3. 以下のDockerfileを　(ルートディレクトリ直下に) 作成して保存する

```
FROM ubuntu

ENV DEBIAN_FRONTEND noninteractive

RUN apt update && apt install -y git
```

4. VSCodeでCtrl-Shift-Pを押して`Reopen in Container`を選択

   → Dockerfileを選択する

5. VSCodeの中でターミナルを開いてユーザー名を確認する (`whoami`)

# ユーザとパーミッション

## *ユーザ, グループ, Root*

- Linuxではユーザごとにファイルの権限 (読み取り, 書き込み, 実行) が設定されている
- ユーザはグループに所属する. グループごとに権限を設定することもできる
- rootユーザ (管理者) はすべてのファイルにアクセスできる

## *Dockerとユーザ*

- Dockerでは通常, rootユーザでコンテナが起動する
- Docker上で作業したファイルがroot権限で作成される
- LinuxとWSL2ユーザーはホスト側からファイルにアクセスできなくなる
- 一つの解決策として, Dockerfile内でユーザを作成するという方法がある

# Handson: Remote Containers with a User

1. VSCodeでCtrl-Shift-Pを押して`Reopen Folder in WSL/Locally`を選択
2. 以下のDockerfileを作成する

```
FROM ubuntu

ENV DEBIAN_FRONTEND noninteractive

RUN apt update && apt install -y git

ARG USERNAME=vscode
ARG USER_UID=1000
ARG USER_GID=$USER_UID

RUN groupadd --gid $USER_GID $USERNAME \
    && useradd --uid $USER_UID --gid $USER_GID -m $USERNAME

USER $USERNAME
```

3. `.devcontainer/devcontainer.json`の`remoteUser`を`vscode`に変更する
4. VSCodeでCtrl-Shift-Pを押して`Rebuild and Reopen in Container`を選択
5. ターミナルを開いてユーザー名を確認する (`whoami`)


on Docker

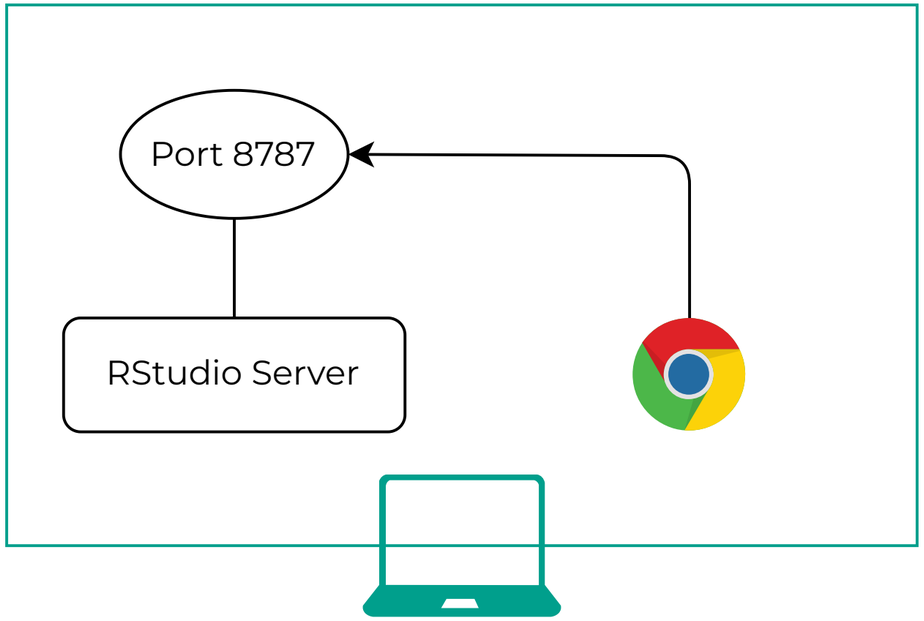
# rockerプロジェクト

- rockerはオフィシャルのRイメージ群
- 私はRのライブラリはキャッシュする (後述) ので, `rocker/rstudio`または`rocker/geospatial`を使うことが多い
- `rstudio`というユーザが用意されている

