



Docker入門

by M-IT



Information Technology Center

Information Technology Center



目次

- Dockerとは
- Dockerのインストール
- Dockerコンテナの実行
- Dockerネットワーク
- Dockerデータ管理
- Docker Compose



ハンズオン参加前提条件、事前準備

前提条件

- ・Linuxの基礎知識、基本的なコマンドの理解
- ・仮想化関連の基礎知識

事前準備

- ・メモリ4G以上の64bit Windows 10 Pro / Mac 10.11以上
- ・ポケット wifi があればベスト
- ・以下の場所から必要なものをダウンロードしてください(git clone or zip download)
 - ・https://github.com/yuntumg/docker_study



制限事項

- Docker ToolboxはインストールできればDocker for Windows/ Mac とほぼ同じく使えますが、ハンズオンでサポートできない場合があります。
- Docker machineはネットワーク管理、データ管理のところで少し使います。必要最小限で説明を行います。
- Docker swarmなどクラスタ構成関連の内容はありません。
- Kubernetesの内容はありません。




本講義の目標

- ・ Docker の概念、一般的なコマンドを理解できるようになる。
- ・ コンテナ、イメージの管理、コンテナの管理ができるようになる。
- ・ Dockerfileで自分で images を作成し、公開できるようになる。
- ・ アプリ → データベースの2レイアのコンテナ環境構築できるようになる。



目次

- Dockerとは 
- Dockerのインストール
- Dockerコンテナの実行
- Dockerネットワーク
- Dockerデータ管理
- Docker Compose



Dockerとは

Docker社が提供する「コンテナ型**仮想化**技術」を実現するソフトウェア。

カーネルの機能を利用して、ホストOS上に**隔離された**アプリケーション実行環境を作られる。

Docker自体はGo言語で書かれている。



Dockerとは

従来の仮想化

PCやサーバのマシンにインストールされているOS(ホストOS)の上に、別のマシンを仮想的に立ち上げる事。

コンピューターの中にさらにコンピューターを動かしているイメージ。



Dockerとは

従来の仮想化

ホストOS上に仮想化ソフトをインストールする

→ 仮想化ソフト上に仮想マシンを作成

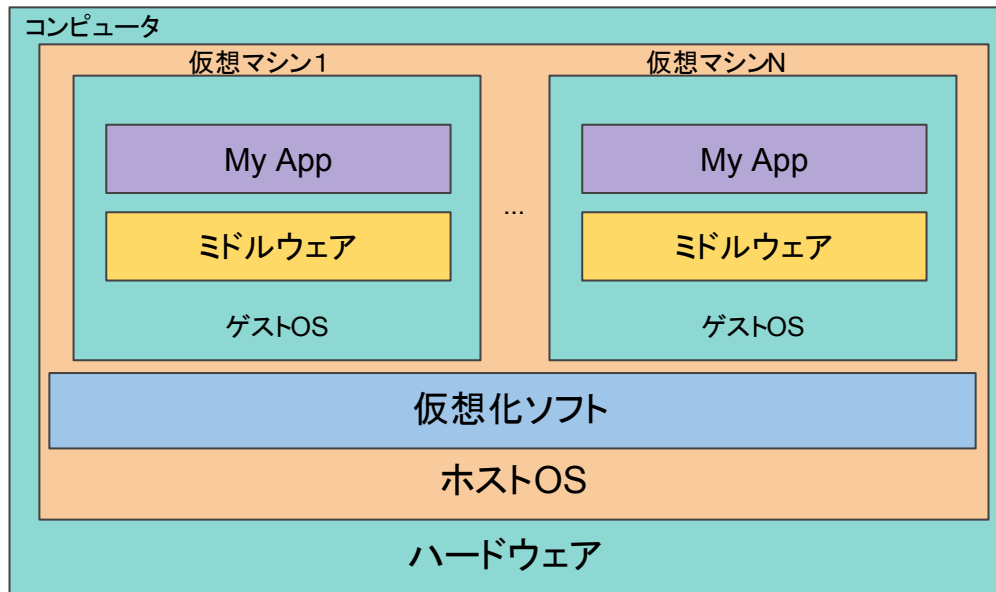
→ 仮想マシンにゲストOSインストール

→ ゲストOSの中でミドルウェアの環境構築する

→ プログラムを実行してアプリケーションを動かす

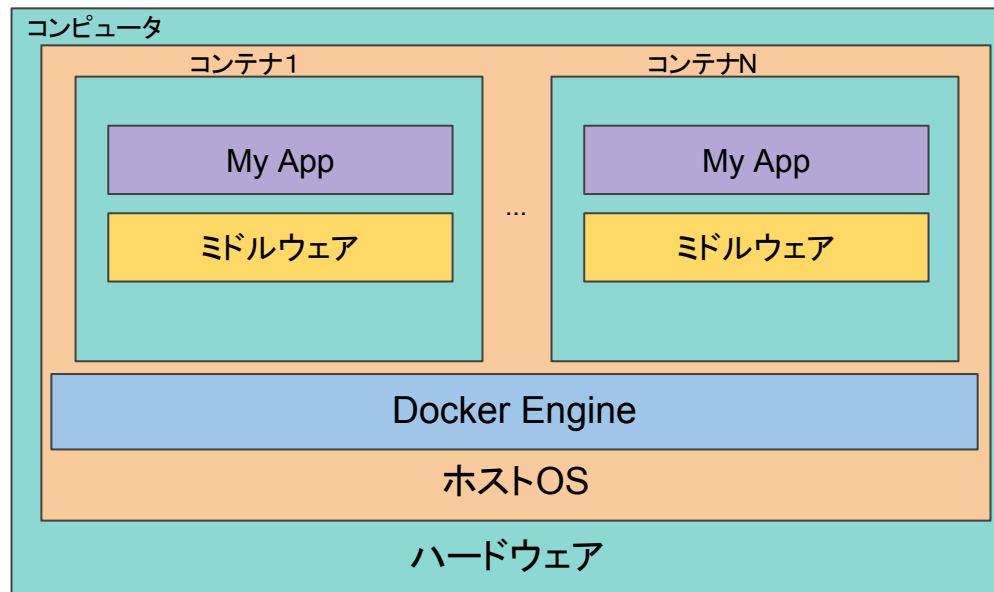
Dockerとは

従来の仮想化



Dockerとは

コンテナ型仮想化





Dockerとは

コンテナ型仮想化

ゲストOSを起動せずに、ホストOSの上に動作しているDocker Engineからコンテナと呼ばれるミドルウェアの環境構築がされた実行環境を作成し、その中でアプリケーションを動作させます。



Dockerとは

良いところ

プログラムコード

→ Git (Github、Bitbucket、Gitlab、gogs)



Dockerとは

良いところ

実行環境は？

→ 開発者みんなバラバラ

具体的

→ Mac OS、Windows

→ java8、java11

→ tomcat8、tomcat8.5 etc...



Dockerとは

良いところ

従来の仮想化を使い、仮想環境を作ってみんなで共有も良いですが。

→サイズがでかい、動作が重たい、共有がむづかしい



Dockerとは

良いところ

Dockerは上記の問題を解決できます。

- imageにアプリを含めて、動かす環境ミドルウェアが
もろもろ入っている。
- image のサイズが小さい、軽量(DL、UPに時間かからない)
- 共有するためDockerHubなどレジストリサービスあり。(自前で構築も可能)



Dockerとは

良いところ

Docker Engineさえあればアプリを動かす環境が簡単に構築できる。



Dockerとは

Docker Engine

Dockerを使うための常駐プログラム。

Docker for mac、Docker for Windows、Docker ToolboxなどのソフトウェアをPCにInstallすると、常駐プログラムとしてDocker Engineが動作し、Dockerを利用することができるようになります。




Dockerとは

デメリット(欠点)

- ・ホストOSと違うシステムをコンテナとして動かせない。
- ・学習コスト。
- ・構成が複雑になるかも。
- ・不具合の調査/切り分けが難しい。



目次

- ・ Dockerとは
- ・ **Dockerのインストール** 
- ・ Dockerコンテナの実行
- ・ Dockerネットワーク
- ・ Dockerデータ管理
- ・ Docker Compose



Dockerのインストール

事前準備

- ・ Docker Hub のアカウント作成してください。
<https://hub.docker.com/> から右上の「Sign Up」で



Dockerのインストール

事前準備

- ・インストーラーをダウンロード
 - ・Docker Hubにログインしたあと、
windows
<https://hub.docker.com/editions/community/docker-ce-desktop-windows>
 - Mac
<https://hub.docker.com/editions/community/docker-ce-desktop-mac>



Dockerのインストール

DockerのEdition

- Docker CE

無料

基本的な機能は使える

StableとEdge版がある

- Docker EE

有料

Docker社認定のコンテナが使える

イメージのセキュリティスキャンが行われる

プライベートリポジトリが使える etc...



Dockerのインストール

Dockerバージョン

Dockerのバージョン

・18.09.2



Dockerのインストール

for Windows

- ・hyper-vの機能を有効にする必要がある。
 1. Windows ボタンを右クリックし、[アプリと機能] を選択します。
 2. 右側にある [関連する設定には、[プログラムと機能を選択します。
 3. [Windows の機能の有効化または無効化] を選択します。
 4. [Hyper-V] を選択して、[OK] をクリックします。



