

# 勉強会

**Docker編** ~基本的な概要や使い方からDBまで触れる~

# 自己紹介

- 氏名:うえ様
- 趣味:ゲーム
- 最近したこと:ゲーム＆ゲーム
- Twitter:下に載せとるわ



# アジェンダ

1. 仮想化とは？
2. Dockerの利点
3. ハンズオン

# 仮想化とは？

仮想化（英語: virtualization）とは、コンピュータのリソースを抽象化することである。主にユーザーに提供するコンピュータそのものをハードウェアの詳細から切り離した状態でソフトウェア化する事を指す。

by [Wikipedia](#)

# つまり。。。。

簡単にいようと、1つのpcであたかももう1つpcがあるように見せる技術。メリットとして、例えばmacOSでもWindowsOSを立てることができる。つまり、OS毎に～用のpcなどの必要性がなくなる。

とはいっても、スペックによって仮想化にも限界がある・・(仮想環境作る時に割り当てるメモリも自由に設定できる。

作成

-タイミングシステム

名前: Windows10\_20190302

場所: C:\Users\ユーザー名\VirtualBox VMs

(I): Microsoft Windows

(V): Windows 10 (64-bit)

(M)



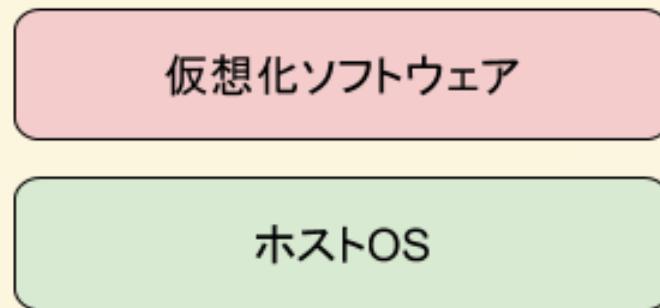
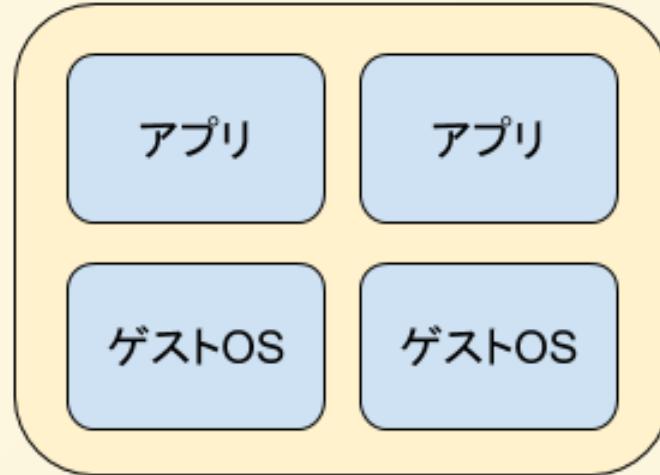
8192 MB

ドディスクを追加しない(D)

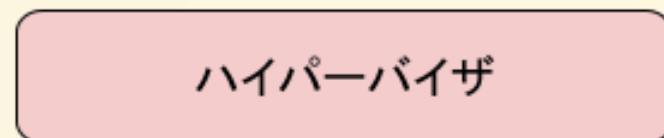
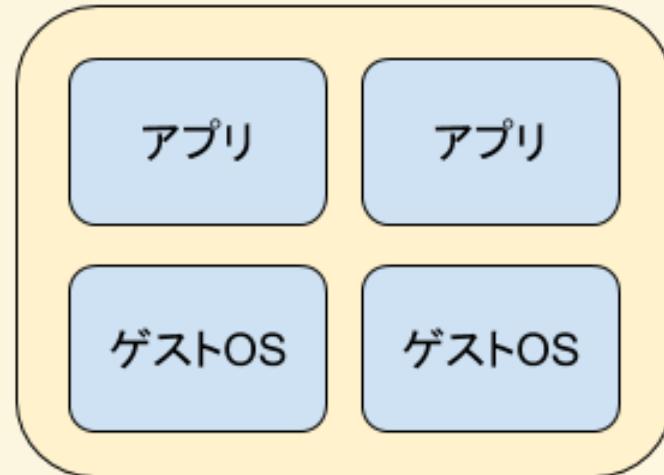
ドディスクを作成する(C)

