Table of Contents

README

docker

gitlab

mastodon

はじめに

本資料は　ぺちぱな。14 〜本を正すと基本が大事。という当然の結論〜　の補足資料というかメインコンテンツです。

つたない私の説明をこちらで補います。何しているか分からなくなったらこちらを参照してください。

本イベントでは、「dockerの基本的な使い方からdockerを使った各種サービスの構築ができること」を目指しています。

このイベントが終わった頃には、参加者の方々は次のようになっていることでしょう（願望）

- dockerの使い方が分かる
- GitLabを使ったCI環境を構築できる
- 自分専用mastodonインスタンスを起動してトゥートできる

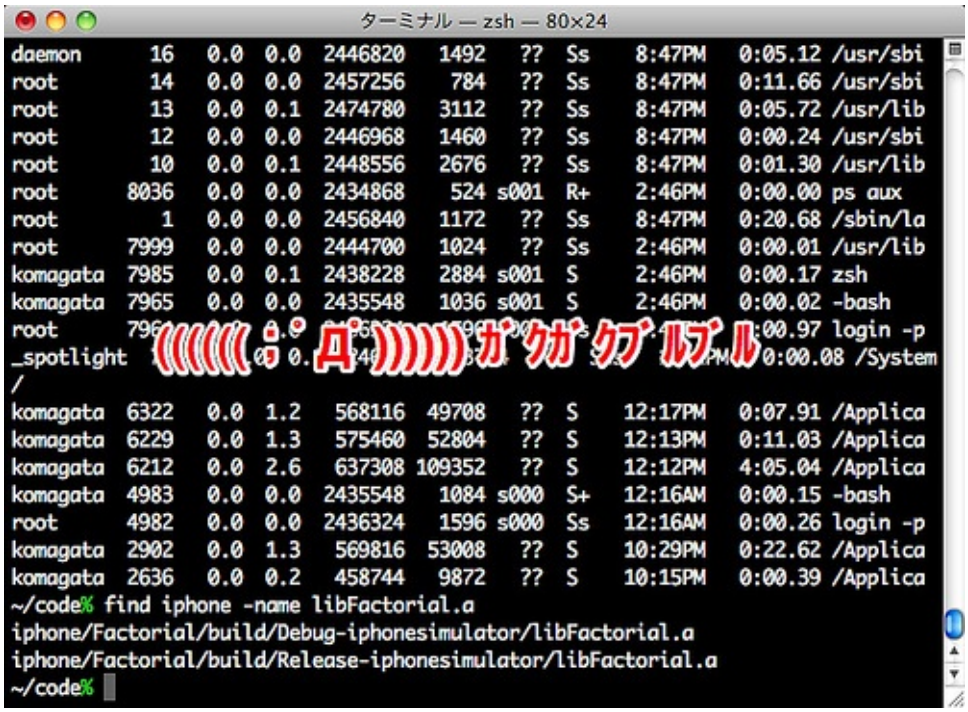
お時間の許す限りお付き合いください！

基礎知識 〜 今日の主役たち

黒い画面（ターミナル）

dockerといえばコレ。いわなくてもコレ。今日はこれしか使いません。ごめんなさい。

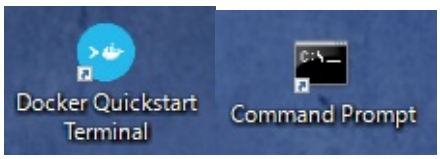
入力したプログラムを実行してくれるナイスガイです。怖がらないでください。



Source: Webデザイナーの為の「本当は怖くない」"黒い画面"入門

下記アイコンをクリックorダブルクリックして起動しておいてください。

windows



mac

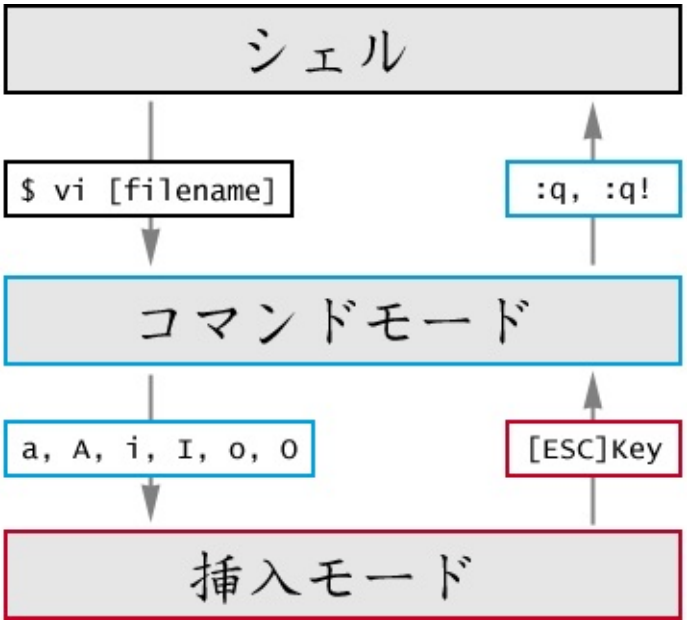
vi（テキストエディタ）

老舗のテキストエディタ。たいていのLinuxにははじめからインストールされている。emacs派と言い争う。

コマンドモードと挿入モードを行き来しながらテキストをいじります。

```
$ vi hoge.txt
```

「わけわかんなくなったら **ESC** キー！」を覚えておくと良いです。



`:q`	**vi** を終了する
`:q!`	バッファでの編集内容を放棄し vi を強制的に終了する
`:w`	バッファの内容をファイルに保存
`:wq`	バッファの内容をファイルに保存し, vi を終了する.

Source: 最低限 vi

docker

今日一番タイピングされる単語。仲良くしてあげてください。

できるだけ新しいバージョンを使うことを推奨します。下記コマンドでバージョンを確かめてください。

古い場合は最新版にアップグレードしましょう。

```
$ docker -v
Docker version 17.05.0-ce, build 89658be
$ docker-compose -v
docker-compose version 1.13.0, build 1719ceb8
```

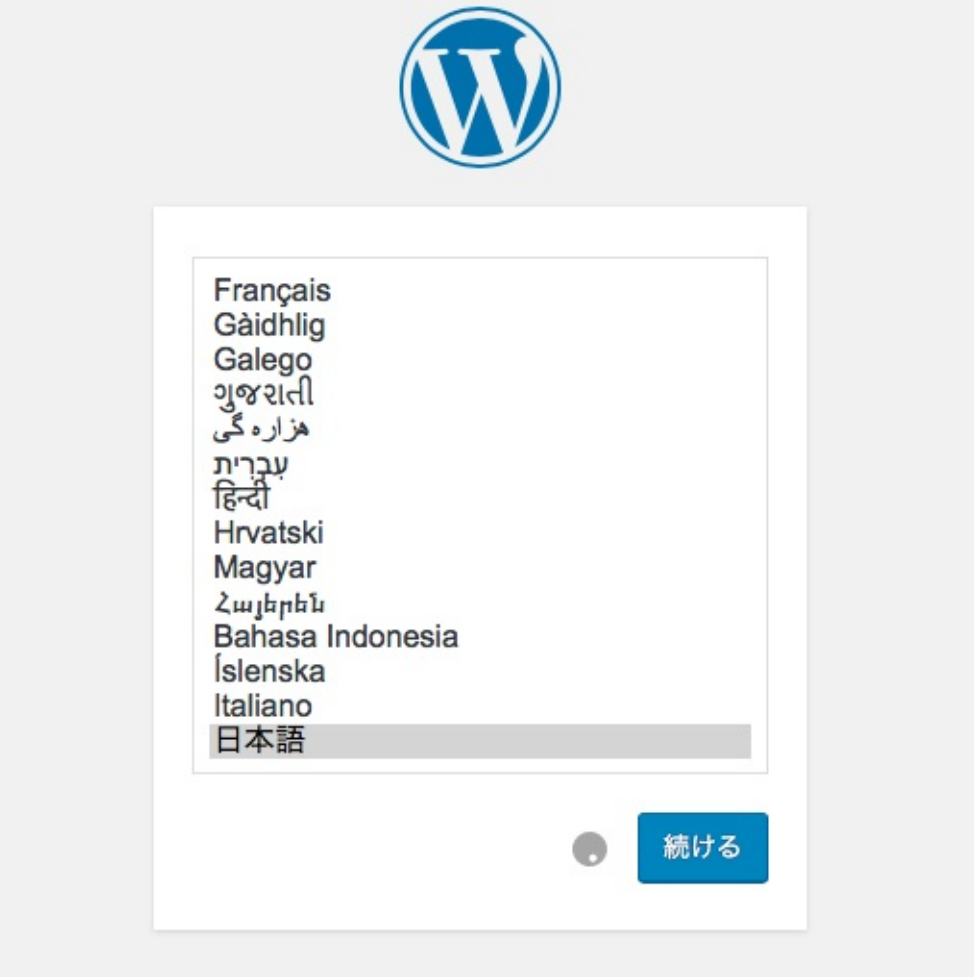
まずはおさわり。

とにかく触ってみないと始まりません。ただ無心にコマンド叩いていきましょう。

wordpressサイトを立ち上げよう！

```
$ cd docker/wordpress
$ ls -la
-rw-r--r--  1 hideAki  staff        184  6  2 23:01 .env
-rw-r--r--  1hideAki  staff        394  6  2 23:33 docker-compos
e.yml
-rw-r--r--  1 hideAki  staff  414653440  6  2 23:39 mysql.tar
-rw-r--r--  1 hideAki  staff  432754176  6  2 23:32 wordpress46.
tar
-rw-r--r--  1 hideAki  staff  418963968  6  2 23:31 wordpress47.
tar
$ docker load < wordpress46.tar
$ docker load < wordpress47.tar
$ docker load < mysql.tar
$ docker-compose up -d
```