Dockerのインストール

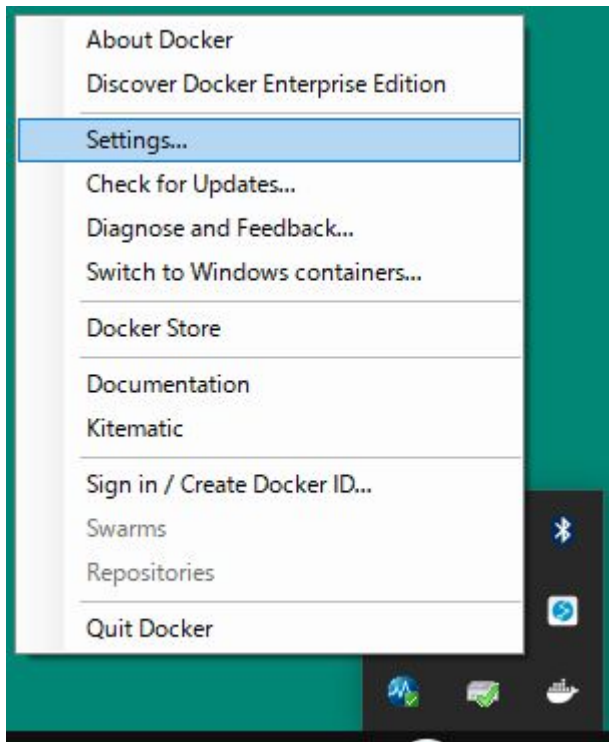
for Windows

管理者権限が求められたり、仮想ネットワークデバイス作成の許可を求められます。

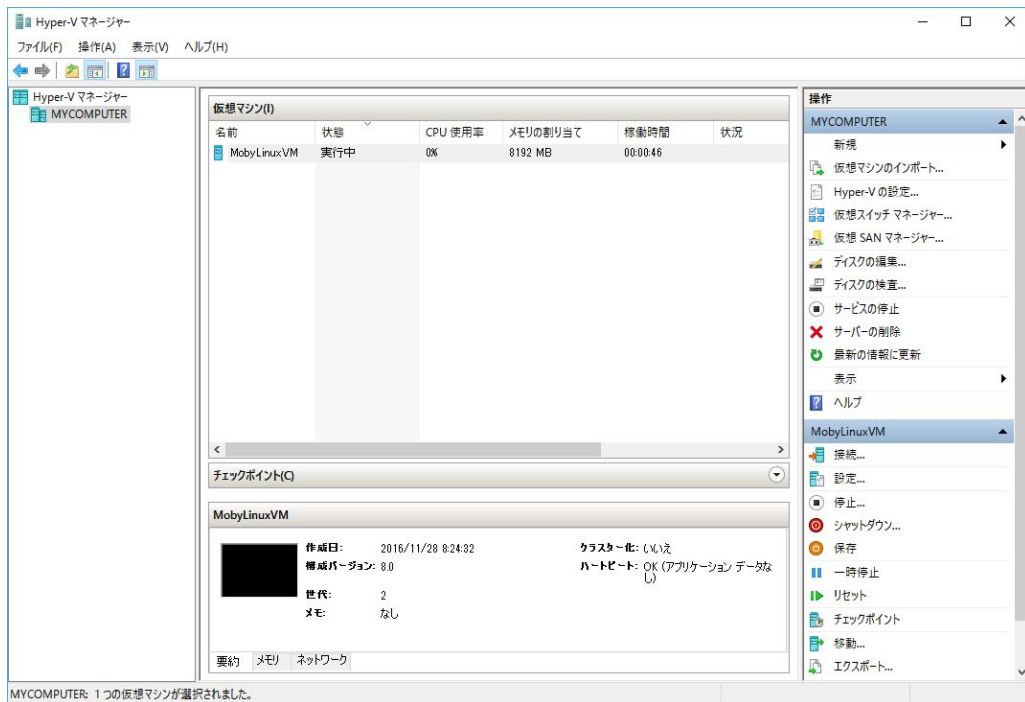
それ以外は普通に「次へ」でいけるはず。

インストールできたら右下のタスクバーでDockerの起動状態の確認ができる。

Dockerのインストール for Windows



Dockerのインストール for Windows





Dockerのインストール

for Mac OS

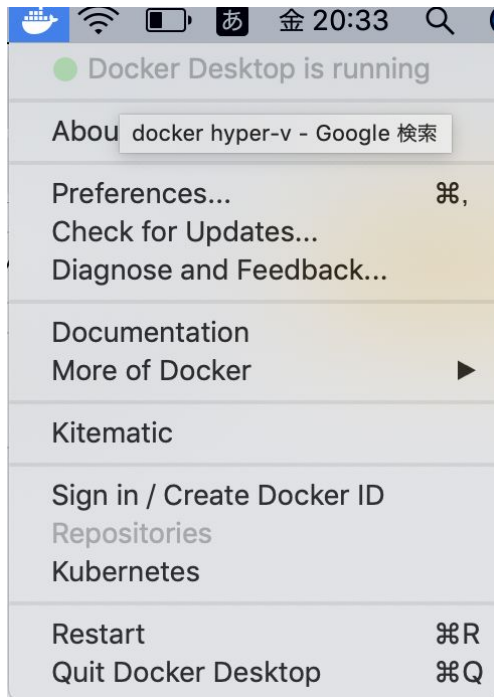
ダウンロード、インストールだけ。特別な設定は不要。
管理者パスワードを求める場合がある。

Docker Machine を使うため、virtual box のインストールも必要。



Dockerのインストール

for Mac OS





Dockerのインストール

インストール確認

「docker version」コマンドが実行できれば可。



Dockerのインストール

インストール確認

```
$ docker version
```

```
Client: Docker Engine - Community
```

```
Version:      18.09.2
```

```
API version:  1.39
```

```
Go version:   go1.10.8
```

```
Git commit:   6247962
```

```
Built:        Sun Feb 10 04:12:39 2019
```


```
OS/Arch:      darwin/amd64
```

```
Experimental: false
```

```
...
```




目次

- ・ Dockerとは
- ・ Dockerのインストール
- ・ **Dockerコンテナの実行** 
- ・ Dockerネットワーク
- ・ Dockerデータ管理
- ・ Docker Compose



Dockerコンテナの実行

実行環境

- Windows
powershell or コマンドプロンプト
- Mac OS
任意のターミナルソフト



Dockerコンテナの実行

hello-world

単純メッセージを表示するイメージからコンテナを起動する。

```
docker run hello-world
```

run:サブコマンド

hello-world:イメージ名



Dockerコンテナの実行

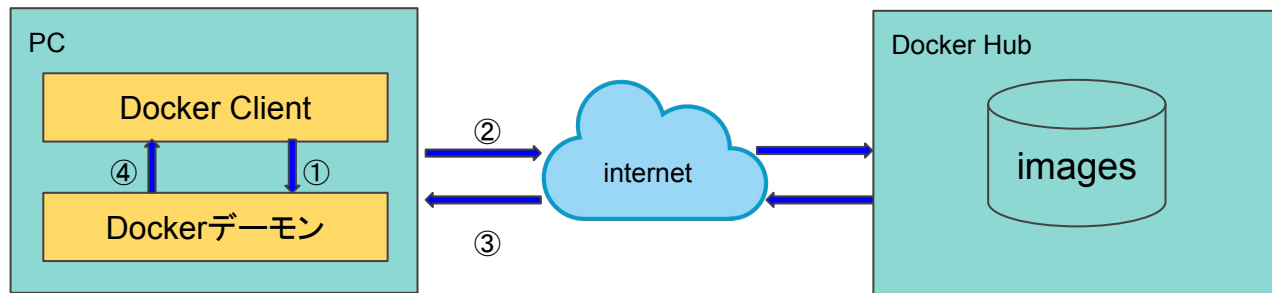
hello-world

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

Dockerコンテナの実行

Dockerイメージ取得手順





Dockerコンテナの実行

runコマンド

- ①docker pull : イメージ取得
- ②docker create : コンテナ作成
- ③docker start : コンテナ起動



Dockerコンテナの実行

docker-sl

```
$ docker run -t yuntung/docker-sl:ver1 /usr/games/sl
```



Dockerコンテナの実行

docker-sl

-t : コンテナの標準出力をホストの標準出力につなげる

yuntumg/docker-sl:ver1 : 名前空間/イメージ名:タグ名

/usr/games/sl : コンテナを起動した後コンテナの中で実行するコマンド



Dockerコンテナの実行

hellomit-java

```
$ docker run yuntumg/hellomit-java:Ver1
```



Dockerコンテナの実行

hellomit-java

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
yuntumg/hellomit-java	Ver1	fb27e3823cf5	9 days ago	50MB



Dockerコンテナの実行

DockerHub

- ・Dockerイメージのレジストリサービス
- ・イメージの公開、検索、ダウンロードができる

<https://hub.docker.com>



Dockerコンテナの実行

Dockerイメージ

Dockerイメージとは

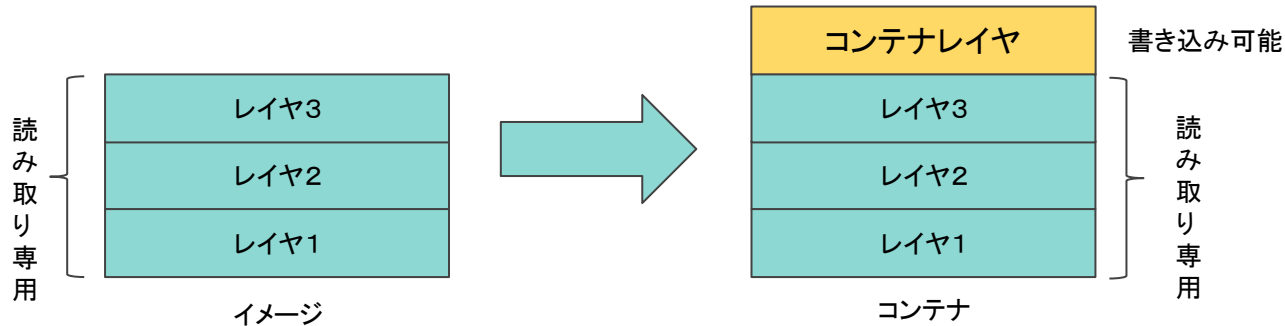
コンテナの実行に必要なものをまとめたファイルシステム

普通のファイルシステムとは少し違う階層型のファイルシステムを使っている

階層はレイヤと呼ばれ、読み取り専用

Dockerコンテナの実行

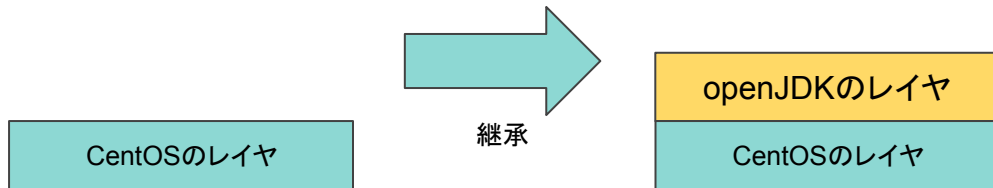
Dockerイメージ





Dockerコンテナの実行

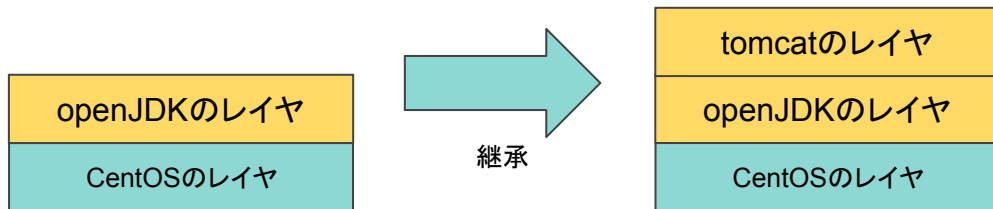
Dockerイメージ





Dockerコンテナの実行

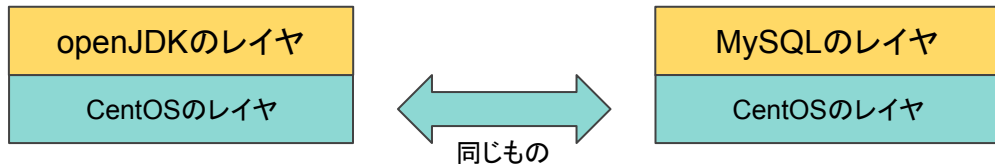
Dockerイメージ





Dockerコンテナの実行

Dockerイメージ





Dockerコンテナの実行

whalesay

```
$ docker run docker/whalesay cowsay hello
Unable to find image 'docker/whalesay:latest' locally
latest: Pulling from docker/whalesay
e190868d63f8: Downloading [=====>          ] 17.73MB/65.77MB
909cd34c6fd7: Download complete
0b9bfabab7c1: Download complete
a3ed95caeb02: Download complete
00bf65475aba: Download complete
...
```