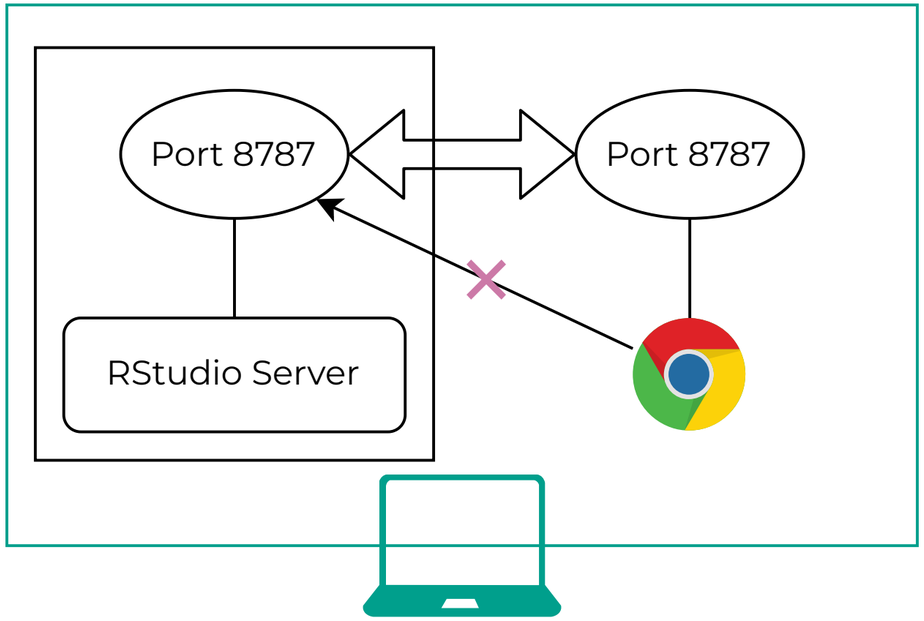
| イメージ | ベースイメージ | 概要 |
|---|---|---|
| rocker/r-ver | ubuntu | Ubuntu + R |
| rocker/rstudio | rocker/r-ver | + RStudio Server |
| rocker/tidyverse | rocker/rstudio | + `tidyverse` & `devtools` |
| rocker/verse | rocker/tidyverse | + `tinytex` & 組版関係のパッケージ |
| rocker/geospatial | rocker/geospatial | + 地理情報用パッケージ |

# ポートフォワーディング

## 通常のRStudio Server



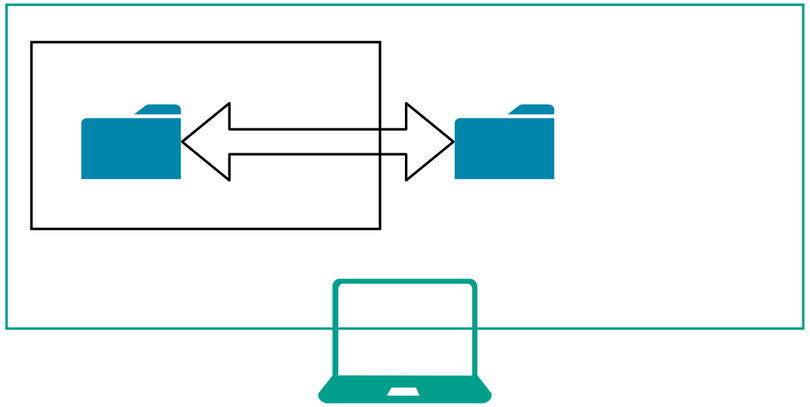

## コンテナ内のRStudio Server



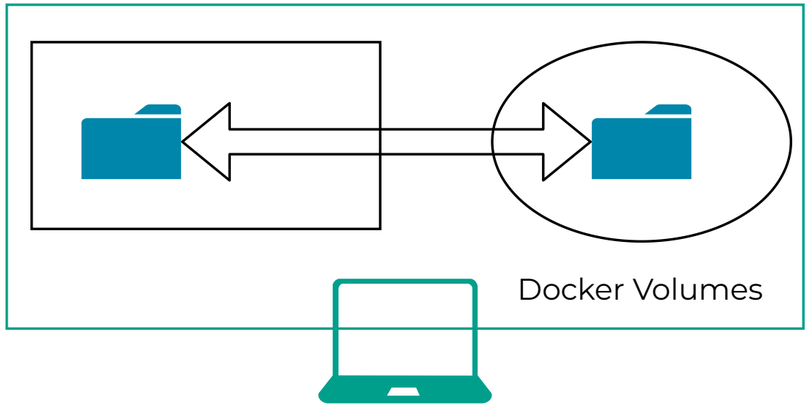

- Docker内のRStudio Serverはホストのブラウザからはアクセスできない
- ポートフォワーディングを設定することで, ホストのポートをDocker内のポートに接続することができる