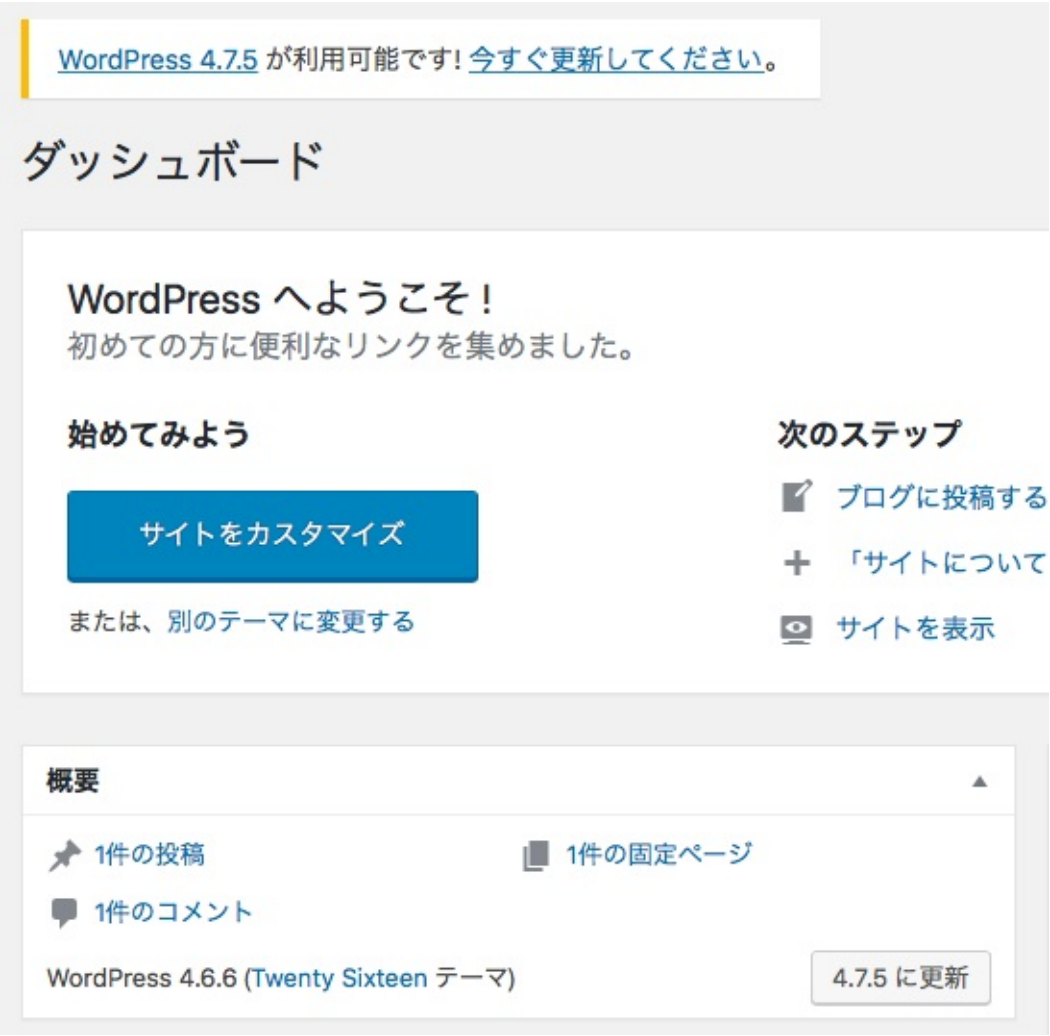
ブラウザを立ち上げて http://localhost へアクセス！



wordpressの初期設定画面が立ち上がってきましたか？？資格情報を登録して管理者画面までいきましょう。

バージョンアップ

おや？wordpressのバージョンが古いみたいです。



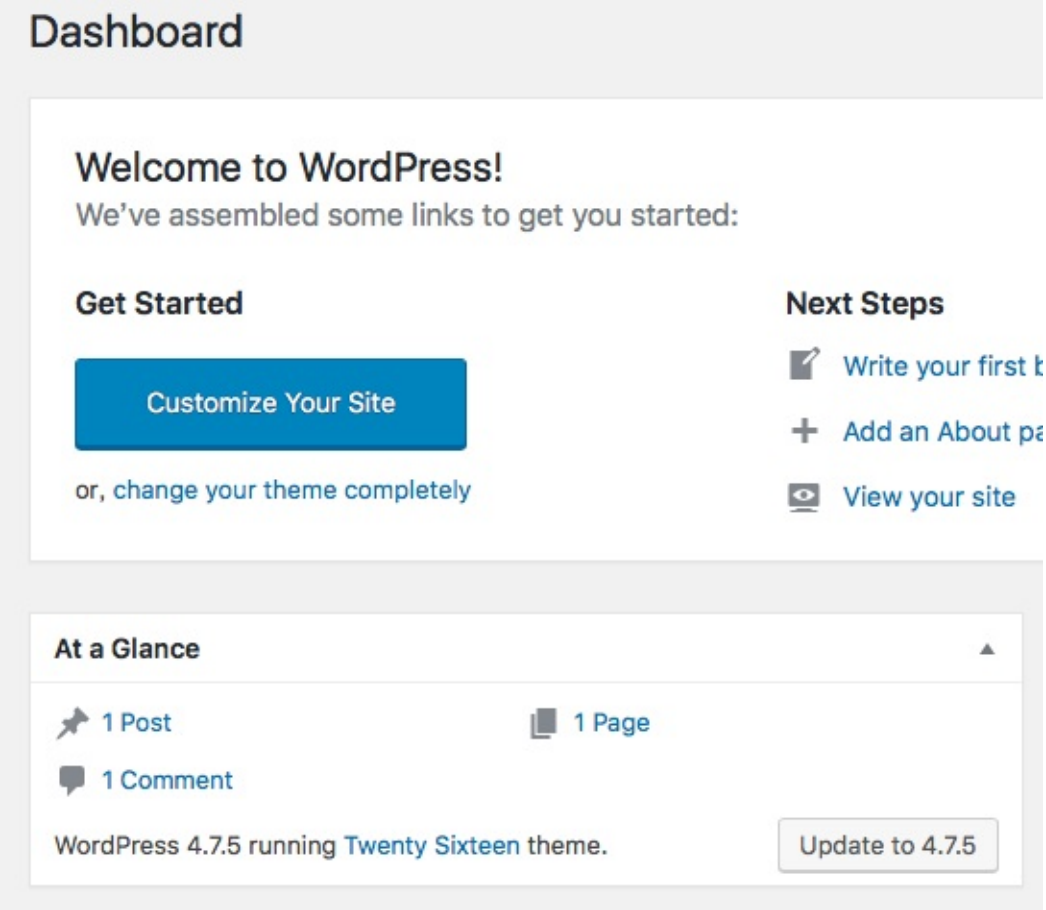
バージョンアップしてみましょう。

```
$ vi docker-compose.yml
version: "2"
services:
  wordpress:
    image: wordpress:4.6 --> この記述を "image: wordpress:latest"
に変更
    ports:
      - "80:80"
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: "db:3306"
    networks:
      - flat-network
    env_file: .env
  db:
    image: mysql:5.7
    volumes:
      - "db-data:/var/lib/mysql"
    networks:
      - flat-network
    env_file: .env
volumes:
  db-data:
networks:
  flat-network:

$ docker-compose stop
Stopping wordpress_wordpress_1 ... done
Stopping wordpress_db_1 ... done

$ docker-compose rm
Removing wordpress_wordpress_1 ... done
Removing wordpress_db_1 ... done

$ docker-compose up -d
```

ふたたびブラウザを立ち上げて http://localhost へアクセス！

お、データベースのアップデートを聞いてきました。

めでたく最新版にアップデートされたようです！



お手軽ですね！

基本の"き"

さて、いきなりなんのこっちゃわからんことをしてきたので基本からみていきましょう。

イメージを探す、持って来る。

dockerはコンテナイメージを手元において、設定通りの環境をいつでも再現できる仕組みです。
まずはコンテナイメージを探して、ダウンロードする必要があります。

search コマンドでイメージを探します。

```
$ docker search alpine
NAME                                                     DESCRIPTION
                                    STARS     OFFICIAL   AUTOMATED
alpine                                                   A minimal Dock
er image based on Alpine Lin...   2241      [OK]
...
```

実際に探すときは https://hub.docker.com で探すことになるでしょう。

それではイメージを **pull** コマンドでダウンロードします。

```
$ docker pull alpine
latest: Pulling from library/alpine
2aecc7e1714b: Pull complete
Digest: sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d8
1490948876e462c
Status: Downloaded newer image for alpine:latest
```

イメージがダウンロードされて登録されていることを **image** コマンドで確認します。

```
$ docker images
REPOSITORY            TAG                 IMAGE ID            CR
EATED             SIZE
alpine                latest              a41a7446062d        7
days ago          3.97M
wordpress             4.6                 ca96afcfa242        2
weeks ago         406MB
wordpress             latest              ca96afcfa242        2
weeks ago         406MB
mysql                 5.7                 e799c7f9ae9c        3
weeks ago         407MB
```

先程 save コマンドで登録したイメージも登録されていますね。

なにはともあれ、hello world!

そう、定番のやつですね。

```
$ docker run alpine echo hello world!
hello world!
```

やった！やりました！

コンテナのいじり方

ここではコンテナの各種操作を体験してみましょう。

docker run - コンテナの起動

Docker コンテナの起動のためのコマンドです。

```
$ docker run alpine echo hello world!
hello world!
```

このコンテナは `echo "Hello Docker"` を実行して終了しています。
この時、ローカルにイメージが存在しない場合、自動的にDocker Hubからイメージを取得した後、コンテナを起動します。

`docker run` を実行するときに `-i（キーボード入力可能）` と `-t（ttyを確保する）` のオプションをつけると実行したターミナル上でコンテナを操作することができます。まとめて `-it` で記述できます。

```
$ docker run -it alpine /bin/sh
# uname -a
Linux 568b642580eb 4.9.30-moby #1 SMP Wed May 31 05:30:34 UTC 20
17 x86_64 Linux
# exit
```

`exit` を入力すると、コンテナのターミナルを終了し、ホストターミナルに戻ります。
実行した `sh` が終了したためコンテナも停止します。

docker ps - コンテナの稼働状況確認

現在稼働しているコンテナの一覧を見ることができます。先程動かしたwordpressのイメージが動いていますね。

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS              PORTS                  NA
MES
bcf000b1a41c        wordpress:latest    "docker-entrypoint..."
 2 hours ago         Up 2 hours          0.0.0.0:80->80/tcp   wo
rdpress_wordpress_1
8626ff5ac374        mysql:5.7           "docker-entrypoint..."
 2 hours ago         Up 2 hours          3306/tcp             wo
rdpress_db_1
```

`-a` オプションをつけて実行すると、先程起動したコンテナが Exitedステータスで残っていることがわかります。

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS                         PORTS
        NAMES
568b642580eb        alpine              "/bin/sh"
 2 minutes ago       Exited (127) 19 seconds ago
        mystifying_kare
bcf000b1a41c        wordpress:latest    "docker-entrypoint..."
 2 hours ago         Up 2 hours                     0.0.0.0:80->8
0/tcp   wordpress_wordpress_1
8626ff5ac374        mysql:5.7           "docker-entrypoint..."
 2 hours ago         Up 2 hours                     3306/tcp
        wordpress_db_1
```

dockerのコンテナは終了した後も明示的に指示しない限り残り続けます。これはコンテナに対して少しでも作業が行われたものはイメージとして利用する可能性があるため、残すように配慮されているものと思われます。実際に、Exitedなコンテナから新たにイメージを作成することもできます。

docker inspect - コンテナ情報の取得

すぐに終了するコンテナと、動き続けるコンテナ。どんな違いがあるのでしょう。
これはイメージの起動時に実行されるコマンドに違いが有ります。`inspect`コマンドで確認してみましょう。

```
$ docker inspect alpine
[
...
        "Config": {
            "Hostname": "9ac68176ac52",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/
usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/bin/sh\"]"
            ],
            "ArgsEscaped": true,
            "Image": "sha256:a96393421091145abdc0ce8f02691166ed0
fe7f769b4dfc7f700b4b11d4a80df",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
...
]

$ docker inspect wordpress | grep CMD
                "CMD [\"apache2-foreground\"]"
```

`CMD` で指定されているものが違います。alpineが `sh` であるのに対し、wordpressは `apache` が指定されています。
apacheは起動後もhttpリクエストを待ち続けるのでコンテナは終了せずUPステータスになっているようです。

`inspect` コマンドはコンテナだけでなく、dockerで扱われる様々なオブジェクトに対して利用できます。
volumeの詳細を確認する例は次のとおりです。

```
$ docker volume ls
DRIVER              VOLUME NAME
local               24e8addfe7e17b0b32edb89a0110bc73fdb89b74ebe4
7b0fc5d0c755ec61e465
local               wordpress_db-data

$ docker inspect wordpress_db-data
[
    {
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/wordpress_db-data
/_data",
        "Name": "wordpress_db-data",
        "Options": {},
        "Scope": "local"
    }
]
```

docker exec - コンテナの稼働状況確認

execを使うと起動中のコンテナに接続してコマンド操作が行えます。名前もしくはコンテナIDで接続できます。
どちらも `docker ps` コマンドで確認できます。

コンテナ内で `ps -el` を実行するとapache2が動作していることが実際に分かります。

```
$ docker exec -it wordpress_wordpress_1 /bin/bash
# ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIM
E CMD
4 S     0     1     0  0  80   0 - 78770 -      ?        00:00:0
0 apache2
# exit
```

docker stop & rm - コンテナの終了と削除の仕方

サービス化したプロセスが動作するコンテナはそのまま動き続けます。終了するには `stop` コマンドを使います。

```
$ docker stop wordpress_wordpress_1
wordpress_wordpress_1
$ docker stop wordpress_db_1
wordpress_db_1
```

コンテナの状態を確認します。

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS                          PORTS
       NAMES
568b642580eb        alpine              "/bin/sh"
 About an hour ago   Exited (137) 22 seconds ago
       mystifying_kare
bcf000b1a41c        wordpress:latest    "docker-entrypoint..."
 3 hours ago         Exited (0) 3 minutes ago
       wordpress_wordpress_1
8626ff5ac374        mysql:5.7           "docker-entrypoint..."
 3 hours ago         Exited (0) 10 seconds ago
       wordpress_db_1
```

残ったもので不要なコンテナは `rm` コマンドで削除できます。稼働中のコンテナは削除できませんのであしからず。

```
$ docker rm mystifying_kare
mystifying_kare

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
 CREATED             STATUS                      PORTS
    NAMES
bcf000b1a41c        wordpress:latest    "docker-entrypoint..."
 3 hours ago         Exited (0) 9 minutes ago
    wordpress_wordpress_1
8626ff5ac374        mysql:5.7           "docker-entrypoint..."
 3 hours ago         Exited (0) 6 minutes ago
    wordpress_db_1
```

wordprssとmysqlのコンテナはもう少しとっておきましょう。

docker system prune - お掃除の仕方

dockerはその性質上、知らない間に色々なデータを残します。意外とリソースに優しくありません。

どれだけリソースを消費しているか `docker system df` コマンドで確認しましょう。

```
$ docker system df
TYPE                TOTAL               ACTIVE              SIZE
                RECLAIMABLE
Images              12                  2                   4.87
GB              4.304GB (88%)
Containers          2                   0                   187B
                187B (100%)
Local Volumes       2                   2                   232.
4MB             0B (0%)
```

最近のdockerさんは自信が管理するリソース（イメージ、コンテナ、ボリューム、ネットワーク）のうちどこからも参照されていないものを一気に削除できるコマンド `docker system prune` を用意してくれました！

```
$ docker system prune
WARNING! This will remove:
    - all stopped containers
    - all volumes not used by at least one container
    - all networks not used by at least one container
    - all dangling images
Are you sure you want to continue? [y/N] y
Deleted Containers:
bcf000b1a41cad0d98753f44bf83ba332ce0eae2f8336158c4bf96c40af10331
8626ff5ac374332c41a1e02df603736e873e394f74cbea6750ef9d628cc5da01