Dockerコンテナの実行

image管理

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
yuntumg/hellomit-java	Ver1	fb27e3823cf5	9 days ago	50MB
yuntumg/docker-sl	ver1	d535de9d177a	3 weeks ago	97.3MB
docker/whalesay	latest	6b362a9f73eb	3 years ago	247MB



Dockerコンテナの実行

image管理

```
$ docker tag docker/whalesay:latest my_whalesay
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
yuntumg/hellomit-java	Ver1	fb27e3823cf5	9 days ago	50MB
yuntumg/docker-sl	ver1	d535de9d177a	3 weeks ago	97.3MB
docker/whalesay	latest	6b362a9f73eb	3 years ago	247MB
my_whalesay	latest	6b362a9f73eb	3 years ago	247MB



Dockerコンテナの実行

image管理

```
docker tag docker/whalesay:latest my_whalesay
```

tag : タグ付けるサブコマンド

docker/whalesay:latest : 元のイメージ名

my_whalesay : 新しいイメージ名



Dockerコンテナの実行

image管理

```
$ docker inspect my_whalesay
[
  {
    "Id":"sha256:6....cce1b6637fc25646728cf7fb0679b2da273c3f4",
    "RepoTags": [
      "docker/whalesay:latest",
      "my_whalesay:latest"
    ],
    ...
  }
]
```



Dockerコンテナの実行

image管理

イメージを削除

```
docker rmi my_whalesay
```

イメージ強制削除

```
docker rmi -f my_whalesay
```

イメージ取得

```
docker pull docker/whalesay
```



Dockerコンテナの実行

Dockerfile

では、ここからは自身でイメージを作成する方法を見ましょう。

Dockerファイルというイメージの定義ファイルを作成する。→イメージビルド



Dockerコンテナの実行

Dockerfile

```
$ mkdir imageBuild
```

```
$ cd imageBuild
```

```
$ vi Dockerfile
```




Dockerコンテナの実行

Dockerfile

```
FROM docker/whalesay:latest
```

```
RUN apt-get -y update && apt-get install -y fortunes
```

```
CMD /usr/games/fortune | cowsay
```



Dockerコンテナの実行

Dockerfile

FROM: 親になるイメージを指定する。

RUN: ビルド時にコンテナ内で実行されるコマンド

CMD: 完成したイメージからコンテナを作成するときに実行される



Dockerコンテナの実行

Dockerfile

`docker build -t my-whale .`

build : イメージをビルドするサブコマンド
-t my-whale : 作成するイメージ名を指定
. : ビルドコンテキストの指定

※ビルドコンテキストとは: イメージに含まれる「内容」



Dockerコンテナの実行

Dockerfile

```
$ docker build -t my-whale .  
Sending build context to Docker daemon 2.048kB  
Step 1/3 : FROM docker/whalesay:latest  
----> 6b362a9f73eb  
Step 2/3 : RUN apt-get -y update && apt-get install -y fortunes  
省略  
Step 3/3 : CMD /usr/games/fortunes | cowsay  
----> b1c37f16e198  
Successfully built b1c37f16e198  
Successfully tagged my-whale:latest
```



Dockerコンテナの実行

Dockerfile

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-whale	latest	b1c37f16e198	3 minutes ago	278MB



Dockerコンテナの実行

historyコマンド

指定したDockerイメージのイメージレイヤのサマリー情報を参照することができます。

Dockerコンテナの実行

Dockerfile

```
$ docker image history my-whale
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
b1c37f16e198	5 minutes ago	/bin/sh -c #(nop) CMD ["/bin/sh" "-c" "/use...		0B
5eb57c401c21	5 minutes ago	/bin/sh -c apt-get -y update && apt-get inst...		30.6MB
6b362a9f73eb	3 years ago	/bin/sh -c #(nop) ENV PATH=/usr/local/bin:/u...		0B
<missing>	3 years ago	/bin/sh -c sh install.sh	30.4kB	
<missing>	3 years ago	/bin/sh -c git reset --hard origin/master	43.3kB	
<missing>	3 years ago	/bin/sh -c #(nop) WORKDIR /cowsay		0B

...



Dockerコンテナの実行

my-whale

```
$ docker run my-whale
```




Dockerコンテナの実行

ビルドキャッシュ

```
$ docker build -t my-whale .  
Sending build context to Docker daemon 2.048kB  
Step 1/3 : FROM docker/whalesay:latest  
----> 6b362a9f73eb  
Step 2/3 : RUN apt-get -y update && apt-get install -y fortunes  
----> Using cache  
Step 3/3 : CMD /usr/games/fortune | cowsay  
----> Using cache  
Successfully built 1f4c7636e7eb  
Successfully tagged my-whale:latest
```



Dockerコンテナの実行

ビルドキャッシュ

```
$ docker build --no-cache -t my-whale .
```



Dockerコンテナの実行

イメージ公開

では自作のイメージを公開して見ましょう。

まずは Docker Hub にログインして、リポジトリを作成します。

Create Repository +



Dockerコンテナの実行

イメージ公開

Create Repository

 yuntumg ▼

my-whale

Description

Visibility

Using 0 of 1 private repositories. [Get more](#)



Public 

Public repositories appear in
Docker Hub search results



Private 

Only you can view private
repositories



Dockerコンテナの実行

イメージ公開

 yuntumg / my-whale

This repository does not have a description 

 Last pushed: never



Dockerコンテナの実行

イメージ公開

```
$ docker login
```

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

Username: xxxxxxxx

Password:

Login Succeeded



Dockerコンテナの実行

イメージ公開

- ・タグ付けのルール

<Docker ID>/image 名:tag名



Dockerコンテナの実行

イメージ公開

```
$ docker tag my-whale yuntumg/my-whale:ver1
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-whale	latest	a0a94cb47b51	33 minutes ago	278MB
yuntumg/my-whale	ver1	a0a94cb47b51	33 minutes ago	278MB



Dockerコンテナの実行

イメージ公開

```
$ docker push yuntumg/my-whale:ver1
The push refers to repository [docker.io/yuntumg/my-whale]
98938bdb23b0: Pushing [=====>] 10.34MB/30.6MB
5f70bf18a086: Pushing 1.024kB
d061ee1340ec: Pushing[=====>] 75.78kB
d511ed9e12e1: Pushing [=====>] 92.67kB
091abc5148e4: Pushing [=====>] 255.5kB
b26122d57afa: Waiting
...
```



Dockerコンテナの実行

イメージ公開

Tags

This repository contains 1 tag(s).

ver1

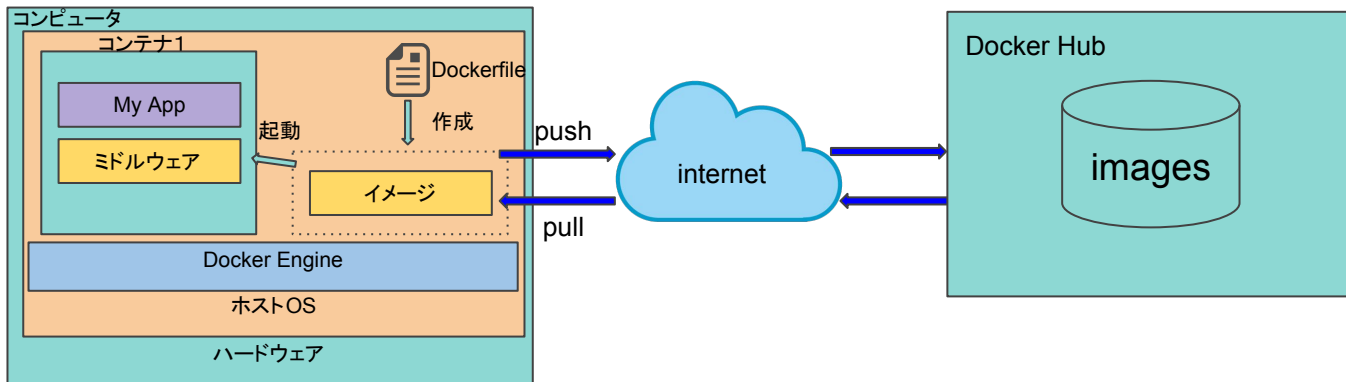


🕒 2 minutes ago

[See all](#)

Dockerコンテナの実行

イメージ公開





Dockerコンテナの実行

ここからは常駐型のコンテナをいくつかみましょう。



Dockerコンテナの実行

Nginxコンテナ



nginx ☆

Docker Official Images

Official build of Nginx.

↓ 10M+

Container

Linux

386

ARM 64

ARM

x86-64

IBM Z

PowerPC 64 LE

Application Infrastructure

Official Image

DESCRIPTION

REVIEWS

TAGS

Supported tags and respective Dockerfile links

- 1.15.9, mainline, 1, 1.15, latest ([mainline/stretch/Dockerfile](#))
- 1.15.9-perl, mainline-perl, 1-perl, 1.15-perl, perl ([mainline/stretch-perl/Dockerfile](#))
- 1.15.9-alpine, mainline-alpine, 1-alpine, 1.15-alpine, alpine ([mainline/alpine/Dockerfile](#))



Dockerコンテナの実行

Nginxコンテナ