5仮想ハードディスクファイルを使用する(U)

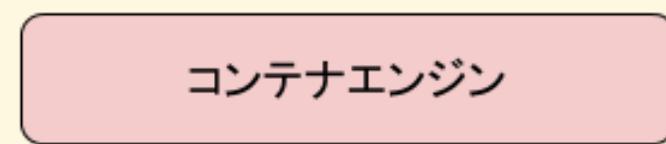
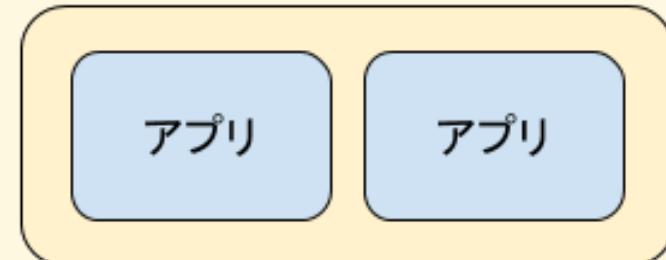
# 仮想化の種類



ホスト型



ハイパーバイザー型



ホストOS

コンテナ型

# ホスト型

ホストOS(普段使ってるOS)上に仮想環境のソフトウェアをインストールして構築していく。

ホストOSの上にゲストOS(仮想化したもの)が乗っかる感じ。

- 有名なソフトウェア・・・Oracle VirtualBox, etc..

## ハイパバイザー型

1つのハードウェアにハイパバイザーと呼ばれる仮想化ソフトウェアを直接インストールし、仮想環境を作るタイプ。

OS自体に仮想環境の機能がついてるもの。

Windows OSではWindows 10 proのみ、Hyper-vという機能がある。

- 有名なソフトウェア(機能)・・・Hyper-v, etc..

## コンテナ型

ホストOSにコンテナエンジンの仮想化ソフトウェアをインストールし、その中でコンテナ環境を作りアプリケーションを実行する。コンテナは複数立てることができ、それぞれ**独立している**。

- 有名なソフトウェア・・・Docker

## ホスト型とハイパバイザーの違い。

ホスト型の処理速度を改善したものがハイパバイザー。  
ハイパーバイザ型は容量そのまま食うので注意。

## ホスト型とコンテナ型の違い。

ホスト型は主にGUI操作が可能だが、コンテナ型は無理。  
コンテナ型は最小の構成なので、言語のversion云々で環境作りたいときはこっちの方が良き。  
OS丸ごといわわけじゃないので。

# Dockerはどういう時に使われるの？いいとこは？

**皆同じ環境で作業を行うことができる！**

Dockerに関するファイルを共有するだけで同じ環境で作業することができます。

**言語のバージョンなどの競合がなくなる。**

言語のバージョンでいうと、最近では\*.env系で管理するのが主流になりつつありますが、

一時的にあるバージョンを使用する場合などは、ホストOSを汚さずにすみます。

要するにバージョンがごちゃごちゃにならなくて良き。

## 他にも。。。.

- 軽量で速い
- 多数のコンテナを起動させることが可能
- *DockerHub*というGitでいうGitHubみたいなステーションがあり、環境が整えるためのファイルが既に転がってる
- 開発環境をそのまま本番環境へ適応することもできる。

では、そろそろ本題へ。。。

# Dockerのインストール

- [docker](#)
- [手順](#)

# 用語などなど

- Dockerイメージ
  - OS・アプリの環境が記述されている。
- コンテナ
  - イメージをベースにした環境。
- Dockerfile
  - Dockerイメージ+a(環境変数や追加パッケージなど)。  
追加したいものがある時に使用する。  
絶対ないとダメなものではない。