Deleted Volumes:
24e8addfe7e17b0b32edb89a0110bc73fdb89b74ebe47b0fc5d0c755ec61e465
wordpress_db-data

Deleted Networks:
wordpress_flat-network

Total reclaimed space: 232.4MB
```

これはうれしい機能です！まめに実行しておいてください。

イメージの作り方

こではイメージの作り方を具体的に体験してみましょう。

準備

```
$ cd ../whalesay/
$ ls -la

$ docker load < whalesey.tar
$ docker run docker/whalesay cowsay phper-nya!
 _____________
< paper-nya! >
 -------------
    \
     \
      \
                    ##         .
              ## ## ##        ==
           ## ## ## ##       ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

くじらさんがしゃべった！！！！！！！

このコンテナを拡張してイメージを作ろうと思います。

cowsayコマンドの使い方

cowの一覧は `-l` オプションで見ることができます。

```
$ docker run docker/whalesay cowsay -l
/usr/local/share/cows:
beavis.zen bong bud-frogs bunny cheese cower daemon default dock
er dragon
dragon-and-cow elephant elephant-in-snake eyes flaming-sheep gho
stbusters
head-in hellokitty kiss kitty koala kosh luke-koala meow milk mo
ofasa moose
mutilated ren satanic sheep skeleton small sodomized squirrel st
egosaurus
stimpy supermilker surgery telebears three-eyes turkey turtle tu
x udder
vader vader-koala www
```

cowは `-f` オプションで変更できます（うしさんになった）

```
$ docker run docker/whalesay cowsay -f www phper-nya!
 _____________
< phper-nya! >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||--WWW |
                ||     ||
```

デフォルトのcowはファイルを書き換えて変更できます（コンテナ内）

```
# cp /usr/local/share/cows/www.cow /usr/local/share/cows/default
.cow
```

docker commit - 起動したコンテナからイメージを作成

いちばん手っ取り早くイメージを作成する方法は docker commit を使う方法です。
イメージ内に接続してデフォルトのcowを変更してみます。

```
$ docker run -it docker/whalesay /bin/bash
# cp /usr/local/share/cows/www.cow /usr/local/share/cows/default
.cow
# exit
```

実行が完了したコンテナのIDを確認します。

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND               CR
EATED             STATUS                          PORTS
      NAMES
45040acafc72        docker/whalesay     "/bin/bash"           9
minutes ago       Up 5 minutes
      dazzling_roentgen
```

docker commit を使ってイメージを作りましょう。コンテナIDもしくは名前で指定しましょう。

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

```
$ docker commit dazzling_roentgen hideaki/cowsay
$ docker images
REPOSITORY            TAG                 IMAGE ID            C
REATED             SIZE
hideaki/cowsay        latest              256199fb2266        2
0 minutes ago      247MB
```

イメージが登録されています！早速実行してみましょう。

```
$ docker run hideaki/cowsay cowsay Moo
 _____
< Moo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||--WWW |
                ||     ||
```

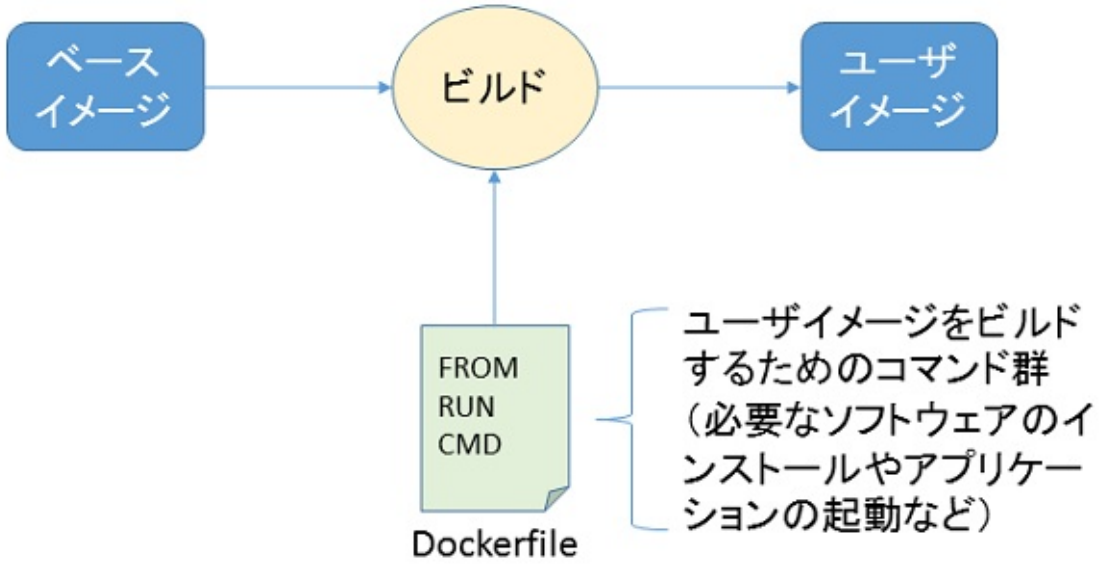
うしさんに変更されたcowsayイメージになりました！

Dockerfile - ファイルに記載した内容をイメージ化

commitは手っ取り早く作ることができますが、手作業が入るため同じイメージを作ることは二度とできません。

そこで突如出現する**Dockerfile**。

これは Infrastructure as code の考え方に則り、構築手順をファイル化しファイル内容に沿ったイメージを作成することができます。



Source: さわって理解するDocker入門

詳しい書式は　Dockerfile リファレンス　や　Dockerfileの命令を理解して、よりDockerを有効活用したい！を参考にしてください。ここでは最低限の説明に留めます。

Dockerfileのコマンド一覧（抜粋）

コマンド	意味	補足
FROM	ベースイメージとなるイメージを指定する	
RUN	イメージをビルドするためのコマンドを指定する	「RUN apk add –update py-pip」の場合、実際には「apk add –update py-pip」の部分が実行される
COPY	ホストからコンテナ内にファイルをコピーする	
CMD	イメージからコンテナを起動するときに実行するコマンドを指定する	Dockerfile内でCMDは1個だけ指定できる

カスタムイメージを作ってみよう！

Dockerfileを利用してカスタムイメージを作ってください。ただし下記条件は守ってください。

- 好きなcowに変更する
- runにコマンドの指定（cowsay hoge）がない場合は何か適当に喋る。

```
FROM docker/whalesay:latest

RUN

CMD ["", ""]
```

Dockerfileが作成できたら下記コマンドでイメージをビルドします。`-t`オプションでタグを付けておきます。

```
$ docker build -t hideaki/cowsay .
$ docker images
REPOSITORY             TAG                 IMAGE ID            C
REATED             SIZE
hideaki/cowsay         latest              819ee733dd50        1
5 minutes ago      247MB
$ docker run hideaki/cowsay
 _____
< boo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||--WWW |
                ||     ||
```

これであなたもイメージマスター！かも。

お掃除 `docker system prune` もお忘れなく。

イメージの連携

docker compose

WordPressをdockerで構築する場合、WordPress 本体が置かれているアプリケーション・サーバ（webサーバ+PHP）と MySQL サーバという大まかに二つのコンテナで構成されることになります。

docker compose とは、複数の Docker コンテナからなるサービスを構築・実行する手順をひとまとめに定義して自動化するという機能です。

普通であればそれらコンテナごとに個別に起動し、さらにそれぞれを関連付ける(`link` オプション) 必要があります。
起動する順序にも気を使わなくてはなりません。コンテナの数が増えてくると面倒になってしまいます。

docker compose は `docker-compose.yml` というファイルで構成を定義します。
YAML で体系的に構成を定義出来るので全体像の見通しが良いのが特徴です。

docker run時のオプションをほぼ同じように定義できる上、コマンド1回実行するだけで複数のコンテナサービスを全て起動できます。

Source: docker-compose を使って WordPress テーマ開発環境を構築しよう

docker-compose.yml

```
$ cd ../wordpress/
$ vi docker-compose.yml
version: "2"

services:

  wordpress:
    image: wordpress:latest
    ports:
      - "80:80"
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: "db:3306"
    networks:
      - flat-network
    env_file: .env

  db:
    image: mysql:5.7
    volumes:
      - "db-data:/var/lib/mysql"
    networks:
      - flat-network
    env_file: .env

volumes:
  db-data:

networks:
  flat-network:
```

.env

```
WORDPRESS_DB_NAME=wordpress
WORDPRESS_DB_USER=wp_user
WORDPRESS_DB_PASSWORD=hogehoge
MYSQL_RANDOM_ROOT_PASSWORD=yes
MYSQL_DATABASE=wordpress
MYSQL_USER=wp_user
MYSQL_PASSWORD=hogehoge
```

起動

```
$ docker-compose up -d
$ docker-compose ps
        Name                       Command               State
       Ports
-------------------------------------------------------------------
------------------
wordpress_db_1          docker-entrypoint.sh mysqld      Up
 3306/tcp
wordpress_wordpress_1   docker-entrypoint.sh apach ...   Up
 0.0.0.0:80->80/tcp
```

終了（コンテナの停止／削除およびボリュームの削除）

```
$ docker-compose down -v
```

もし**docker-compse**を使わなかったら**...**

```
$ docker run --name db -v db-data:/var/lib/mysql -e MYSQL_USER=w
p_user -e MYSQL_PASSWORD=hogehoge -e MYSQL_DATABASE=wordpress -d
 mysql:5.7
$ docker run -p 80:80 -e WORDPRESS_DB_PASSWORD=hogehoge -d --nam
e wordpress --link db:mysql wordpress
$ docker stop wordpress
$ docker stop db
$ docker rm wordpress
$ docker rm db
$ docker volume rm db-data
```

環境設定を変えてみよう！

docker-compose.ymlの設定を変更して実行される環境を変えてみましょう。ただし下記条件は守ってください。

- ブラウザからアクセスされるポート番号は9090
- DBのデータ保存先を実行中のフォルダ内に指定する

```
$ docker-compose up -d
Creating network "wordpress_flat-network" with the default drive
r
Creating volume "wordpress_db-data" with default driver
Creating wordpress_db_1 ...
Creating wordpress_db_1 ... done
Creating wordpress_wordpress_1 ...
Creating wordpress_wordpress_1 ... done

$ ls -la
-rw-r--r--  1 hideAki  staff        184  6  2 23:01 .env
drwxr-xr-x@ 9 hideAki  staff        306  6  3 05:57 db-data <--
できている
-rw-r--r--  1 hideAki  staff        401  6  3 05:56 docker-compo
se.yml
-rw-r--r--  1 hideAki  staff  414653440  6  2 23:39 mysql.tar
-rw-r--r--  1 hideAki  staff  432754176  6  2 23:32 wordpress46.
tar
-rw-r--r--  1 hideAki  staff  418963968  6  2 23:31 wordpress47.
```

```
tar

$ ls -la db-data/
total 376912
drwxr-xr-x@  20 hideAki  staff       680  6  3 05:57 .
drwxr-xr-x    8 hideAki  staff       272  6  3 06:00 ..
-rw-r-----    1 hideAki  staff        56  6  3 05:57 auto.cnf
-rw-------    1 hideAki  staff      1679  6  3 05:57 ca-key.pem
-rw-r--r--    1 hideAki  staff      1074  6  3 05:57 ca.pem
-rw-r--r--    1 hideAki  staff      1078  6  3 05:57 client-cert
.pem
-rw-------    1 hideAki  staff      1679  6  3 05:57 client-key.
pem
-rw-r-----    1 hideAki  staff      1321  6  3 05:57 ib_buffer_p
ool
-rw-r-----    1 hideAki  staff  50331648  6  3 05:57 ib_logfile0
-rw-r-----    1 hideAki  staff  50331648  6  3 05:56 ib_logfile1
-rw-r-----    1 hideAki  staff  79691776  6  3 05:57 ibdata1
-rw-r-----    1 hideAki  staff  12582912  6  3 05:58 ibtmp1
drwxr-x---   77 hideAki  staff      2618  6  3 05:57 mysql
drwxr-x---   90 hideAki  staff      3060  6  3 05:57 performance
_schema
-rw-------    1 hideAki  staff      1679  6  3 05:57 private_key
.pem
-rw-r--r--    1 hideAki  staff       451  6  3 05:57 public_key.
pem
-rw-r--r--    1 hideAki  staff      1078  6  3 05:57 server-cert
.pem
-rw-------    1 hideAki  staff      1679  6  3 05:57 server-key.
pem
drwxr-x---  108 hideAki  staff      3672  6  3 05:57 sys
drwxr-x---    3 hideAki  staff       102  6  3 05:57 wordpress