```
docker run --name some-nginx -d -p 8080:80 some-content-nginx
```

-- name : 起動するコンテナの名前

-d : デタッチモード、コンテナをバックグラウンドで実行する

-p 8080:80 : ホスト側のポート:コンテナ側のポート

「-d」: デーモンのような常駐アプリのコンテナを起動する場合使うことが多い



Dockerコンテナの実行

Nginxコンテナ

```
$ docker run --name my-nginx -d -p 8080:80 nginx
```



Dockerコンテナの実行

Nginxコンテナ

localhost:8080

Welcome to nginx!

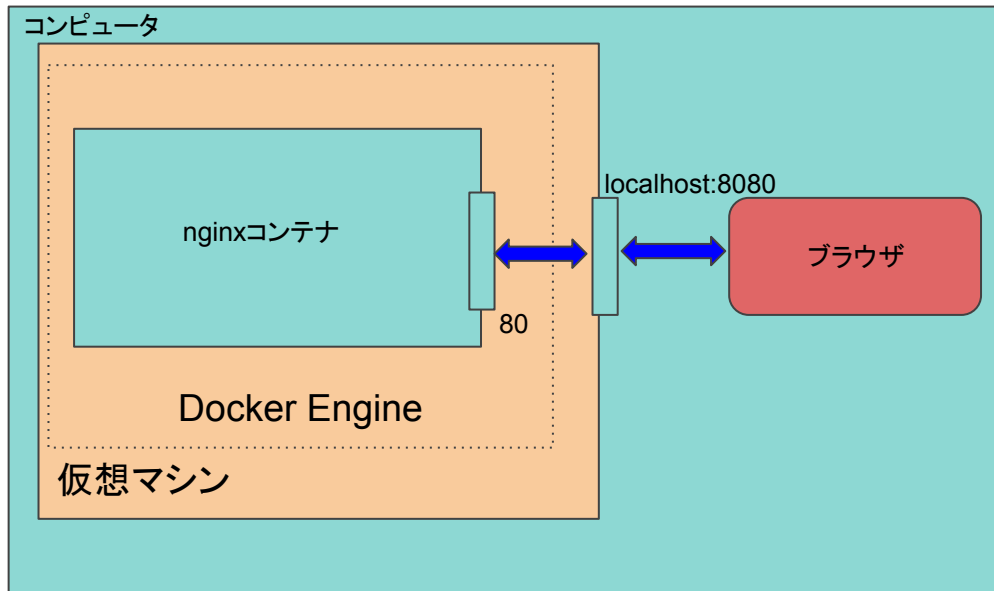
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Dockerコンテナの実行

Nginxコンテナ





Dockerコンテナの実行

Nginxコンテナ

-d の指定がない場合は

Demo参照ください。



Dockerコンテナの実行 バインドマウント

自分で作成したhtml ファイルをnginx で公開する



Dockerコンテナの実行 バインドマウント

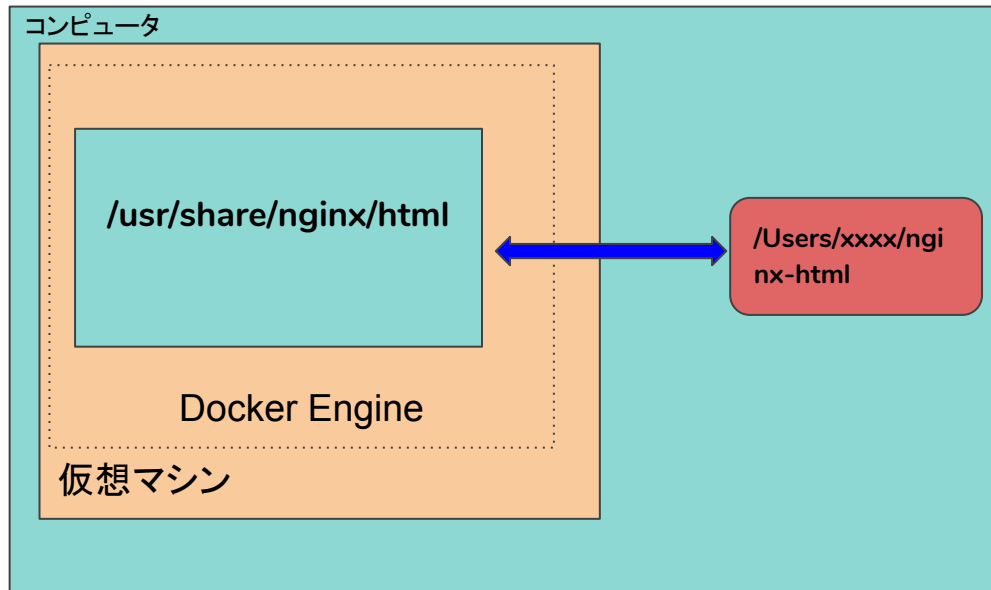
```
docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

-v : ホスト側のdir:コンテナ側のdir:オプション

※ dir のパスは絶対パスでの指定が必要。

Dockerコンテナの実行

バインドマウント





Dockerコンテナの実行

バインドマウント

```
$ pwd  
/Users/oyont/Docker  
  
$ mkdir nginx-html  
$ cd nginx-html  
  
$ vi index.html  
  
$ docker run --name my-nginx-html -d -p 8080:80 -v  
/Users/oyont/Docker/nginx-html:/usr/share/nginx/html:ro nginx
```



Dockerコンテナの実行

バインドマウント

← → ↻ ⓘ localhost:8080

hello M-IT



Dockerコンテナの実行

バインドマウント

index.html を修正して、再度 localhost:8080 にアクセス



Dockerコンテナの実行

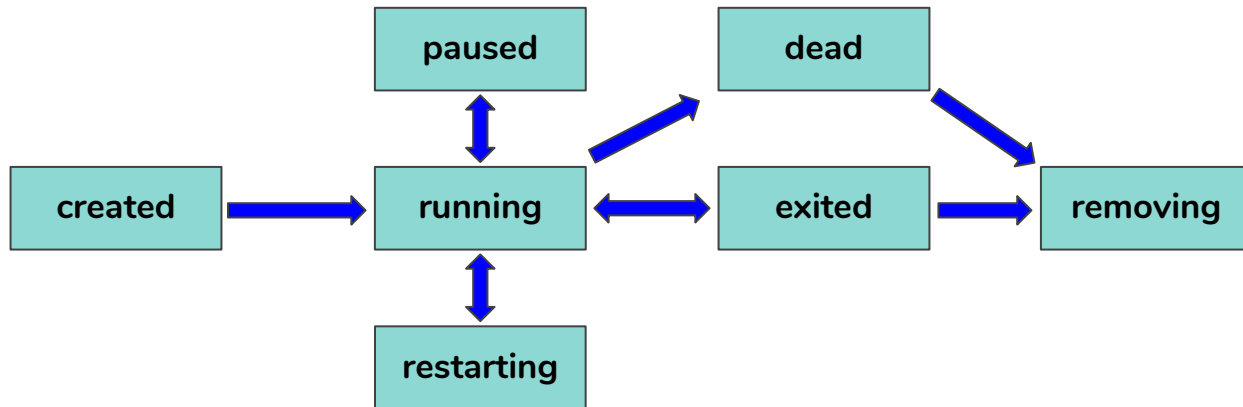
バインドマウント

← → ↺ ⓘ localhost:8080

hello M-IT hello M-IT hello M-IT hello M-IT hello M-IT hello M-IT hello M-IT

Dockerコンテナの実行

コンテナライフサイクル





Dockerコンテナの実行

コンテナライフサイクル

```
$ docker create --name status -it alpine /bin/sh
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6a66011fd19f	alpine	"/bin/sh"	3 seconds ago	Created	status	



Dockerコンテナの実行

コンテナライフサイクル

```
$ docker start status
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4c9578b83501	alpine	"/bin/sh"	About a minute ago	<u>Up 3 seconds</u>		status

```
$ docker inspect status
```



Dockerコンテナの実行

コンテナライフサイクル

```
$ docker pause status
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4c9578b83501	alpine	"/bin/sh"	5 minutes ago	Up 3 minutes (Paused)		status



Dockerコンテナの実行

コンテナライフサイクル

```
$ docker unpause status
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4c9578b83501	alpine	"/bin/sh"	6 minutes ago	<u>Up 5 minutes</u>	status	



Dockerコンテナの実行

コンテナライフサイクル

```
$ docker stop status
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4c9578b83501	alpine	"/bin/sh"	8 minutes ago	<u>Exited</u> (137) 3 seconds ago		status

```
$ docker rm status
```



Dockerコンテナの実行

コンテナに接続

`docker exec コンテナ名/ID -it /bin/bash`



Dockerコンテナの実行

コンテナに接続

```
$ docker run --name my-ubuntu -it -d ubuntu /bin/bash
```

```
$ docker exec -it my-ubuntu /bin/bash
```

```
root@783de44d5cdd:/# ls -l
```

```
total 64
```

```
drwxr-xr-x  2 root root 4096 Feb  4 21:05 bin
```

```
drwxr-xr-x  2 root root 4096 Apr 24  2018 boot
```

```
...
```



Dockerコンテナの実行

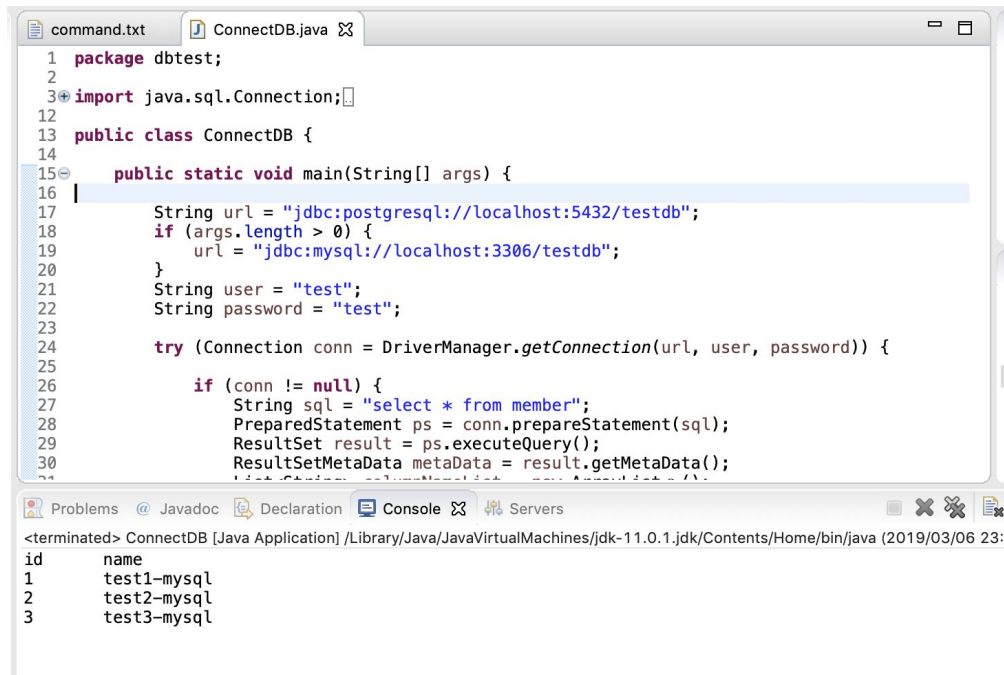
DBコンテナ構築

```
$ docker run --name my-mysql-db -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root  
-d mysql:5.5
```

