はじめに

DockerMachineユーザガイド～基礎編

このガイドについて

Docke ドキュメント（https://docs.docker.com/）の日本語翻訳版（http://docs.docker.jp/）をもとに電子データ化しました。

免責事項

現時点ではベータ版であり、内容に関しては無保証です。PDF の評価用として公開しました。Docker は活発な開発と改善が続いています。同様に、ドキュメントもオリジナルですら日々書き換えが進んでいますので、あらかじめご了承願います。

このドキュメントに関する問題や翻訳に対するご意見がありましたら、GitHub リポジトリ上の Issue までご連絡いただくか、pull request をお願いいたします。

- https://github.com/zembutsu/docs.docker.jp

履歴

- 2016 年 5 月 12 日　beta1 を公開

目次

1章

DockerMachine概要

Docker Machine（マシン）を使いますと、以下の操作ができます。

- Mac や Windows 上に Docker をインストール・実行
- 複数のリモート Docker ホストを構築・管理
- Swarm クラスタの構築（プロビジョン）

1.1 Docker Machine とは何でしょうか

Docker Machine（ドッカー・マシン）は仮想マシン上に Docker Engine をインストールするツールであり、`docker-machine` コマンドを使ってホストを管理します。Machine を使えば、自分のローカルの Mac や Windows 上に Docker ホストを作れるだけでなく、あなたの会社のネットワーク上や、データセンターや、AWS、Digital Ocean のようなクラウド・プロバイダ上でも作れます。

`docker-machine` コマンドで、管理ホストの起動（start）、調査（inspect）、停止（stop）、再起動（restart）ができます。他にも Docker クライアントとデーモンの更新や、Docker クライアントが対象のホストへ接続できるような設定もできます。

Machine のコマンドライン上で管理対象のホストを参照するように指定したら、`docker` コマンドが対象ホストを直接管理します。例えば、`docker-machine env default` を実行したら、操作対象のホストは `default` という名前のホストにするため、画面上に `env` コマンドの指示が表示されます。これを使ってセットアップしたあとは、docker ps や docker run hello-world コマンドなど、指定したホスト上で直接処理できます。

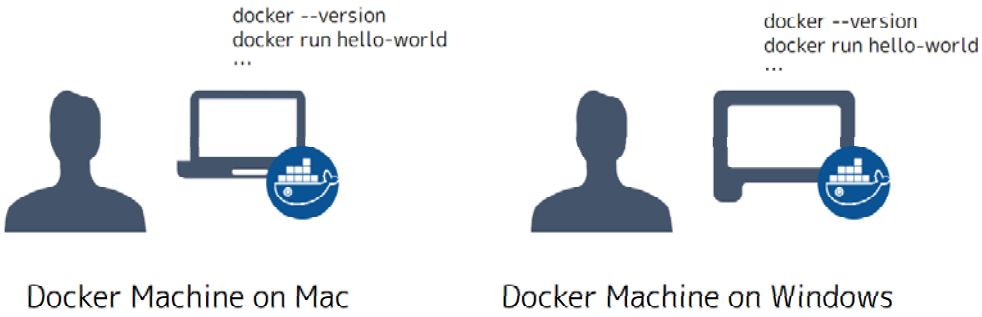
1.2 なぜ使うべきでしょうか

Mac や Windows 上では、Docker Machine を使うのが Docker を実行する唯一の方法です。そして、様々な Linux で動作するリモート Docker ホストを複数プロビジョン[*1] するのにも、ベストの方法です。

Docker Machine は主に 2 つの使い方があります。

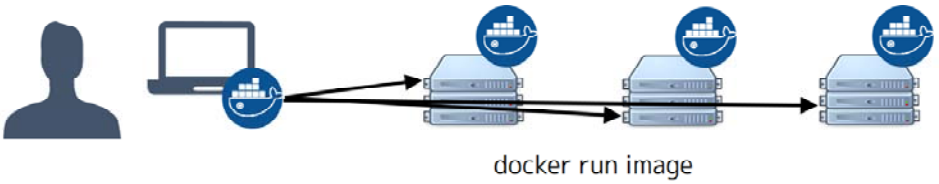
*1 訳者注：プロビジョンする（プロビジョニング）とは、環境とソフトウエアの自動設定です。

Mac や Windows 上で Docker を実行したい



主に Mac や Windows をノート PC 上で動かす場合は、「Docker を実行」するために（つまり DockerEngine を動かすために）Docker Machine をローカルにインストールする必要があります。Mac や Windows 環境上で Docker Machine を使えば、ローカルに Docker Engine が動く仮想マシンをプロビジョニングします。そして、その環境に接続したら、docker コマンドを実行可能になります。

リモート・システム上に Docker ホストをプロビジョンしたい

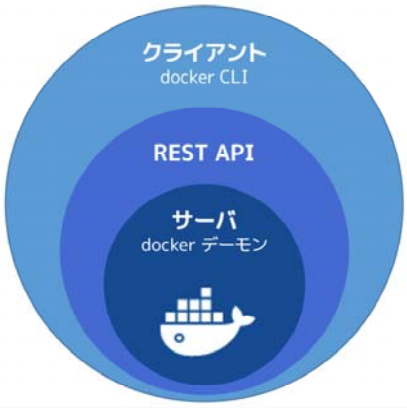


Docker Engine は Linux システム上でネイティブに動きます。主な利用環境が Linux であれば、docker コマンドを実行したい場合に必要になるのは Docker Engine のダウンロードとインストールです。それだけでなく、ネットワーク上で複数の Docker ホストを効率的にプロビジョンしたい場合、それがクラウドでも、ローカル環境でも実現したいのであれば、Docker Machine が必要になるでしょう。