$ docker-compose down -v
```

`docker-compose` の使い方、イメージできましたか？

（補足）データボリュームの指定書式について

公式リファレンスに記載があります。参考にしてください

Compose ファイル・リファレンス

GitLabとは？

本章のゴール

- GitLabをローカルに立てて、ソースとコンテナの管理ができる
- ローカルの自動CI環境を構築できる

GitLab（ギットラボ）

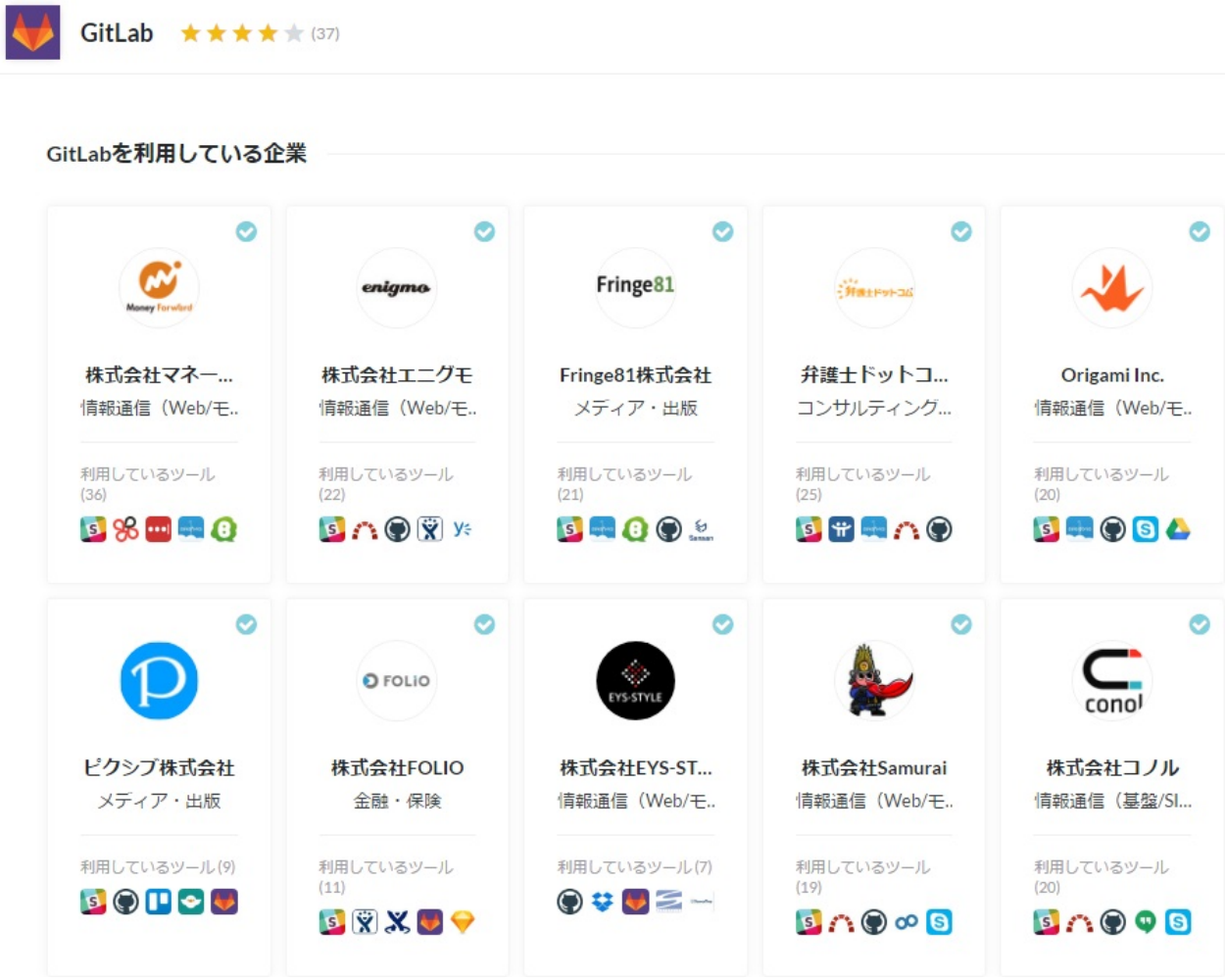
ソフトウェア開発支援環境です。「GitHub」のようなサービスを社内などのクローズド環境に独自で構築できるGitリポジトリマネージャーです。Gitベースのソースコード管理機能、マージリクエスト、レビュー機能なども備えています。

さらにCI機能・コンテナレジストリ機能などが追加され、オールインワンのCI/DIソリューションとして注目されています。

ローカル運用も可能なため、セキュリティおよびコスト面で導入が難しかった企業でも使われるようになっています。

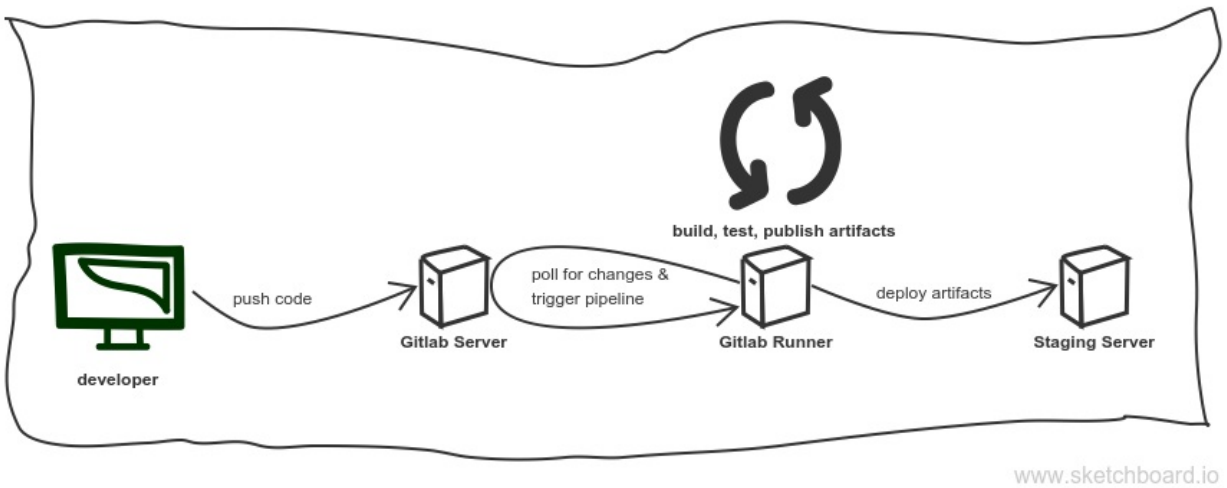
GitLabを導入している企業

結構ありますね。

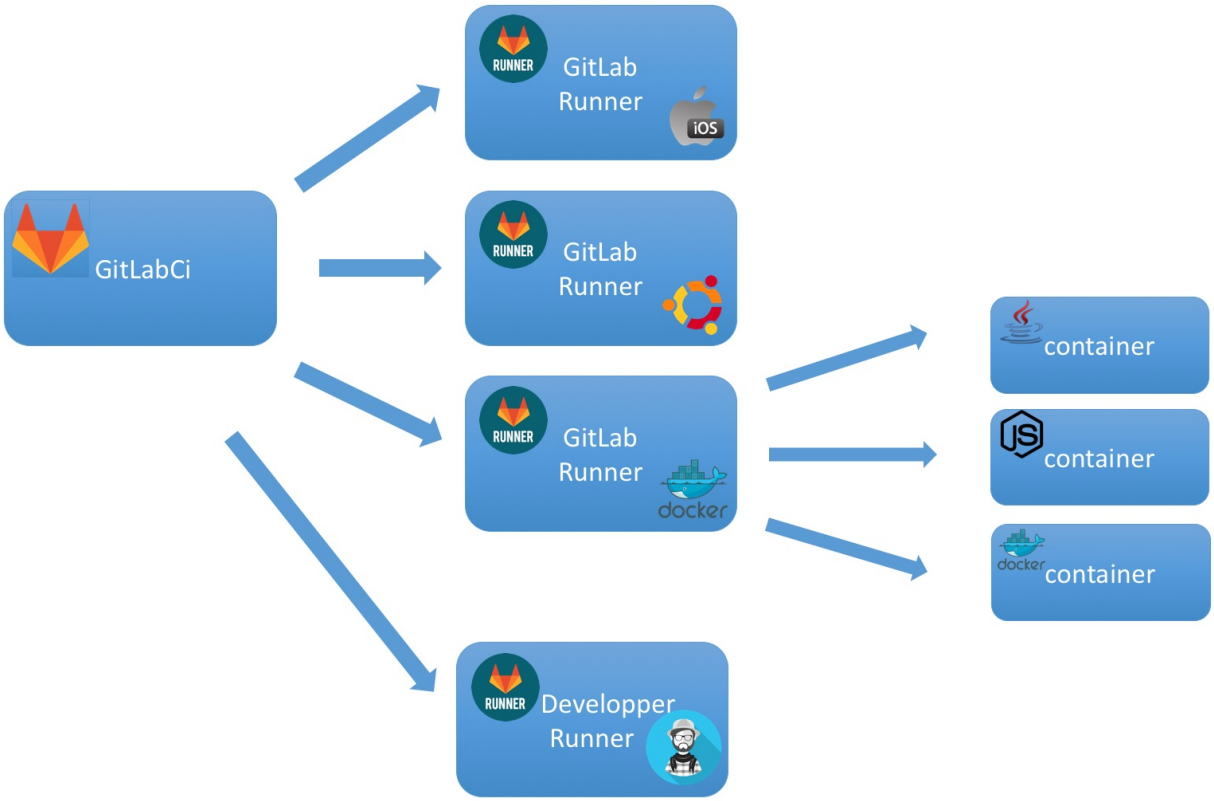


Source: GitLabを利用している企業

システム構成



Source: Continuous Delivery with GitLab CI and Ansible (part 1)



Source: Coding the Next Build

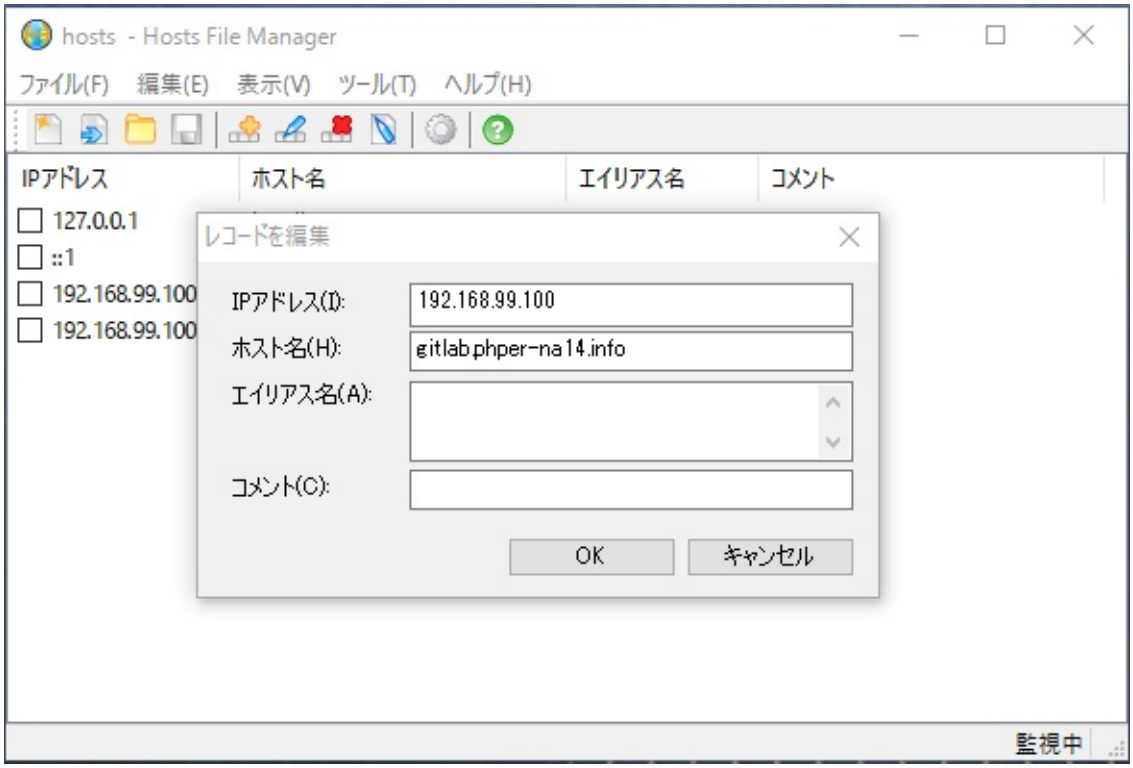
準備

gitlabへのアクセスに使うドメイン名（phper-na14.info）を名前解決できるよう設定を行います。

hostsファイルの編集（ホスト）

windows

- Hosts File Manager




```
192.168.99.100 gitlab.phper-na14.info
192.168.99.100 registry.phper-na14.info
192.168.99.100 ci.phper-na14.info
```

mac