```
$ docker exec -it my-mysql-db bash
```

Dockerコンテナの実行

DBコンテナ構築



```
1 package dbtest;
2
3 import java.sql.Connection;
4
12 public class ConnectDB {
13
14     public static void main(String[] args) {
15         String url = "jdbc:postgresql://localhost:5432/testdb";
16         if (args.length > 0) {
17             url = "jdbc:mysql://localhost:3306/testdb";
18         }
19         String user = "test";
20         String password = "test";
21
22         try (Connection conn = DriverManager.getConnection(url, user, password)) {
23
24             if (conn != null) {
25                 String sql = "select * from member";
26                 PreparedStatement ps = conn.prepareStatement(sql);
27                 ResultSet result = ps.executeQuery();
28                 ResultSetMetaData metaData = result.getMetaData();
29             }
30         }
31     }
32 }
```

Problems @ Javadoc Declaration Console Servers

<terminated> ConnectDB [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java (2019/03/06 23:00:00)

id	name
1	test1-mysql
2	test2-mysql
3	test3-mysql



Dockerコンテナの実行

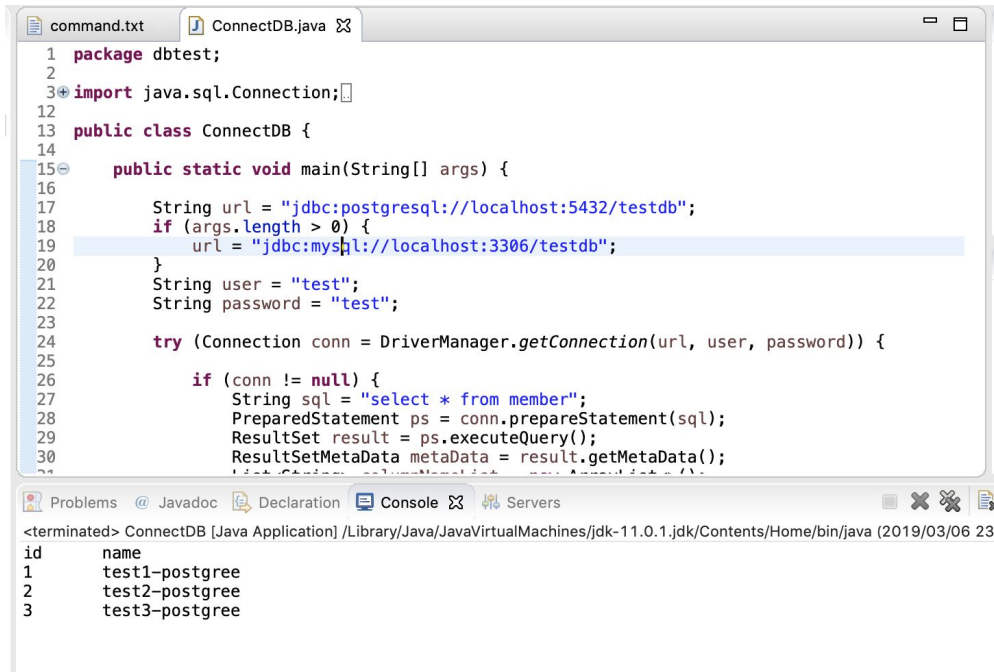
DBコンテナ構築

```
$ docker run --name my-postgress-db -p 5432:5432 -e POSTGRES_USER=root -e  
POSTGRES_PASSWORD=root -d postgres:9.6
```

```
$ docker exec -it my-postgress-db bash
```

Dockerコンテナの実行

DBコンテナ構築



The screenshot shows an IDE with two tabs: 'command.txt' and 'ConnectDB.java'. The 'ConnectDB.java' tab is active, displaying the following Java code:


```
1 package dbtest;
2
3 import java.sql.Connection;
4
12
13 public class ConnectDB {
14
15     public static void main(String[] args) {
16
17         String url = "jdbc:postgresql://localhost:5432/testdb";
18         if (args.length > 0) {
19             url = "jdbc:mysql://localhost:3306/testdb";
20         }
21         String user = "test";
22         String password = "test";
23
24         try (Connection conn = DriverManager.getConnection(url, user, password)) {
25
26             if (conn != null) {
27                 String sql = "select * from member";
28                 PreparedStatement ps = conn.prepareStatement(sql);
29                 ResultSet result = ps.executeQuery();
30                 ResultSetMetaData metaData = result.getMetaData();
31                 int columnCount = metaData.getColumnCount();
32                 for (int i = 1; i <= columnCount; i++) {
33                     System.out.print(metaData.getColumnLabel(i) + " ");
34                 }
35                 System.out.println();
36                 while (result.next()) {
37                     for (int i = 1; i <= columnCount; i++) {
38                         System.out.print(result.getString(i) + " ");
39                     }
40                     System.out.println();
41                 }
42             }
43         } catch (SQLException e) {
44             e.printStackTrace();
45         }
46     }
47 }
```

The console output at the bottom shows the result of the program execution:

```
<terminated> ConnectDB [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java (2019/03/06 23
id      name
1       test1-postgree
2       test2-postgree
3       test3-postgree
```



目次

- ・ Dockerとは
- ・ Dockerのインストール
- ・ Dockerコンテナの実行
- ・ **Dockerネットワーク** 
- ・ Dockerデータ管理
- ・ Docker Compose



Dockerネットワーク

デフォルトのブリッジネットワーク

単一ホスト内で構成されるネットワーク。
比較的に小規模のネットワーク構成で使われます。

確認しやすいため、**docker-machine** で VM を作成し、その中でネットワークの確認を行います。

Mac OSで docker-machine を使うには virtual box のインストールが必要。



Dockerネットワーク

デフォルトのブリッジネットワーク

Docker Machine とは

Dockerコンテナを動かすためのホスト環境を構築するためのツールです。

第6回目の勉強会で使う予定。



Dockerネットワーク

デフォルトのブリッジネットワーク

```
$ docker-machine create network-test
```

Dockerネットワーク

デフォルトのブリッジネットワーク





Dockerネットワーク

デフォルトのブリッジネットワーク

```
$ docker-machine ssh network-test  
docker@network-test:~$
```

注意！！！！

以下「network-test」というVMの中で操作する。



Dockerネットワーク

デフォルトのブリッジネットワーク

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
cf3b069d6924	bridge	bridge	local
06b5e0fee500	host	host	local
0c04b44e3b29	none	null	local



Dockerネットワーク

デフォルトのブリッジネットワーク

```
$ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "5bad5d0a88c3a15e...45ab81d0469ec77d8ce1520b",
    "Created": "2019-02-28T04:24:05.041127053Z",
    "Scope": "local",
    "Driver": "bridge",
    ...
  }
]
```



Dockerネットワーク

デフォルトのブリッジネットワーク

```
...  
"Config": [  
  {  
    "Subnet": "172.17.0.0/16",  
    "Gateway": "172.17.0.1"  
  }  
]  
...
```



Dockerネットワーク

デフォルトのブリッジネットワーク

```
...  
"Options": {  
  ...  
  "com.docker.network.bridge.name": "docker0",  
  ...  
},  
...
```



Dockerネットワーク

デフォルトのブリッジネットワーク

```
# コンテナ作成
$ docker run -itd --name alpine alpine /bin/sh

$ docker network inspect bridge
```




Dockerネットワーク

デフォルトのブリッジネットワーク

```
"Containers": {  
  "983dfe87667175a90...2623e6906f5574351570d5": {  
    "Name": "alpine",  
    "EndpointID": "296f9ba5a4df0b554ab506a...31dd80f8ec015d18",  
    "MacAddress": "02:42:ac:11:00:02",  
    "IPv4Address": "172.17.0.2/16",  
    "IPv6Address": ""  
  }  
},
```



Dockerネットワーク

デフォルトのブリッジネットワーク

```
$ docker run -itd --name alpine2 alpine /bin/sh
```

```
$ docker exec -it alpine2 /bin/sh
```

```
/ # ping -c 3 172.17.0.2
```

```
PING 172.17.0.2 (172.17.0.2): 56 data bytes
```

```
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.042 ms
```

```
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.075 ms
```

```
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.064 ms
```



Dockerネットワーク

デフォルトのブリッジネットワーク

```
/ # ping -c 3 alpine
```

```
ping: bad address 'alpine'
```

↓

ユーザ定義のネットワークを使うことで解決できる



Dockerネットワーク

ユーザ定義のブリッジネットワーク

```
$ docker network create my-network
```

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
cf3b069d6924	bridge	bridge	local
06b5e0fee500	host	host	local
702f20fb0bff	my-network	bridge	local
0c04b44e3b29	none	null	local



Dockerネットワーク

ユーザ定義のブリッジネットワーク

```
$ docker network connect my-network alpine
```

```
$ docker network connect my-network alpine2
```



Dockerネットワーク

ユーザ定義のブリッジネットワーク

```
$ docker exec -it alpine2 /bin/sh
```

```
/ # ping alpine
```

```
PING alpine (172.18.0.2): 56 data bytes
```

```
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.076 ms
```

```
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.091 ms
```

```
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.127 ms
```

```
...
```



Dockerネットワーク

ノンネットワーク

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
cf3b069d6924	bridge	bridge	local
06b5e0fee500	host	host	local
702f20fb0bff	my-network	bridge	local
0c04b44e3b29	none	null	local



Dockerネットワーク

ノンネットワーク