# マウント

## バインドマウント



## ボリューム




- デフォルトではコンテナが削除されるとファイルも削除される
- バインドマウントはホストのフォルダをコンテナにマウントする
- ボリュームはDockerが管理するフォルダをコンテナにマウントする
  - → ホスト側からは (基本的に) 見えない
  - → コンテナに最適化されたファイルシステムなので, パフォーマンスが良い

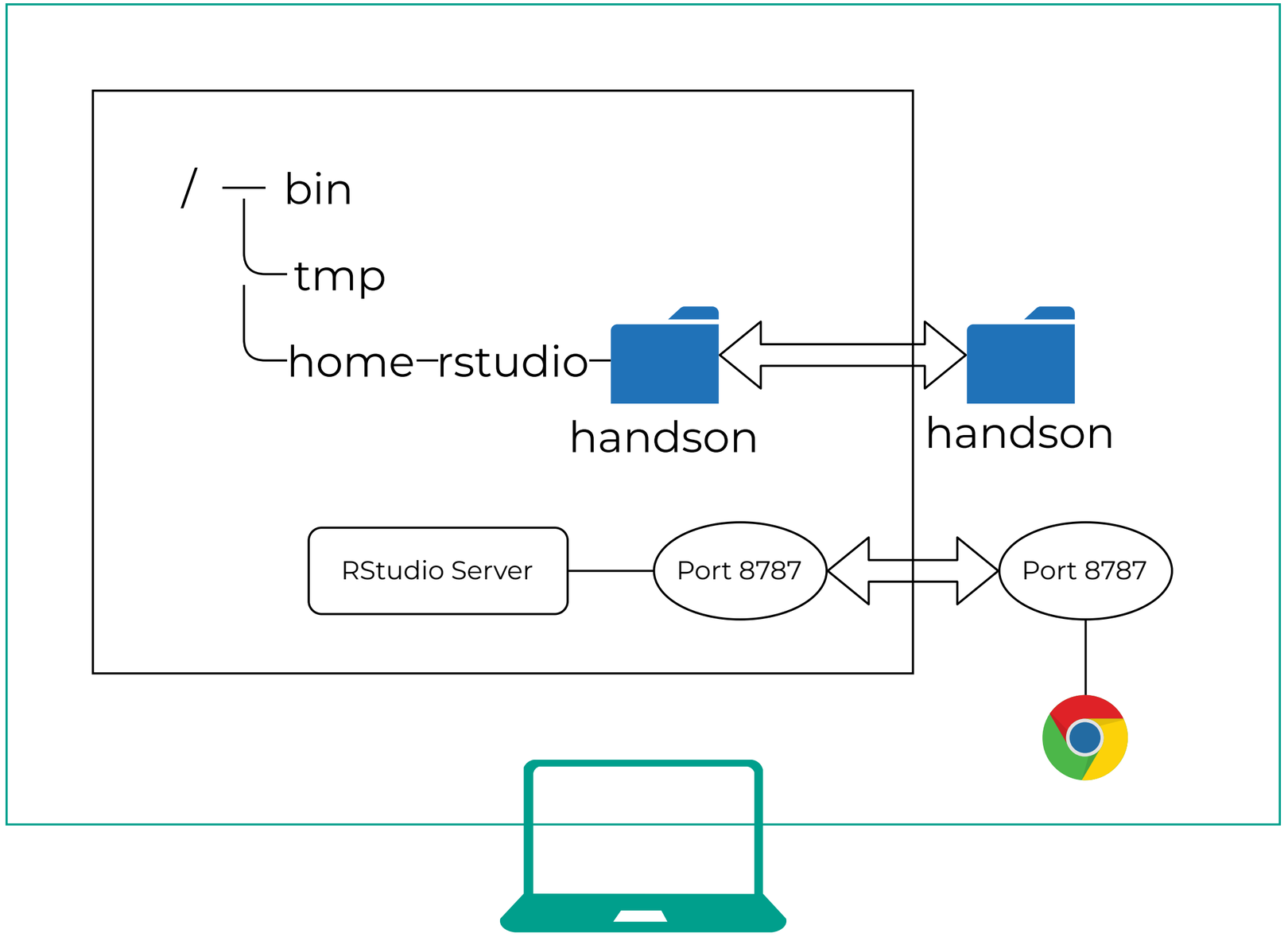
# docker-compose.yml

- `docker-compose.yml`ファイルにコンテナ起動の際の設定を記述することができる
- 複数のコンテナを起動することもできる

```
services:
  rstudio:
    build:
      context: .
    environment:
      - TZ=Asia/Tokyo
      - DISABLE_AUTH=true
    volumes:
      - .:/home/rstudio/handson2
```