```
$ sudo vi /etc/hosts
127.0.0.1 gitlab.phper-na14.info registry.phper-na14.info ci.php
er-na14.info
```

hostsファイルの編集（ゲスト）***docker toolbox** のみ

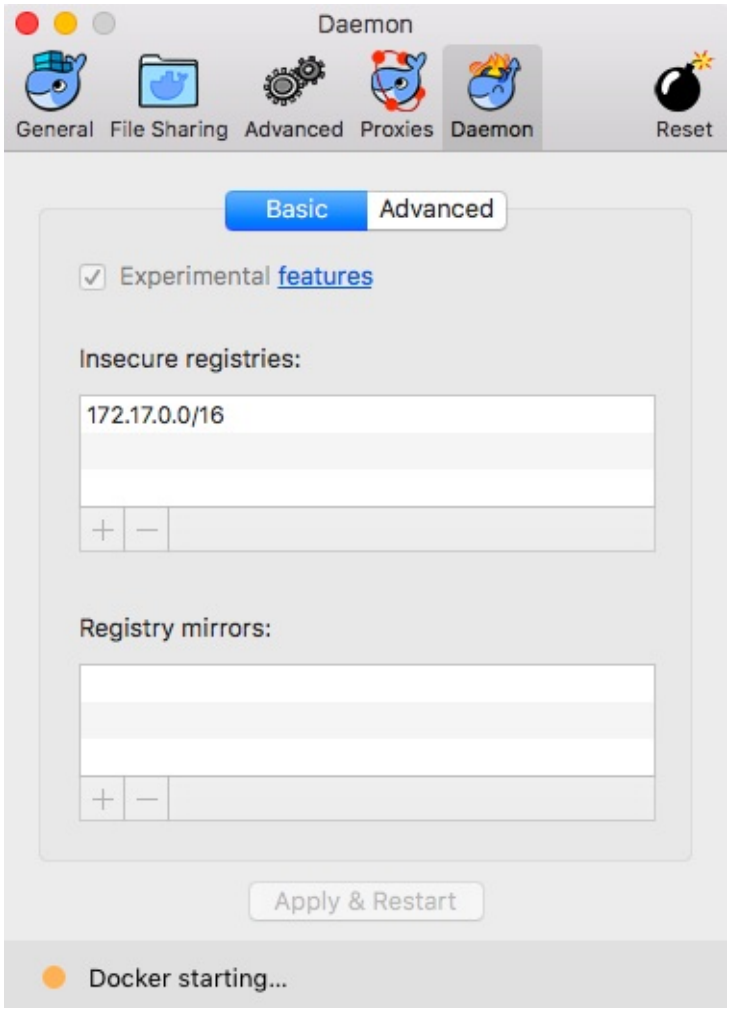
```
$ docker-machine ssh
$ sudo vi /etc/hosts
127.0.0.1 gitlab.phper-na14.info registry.phper-na14.info ci.php
er-na14.info
$ exit
```

プライベートレジストリの設定（**https**通信から**http**通信へ切替）

windows

```
$ docker-machine ssh
$ vi /var/lib/boot2docker/profile
EXTRA_ARGS='
--label provider=virtualbox
--insecure-registry 192.168.99.0/24
--insecure-registry 172.17.0.0/16
'
$ exit
```

mac



イメージのインポート

本章で使うイメージをインポートしておきます。

```
$ cd gitlab
$ docker load < docker.tar
$ docker load < gitlab-ce.tar
$ docker load < gitlab-runner.tar
$ docker load < redis.tar
$ docker load < dnmonster.tar
$ docker load < python.tar
```

ユニットテストとは？

identidock（webアプリ）起動

```
$ cd gitlab/files

$ docker-compose up -d

localhost:9090 へアクセス
```

テストの中身確認

```
$ vi app/tests.py

import unittest
import identidock


class TestCase(unittest.TestCase):

    def setUp(self):
        identidock.app.config["TESTING"] = True
        self.app = identidock.app.test_client()

    def test_get_mainpage(self):
        page = self.app.post("/", data=dict(name="Moby Dock"))
        assert page.status_code == 200
        assert 'Hello' in str(page.data)
        assert 'Moby Dock' in str(page.data)

    def test_html_escaping(self):
        page = self.app.post("/", data=dict(name='"><b>TEST</b><
!--'))
        assert '<b>' not in str(page.data)


if __name__ == '__main__':
    unittest.main()
```

テスト実行

```
$ docker run files_identidock python /app/tests.py
```

GitLabの起動

作業フォルダへ移動しdocker-composeを使ってGitlabを起動します。

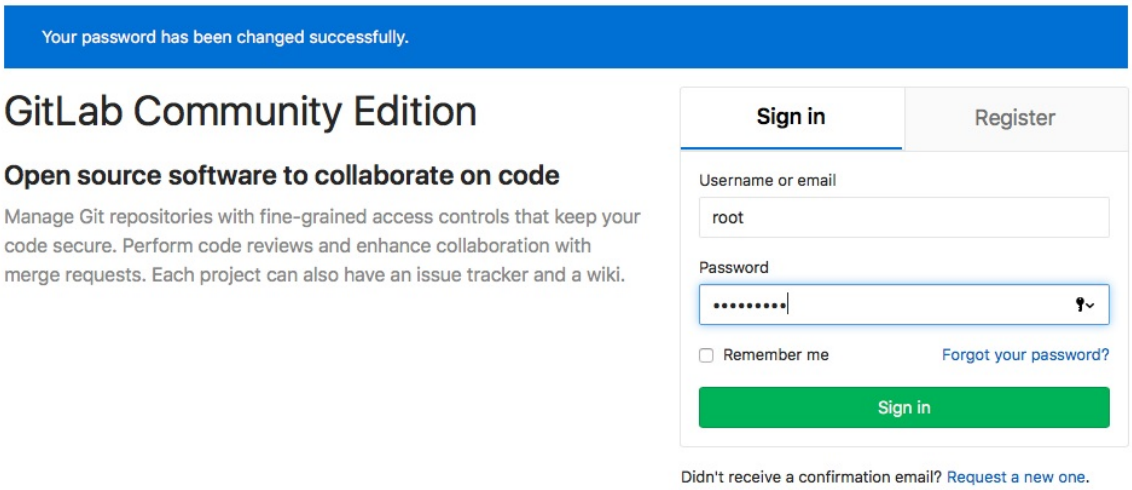
```
$ cd gitlab
$ ls -l
-rwxrwxrwx  docker-compose.yml
drwxrwxrwx  files
$ docker-compose up -d
```

起動に結構時間がかかります。起動の状況を確認するためにログをリアルタイムに表示しましょう。

```
$ docker-compose log -f
```

ログの出力が頻繁になったらCtrl+cでログの出力を停止し http://gitlab.phper-na14.info へアクセスします。

下記の画面がでたら起動完了です。



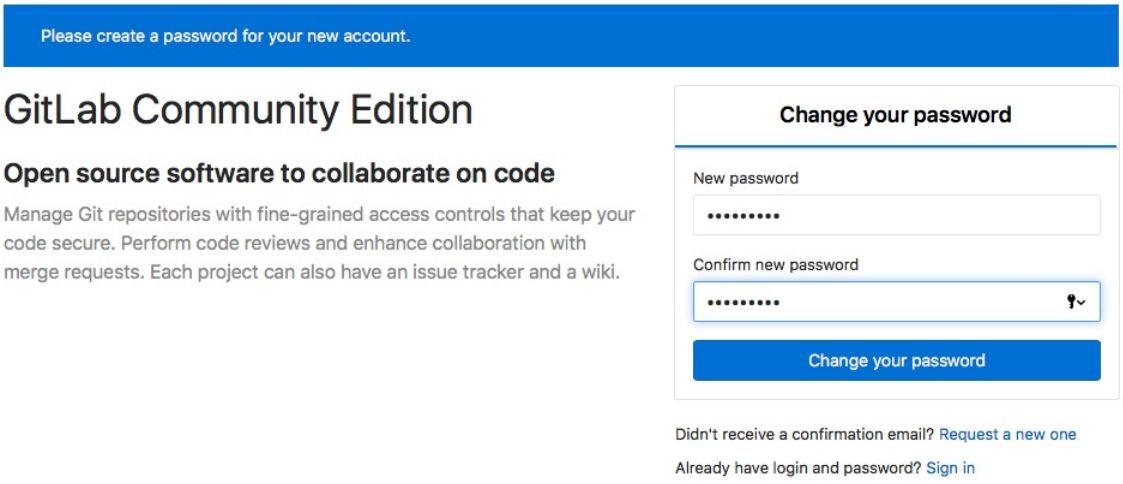
アクセスできない？名前解決ができているか確認してみましょう。応答がなければ前項の設定をもう一度行ってみてください。