```
$ docker run -itd --name none-network --network none alpine /bin/sh

$ docker exec -it none-network /bin/sh
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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
```




Dockerネットワーク

ホストネットワーク

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
cf3b069d6924	bridge	bridge	local
06b5e0fee500	host	host	local
702f20fb0bff	my-network	bridge	local
0c04b44e3b29	none	null	local



Dockerネットワーク

ホストネットワーク

ホストネットワークに接続したコンテナはホストと同じネットワーク設定を持ちます。

コンテナにwebサーバを立ち上げた場合、ホストの80 のポートで待ちうけていると同じ動作になります。そのため、コンテナの起動には-p の指定が不要になります。



Dockerネットワーク

ホストネットワーク

```
$ docker-machine ip network-test  
192.168.99.112
```



Dockerネットワーク

ホストネットワーク

① 192.168.99.112



このサイトにアクセスできません

192.168.99.112 で接続が拒否されました。

次をお試しください:

- 接続を確認する
- [プロキシとファイアウォールを確認する](#)

ERR_CONNECTION_REFUSED



Dockerネットワーク

ホストネットワーク

```
$ docker-machine ssh network-test
```

```
$ docker run -d --name myweb service --network host nginx
```



Dockerネットワーク

ホストネットワーク

192.168.99.112

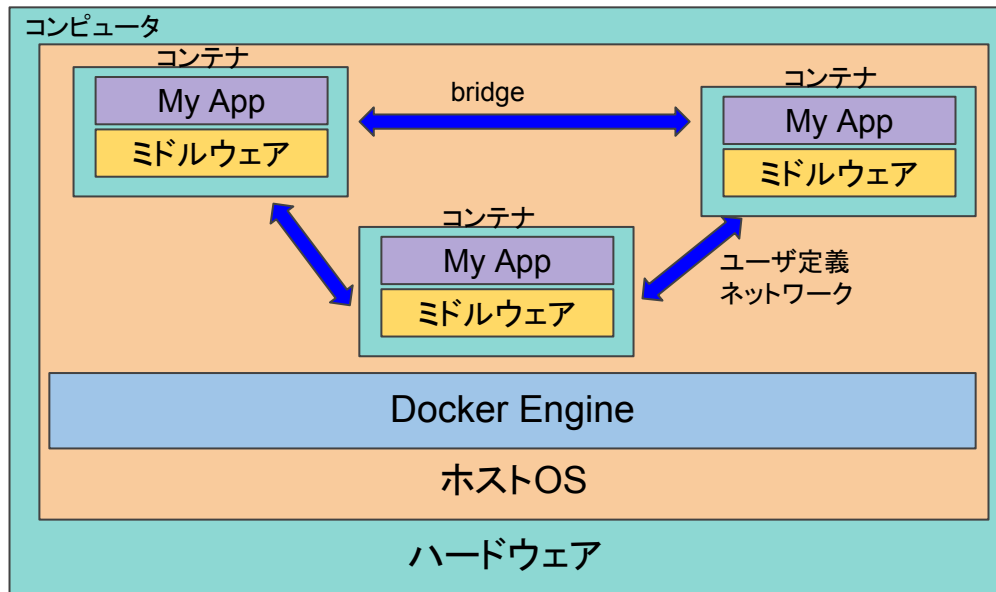
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.


Thank you for using nginx.

Dockerネットワーク





目次

- ・ Dockerとは
- ・ Dockerのインストール
- ・ Dockerコンテナの実行
- ・ Dockerネットワーク
- ・ **Dockerデータ管理** 
- ・ Docker Compose



Dockerデータ管理

データ管理の概要

コンテナで扱う動的なデータはコンテナの書き込みなレイヤ上におくことができます。



Dockerデータ管理

データ管理の概要

ですが

- ・コンテナが削除されたタイミングで消える。
- ・コンテナ間のデータ共有ができない。
- ・書き込みのパフォーマンスもよくないらしい。



Dockerデータ管理

データ管理の概要

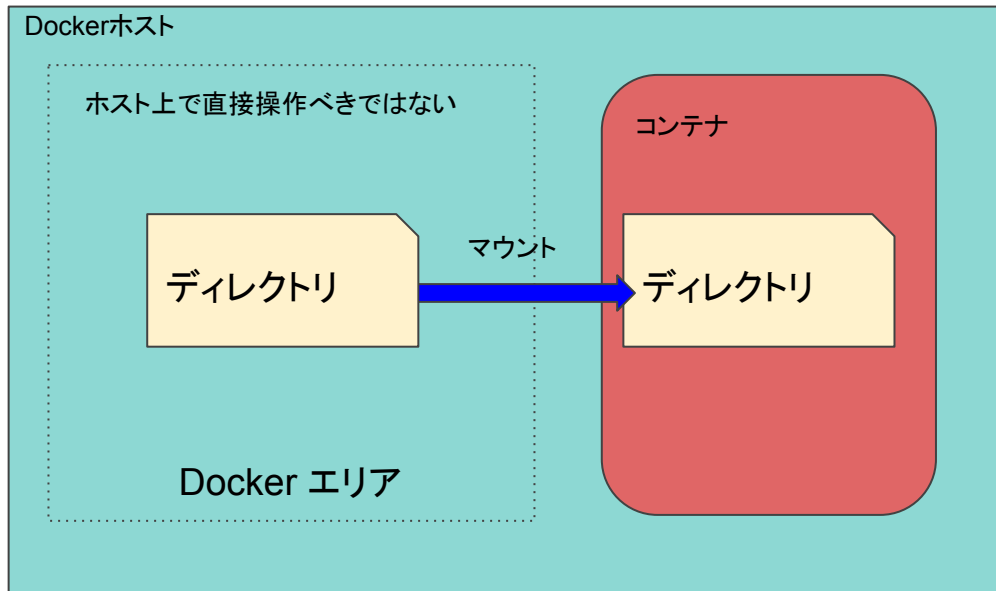
Dockerにはホスト上のファイルやディレクトリをコンテナにマウントする仕組み

ホストのメモリをファイルシステムとしてコンテナにマウントする仕組みが用意されています。

- volumeマウント
- bindマウント
- tmpfs

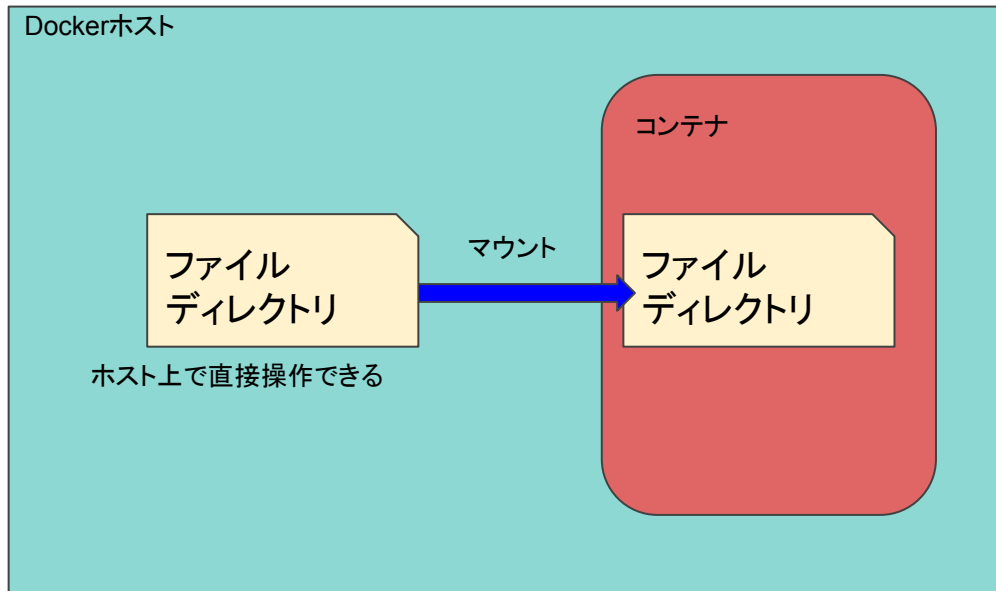
Dockerデータ管理

データ管理の概要 volume



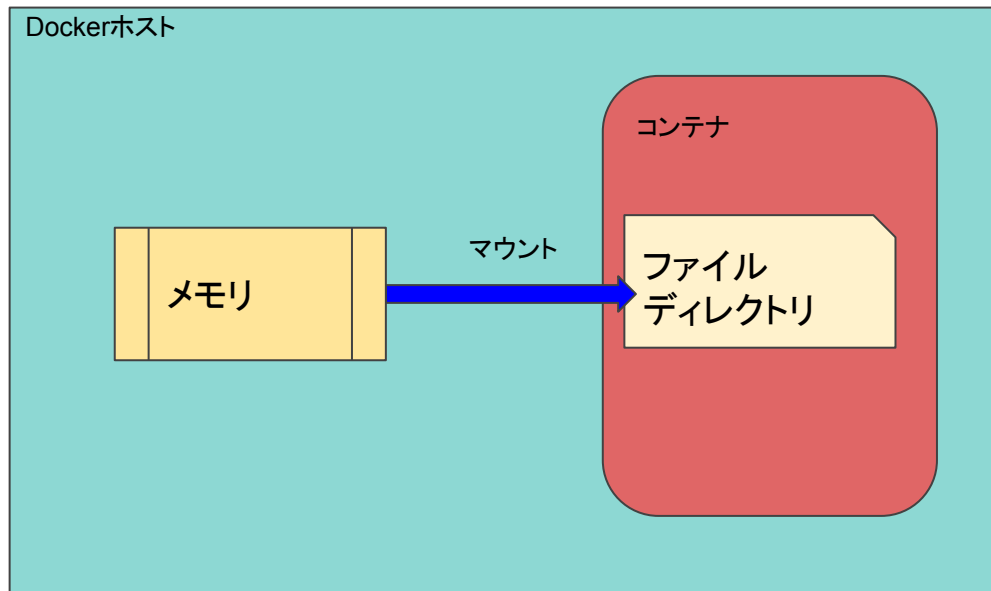
Dockerデータ管理

データ管理の概要 bind mount



Dockerデータ管理

データ管理の概要





Dockerデータ管理

volumeマウント

```
$ docker-machine create vol-test
```

```
$ docker-machine ssh vol-test  
docker@vol-test:~$
```

注意！！！！

以下「vol-test」というVMの中で操作する。



Dockerデータ管理

volumeマウント

```
$ docker volume create my-vol  
my-vol
```

```
$ docker volume ls  
DRIVER          VOLUME NAME  
local           my-vol
```

```
$ docker volume inspect my-vol
```

```
$ docker volume rm my-vol
```




Dockerデータ管理

volumeマウント

```
$ docker run -itd --name mount-test1 --mount source=my-vol,target=/app nginx
```

```
$ docker inspect mount-test1
```

```
$ docker exec -it mount-test1 /bin/bash
```

```
root@eccce8d194b0:/# df
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
------------	-----------	------	-----------	------	------------

...