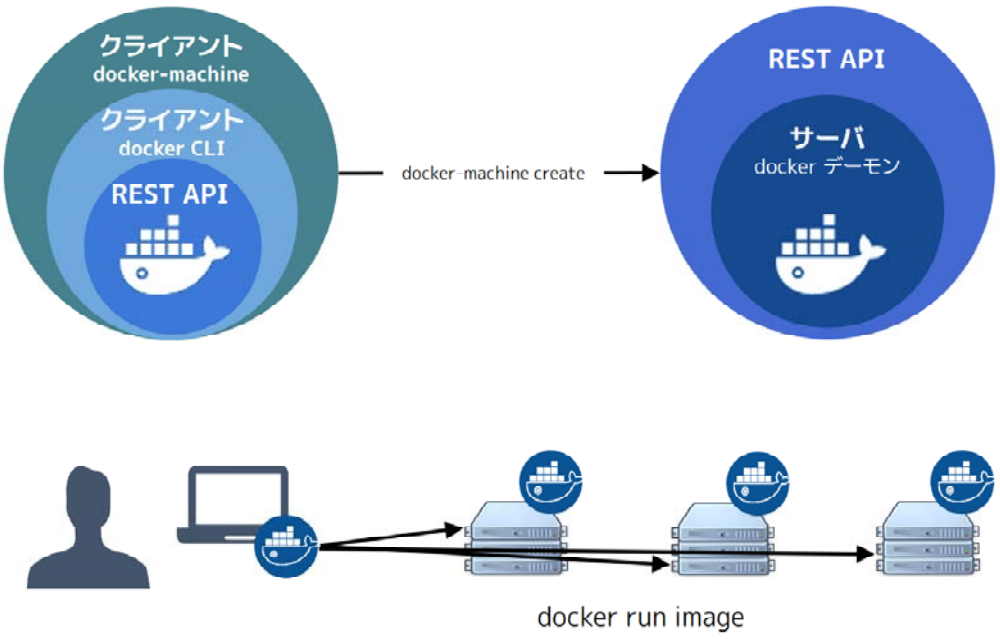
Docker Machine をインストールしたら、Mac・Windows・Linux の環境を問わず `docker-machine` コマンドを使って多くの Docker ホストのプロビジョンと管理ができます。Docker Machine は自動的にホストを作成し、そこに Docker Engine をインストールし、 docker クライアントの設定を行います。管理対象の各ホスト（「マシン」）が Docker ホストと設定済みのクライアントを結び付けます。

1.3 Docker Engine と Machine の違い

一般的に「Docker」と呼ばれるのは **Docker Engine** を意味します。Docker デーモンはクライアント・サーバ型アプリケーションです。デーモンは特定の **REST API**（レストエーピーアイ） **インターフェース**と**コマンド・ライン・インターフェース**（**CLI**）で、デーモンと通信します（正確には、REST API ラッパーを経由します）。Docker Engine は CLI からの `docker` コマンドを受け付けます。`docker run <イメージ名>` や `docker ps` でのコンテナ実行や、`docker images` でイメージの一覧などを処理します。



Docker Machine は Docker ホスト（Docker Engine が動くホスト環境）のプロビジョニングと管理用のツールです。一般的には Docker Machine を自分のローカルシステム上にインストールします。Docker Machine とは `docker-machine` というコマンド・ラインのクライアントと `docker` という名称の Docker Engine クライアントです。Docker Machine を使えば仮想システム上に Docker Engine をインストールできます。仮想システムとはローカル（Machine を使い、Mac または Windows 上の VirtualBox に Docker Engine をインストール、実行します）でも、リモート（Machine を使い、クラウド・プロバイダ上に Docker をプロビジョン）でも構いません。Docker に対応したホストであれば、何でも常に「マシン」として管理できる[*1] のです。



*1 訳者注：Docker Machine の管理対象のホストは、ローカルでもクラウド上でも「マシン」という概念で抽象化できる、という意味です

1.4 Docker Machine 概念の理解

Docker Machine を使えば、様々な環境・様々な仮想マシン上で Docker が動くマシンを自動作成できます。これはローカルのシステム上だけでなく、クラウド・プロバイダ上でも、ベアメタル・サーバ（物理コンピュータ）上でも可能です。Docker Machine で Docker ホストを作成したら、Docker Engine クライアントを使えば、必要に応じてホスト上でイメージの構築やコンテナ作成が可能になります。

マシン作成用のドライバ

仮想マシンを作成するには、Docker Machine 実行時に使いたい**ドライバ**の名前を指定します。ドライバの選択は、仮想マシンをどこで作成するかによります。例えば、ローカルの Mac や Windows システム上であれば、Oracle VirtualBox が一般的なドライバになります。物理マシン上にプロビジョニングする場合は generic（ジェネリック）ドライバです。クラウド・プロバイダであれば、Docker Machine は AWS・Microsoft Azure・Digital Ocean 等々に対応してます。Docker Machine のリファレンスに サポートしているドライバ一覧 があります。

ローカルまたはクラウド上でデフォルトのベース OS

Docker を Linux 上で動かす場合、Docker Machine のプロビジョニングは仮想マシンのベースとなるオペレーティング・システムに依存します。簡単なのはデフォルトのベース・オペレーティング・システムを使うことです。Oracle Virtual Box ドライバであれば、ベースにするオペレーティング・システムは boot2docker です。クラウド・プロバイダを使う場合はベース・オペレーティング・システムは Ubuntu 12.04 以上です。このデフォルトはマシン作成時に変更できます。Docker Machine リファレンスの「サポートしているオペレーティング・システムの一覧」をご覧ください。

Docker ホストの IP アドレス

各マシンの作成時、Docker ホストに対して Linux 仮想マシンの IP アドレスが割り当てられます。これは `docker-machine create` サブコマンドの実行時に割り当てられます。作成したマシンの一覧は `docker-machine ls` コマンドで確認できます。`docker-machine ip <マシン名>` コマンドは、指定したホストの IP アドレスを返します。

CLI の操作対象 Docker ホストを環境変数で指定

マシン上で `docker` コマンドを実行する前に、コマンドライン上で対象のマシンを指定する必要があります。`docker-machine env <マシン名>` サブコマンドを実行したら、適切な命令を出力します。

`docker-machine` サブコマンドの一覧は、リファレンスの「Docker Machine サブコマンド」をご覧ください。

1.5 クラッシュ報告

ホストのプロビジョニングには複雑な問題があり、様々な理由により失敗する場合があります。皆さんの作業環境ではシェル、ネットワーク設定、VPN、プロキシやファイアウォールに関する多くの問題があるかもしれません。また、これが他との問題の切り分け箇所（クラウド・プロバイダ、あるいは、ネットワーク間での問題か）になります。

`docker-machine` が安定するための手助けとなるよう、私たちはホスト上で `create` や `upgrade` を試みるときクラッシュ（障害情報）を監視できるようにしました。クラッシュ報告は HTTPS を経由して送信します。送信される情報は `docker-machine` のバージョン、構築に関して、OS 、アーキテクチャ、現在のシェル上のパス、ホスト上での直近の履歴といった、 `--debug` オプション指定時に参照できるものです。送信されたデータは `docker-machine` 実行時にどのような問題が発生しているかを、ピンポイントで把握できるようにします。送信されるのは `docker-machine` がクラッシュした場合のみです。

エラーメッセージを送りたくなければ、自分の `$HOME/docker/machine` ディレクトリに `no-error-report` ファイルを置きます。そうしておけば Docker Machine はレポートを報告しません。

```
$ mkdir -p ~/.docker/machine && touch ~/.docker/machine/no-error-report
```

このファイルは空のままで構いません。Docker Machine はファイルの存否のみ確認します。

ヘルプが必要ですか？

Docker Machine は開発途上であり、積極的に開発が行われています。ヘルプが必要であれば、あるいは貢献したい場合、一番簡単なのはプロジェクトの仲間に声をかけることです。私たちはコミュニケーションのために開かれたチャンネルを用意しています。