```
$ ping gitlab.phper-na.info
PING gitlab.phper-na14.info (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.082 ms
64 bytes from 127.0.0.1: seq=2 ttl=64 time=0.079 ms
```

GitLabの設定

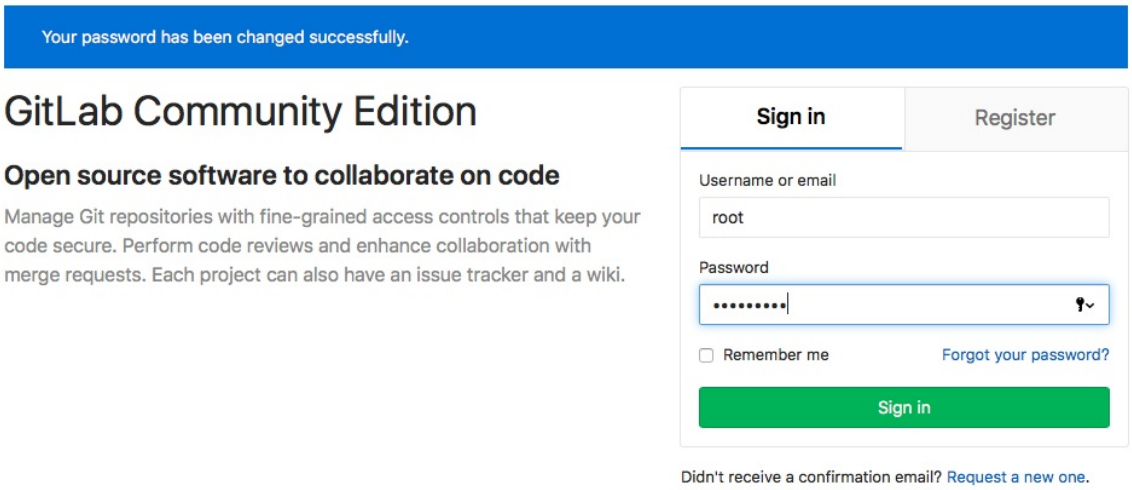
GitLabの初期設定を行います。

管理者パスワードの設定

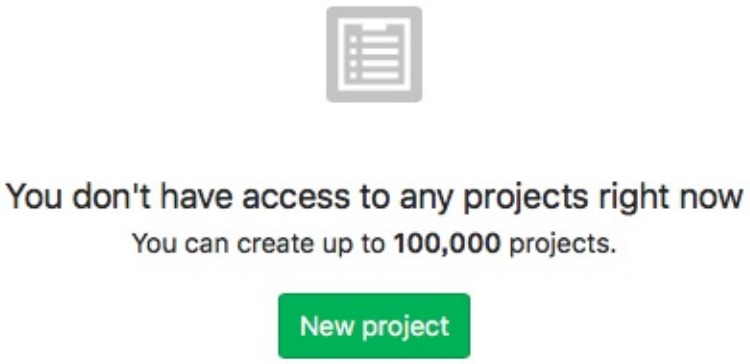


ログイン

ユーザー名は **root** 先に設定したパスワードを入力してログインします。



プロジェクトの新規作成

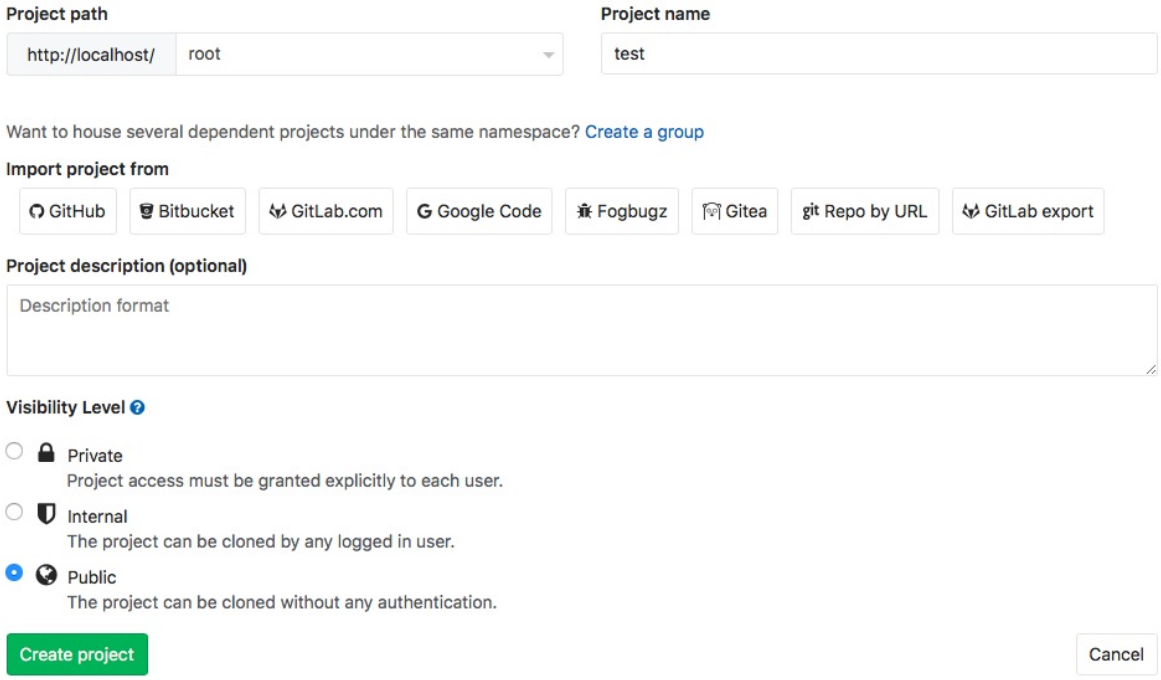


プロジェクトの設定

project name（プロジェクト名）： test

Visiblity Level（公開範囲）：public

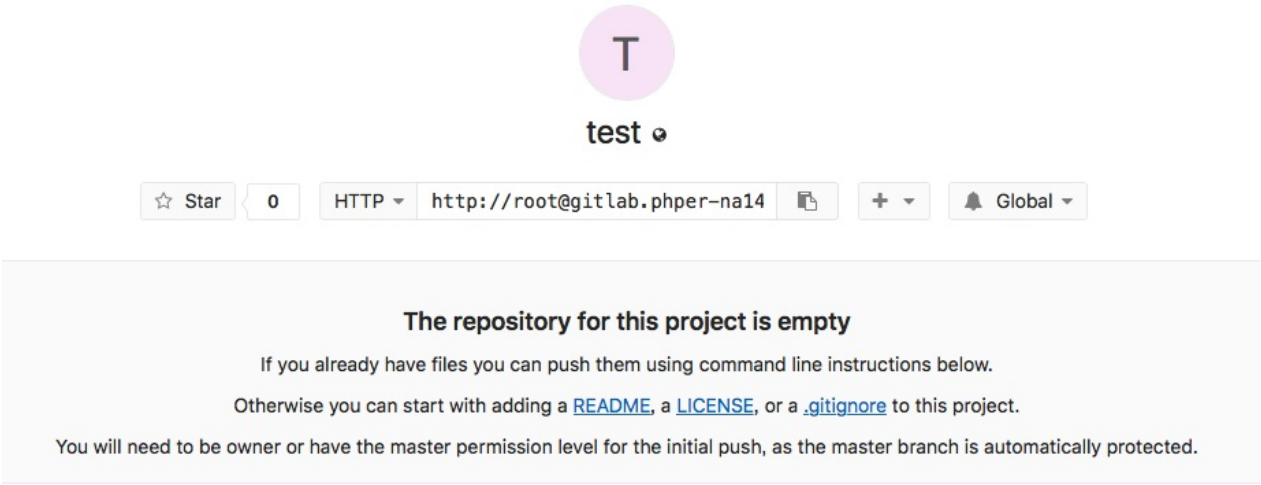
にて設定してください。


Project path
http://localhost/
root
Project name
test
Want to house several dependent projects under the same namespace? Create a group
Import project from
GitHub
Bitbucket
GitLab.com
Google Code
Fogbugz
Gitea
Repo by URL
GitLab export
Project description (optional)
Description format
Visibility Level
Private
Project access must be granted explicitly to each user.
Internal
The project can be cloned by any logged in user.
Public
The project can be cloned without any authentication.
Create project
Cancel

動作確認

リポジトリの確認

http://gitlab.phper-na14.info/root/test へアクセスしリポジトリが空であることを確認します。



ソースコード登録

git グルーバル設定（既に実施している場合は不要）

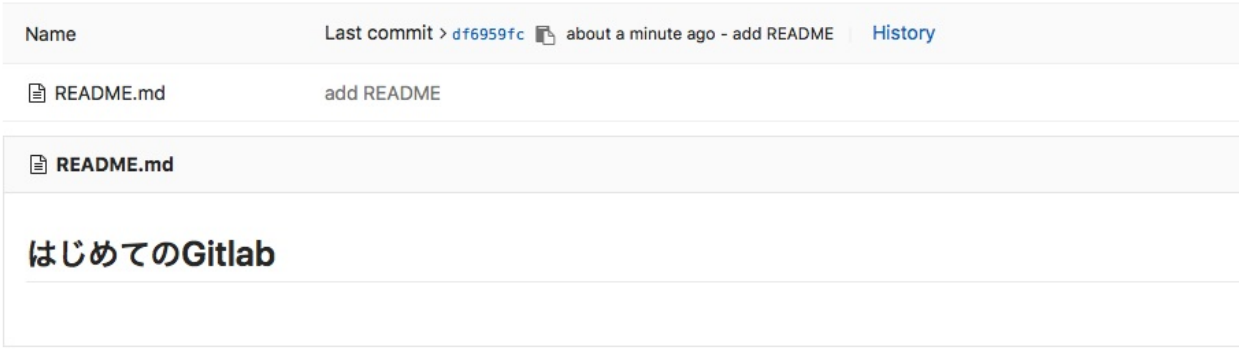
```
$ git config --global user.name "Administrator"
$ git config --global user.email "admin@example.com"
```

ソースコードをリポジトリに登録します。README.mdはコピーします。

```
$ git clone http://root@gitlab.phper-na14.info/root/test.git
$ cd test
$ cp ../files/README.md .
$ git add README.md
$ git status
$ git commit -m "add README"
$ git push -u origin master
```

リポジトリの確認

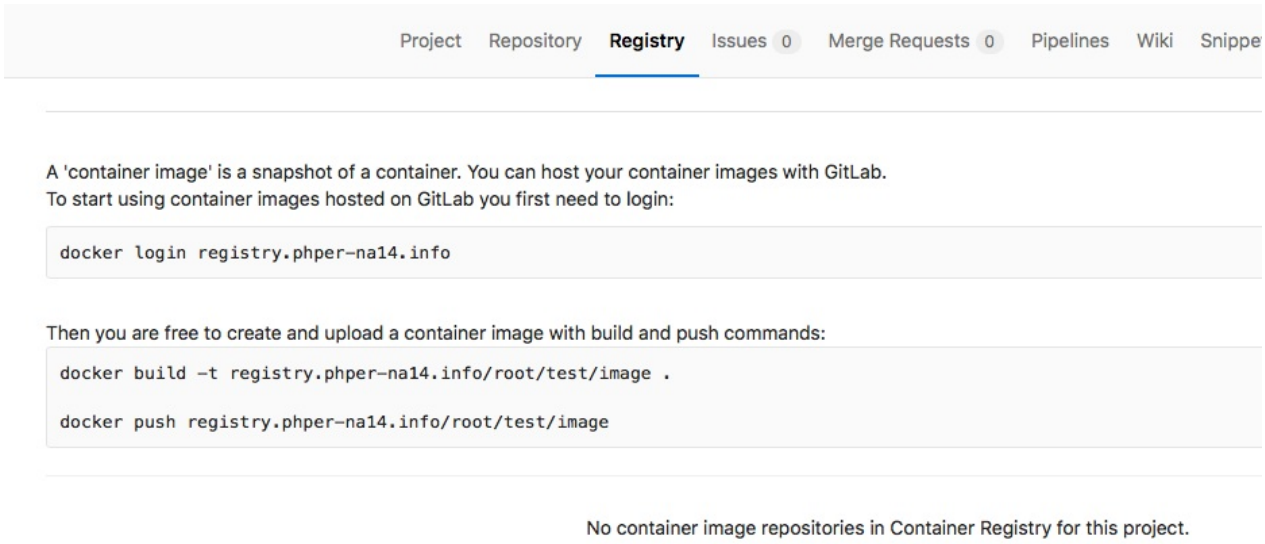
http://gitlab.phper-na14.info/root/test へアクセスし、先程pushしたソースが登録されていることを確認します。



レジストリの動作確認

リポジトリの確認

http://gitlab.phper-na14.info/root/test/container_registry へアクセスしレジストリが空であることを確認します。



プライベートレジストリへのログイン

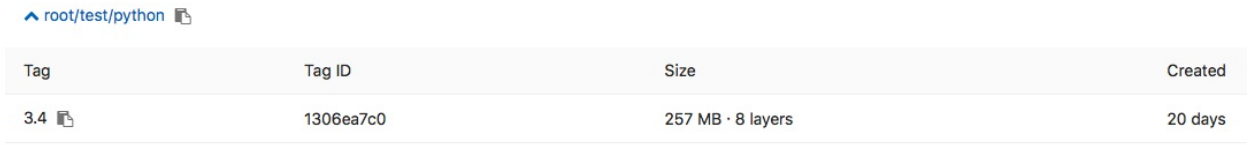
```
$ docker login registry.phper-na14.info
Username (root): root
Password:
Login Succeeded
```

イメージの登録

```
$ docker tag library/python:3.4 registry.phper-na14.info/root/te
st/python:3.4
$ docker push registry.phper-na14.info/root/test/python:3.4
e5ba092876c2: Pushed
fafff35446a4: Pushed
e1013fcbd78a: Pushed
ab0825ad21e8: Pushed
e7b0b4cd055a: Pushed
445ed6ee6867: Pushed
c59fa6cbcbd9: Pushed
8d4d1ab5ff74: Pushed
3.4: digest: sha256:b5422da942dc1480c28267865b852b7a46a8b91a99de
824a1da23c6d9267cf27 size: 2007
```

イメージの登録確認

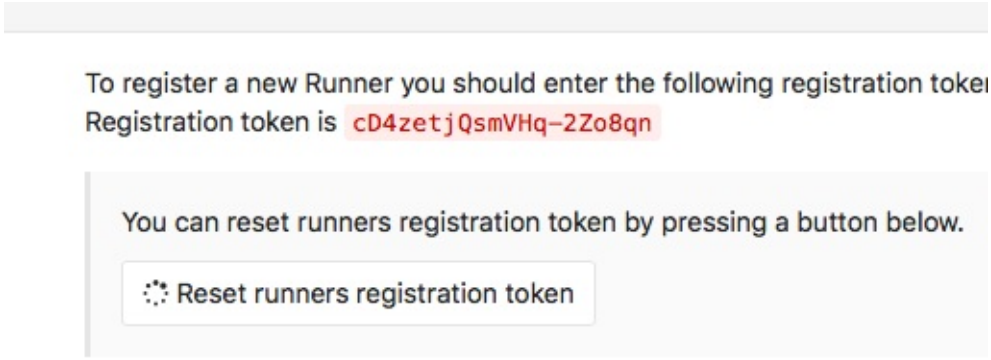
http://gitlab.phper-na14.info/root/test/container_registry へアクセスし、先程pushしたイメージが登録されていることを確認します。



GitLab CI の設定

Registration tokenの確認

http://gitlab.phper-na14.info/admin/runners へアクセスし"Registration token" を控えておきます。



gitlab-runnerの登録

既に起動している gitlab-runner にログインして設定を行います。

先にmacの方はipアドレスを調べておきましょう

mac

gitlab-ci coordinator URLに指定する gitlab-ceのipアドレスの調べ方

```
$ docker inspect gitlab-ce | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.18.0.2",
```

extra_hostsに指定するgitlab-runner-dockerのipアドレスの調べ方

```
$ docker network inspect bridge | grep Gateway
                    "Gateway": "172.17.0.1"
```

さて、準備ができました。

既に起動している gitlab-runner にログインして設定を行います。

```
$ docker ps
    Name                    Command               State
              Ports
------------------------------------------------------------------
----------------------------------------
gitlab-ce       /assets/wrapper                  Up      22/tcp,
 0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
gitlab-runner   /usr/bin/dumb-init /entryp ...   Up

$ docker exec -it gitlab-runner /bin/bash
```

下記コマンド実行後、対話式で入力しENTERキーを押していきます #以降はコメントですので入力はしないでください

```
# gitlab-ci-multi-runner register

Running in system-mode.
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.
com/\:

--> http://192.168.99.100/ci # docker toolbox
--> http://172.18.0.2/ci # docker for mac 注）gitlab-ceコンテナのip
を指定

Please enter the gitlab-ci token for this runner:

--> cD4zetjQsmVHq-2Zo8qn # 先ほど控えたトークンを入力

Please enter the gitlab-ci description for this runner:

--> [1e57eefd7e33]: ci
```

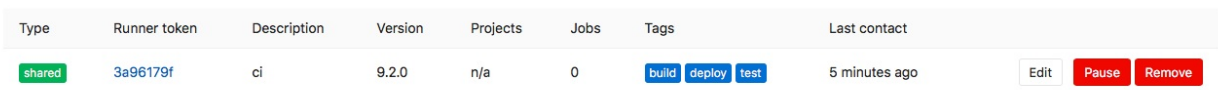
```
Please enter the gitlab-ci tags for this runner (comma separated
):