- `build context`: `Dockerfile`のパス
- `environment`: 環境変数の設定
  - → `TZ`: タイムゾーン
  - → `DISABLE_AUTH`: RStudioのパスワード認証を無効化
- `volumes`: マウント
  - → HOST_PATH:CONTAINER_PATH
  - → 上記の例はバインドマウント

# Handson: RStudio Server



1. 新しいHandson用ののディレクトリを作成する (handson2)
2. VSCodeで新しく作成したディレクトリを開く
3. Dockerfileを作成しFROM rocker/rstudioと記述し保存
4. 以下のdocker-compose.ymlと.devcontainer/devcontainer.jsonを作成する

```yaml
services:
  rstudio:
    build:
      context: .
    environment:
      - TZ=Asia/Tokyo
      - DISABLE_AUTH=true
    volumes:
      - .:/home/rstudio/handson2
```

```json
{
    "name": "${localWorkspaceFolderBasename}",
    "dockerComposeFile": "../docker-compose.yml",
    "service": "rstudio",
    "remoteUser": "rstudio",
    "forwardPorts": [8787],
    "workspaceFolder": "/home/rstudio/handson2"
}
```

4. VSCodeでCtrl-Shift-PからRebuild and Reopen in Containerを選択する
5. ブラウザでlocalhost:8787にアクセスする

# Rパッケージのインストール

以下のようなDockerfileでコンテナの中にRパッケージをインストールできる

```
1 FROM rocker/rstudio
2
3 RUN R -e "install.packages(c('here', 'modelsummary', 'janitor'))"
```

しかしこの方法はいくつかの問題がある

- パッケージのバージョンを指定しているわけではない
- パッケージを追加するごとにDockerfileを書き換える必要がある
- ビルドのたびにパッケージをインストールするので時間がかかる
- コンテナごとにパッケージをインストールするのでストレージを圧迫する