- バグ報告や機能リクエストを送りたい：GitHub の課題トラッカー[*1] をご利用ください。
- プロジェクトの人々とリアルタイムに会話したい：IRC の`#docker-machine` チャンネルにご参加ください。
- コードやドキュメント変更に貢献したい：GitHub にプル・リクエスト[*2] を送信ください。

更に詳しい情報やリソースについては、プロジェクトのヘルプページ[*3] をご覧ください。

*1 https://github.com/docker/machine/issues

*2 https://github.com/docker/machine/pulls

*3 https://docs.docker.com/project/get-help/

2 章

セットアップ

2.1 Docerk Machine のインストール

OS X と Windows の場合は、Docker Toolbox をインストールしたら、他の Docker プロダクトと一緒にインストールします。Docker Toolbox の詳細は、「Mac OS X インストールガイド」か「Windows インストールガイド」をご覧ください。

Docker Machine だけインストールしたい場合は、Machine のバイナリを直接インストールできます。詳細は次のセクションをご覧ください。また、最新版のバイナリは GitHub 上の docker/machine リリース・ページ[*1] 上で確認できます。

2.1.1 Machine を直接インストール

1. Docker クライアント（docker という名称のバイナリ・ファイル）をインストールします。

2. Docker Machine のバイナリ・ファイル（docker-machine）をダウンロードし、PATH に展開[*2] します。

Mac OS X もしくは Windows の場合：

```
$ curl -L https://github.com/docker/machine/releases/download/v0.7.0/docker-machine-`uname -s`-`uname
-m` > /usr/local/bin/docker-machine && \ ←１行で入力
chmod +x /usr/local/bin/docker-machine
```

Windows 上の git bash の場合：

```
$ if [[ ! -d "$HOME/bin" ]]; then mkdir -p "$HOME/bin"; fi && \
curl -L https://github.com/docker/machine/releases/download/v0.7.0/docker-machine-Windows-x86_64.exe
> "$HOME/bin/docker-machine.exe" && \ ←１行で入力
chmod +x "$HOME/bin/docker-machine.exe"
```

*1 https://github.com/docker/machine/releases/

*2 訳者注：環境変数 PATH（パス）に指定されているディレクトリ上に置きます。

あるいは、docker/machine リリース・ページから直接ダウンロードします。

3. Machine のバージョンを表示して、インストールを確認します。

```
$ docker-machine version
docker-machine version 0.7.0, build 61388e9
```

2.1.2 bash 補完スクリプトのインストール

Machine 用のリポジトリには次の機能を持つ bash スクリプトがあります。

- コマンド補完
- シェル・プロンプトにアクティブなホスト[*1] を表示
- docker-machine use サブコマンドを追加し、アクティブなマシンを切り替えるラッパー

スクリプトをインストールするには、/etc/bash_completion.d か /usr/local/etc/bash_completion.d にファイルをコピーするかリンクします。docker-machine シェル・プロンプトを有効化するには、~/.bashrc の PS1 に $(__docker-machine-ps1) を追加します。

```
PS1='[\u@\h \W$(__docker-machine-ps1)]\$ '
```

詳細なドキュメントは、各スクリプト[*2] の文頭にあるコメントをご覧ください。

*1 訳者注：Docker クライアントの直接操作対象です。

*2 https://github.com/docker/machine/tree/master/contrib/completion/bash

2.2 Docker Machine をローカル VM で始める

2.2.1 動作条件

システム上に正しくインストールするには、最新バージョンの VirtualBox(バーチャルボックス)[*1] をインストールする必要があります。Mac または Windows で Docker Machine のインストールに Docker Toolbox[*2] を使えば、VirtualBox を自動的にインストールします。

1台目のマシンを Quickstart Terminal で作成したら、ターミナル上に default という名称を持つ環境が自動的に用意されます。この場合、以下の手順をそのまま読み進めても構いませんが、「default」以外の名前（staging(ステージング) や sandbox(サンドボックス)）で別のマシンの作成も可能です。

2.2.2 Machine を使って Docker コンテナを実行

Docker コンテナを実行するには、

- 新しい Docker 仮想マシン[*3] を作成します（あるいは既存マシンを開始します)。
- 環境変数を新しい仮想マシンに切り替えます。
- docker クライアントを使い、コンテナの作成、読み込み、管理を行います。

Docker Machine で作成したマシンは、必要に応じて何度も再利用できます。マシンは VirtualBox 上の仮想マシンと同じで環境あり、どちらでも同じ設定が使われます。

以下の例で、マシンの作成・起動方法、Docker コマンドの実行方法、コンテナの使い方を見ていきます。

2.2.3 マシンの作成

1. コマンド・シェルやターミナル画面を開きます。

以下の例では Bash シェルを扱います。C シェルのような他のシェルでは、いくつかのコマンドが動作しない可能性がありますので、ご注意ください。

2. `docker-machine ls` を使い、利用可能なマシンの一覧を表示します。

以下の結果から、マシンがまだ1台も作成されていないことが分かります。

```
$ docker-machine ls
NAME   ACTIVE   DRIVER   STATE   URL   SWARM   DOCKER   ERRORS
```

3. マシンを作成します。

---

*1 https://www.virtualbox.org/wiki/Downloads

*2 https://www.docker.com/products/docker-toolbox

*3 訳者注：以降のドキュメントでは「Docker Machine」のツールを「Machine」と表記し、仮想マシンを「マシン」と表記します。

コマンド docker-machine create（クリエイト）実行時、 --driver（ドライバ）フラグに virtualbox の文字列を指定します。そして、最後の引数がマシン名になります。これが始めてのマシンであれば、名前を default にしましょう。既に「default 」という名前のマシンが存在している場合は、別の新しいマシン名を指定します。

```
$ docker-machine create --driver virtualbox default
Running pre-create checks...
Creating machine...
(staging) Copying /Users/ripley/.docker/machine/cache/boot2docker.iso to
/Users/ripley/.docker/machine/machines/default/boot2docker.iso...
(staging) Creating VirtualBox VM...
(staging) Creating SSH key...
(staging) Starting the VM...
(staging) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Machine is running, waiting for SSH to be available...
Detecting operating system of created instance...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect Docker to this machine, run: docker-machine env default
```

このコマンドは Docker デーモンをインストールする軽量 Linux ディストリビューション（boot2docker（ブートツードッカー）[*1]）をダウンロードし、Docker を動かすための VirtualBox 仮想マシンを作成・起動します。

4. 再び利用可能なマシン一覧表示したら、新しいマシンが出てきます。

```
$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER   ERRORS
default   *        virtualbox   Running   tcp://192.168.99.187:2376           v1.9.1
```