--> build,test,deploy

Whether to run untagged builds [true/false]:

--> [false]: # 入力なし

Whether to lock Runner to current project [true/false]:

--> [false]: # 入力なし

Registering runner... succeeded                     runner=Rx93N
8ie
Please enter the executor: docker, parallels, shell, ssh, virtua
lbox, docker-ssh, docker+machine, docker-ssh+machine, kubernetes
:

--> docker

Please enter the default Docker image (e.g. ruby:2.1):

--> docker:latest

Runner registered successfully. Feel free to start it, but if it
's running already the config should be automatically reloaded!
```

続いて不足している設定を追記／変更します。

```
# vi /etc/gitlab-runner/config.toml

concurrent = 1
check_interval = 0

[[runners]]
  name = "ci"
  url = "http://192.168.99.100/ci"
  token = "a30c281805be2666c8ddb048e0331f"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "docker latest"
変更-->    privileged = true # trueに変更
    disable_cache = false
変更-->    volumes = ["/var/run/docker.sock:/var/run/docker.sock"
, "/cache"] # 既にvolumesが定義されているので上書きする
    shm_size = 0
追加toolboxの場合-->    extra_hosts = ["gitlab.phper-na14.info:192
.168.99.100", "registry.phper-na14.info:192.168.99.100"] # 追記 d
ocker toolbox
追加formacの場合-->    extra_hosts = ["gitlab.phper-na14.info:172.
17.0.1", "registry.phper-na14.info:172.17.0.1"] # 追記 docker for
 mac  注）gitlab-runner-dockerのipを指定  環境によって変わる
  [runners.cache]

# exit
```

gitlab-runnerの登録確認

http://gitlab.phper-na14.info/admin/runners へアクセスしgitlab-runnerが登録されていることを確認します。

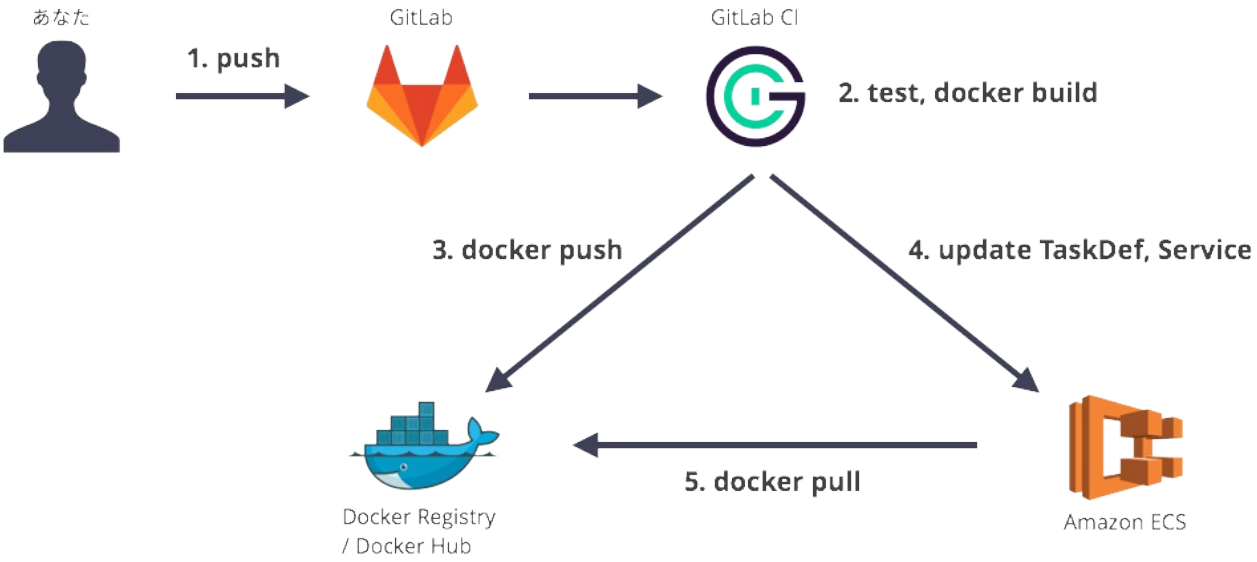


CIを体験しよう

これからやること

1. ソースコードをGitLabにpushします
2. GitLabCIがpushイベントを検知し、設定されているジョブを自動的に実行
    i. イメージのビルド
    ii. アプリケーションのテスト
    iii. テストをパスしたソースを含んだイメージを自動的にレジストリに登録

つまりプログラムソースをpushすると自動でテストが走り、実行環境まで整備してくれます。



Source: Git プッシュから Amazon ECS に自動デプロイする仕組みを構築する

さぁ、体験してみましょう。

ソースコードのcopy

filesフォルダからtestフォルダへファイルを一式コピーします。

```
$ cp -r ../files/. .
```

Jobの内容

Jobの内容が上記の仕様になっていることを確認してください。

```
$ vi .gitlab-ci.yml

variables:
    DOCKER_DRIVER: overlay

stages:
  - build
  - test
  - deploy

build: # イメージのビルド

  image: docker:latest
  stage: build
  before_script:
    - docker info
  script:
    - docker build -t $CI_REGISTRY_IMAGE/identidock:latest .
  tags:
    - build

test: # イメージのテスト

  image: docker:latest
  stage: test
  script:
    - docker run $CI_REGISTRY_IMAGE/identidock:latest python /ap
p/tests.py
  tags:
    - test

deploy: # イメージの登録（動作環境へのデプロイなど）

  image: docker:latest
  stage: deploy
  script:
```

```
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN $CI_REG
ISTRY
    - docker push $CI_REGISTRY_IMAGE/identidock:latest
  tags:
    - deploy
```

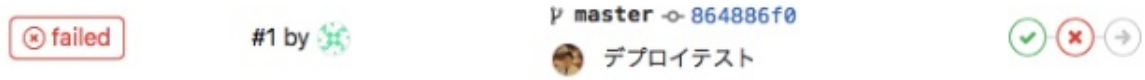
ソースコードのpush

```
$ git add .
$ git status
$ git commit -m "デプロイテスト"
$ git push -u origin master
```

Jobの確認

ソースコードのpushを検知したGitLabは、Jobを登録してgitlab-runnerにてそのJobを実行します。

http://gitlab.phper-na14.info/root/test/pipelines にてJobを確認してみましょう。



おや？テストJobで失敗しています。詳細を見てみましょう。

```
$ docker run $CI_REGISTRY_IMAGE/identidock:latest python /app/te
sts.py
.F
======================================================================
FAIL: test_html_escaping (__main__.TestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/app/tests.py", line 19, in test_html_escaping
    assert '<b>' not in str(page.data)
AssertionError

----------------------------------------------------------------------
Ran 2 tests in 0.006s

FAILED (failures=1)
ERROR: Job failed: exit code 1
```

サニタイズ処理のテストが失敗していました。

ソース修正

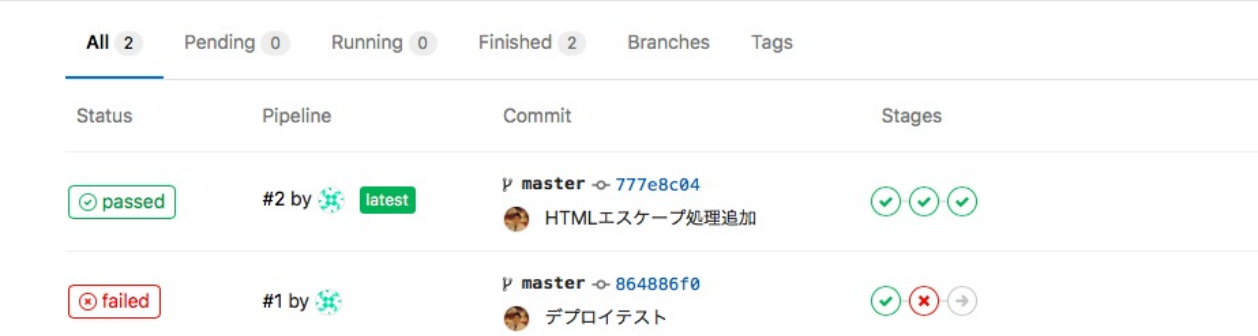
エラーの原因が分かったのでサニタイズ処理を追加しましょう。修正後再度pushします。

```
$ vi app/identidock.py
name = html.escape(request.form['name'], quote=True) # 18行目修正