これらはrenvのキャッシュをDocker Volumesに保存することで解決できる

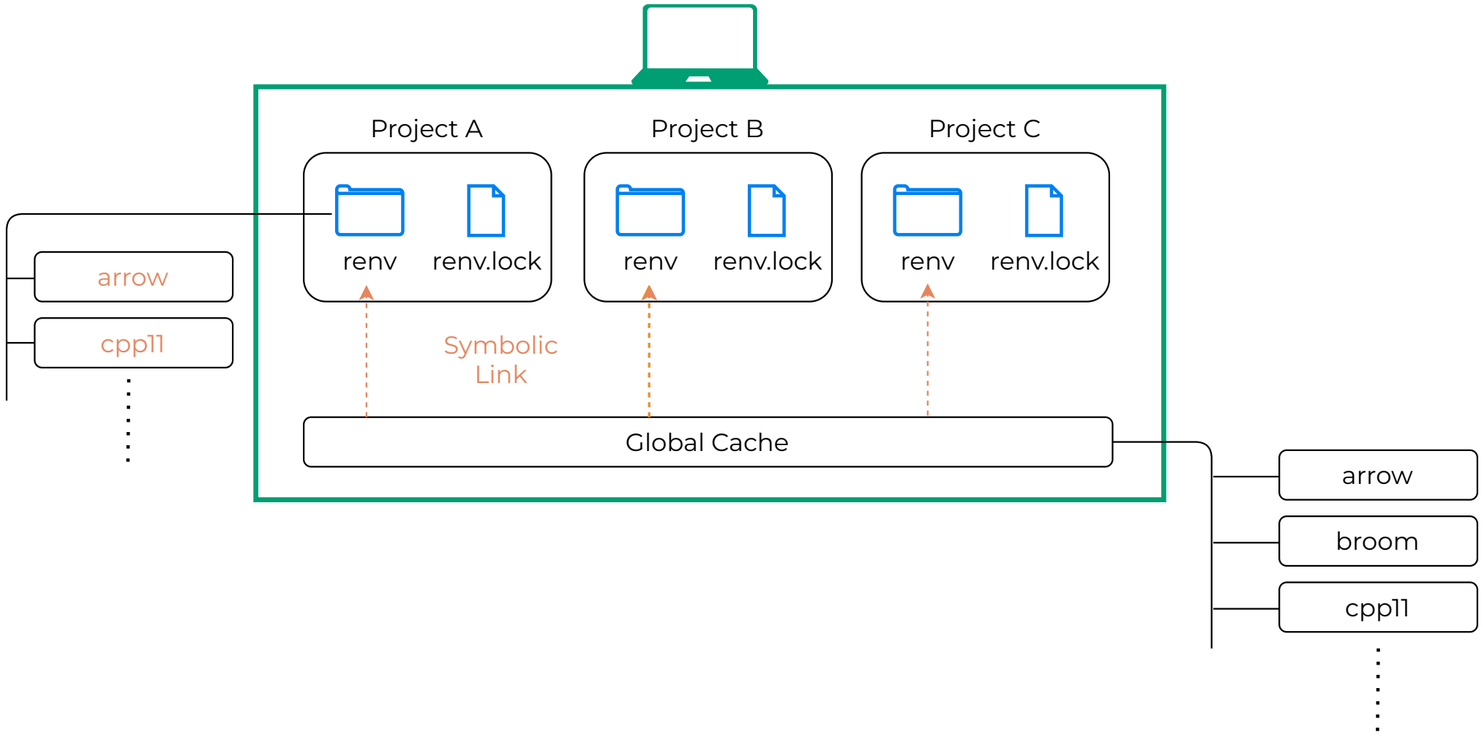
# renvによるパッケージ管理

1. `renv::init()`
   → Rプロジェクトを作成したあとにを実行する
   → `renv/`フォルダと`renv.lock`ファイルを作る
2. `renv::snapshot()`
   → 実行すると, その時にプロジェクトのソースコード中で使用されているパッケージ情報が`renv.lock`ファイルに自動的に記録される
3. `renv::restore()`
   → 共著者やコラボレーターは`renv::restore()`を実行することで`renv.lock`の情報からパッケージを自動でインストールできる