5. コマンドの環境変数を新しい仮想マシンに設定します。

コマンド docker-machine create を実行しても、そのまま新しいマシンを操作できないので注意が必要です。新しいマシンの操作には docker-machine env コマンドを使います。

```
$ docker-machine env default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://172.16.62.130:2376"
export DOCKER_CERT_PATH="/Users/<yourusername>/.docker/machine/machines/default"
export DOCKER_MACHINE_NAME="default"
# Run this command to configure your shell:
# eval "$(docker-machine env default)"
```

*1 https://github.com/boot2docker/boot2docker

6. シェルを新しいマシンに接続します[*1]。

```
$ eval "$(docker-machine env default)"
```

fish や Powershell、あるいは cmd.exe のような Windows シェルでは、先ほどのコマンドは実行できません。自分の使っているシェルで環境変数を有効にする方法は、env コマンドのドキュメントをご覧ください。

このシェル上で指定した環境変数を使えば、クライアントは指定された TLS 設定を読み込みます。新しいシェルの起動時やマシン再起動時には、再度指定する必要があります。

あとはホスト上で Docker コマンドを実行できます。

2.2.4 Machine コマンドを使ってコンテナを実行

セットアップが完了したことを確認するため、docker run コマンドを使ってコンテナを起動しましょう。

1. docker run コマンドを使い、busybox イメージをダウンロードし、簡単な echo コマンドを実行します。

```
$ docker run busybox echo hello world
Unable to find image 'busybox' locally
Pulling repository busybox
e72ac664f4f0: Download complete
511136ea3c5a: Download complete
df7546f9f060: Download complete
e433a6c5b276: Download complete
hello world
```

2. ホストの IP アドレスを確認します。

Docker ホスト上でポート番号が利用可能な IP アドレスの確認は、docker-machine ip コマンドを使います。

```
$ docker-machine ip default
192.168.99.100
```

3. コンテナでウェブサーバ（https://www.nginx.com/）を実行するため、次のコマンドを実行します。

```
$ docker run -d -p 8000:80 nginx
```

イメージの取得が完了したら、docker-machine ip で確認した IP アドレス上のポート 8000 でサーバにアクセスできます。実行例：

```
$ curl $(docker-machine ip default):8000
<!DOCTYPE html>
<html>
<head>
```

*1 訳者注：ドキュメントでは「接続」という表現を使っていますが、常に通信セッションを継続している状態ではありません。クライアントが操作対象の Docker Engine を指し示す状態を「接続」と表現しています。同様に、対象ホストを指さない状態を「切断」と表現しています。

```
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

あとは、好きなだけ実行したいローカル仮想マシンを作成・管理できます。そのためには docker-machine create を実行するだけです。作成されたマシン全ての情報を確認するには docker-machine ls を使います。

2.2.5 マシンの起動と停止

ホストを使い終わり、しばらく使わないのであれば、docker-machine stop を実行して停止できます。あとで起動したい場合は docker-machine start を実行します。

```
$ docker-machine stop default
$ docker-machine start default
```

2.2.6 マシンの名前を指定せずに操作するには

いくつかの docker-machine コマンドは、マシン名を明示しなければ default という名称のマシン（が存在している場合）に対して処理を行います。そのため、default ローカル仮想マシンは一般的なパターンとして、頻繁に利用できるでしょう。

実行例：

```
$ docker-machine stop
Stopping "default"....
Machine "default" was stopped.

$ docker-machine start
Starting "default"...
(default) Waiting for an IP...
Machine "default" was started.
Started machines may have new IP addresses.  You may need to re-run the `docker-machine env` command.

$ eval $(docker-machine env)

$ docker-machine ip
```

```
192.168.99.100
```

コマンドは以下の形式でも利用可能です。

- docker-machine config
- docker-machine env
- docker-machine inspect
- docker-machine ip
- docker-machine kill
- docker-machine provision
- docker-machine regenerate-certs
- docker-machine restart
- docker-machine ssh
- docker-machine start
- docker-machine status
- docker-machine stop
- docker-machine upgrade
- docker-machine url

default 以外のマシンでは、常に特定のマシン名をコマンドの引数として明示する必要があります。

2.2.7 起動時にローカル・マシンの自動起動

シェルのセッションを開く度に、Docker クライアントが自動的に毎回設定された状態にするには、対象ユーザのシェル profile（例：~/.bash_profile ファイル）に追記（eval $(docker-machine env default)）します。しかし、default のマシンが起動されていなければコマンドは実行できません。そのような場合は、システム起動時に default マシンが自動的に起動するよう設定します。

以下の例は OS X 上での設定です。

~/Library/LaunchAgents 以下に com.docker.machine.default.plist ファイルを作成します。内容は次の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>EnvironmentVariables</key>
        <dict>
            <key>PATH</key>
            <string>/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin</string>
        </dict>
        <key>Label</key>
        <string>com.docker.machine.default</string>
        <key>ProgramArguments</key>
        <array>
            <string>/usr/local/bin/docker-machine</string>
            <string>start</string>
            <string>default</string>
        </array>
        <key>RunAtLoad</key>
```

```
        <true/>
    </dict>
</plist>
```

この中にある LaunchAgent の default を書き換えれば、任意のマシン（群）を起動できます。

2.3 クラウド・プロバイダで Machine を使う

Docker Machine は様々なクラウド・プラットフォームに対応したプラグインを扱えます。このプラグインに対応したドライバを使えば、Machine でクラウド・ホストを自動作成します。そして、作成するホスト上に Docker Engine も自動インストールできます。

必要になるのは Docker Machine のインストール・実行と、利用するクラウド・プロバイダ上でのアカウント作成です。

次にアカウント証明書、セキュリティ証明書など、`docker-machine create` コマンドのオプション用フラグで必要なものを確認します。例えば、Digital Ocean のアクセス・トークンを指定するには `--digitalocean-access_token` フラグを使います。以降で Digital Ocean と AWS の設定例を見ていきます。

2.3.1 サンプル

Digital Ocean

次のコマンドは Digital Ocean 上に「docker-sandbox」という名前のドロップレット（クラウド・ホスト）を作成します。

```
$ docker-machine create --driver digitalocean --digitalocean-access-token xxxxx docker-sandbox
```

Machine を使い、Digital Ocean 上のホストで更なる操作をするには「Digital Ocean サンプル」のセクションをご覧ください。

Amazon Web Services (AWS)

次のコマンドは AWS EC2 上に「aws-sandbox」という名前のドロップレット（クラウド・ホスト）を作成します。

```
$ docker-machine create --driver amazonec2 --amazonec2-access-key AKI******* --amazonec2-secret-key
8T93C*******  aws-sandbox
```

Machine を使い、AWS 上の Docker 対応インスタンスで更なる操作をするには「Amazon Web Services (AWS) サンプル」のセクションをご覧ください。