name = html.escape(name, quote=True) # 38行目付近に追記
$ git add .
$ git status
$ git commit -m "HTMLエスケープ処理追加"
$ git push -u origin master
```

再度Jobの確認

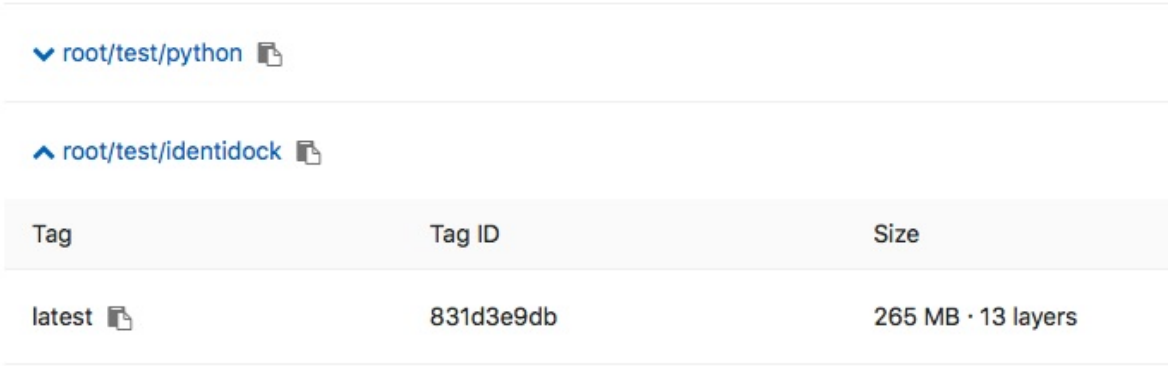
http://gitlab.phper-na14.info/root/test/pipelines

暫く待つと...お！テストがパスされて最後までJobが動作しています。



イメージ登録確認

レジストリにイメージが登録されています！



Special Thanks

- SaaSなしでもインフラCIを！GitLab CI + Docker + Ansible + serverspecでインフラCIを試す
- プライベートレジストリ --insecure-registry の設定方法（https通信→http通信への切り替え）
- Docker in Docker のベタープラクティス
- Git プッシュから Amazon ECS に自動デプロイする仕組みを構築する
- Dockert調査　~ログ編~
- --insecure-registryOSごと設定まとめ

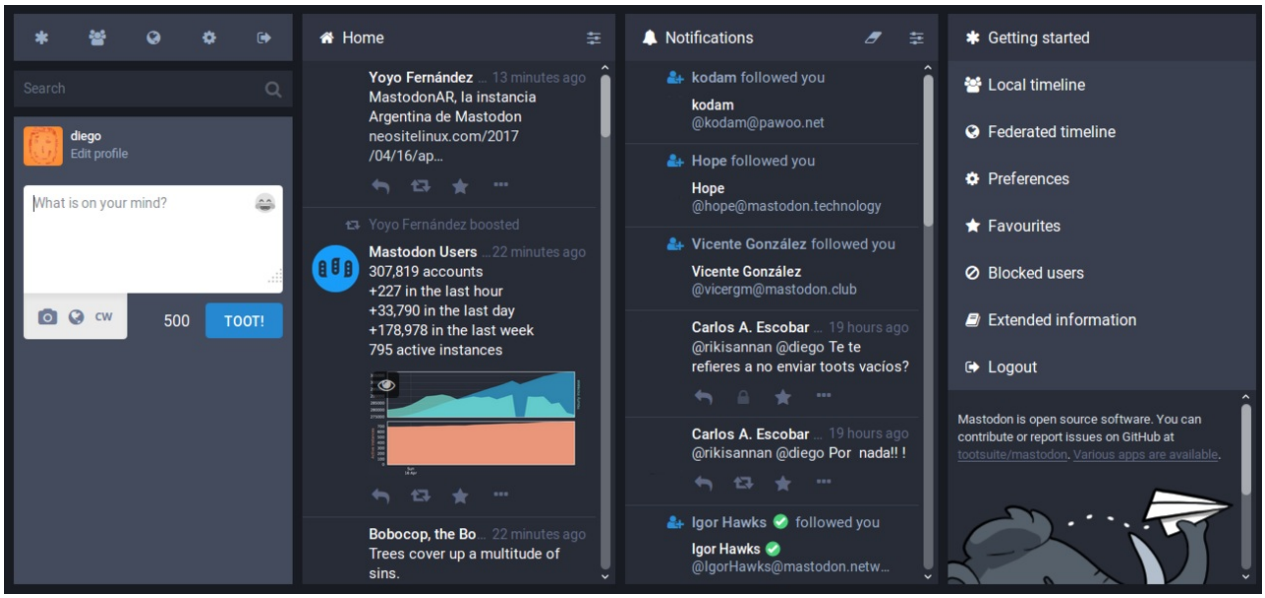
mastodonとは？

本章のゴール

- dockerを使ってmastodonインスタンスをローカルに立てる
- ローカルに立てたmastodonインスタンスをインターネットに公開する

mastodon（マストドン）

（Mastodon）はミニブログサービスを提供するためのフリーソフトウェア、またはこれが提供する連合型のソーシャルネットワークサービスである。「脱中央集権型」(decentralized) のマストドンのサーバーはだれでも自由に運用する事が可能であり、利用者は通常このサーバーの一つを選んで所属するが、異なるサーバーに属する利用者間のコミュニケーションも容易である。



Source: マストドン (ミニブログ) wikipedia

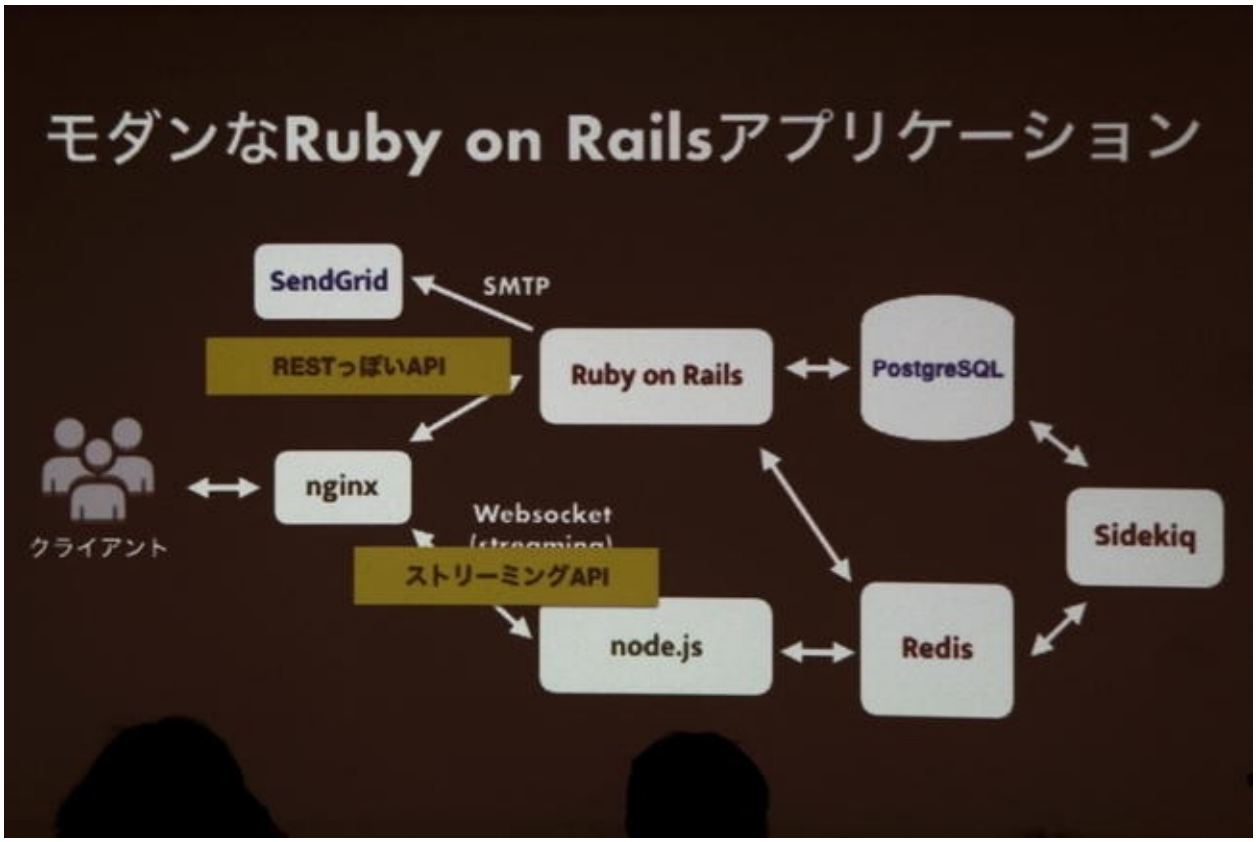
日本で有名なマストドン インスタンス

- mstdn.jp
- pawoo.net　(pixiv公式)
- friends.nico　(niconico/ドワンゴ公式)

Source: 日本のマストドン インスタンス一覧

nullkal氏...ドワンゴ...

システム構成



Source: 勉強会に行ってみた！第51回「Mastodon/Pawooの運用&開発技術

準備

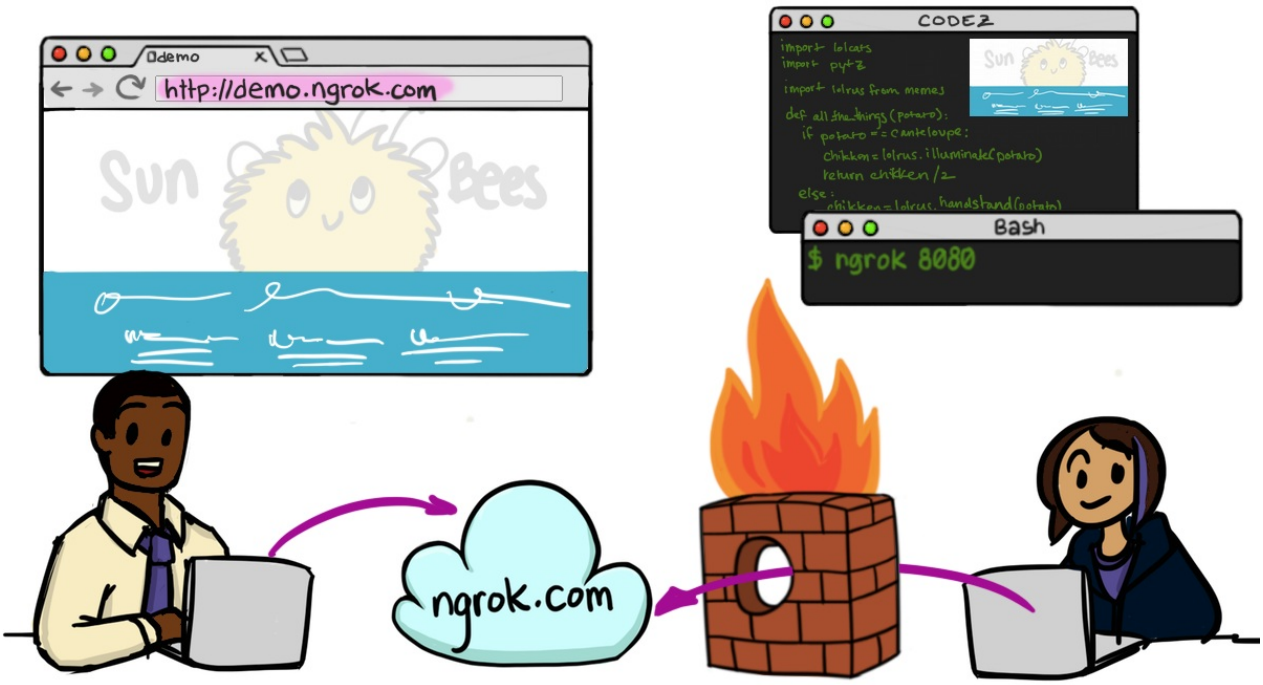
イメージをインポート

```
$ cd mastodon
$ docker load < mastodon.tar
$ docker load < postgresql.tar
```

ngrokで公開用URLを取得

ngrok（エヌジーロック）はNATやファイヤーウォール以下にあるローカルサーバーを、インターネット越しにアクセス可能にしてくれるサービスです。

今回は、このサービスを使ってmastodonインスタンスを外部に公開します。



Source: ngrok

> **windows**

DockerTerminalではなくコマンドプロンプトを起動して実行してください。

```
> cd path-to-ngrok
> ngrok.exe http -region=ap 192.168.99.100:3000
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Version                       2.2.4
Region                        Asia Pacific (ap)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://ee9e6dd7.ap.ngrok.io -> 192
.168.99.100:3000
Forwarding                    https://ee9e6dd7.ap.ngrok.io -> 19
2.168.99.100:3000

Connections                   ttl     opn     rt1     rt5     p5
0     p90
                              0       0       0.00    0.00    0.
00    0.00
```

mac

別ターミナルで実行してください

```
$ chmod +x ngrok
$ ./ngrok http -region=ap 3000
ngrok by @inconshreveable
                                              (Ctrl+C to quit)

Session Status                online
Version                       2.2.4
Region                        Asia Pacific (ap)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://ee9e6dd7.ap.ngrok.io -> loc
alhost:3000
Forwarding                    https://ee9e6dd7.ap.ngrok.io -> lo
calhost:3000

Connections                   ttl     opn     rt1     rt5     p5
0     p90
                              0       0       0.00    0.00    0.
00    0.00
```

設定ファイル編集

公開URL設定

ユーザーがアクセスする公開URLを設定します。

出力されたForwarding行に記載されたURLを設定ファイルに転記します。

```
$ vi .env.production
WEB_DOMAIN=https://ee9e6dd7.ap.ngrok.io
```

メール受信設定

自分のgmail宛にメールが受信できるように設定します。

下記URLを参考にアプリパスワードの作成を行った上設定してください（要2段階認証設定）

Source: Googleアプリパスワードを利用して2段階認証をより便利に

```
$ vi .env.production
SMTP_SERVER=smtp.gmail.com
SMTP_PORT=587
SMTP_LOGIN=xxxxxx@gmail.com <--自分のURL
SMTP_PASSWORD=アプリパスワード
SMTP_FROM_ADDRESS=xxxxxx@gmail.com
SMTP_DOMAIN=gmail.com
```

もし、設定できない場合は...Skipしましょう。

mastodon...いきまーす！

さぁ、mastodonのインスタンスを立てますよ！

データベースの作成

```
$ docker-compose run --rm web rails db:migrate
```

アセットファイルの作成

```
$ docker-compose run --rm web rails assets:precompile
```

起動

もうおなじみのコマンドで！と中身も確認しておきましょう。

```
$ vi docker-compose.yml
version: '3'
services:

  db:
    restart: always
    image: postgres:alpine
### Uncomment to enable DB persistance
    volumes:
      - postgres:/var/lib/postgresql/data

  redis:
    restart: always
    image: redis:alpine
### Uncomment to enable REDIS persistance
    volumes:
```

```
      - redis:/data

  web:
    build: .
    image: gargron/mastodon
    restart: always
    env_file: .env.production
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    ports:
      - "3000:3000"
    depends_on:
      - db
      - redis
    volumes:
      - public-assets:/mastodon/public/assets
      - public-packs:/mastodon/public/packs
      - public-system:/mastodon/public/system

# タイムラインや通知を Websocket で受け取ることができます
  streaming:
    build: .
    image: gargron/mastodon
    restart: always
    env_file: .env.production
    command: npm run start
    ports:
      - "4000:4000"
    depends_on:
      - db
      - redis

# メッセージパッシング
  sidekiq:
    build: .
    image: gargron/mastodon
    restart: always
    env_file: .env.production
    command: bundle exec sidekiq -q default -q mailers -q pull -
q push
    depends_on:
```

```
      - db
      - redis
    volumes:
      - public-system:/mastodon/public/system

volumes:
      postgres:
      redis:
      public-assets:
      public-packs:
      public-system:
```

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

起動状況確認

こちらも起動に時間がかかるので起動状況を確認します。

```
$ docker-compose logs -f
```

起動確認

あの画面がでましたか？？

windows

http://192.168.99.100:3000/

mac

http://localhost:3000/

公開確認

携帯などから確認してみましょう！表示されましたか？？（ngrokのアドレスは各自で読み替えてください）

https://ee9e6dd7.ap.ngrok.io

アカウント作成

初期表示画面にてアカウントを作成（参加）してください。メール認証後ログインしましょう。
もちろんトゥートもできるはず！

なおメール設定ができていない方は、下記コマンドにてメール認証を行ってログインしましょう。

```
$ docker-compose run --rm web rails mastodon:confirm_email USER_EMAIL=アカウント作成時に登録したメールアドレス
```

管理者の登録

下記コマンドで管理者を作成できます。ここでは既に作成したユーザーに管理者権限を与えています

```
$ docker-compose run --rm web rails mastodon:make_admin USERNAME
=アカウント作成時に登録したユーザー名
```

サイトの設定は下記URLにて行います。（ngrokのアドレスは各自で読み替えてください）

https://ee9e6dd7.ap.ngrok.io/admin/settings/

Special Thanks

- DockerでMastodonをローカルで動かしてみた！ ので、その方法をご紹介。
- Mastodon を Docker で起動する方法
- 小規模Mastodonインスタンスを運用するコツ
- 勉強会に行ってみた！第51回「Mastodon/Pawooの運用&開発技術