/dev/sda1	18714044	165900	17559016	1%	/app
-----------	----------	--------	----------	----	------



Dockerデータ管理

volumeマウント

```
$ echo "test by yuntumg." > /app/testfile
```

```
$ docker run -itd --name mount-test2 --mount source=my-vol,target=/app nginx
```

```
$ docker exec -it mount-test2 /bin/bash
```

```
root@7b08ca203600:/#
```

```
root@7b08ca203600:/# cat /app/testfile
```

```
test by yuntumg.
```



Dockerデータ管理

volumeマウント

ただし、volume を共有できるのが同じホスト内のコンテナのみです。

volumeを明示的削除しない限り、データは消えません。



Dockerデータ管理

volumeマウント

```
# マウント先に dir や ファイルが存在する場合は
$ docker run -itd --name mount-test3 --mount source=my-vol2,target=/etc/nginx nginx

$ docker volume inspect my-vol2

$ sudo ls /mnt/sda1/var/lib/docker/volumes/my-vol2/_data
conf.d      koi-utf     mime.types  nginx.conf  uwsgi_params
fastcgi_params koi-win     modules     scgi_params win-utf
```



Dockerデータ管理

volumeマウント

readonly のフラグを指定すれば、volumeを読み取り専用でマウントできます。



Dockerデータ管理

bind mount

volumeはコンテナ上の管理領域であり、ホストから直接操作はNG。

bind mount はホス上上の任意のファイルやディレクトリをマウントでき、ホスト側で直接編集することができます。



Dockerデータ管理

bind mount

```
$ docker run -itd --name bind-test --mount  
type=bind,source="$(pwd)"/bindmount-test,target=/app nginx  
  
$ echo "xxxxxxxx" > bindmount-test/yyyyyyy  
$ docker exec -it bind-test /bin/bash  
root@e3db6d08380c:/# cat /app/yyyyyyy  
xxxxxxxx
```



Dockerデータ管理

tmpfs

ホストのメモリ上の領域をコンテナにマウントするタイプです。

コンテナが停止すると保持しているデータが解放されます。



Dockerデータ管理

tmpfs

```
$ docker run -itd --name tmpfs-test --mount type=tmpfs,target=/app nginx
```

```
$ docker inspect tmpfs-test
```



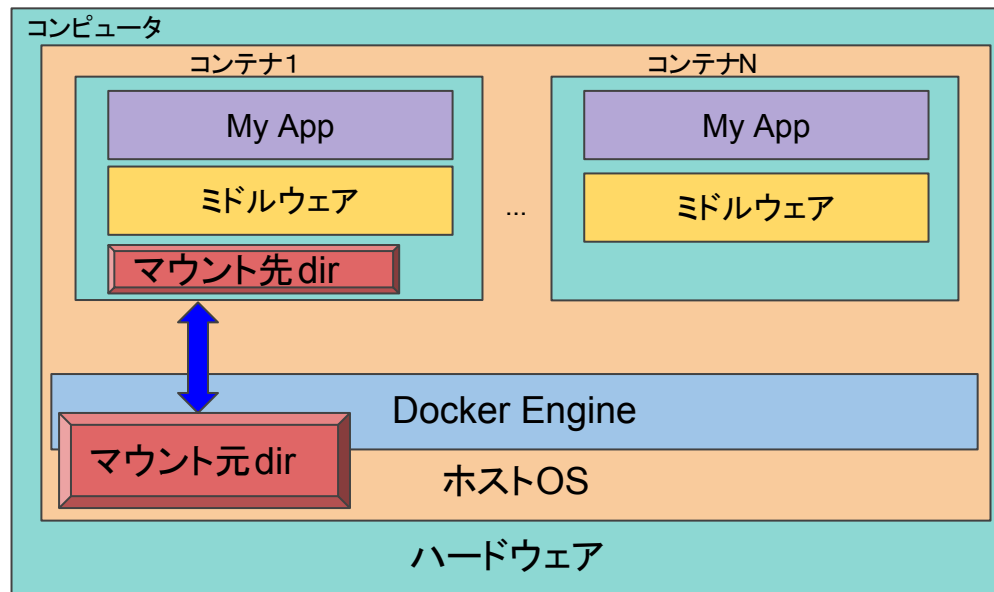
Dockerデータ管理

tmpfs

デフォルトではマウントサイズの制限がなく、メモリが足りなくなったらswap 領域が使われます。


tmpfs-sizeオプションで制限を指定することもできます。

Dockerデータ管理





目次

- ・ Dockerとは
- ・ Dockerのインストール
- ・ Dockerコンテナの実行
- ・ Dockerネットワーク
- ・ Dockerデータ管理
- ・ **Docker Compose** 



Docker Compose

Docker Composeとは

マルチコンテナの Docker アプリケーションを設定ファイルとして定義して、実行するツールです。

設定ファイルは yml 形式で記載します。

開発環境、自動テスト環境などの立ち上げによく使います。

※ yml とは構造化されたデータを表現するためのフォーマット。使ったことがなくても直感的に記述することができますので、ファイルを見れば大体の構造がわかります。



Docker Compose

Docker Composeとは

DBサーバ、アプリサーバ、プロキシサーバなどの定義をファイルに記載して、コンテナをまとめて起動することができます。



Docker Compose

Docker Composeとは

docker compose でアプリケーションを立ち上げる手順

1. Dockerfile or image を用意する。
2. docker-compose.yml ファイルを作成。
3. docker-compose up を実施。



Docker Compose

Docker Composeとは

```
$ docker-compose -v  
docker-compose version 1.23.2, build 1110ad01
```