2.3.2 docker-machine create コマンド

`docker-machine create` コマンドの実行時、いくつか最小限の指定が必要となります。

- `--driver` で、マシンを作成するプロバイダを明示します（VirtualBox、Digital Ocean AWS、等)。
- クラウド・サービスを使う場合は、(クラウド・プロバイダの）アカウント証明書やセキュリティ証明書の指定。
- `<マシン名>` で、作成したいホスト名を指定

扱いやすいように、`docker-machine` でサーバ作成時に一般的なオプションがデフォルトで適用されます。しかし、これらのデフォルト値はフラグを使って上書きできます（例： `--digitalocean-image`)。そのため、クラウド・サーバに多くのメモリや CPU を割り当てたい場合には便利でしょう（デフォルトの `docker-machine` は小さなサーバを作成します)。

デフォルトの値、あるいは利用可能なフラグや設定を全て確認したい場合は、コマンドラインで `docker-machine create -h` を使います。他にも、Machine コマンドライン・リファレンス の `create` コマンドや、Machine ドライ

バ・リファレンスの「ドライバのオプション」と「デフォルト OS」をご覧ください。

2.3.3 クラウド・プロバイダ向けのドライバ

Docker Machine をインストールしたら、様々なクラウド・プロバイダに対応したドライバ（Amazon Web Services 、Digital Ocean 、Microsoft Azure 等）と、ローカルのプロバイダ（Oracle VirtualBox 、VMware Fusion 、Microsoft Hyper-V）が利用可能になります。

各ドライバの詳細は Docker Machine ドライバ・リファレンス から、必要なフラグや設定オプション（プロバイダごとにかなり違います）をご確認ください。

2.3.4 サード・パーティのドライバ・プラグイン

サード・パーティの貢献者による様々なクラウド・プラットフォームに対応した Docker Machine プラグインがあります。これらのプラグイン利用にあたっては、利用者の皆さん自身でリスクを取ってください。Docker によって直接メンテナンスされているものではありません。

使うには、GItHub の docker/machine リポジトリ上の利用可能なドライバ・プラグイン[*1] をご覧ください。

2.3.5 ドライバを使わずにホストを追加

Docker の場所を指定したら、ドライバがないホストの追加が可能です。既存のホストに関するマシン名を指定することにより、Docker コマンド使用時に毎回オプションを指定する必要がなくなります。

```
$ docker-machine create --url=tcp://50.134.234.20:2376 custombox
$ docker-machine ls
NAME        ACTIVE   DRIVER    STATE     URL
custombox   *        none      Running   tcp://50.134.234.20:2376
```

2.3.6 Machine で Docker Swarm クラスタの自動構築

Docker Machine を使えば Docker Swarm クラスタのプロビジョンもできます。どのドライバを使っても TLS で安全にします。

- Swarm を使うには「Docker Swarm の入手方法」のドキュメントをご覧ください。
- Machine で Swarm クラスタを構築する方法は「Docker Machine で Swarm クラスタをプロビジョン」のドキュメントをご覧ください。

*1 https://github.com/docker/machine/blob/master/docs/AVAILABLE_DRIVER_PLUGINS.md

3 章

使い方

3.1 Digital Ocean

以下の例では Docker に対応した Digital Ocean ドロップレット（クラウド・ホスト）を作成します。

ステップ１：Digital Ocean アカウントの作成

アカウントの取得がまだであれば、Digital Ocean 上にアカウントを作成し、それからログインします。

ステップ２：自分のアクセス・トークンを生成

アクセス・トークンを生成します。

1. Digital Ocean 管理コンソールに移動し、ページ上方にある［ API ］をクリックします。

2. ［ Generate New Token ］（新しいトークンの生成）をクリックして、トークンの生成に進みます。

3. トークン名を指定し（例「machine」)、［ Write ］(Optional）にチェックが入っているのを確認してから［ Generate Token ］（トークン生成）をクリックします。

4. 生成された長いバイナリ文字列を取得し（クリップボードにコピーします)、どこか安全な場所に保管します。

これが次のクラウド・サーバの作成に必要となる、個人のアクセス・トークンです。

ステップ３：Machine を使ってドロップレットを作成

1. `docker-machine create` コマンドの実行時に、`digitalocean` ドライバと `--digitalocean-access-token` フラグで自分のキーを指定します。あわせて新しいクラウド・サーバ名も指定します。

次の例は、「docker-sandbox」という名前の新しいドロップレットを作成します。

```
$ docker-machine create --driver digitalocean --digitalocean-access-token xxxxx docker-sandbox
Running pre-create checks...
Creating machine...
```

```
(docker-sandbox) OUT | Creating SSH key...
(docker-sandbox) OUT | Creating Digital Ocean droplet...
(docker-sandbox) OUT | Waiting for IP address to be assigned to the Droplet...
Waiting for machine to be running, this may take a few minutes...
Machine is running, waiting for SSH to be available...
Detecting operating system of created instance...
Detecting the provisioner...
Provisioning created instance...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
To see how to connect Docker to this machine, run: docker-machine env docker-sandbox
```

Droplet が作成されたら、Docker はユニークな SSH 鍵を生成し、自分のローカル・システム上の ~/.docker/machines に保存します。当初、この鍵はホストのプロビジョニング用に使われます。後ほど、docker-machine ssh コマンドでドロップレットに簡単にアクセスするときにも使います。Docker Engine はクラウド・サーバ上にインストールされます。そして、TCP を通してリモートからの通信を受け付けられるように TLS 認証を使います。

2. Digital Ocean コンソールに移動し、新しいドロップレットの情報を確認します。

3. コマンド・ターミナル上で docker-machine ls を実行します。

```
$ docker-machine ls
NAME             ACTIVE   DRIVER         STATE     URL                         SWARM
default          -        virtualbox     Running   tcp://192.168.99.100:2376
docker-sandbox   *        digitalocean   Running   tcp://45.55.139.48:2376
```

新しい docker-sandbox マシンが実行されています。そして、アクティブなホストはアスタリスク（*）印が付いています。新しいマシンを作成したら、コマンド・シェルから自動的に接続できます。何らかの理由により、新しいマシンがアクティブなホストでない場合は docker-machine env docker-sandbox を実行し、反映するためには eval $(docker-machine env docker-sandbox) の実行が必要です。

ステップ４：ドロップレット上で Docker コマンドを実行

1. docker-machine コマンドを使ってリモート・ホストの情報を確認できます。例えば、docker-machine ip <マシン名> はホスト側の IP アドレスを取得します。より詳しい情報は docker-machine inspect <マシン名> で確認できます。

```
$ docker-machine ip docker-sandbox
104.131.43.236

$ docker-machine inspect docker-sandbox
{
    "ConfigVersion": 3,
    "Driver": {
    "IPAddress": "104.131.43.236",
    "MachineName": "docker-sandbox",
    "SSHUser": "root",
    "SSHPort": 22,
    "SSHKeyPath": "/Users/samanthastevens/.docker/machine/machines/docker-sandbox/id_rsa",
    "StorePath": "/Users/samanthastevens/.docker/machine",
```

```
"SwarmMaster": false,
"SwarmHost": "tcp://0.0.0.0:3376",
"SwarmDiscovery": "",
...
```