# Dockerのコマンド

- docker run ~
  - コンテナ起動
- docker exec ~
  - コンテナに接続
- docker images
  - 手元にあるイメージ一覧
- docker ps
  - 稼働中のコンテナを表示

- docker start ~
  - ~のコンテナを起動
- docker stop ~
  - ~のコンテナの終了

## まあとりあえず触ってみようや(step1)

ってことでハンズオン挟みます[c-a-c/summer\_study\_2020]  
([https://github.com/c-a-c/Summer\\_study\\_2020](https://github.com/c-a-c/Summer_study_2020))

(作業環境はどのディレクトリでも構いません。)

CACのGitHub Organization入ってるかな？

```
git clone https://github.com/c-a-c/Summer_study_2020.git
```

してもらって。

ハンズオンの資料はREADMEに載せてるので、そっちみてやるよ～!

モクモク默默



# step1終わり

いかがでしたでしょうか？

ちょっと難しいかも？

とはいっても、(今回はMySQLを)1から環境構築するよりかは全然時間がかかるないはずです。

では次のステップへ！

## 問題点

先ほどのstep 1 では、 MySQLを起動できたのはいいものの、何らかの理由でコンテナを消した時にデータが残らず、消えてしまいます。

そうならないために、 **永続化**という作業を行います。

# マウント(mount)

と、その前にマウントの話をしないといけません。

Dockerでいうマウントとは、ホストOS側のファイルをゲストOS側で使えるようにする機能です。

要するにホストとゲストでファイルを共有できます。

## ということは。。。。

コンテナ内部で作成したデータベースなどがホスト側にも共有され、情報を残すことができます。

この概念を**永続化**と言います。

# マウントを使わないで共有する方法

もちろんあります。

コンテナ側からホスト側へ・ホスト側からコンテナ側へ、両方できます。

ex) `docker cp コンテナID:/hoge.txt purpose/`

ファイル1つ1つコピーするなんてナンセンスだし、面倒くさいよね。。。

(過去の私はマウントというものを知らなかったので、このコマンドを愛していました。)

# もう1つ新しいこと

コンテナの起動には `docker run ~` としていましたが、  
`docker-compose` というコマンドで起動する方法もあります。

今回データベースの永続化にあたって、このコマンドを使用します。  
このコマンドを使用するには `docker-compose.yml` が必要になります。

また、このファイルがある場所でしか起動できません。

## では、コンテナを消したらデータベースの情報が消えることを確かめましょう。(step2)

前述のcomposeファイルは頭の片隅に置いて置いてもろて、  
とりあえず、データベースの情報が消えるのかをまず確かめてみま  
す。

再びハンズオン(作業環境はどのディレクトリでも構いません。)

README.mdのstep2をご参照^^

## 確認できたので、続いてcomposeファイルで起動してみる。(step3)

最後のハンズオン--;

ディレクトリはcloneした `docker @ 79ead41/` で行います。

READMEのstep3を参照してください。

# docker-composeの本当の使い方

composeコマンドは通常今回のハンズオンで使うのにあまり恩恵を受けていません。

本来の使い方はdocker runを複数する、即ち、複数のコンテナを立ち上げる際に、コマンド1つで立ち上げることができる場合に利用します。 (ex) PHPのコンテナとMySQLのコンテナを同時に立ち上げる。

docker runではオプションを多くつけすぎてわからなくなることが多いので、1つのコンテナを立ち上げるのにcomposeファイルを利用することもあります。(今回がそれ)

1番の利点は、複数のコンテナを1つのコマンドで立ち上げができるということを頭の隅っこに入れておこう。

**step3終了**

## まとめ

- Dockerを使うと簡単に環境を構築できる。
- コンテナはそれぞれ独立しているが、連携させることで1つのアプリとして使えるようになる。

## 参考資料

- 仮想化
  - [aaa](#)
  - [bbb](#)
- Dockerの永続化
  - [ccc](#)

THANK YOU FOR LISTNING 😊