Docker Compose

wordpress環境作成

サンプルファイルを見ましょう。

https://hub.docker.com/_/wordpress



Docker Compose

wordpress環境作成

```
$ mkdir wp  
$ cd wp/  
$ vi docker-compose.yml
```

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

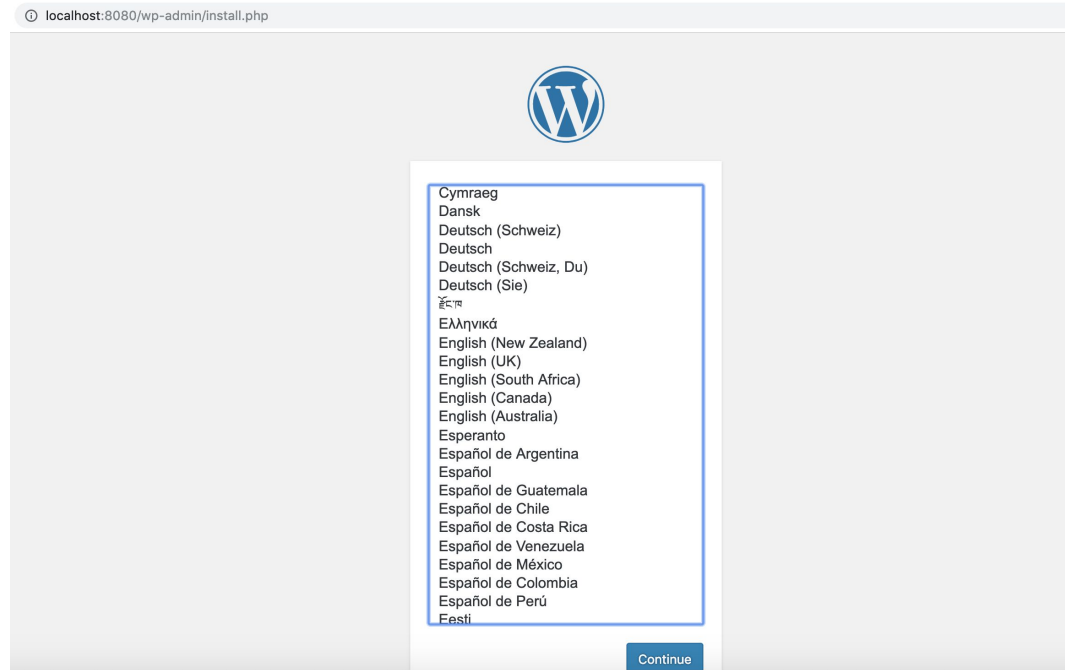
```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
cd12741275cd	wordpress	"docker-entrypoint.s..."	About a minute ago	Up	About a minute	0.0.0.0:8080->80/tcp wp_wordpress_1
34243255439d	mysql:5.7	"docker-entrypoint.s..."	About a minute ago	Up	About a minute	3306/tcp, 33060/tcp wp_db_1



Docker Compose

wordpress環境作成





Docker Compose

wordpress環境作成

ようこそ

WordPress の有名な5分間インストールプロセスへようこそ！以下に情報を記入するだけで、世界一拡張性が高くパワフルなパーソナル・パブリッシング・プラットフォームを使い始めることができます。

必要情報

次の情報を入力してください。ご心配なく、これらの情報は後からいつでも変更できます。

サイトのタイトル

ユーザー名

ユーザー名には、半角英数字、スペース、下線、ハイフン、ピリオド、アットマーク (@) のみが使用できます。

パスワード

普通

重要: ログイン時にこのパスワードが必要になります。安全な場所に保管してください。

メールアドレス

次に進む前にメールアドレスをもう一度確認してください。

検索エンジンでの表示

☐ 検索エンジンがサイトをインデックスしないようにする

このリクエストを尊重するかどうかは検索エンジンの設定によります。



Docker Compose

wordpress環境作成

localhost:8080/wp-login.php



ユーザー名またはメールアドレス
docker

パスワード
●●●●●●●●●●

☐ ログイン状態を保存する

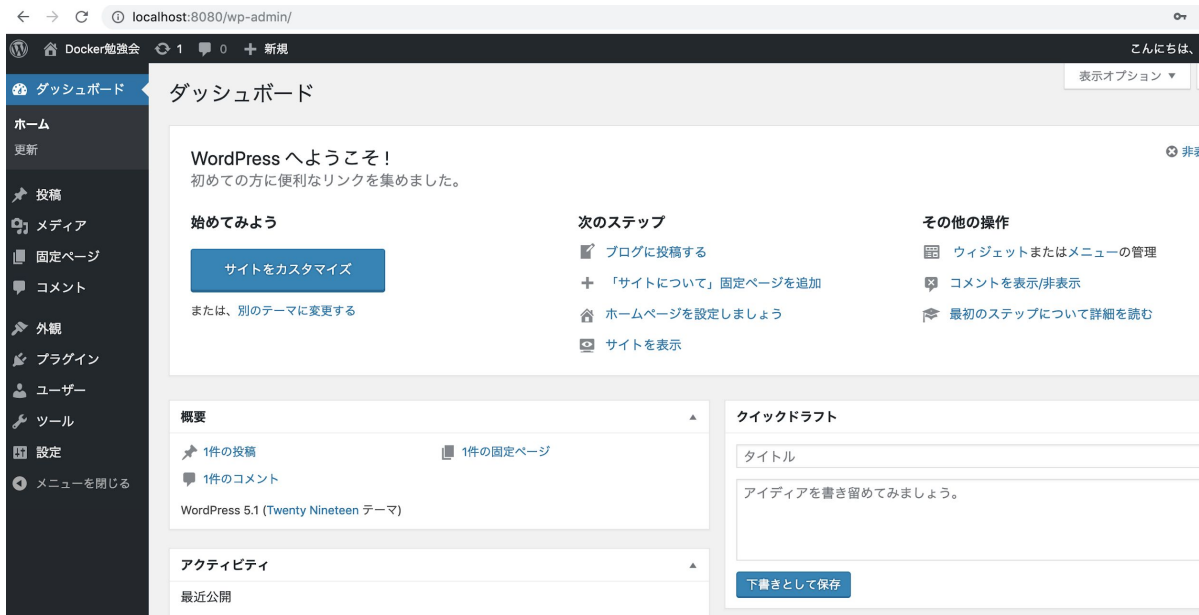
[パスワードをお忘れですか？](#)

[← Docker勉強会に戻る](#)



Docker Compose

wordpress環境作成



The screenshot shows the WordPress dashboard interface. The top navigation bar includes the WordPress logo, the site name 'Docker勉強会', and the user name 'こんにちは、'. The left sidebar contains a menu with options like 'ダッシュボード', 'ホーム', '更新', '投稿', 'メディア', '固定ページ', 'コメント', '外観', 'プラグイン', 'ユーザー', 'ツール', '設定', and 'メニューを開く'. The main content area is titled 'ダッシュボード' and features a welcome message 'WordPress へようこそ!' with a link to 'サイトをカスタマイズ'. Below this, there are sections for '始めてみよう', '次のステップ', and 'その他の操作'. The '概要' (Overview) section shows statistics for posts, pages, and comments. The 'アクティビティ' (Activity) section shows the latest public activity. The 'クイックドラフト' (Quick Draft) section allows for creating a new post.



Docker Compose

モンゴル語wordpress

orhon cms

- ・オープンソース
- ・<https://github.com/dotpub/orhoncms>




Docker Compose

モンゴル語wordpress

开源蒙古文内容管理系统 <http://www.orhoncms.org>

4 commits
2 branches
0 releases
1 contributor
View license

Branch: master
New pull request
Find file
Clone or download


XuYS update read me

Latest commit 0853a7d on 15 Mar 2016

wp-admin	OrhonCMS v0.0.1	3 years ago
wp-content	OrhonCMS v0.0.1	3 years ago
wp-includes	OrhonCMS v0.0.1	3 years ago
.gitignore	OrhonCMS v0.0.1	3 years ago
README.md	update read me	3 years ago
index.php	OrhonCMS v0.0.1	3 years ago
license.txt	OrhonCMS v0.0.1	3 years ago
wp-activate.php	OrhonCMS v0.0.1	3 years ago
wp-blog-header.php	OrhonCMS v0.0.1	3 years ago
wp-comments-post.php	OrhonCMS v0.0.1	3 years ago
wp-config-sample.php	OrhonCMS v0.0.1	3 years ago
wp-cron.php	OrhonCMS v0.0.1	3 years ago
wp-links-opml.php	OrhonCMS v0.0.1	3 years ago



Docker Compose

モンゴル語wordpress

Dockerfile

```
FROM wordpress:latest
```

```
ADD orhoncms-master.tar /var/www/html/
```





Docker Compose

モンゴル語wordpress

← → ↺ ⓘ localhost:8000/wp-login.php

Warning: Illegal string offset 'remember' in `/var/www/html/wp-includes/user.php` on line **39**

Warning: Cannot assign an empty string to a string offset in `/var/www/html/wp-includes/user.php` on line **39**

Warning: Illegal string offset 'user_login' in `/var/www/html/wp-includes/user.php` on line **54**

Fatal error: Uncaught Error: Cannot create references to/from string offsets in `/var/www/html/wp-includes/user.php:54` Stack trace: #0 `/var/www/html/wp-logi`
`wp_signon(", ")` #1 {main} thrown in `/var/www/html/wp-includes/user.php` on line **54**



Docker Compose

モンゴル語wordpress

諦めました。



Docker Compose

モンゴル語wordpress

古い php バージョンの環境で動かすより、orhon cms を最新 php バージョンに対応したい気持ちでいっぱい。

ただし、php ビギナーなので。。。



Docker Compose

各種コマンド

- `docker-compose ps`: docker compose で起動したコンテナ一覧を表示
- `docker-compose run db xxx`: docker compose で起動したコンテナの中でコマンド実行
- `docker-compose stop`: docker compose で起動したコンテナを停止
- `docker-compose start`: docker compose で起動したコンテナを起動
- `docker-compose down`: docker compose で起動したコンテナを停止、削除
Docker Composeで作成したネットワークも一緒に削除します。
-v オプションをつけると作成したvol も削除してくれます。



Docker Compose

各種コマンド

```
$ docker-compose ps
```

Name	Command	State	Ports

wp_db_1	docker-entrypoint.sh mysqld	Up	3306/tcp, 33060/tcp
wp_wordpress_1	docker-entrypoint.sh apach ...	Up	0.0.0.0:8080->80/tcp



Docker Compose

各種コマンド

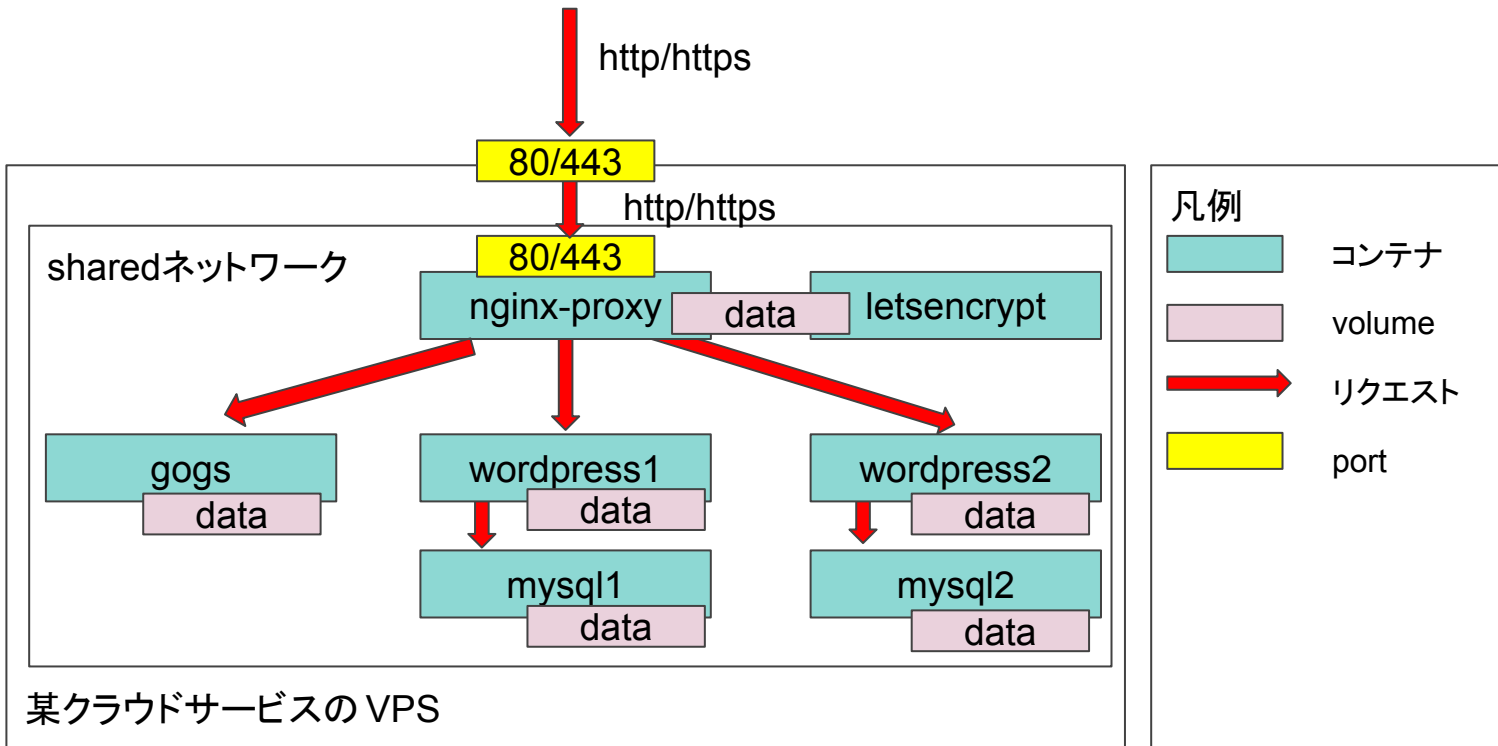
```
$ docker-compose down
Stopping wp_wordpress_1 ... done
Stopping wp_db_1      ... done
Removing wp_wordpress_1 ... done
Removing wp_db_1      ... done
Removing network wp_default
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

最後に

僕のDocker環境





参考情報

qiita記事

【図解】Dockerの全体像を理解する(前編、中編)
その他色々

書籍

Docker/Kubernetesコンテナ開発入門
Dockerを支える技術

udemy

ゼロからはじめるDockerによるアプリケーション実行環境構築

ありがとうございました。