2. Docker Engine が正しくインストールされたかどうか確認するため、docker コマンドを実行します。

docker run hello-world のような基本的なコマンドを、新しいリモート・マシン上で実行します。あるいは、より面白いテストとなるよう Docker に対応したウェブサーバを実行します。

次の例は -p オプションで nginx コンテナのポート 80 を公開できるようにし、それを docker-sandbox ホスト上のポート 8000 に割り当てます。

```
$ docker run -d -p 8000:80 --name webserver kitematic/hello-world-nginx
Unable to find image 'kitematic/hello-world-nginx:latest' locally
latest: Pulling from kitematic/hello-world-nginx
a285d7f063ea: Pull complete
2d7baf27389b: Pull complete
...
Digest: sha256:ec0ca6dcb034916784c988b4f2432716e2e92b995ac606e080c7a54b52b87066
Status: Downloaded newer image for kitematic/hello-world-nginx:latest
942dfb4a0eaae75bf26c9785ade4ff47ceb2ec2a152be82b9d7960e8b5777e65
```

ウェブブラウザで http://<ホスト IP>:8000 を開き、ウェブサーバのホームページを開きます。 ホスト IP の確認は、先ほどの docker-machine ip <マシン名> コマンドで行いました。docker run コマンドを実行したら、指定したポートを開きます。

ステップ５：Machine でドロップレットを削除

ホストだけでなく全てのコンテナとイメージを削除するには、マシンを停止するために docker-machine rm を使います。

```
$ docker-machine stop docker-sandbox
$ docker-machine rm docker-sandbox
Do you really want to remove "docker-sandbox"? (y/n): y
Successfully removed docker-sandbox

$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM
default   *        virtualbox   Running   tcp:////xxx.xxx.xx.xxx:xxxx
```

コマンドを実行後に Digital Ocean コンソールを確認したら、すぐにドロップレットが停止し、削除されるのが分かるでしょう。

Docker Machine は作成したホストは、クラウド・プロバイダのコンソールからも削除できます。ただし Machine からは状況が追跡できなくなります。そのため、docker-machine create で作成したホストは docker-machine rm をお使いください。

3.2 Amazon Web Services (AWS) EC2 の例

以下の例では Docker に対応した Amazon Web Services (AWS) EC2 インスタンスを作成します。

ステップ１：AWS にサインアップして証明書を取得

1. まだ AWS の利用者でなければ、AWS にサインアップし、EC2 クラウド・コンピュータに対して root アクセスを持つアカウントを作成します。

既に Amazon アカウントをお持ちであれば、自分の root ユーザ・アカウントを利用できます。

2. IAM (Identity and Access Management) 管理ユーザと管理グループを作成し、リージョンに鍵ペアを関連づけます。

まず、AWS メニューの［ サービス ］から［ IAM ］を選びます。

AWS 上でマシンを作成するには、２つのパラメータが必要です。

- AWS アクセスキー
- AWS シークレットアクセスキー

AWS にある Amazon EC2 でのセットアップ[1] のドキュメントをご覧ください。この中にある「IAM ユーザーを作成する」「キーペアを作成する」の各手順を進めます。

ステップ２：Machine でインスタンスを作成

1. オプションで AWS 認証用ファイルを作成できます。

`~/.aws/credentials` ファイルを作成し、AWS 鍵を記述できます。そうしておけば、`docker-machine create` コマンドを実行する度に入力する必要はありません。以下が認証用ファイルの例です。

```
[default]
aws_access_key_id = AKID1234567890
 aws_secret_access_key = MY-SECRET-KEY
```

2. `docker-machine create` コマンドの実行時、`amazonec2` ドライバと鍵と新しいインスタンス名を指定します。

認証用ファイルを使う場合

鍵を認証用ファイルに入れている場合は、次のコマンドを実行すると `aws-sandbox` という名前のインスタンスを起動します。

```
$ docker-machine create --driver amazonec2 aws-sandbox
```

*1 http://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html

鍵をコマンドラインで指定する場合

認証用ファイルを使わない場合は、コマンドラインで --amazonec2-access-key と --amazonec2-secret-key を指定します。

```
$ docker-machine create --driver amazonec2 \
  --amazonec2-access-key AKI******* --amazonec2-secret-key 8T93C*******  aws-sandbox
```

リージョンの指定

デフォルトでは、ドライバは新しいインスタンスを us-east-1 (North Virginia) リージョンで作成します。別のリージョンで作成するには --amazonec2-region フラグを使います。例えば「aws-01」マシンを us-west-1 (Northern California)で作成するには、次のように実行します。

```
$ docker-machine create --driver amazonec2 --amazonec2-region us-west-1 aws-01
```

3. AWS EC2 ダッシュボードに移動し、新しいインスタンスを確認します。

AWS に IAM 証明書でログインし、EC2 実行中のインスタンスの画面に移動します。

メニュー右上で対象のリージョンを選択してください。そうすると、インスタンスが見えるでしょう。docker-machine create 実行時にリージョンを指定しなければ（オプションの --amazonec2-region フラグを使う)、デフォルトでは US East リージョンになります。

4. コマンド・ターミナル上で docker-machine ls を実行します。

```
$ docker-machine ls
NAME            ACTIVE   DRIVER        STATE     URL                         SWARM   DOCKER
ERRORS
aws-sandbox     *        amazonec2     Running   tcp://52.90.113.128:2376            v1.10.0
default         -        virtualbox    Running   tcp://192.168.99.100:2376           v1.10.0-rc4
aws-sandbox     -        digitalocean  Running   tcp://104.131.43.236:2376           v1.9.1
```

新しい aws-sandbox マシンが実行されています。そして、アクティブなホストはアスタリスク (*) 印が付いています。新しいマシンを作成すると、コマンド・シェルから自動的に接続できます。何らかの理由により、新しいマシンがアクティブなホストでない場合は docker-machine env aws-sandbox を実行し、反映するためには eval $(docker-machine env aws-sandbox) の実行が必要です。

ステップ３：インスタンス上で Docker コマンドを実行

1. docker-machine コマンドを使ってリモート・ホストの上方を確認できます。例えば、docker-machine ip <マシン名> はホスト側の IP アドレスを取得します。より詳しい情報は docker-machine inspect <マシン名> で確認できます。

```
$ docker-machine ip
192.168.99.100

$ docker-machine inspect aws-sandbox
{
    "ConfigVersion": 3,
    "Driver": {
```

```
    "IPAddress": "52.90.113.128",
    "MachineName": "aws-sandbox",
    "SSHUser": "ubuntu",
    "SSHPort": 22,
    ...
```