renv.lock

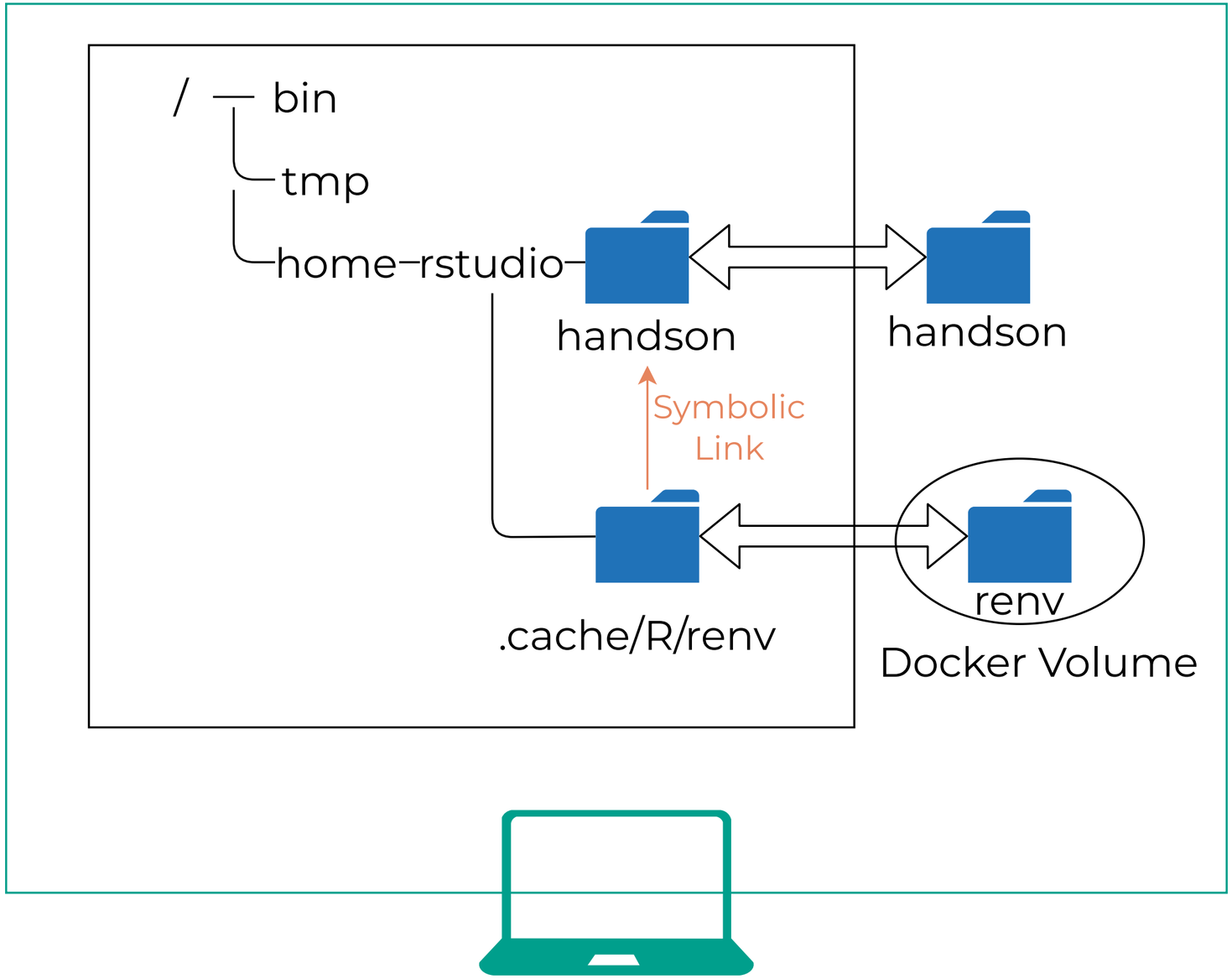
```
{
  "R": {
    "Version": "4.3.1",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://packagemanager.posi
      }
    ]
  },
  "Packages": {
    "dplyr": {
      "Package": "dplyr",
      "Version": "1.0.10",
      "Source": "Repository",
      "Repository": "RSPM",
      "Hash": "539412282059f7f0c07295723d2
      "Requirements": [
```

# renv のキャッシュ




- `renv`は各Rの各パッケージのバージョンごとにグローバルキャッシュを持っている
- プロジェクト内で利用するパッケージは`renv/`フォルダ内に収められているが, 実体はシンボリックリンクが張られたグローバルキャッシュにある
- キャッシュをシェアすることで, パッケージを無駄なくインストールできる

# Handson: RStudio Server with `renv`



1. ホスト側でrenv用のDocker Volumeを作成する

   → `docker volume create renv`

2. docker-compose.yml, Dockerfileを以下のように書き換える

   → chownでrenvディレクトリの所有者をrstudioにしている

```yaml
services:
  rstudio:
    build:
      context: .
    environment:
      - TZ=Asia/Tokyo
      - DISABLE_AUTH=true
    volumes:
      - .:/home/rstudio/handson2
      - renv:/home/rstudio/.cache/R/renv

volumes:
  renv:
    external: true
```

```dockerfile
FROM rocker/rstudio

RUN R -e "install.packages('renv')"

RUN cd /home/rstudio && \
    mkdir .cache .cache/R .cache/R/renv && \
    chown rstudio:rstudio .cache .cache/R .cache/R/renv
```

3. VSCodeでCtrl-Shift-Pから`Rebuild and Reopen in Container`を選択する

4. RStudioでプロジェクト作成し, renvを用いてパッケージを記録する

# Docker + VSCodeによる研究環境

# Handson: kazuyanagimoto/dockerR

*管理者編*

1. kazuyanagimoto/dockerRをテンプレートにGitHubのレポジトリを作成する
2. 作成したレポジトリをクローンする
3. `Dockerfile`, `docker-compose.yml`, `.devcontainer.json` を修正する.
   → Python, Julia, TinyTeXなど今回関係ないものは削除する
4. Docker Volumeが作られていることを確認する
5. VSCodeで`Rebuild and Reopen in Container`を選択する
6. RStudioでプロジェクトを作成し, `renv`を用いてパッケージをインストールする