2. Docker Engine が正しくインストールされたかどうか確認するため、docker コマンドを実行します。

docker run hello-world のような基本的なコマンドを、新しいリモート・マシン上で実行します。あるいは、より面白いテストとなるよう Docker に対応したウェブサーバを実行します。

次の例は -p オプションで nginx コンテナのポート 80 を公開できるようにし、それを aws-sandbox ホスト上のポート 8000 に割り当てます。

```
$ docker run -d -p 8000:80 --name webserver kitematic/hello-world-nginx
Unable to find image 'kitematic/hello-world-nginx:latest' locally
latest: Pulling from kitematic/hello-world-nginx
a285d7f063ea: Pull complete
2d7baf27389b: Pull complete
...
Digest: sha256:ec0ca6dcb034916784c988b4f2432716e2e92b995ac606e080c7a54b52b87066
Status: Downloaded newer image for kitematic/hello-world-nginx:latest
942dfb4a0eaae75bf26c9785ade4ff47ceb2ec2a152be82b9d7960e8b5777e65
```

ウェブブラウザで http://<ホスト IP>:8000 を開き、ウェブサーバのホームページを開きます。ホスト IP の確認は、先ほどの docker-machine ip <マシン名> コマンドで行いました。docker run コマンドを実行すると、指定したポートを開きます。

ステップ４：Machine でインスタンスを削除

ホストだけでなく全てのコンテナとイメージを削除するには、マシンを停止するために docker-machine rm を使います。

```
$ docker-machine stop aws-sandbox
$ docker-machine rm aws-sandbox
Do you really want to remove "aws-sandbox"? (y/n): y
Successfully removed aws-sandbox
```

3.3 コマンドライン補完

Docker Machine は bash シェルで**コマンド補完(command-line completion)**が使えます。

コマンド補完のインストール

bash

bash 補完（bash completion）がインストールされているかどうか確認します。お使いの Linux 環境が最小インストールでなければ、おそらく補完機能が利用できます。Mac では `brew install bash-completion` でインストールします。

補完スクリプトを `/etc/bash_completion.d/` に置きます（Mac の場合 `brew --prefix`/etc/bash_completion.d/`）。例：

```
files=(docker-machine docker-machine-wrapper docker-machine-prompt)
for f in "${files[@]}"; do
  curl -L https://raw.githubusercontent.com/docker/machine/v$(docker-machine --version | tr -ds ',' ' '
| awk 'NR==1{print $(3)}')/contrib/completion/bash/$f.bash > `brew --prefix`/etc/bash_completion.d/$f
done
```

次回ログイン時から補完機能が使えます。

3.4 Boot2Docker から Machine への移行

これまで Boot2Docker を使っていた場合は、既に Docker の boot2docker-vm 仮想マシンがローカルシステム上に存在しています。Docker Machine で古い仮想マシンを管理する場合は、移行が必要です。

1. ターミナルか、システム上の Docker CLI を開きます。
2. 次のコマンドを実行します。

```
$ docker-machine create -d virtualbox --virtualbox-import-boot2docker-vm boot2docker-vm docker-vm
```

3. docker-machine コマンドを使い、対話式に仮想マシンを移行します。

サブコマンドの比較

docker-machine サブコマンドは、boot2docker サブコマンドと若干の違いがあります。次の表は docker-machine サブコマンドとの互換性を比較したものです。

boot2docker	docker-machine	docker-machine の説明
init	create	新しい docker ホストの作成
up	start	停止しているマシンの起動
ssh	ssh	コマンドの実行やマシンとの双方向 ssh セッション
save	−	使用不可
down	stop	実行中のマシンの停止
poweroff	stop	実行中のマシンの停止
reset	restart	実行中のマシンの再起動
config	inspect	マシン設定詳細の表示
status	ls	マシン一覧と状態の表示
info	inspect	マシンの詳細を表示
ip	ip	マシンの IP アドレスを表示
shellinit	env	シェルがマシンと対話するために必要なコマンドの表示
delete	rm	マシンの削除
download	−	使用不可
upgrade	upgrade	マシン上の Docker クライアントを最新安定版に更新

参

4 章

Machine コマンドライン・リファレンス

4.1 docker machine サブコマンド

active（アクティブ）

どのマシンが「アクティブ」かを表示します（Docker Machine は DOCKER_HOST 環境変数が示すところをアクティブとみなします）。

```
$ docker-machine ls
NAME      ACTIVE   DRIVER         STATE     URL
dev       -        virtualbox     Running   tcp://192.168.99.103:2376
staging   *        digitalocean   Running   tcp://203.0.113.81:2376
$ echo $DOCKER_HOST
tcp://203.0.113.81:2376
$ docker-machine active
staging
```

コンフィグ
config

対象マシンに対する Docker クライアントの設定を表示します。

使い方: docker-machine config [オプション] [引数...]

マシンに接続する設定を表示

説明:
引数はマシン名。

オプション:

--swarm Docker デーモンの代わりに Swarm 設定を表示

実行例：

```
$ docker-machine config dev
--tlsverify
--tlscacert="/Users/ehazlett/.docker/machines/dev/ca.pem"
--tlscert="/Users/ehazlett/.docker/machines/dev/cert.pem"
--tlskey="/Users/ehazlett/.docker/machines/dev/key.pem"
-H tcp://192.168.99.103:2376
```

クリエイト
create

マシンを作成します。どのプロバイダ（VirtualBox、Digital Ocean、AWS 等）でマシンを作成するかを --driver フラグで指定します。更に、引数で作成するマシンの名前も指定します。

```
$ docker-machine create --driver virtualbox dev
Creating CA: /home/username/.docker/machine/certs/ca.pem
Creating client certificate: /home/username/.docker/machine/certs/cert.pem
Image cache does not exist, creating it at /home/username/.docker/machine/cache...
No default boot2docker iso found locally, downloading the latest release...
Downloading https://github.com/boot2docker/boot2docker/releases/download/v1.6.2/boot2docker.iso to
/home/username/.docker/machine/cache/boot2docker.iso...
Creating VirtualBox VM...
Creating SSH key...
Starting VirtualBox VM...
Starting VM...
To see how to connect Docker to this machine, run: docker-machine env dev
```