## *共著者編*

1. kazuyanagimoto/rabootcamp-docker-2023をクローンする
2. Docker Volumeが作られていることを確認する (今回は必要ない)
3. VSCodeで`Rebuild and Reopen in Container`を選択する
4. Rコンソールで`install.packages(c("rlang", "jsonlite", "rmarkdown"))`を実行する
5. Rプロジェクトを開き, `renv::restore()` を実行する
6. `index.qmd` を開き, Ctrl-Shift-Kでスライドがビルドされることを確認する

より詳しい使い方はZenn記事を参考のこと

# Docker Desktop for Windows/Mac

## *DockerはLinux上の技術*

- Docker Desktop for Windows/Macでは, 仮想マシン上のLinuxを用いてDockerを使う
- WindowsはWSL2という軽量かつ高速なLinuxエミュレータを用いる
- Mac上のDockerは一般にLinux上のDockerよりも数段遅くなる

## *ファイルシステムの違い*

- Windows/Mac上のファイルをDocker (Linux) で用いようとするとファイルの変換作業が必要
- Windowsの場合はWSL2の領域でファイルを保存すれば解決する
- Macの場合は長らく解決策がなかったが, virtiofsという新しいファイルシステムを用いることで高速化した (らしい)

# 発展: Docker on AWS

# SSH接続

0. AWSアカウントを作成する. クレジットカード登録も必要.
1. AWSでEC2インスタンスを作成. Ubuntuを選ぶ.
2. インスタンス作成時にキーペアを作成すると, `.pem`ファイルがダウンロードされる
   → 基本的にユーザー直下の`.ssh`フォルダに保存する
   → Mac/Linuxの場合のみ, `chmod 400`で`.pem`ファイルのパーミッションを変更する
3. インスタンスを起動する. インスタンスのパブリックIPアドレスをメモしておく
4. インスタンスにSSHでログインする. ターミナルを開き, SSH接続ができることを確認

```
ssh -i PATH_TO_PEM ubuntu@IP_ADDRESS
```

**Speedtest CLI**

- speedtest-cliをインストールすると, インターネットの速度を計測できる
- Ubuntu向けのインストールを完了した後, `speedtest`をターミナルで実行する
- 高速なインターネット環境 (1Gbps前後) であることを確認する

# Docker環境の構築

## dockerのインストール

```
sudo amazon-linux-extras install -y docker
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -a -G docker ubuntu
```

## docker-composeのインストール

```
sudo mkdir -p /usr/local/lib/docker/cli-plugins
sudo curl -L https://github.com/docker/compose/releases/download/v2.20.3/docker-compose-$(uname -s)-$(uname -m) \
  -o /usr/local/lib/docker/cli-plugins/docker-compose
sudo chmod +x /usr/local/lib/docker/cli-plugins/docker-compose
sudo ln -s /usr/local/lib/docker/cli-plugins/docker-compose /usr/bin/docker-compose
```

- `ubuntu`ユーザーを`docker`をsudoなしで実行できるようにしている
- v2.20.3の部分は最新版のリリースを確認
- ここまで終了したら, `exit`で一度ログアウトする

# VSCode SSH接続

0. VSCodeに`Remote-SSH`拡張機能をインストール
1. `~/.ssh/config`に以下のように記述する

```
Host rabootcamp
  HostName IP_ADDRESS
  User ubuntu
  IdentityFile PATH_TO_PEM
```

2. VSCodeで`Remote-SSH: Connect to Host ...`で`rabootcamp`を選択
3. kazuyanagimoto/dockerRのハンズオンの共著者編を実行する

> **Caution**
>
> ハンズオンの終了時には必ずインスタンスを停止すること. 今後使う予定がない場合は終了してもよい.

# Docker on AWS の実践的アドバイス

### インスタンスの切り忘れが怖い

- AWSのバジェットアラートを設定すると, 予算を超えた場合にメールで通知してくれる

### IPアドレスが毎回変更されてしまうのが面倒

- AWSのElastic IP (有料) を使うと, IPアドレスを固定できる
- GoogleのGCPは現在は無料でIPアドレスを固定できる

### AWS上のコンテナ内でGitHubと通信したい

- ローカルでSSH Agentの設定とSSHコンフィグでAgent Forwardの設定を行うと, ローカルのSSHキーをAWS上のコンテナで使えるようになる

# Learn More

Udemy講座 (かめ) ・ Zenn記事 (柳本) ・ Notion記事 (神戸大 山﨑先生)