ヘルプ・テキストで、ドライバを指定するフラグを使う

docker-machine create コマンドには全てのドライバで適用できる共通のフラグがあります。主に、マシンのプロビジョニング手順における挙動を制御するもので（Docker Swarm コンテナの作成も含みます)、利用者がカスタマイズできます。

```
$ docker-machine create
Docker Machine Version: 0.5.0 (45e3688)
使い方: docker-machine create [オプション] [引数...]

マシンを作成する。

'docker-machine create --driver 名前' を実行すると、対象ドライバで作成時のヘルプ文字列を表示。

オプション:

   --driver, -d "none"
   マシン作成に使うドライバ
   --engine-install-url "https://get.docker.com"
   エンジンをインストールするカスタム URL [$MACHINE_DOCKER_INSTALL_URL]
   --engine-opt [--engine-opt option --engine-opt option]
  engine 作成時に任意のフラグを flag=value 形式で指定
   --engine-insecure-registry [--engine-insecure-registry option --engine-insecure-registry option]
  作成するエンジンで安全では無いレジストリ（insecure registry）を指定
   --engine-registry-mirror [--engine-registry-mirror option --engine-registry-mirror option]
  レジストリのミラーを使う指定 [$ENGINE_REGISTRY_MIRROR]
   --engine-label [--engine-label option --engine-label option]
  engine 作成時にラベルを指定
   --engine-storage-driver
   engine が使うストレージ・ドライバの指定
   --engine-env [--engine-env option --engine-env option]
  engine で使う環境変数を指定
   --swarm
   Swarm と Machine を使う設定
   --swarm-image "swarm:latest"
   Swarm が使う Docker イメージの指定 [$MACHINE_SWARM_IMAGE]
```

```
   --swarm-master
   Machine で Swarm マスタ用の設定
   --swarm-discovery
   Swarm で使うディスカバリ・サービス
   --swarm-strategy "spread"
   Swarm のデフォルト・スケジューリング・ストラテジを指定
   --swarm-opt [--swarm-opt option --swarm-opt option]
  swarm に任意のフラグを指定
   --swarm-host "tcp://0.0.0.0:3376"
   Swarm マスタ上でリッスンする ip/socket
   --swarm-addr
   Swarm のアドバタイズ・アドレス（デフォルト: 検出、もしくはマシン IP を使用）
   --swarm-experimental
```

更に、Machine は各プラグイン・コードに含むフラグも受け付けることができ、これをドライバのフラグで指定できます。これにより、利用者は作成するマシン向けプロバイダ固有のパラメータをカスタマイズできます。例えば、容量（ `--amazonec2-instance-type m1.medium` ）や地理的なリージョン（ `--amazonec2-region us-west-1` ）などです。

プロバイダ固有のフラグを確認するには `create` と `--driver` にヘルプ・テキストの表示を単純に指定するだけです。

```
$ docker-machine create --driver virtualbox --help
使い方: docker-machine create [オプション] [引数...]

マシンを作成。

'docker-machine create --driver 名前' を実行すると、対象ドライバで作成時のヘルプ文字列を表示。

オプション:

   --driver, -d "none"
   マシン作成に使うドライバ
   --engine-env [--engine-env option --engine-env option]
  engine で使う環境変数を指定
   --engine-insecure-registry [--engine-insecure-registry option --engine-insecure-registry option]
  作成するエンジンで安全では無いレジストリ（insecure registry）を指定
   --engine-install-url "https://get.docker.com"
   エンジンをインストールするカスタム URL [$MACHINE_DOCKER_INSTALL_URL]
   --engine-label [--engine-label option --engine-label option]
  engine 作成時にラベルを指定
   --engine-opt [--engine-opt option --engine-opt option]
  engine 作成時に任意のフラグを flag=value 形式で指定
   --engine-registry-mirror [--engine-registry-mirror option --engine-registry-mirror option]
  レジストリのミラーを使う指定 [$ENGINE_REGISTRY_MIRROR]
   --engine-storage-driver
   engine が使うストレージ・ドライバの指定
   --swarm
   Swarm と Machine を使う設定
   --swarm-addr
   Swarm のアドバタイズ・アドレス（デフォルト: 検出、もしくはマシン IP を使用）
   --swarm-discovery
   Swarm で使うディスカバリ・サービス
   --swarm-experimental
   Swarm の実験的機能を有効化
   --swarm-host "tcp://0.0.0.0:3376"
```

```
   Swarm マスタ上でリッスンする ip/socket
   --swarm-image "swarm:latest"
   Swarm が使う Docker イメージの指定 [$MACHINE_SWARM_IMAGE]
   --swarm-master
   Machine で Swarm マスタ用の設定
   --swarm-opt [--swarm-opt option --swarm-opt option]
  swarm に任意のフラグを指定
   --swarm-strategy "spread"
   Swarm のデフォルト・スケジューリング・ストラテジを指定
   --virtualbox-boot2docker-url
   boot2docker イメージの URL を指定。デフォルトは利用可能な最新バージョン
[$VIRTUALBOX_BOOT2DOCKER_URL]
   --virtualbox-cpu-count "1"
   マシンで使う CPU 数（-1 は利用可能な CPU 全て）[$VIRTUALBOX_CPU_COUNT]
   --virtualbox-disk-size "20000"
   ホストのディスク容量を MB 単位で指定 [$VIRTUALBOX_DISK_SIZE]
   --virtualbox-host-dns-resolver
   ホストが使う DNS リゾルバ [$VIRTUALBOX_HOST_DNS_RESOLVER]
   --virtualbox-dns-proxy
   全ての DNS リクエストをホストへプロキシ [$VIRTUALBOX_DNS_PROXY]
   --virtualbox-hostonly-cidr "192.168.99.1/24"
   ホスト・オンリー CIDR の指定 [$VIRTUALBOX_HOSTONLY_CIDR]
   --virtualbox-hostonly-nicpromisc "deny"
   ホスト・オンリー・ネットワーク・アダプタをプロミスキャスト・モードに指定
[$VIRTUALBOX_HOSTONLY_NIC_PROMISC]
   --virtualbox-hostonly-nictype "82540EM"
   ホスト・オンリー・ネットワーク・アダプタの種類を指定 [$VIRTUALBOX_HOSTONLY_NIC_TYPE]
   --virtualbox-import-boot2docker-vm
   取り込む　Boot2Docker VM のイメージ名
   --virtualbox-memory "1024"
   ホスト側のメモリ容量を MB で指定 [$VIRTUALBOX_MEMORY_SIZE]
   --virtualbox-no-share
```

環境変数を使ってもフラグと同様の指定ができますので、覚えておいてください（列の左側にあります）。環境変数は `docker-machine create` の実行時に読み込まれ、Docker machine はフラグのデフォルト値を上書きします。

Docker エンジン作成用のオプションを指定

作成時の手順において、Docker　Machine は Docker をインストールし、適切な初期設定をします。例えば、外の世界から TLS をベースとした暗号化 TCP を通して通信できるようにし、ストレージ・ドライバ が利用可能であれば AUFS を設定します。

Docker エンジン（あるいは Docker デーモン）に対して、利用者は自分自身でオプションを設定すべきケースが複数あります。例えば、自分たちで実行しているレジストリに接続するには、デーモンに対して `--insecure-registry` フラグを使う必要があります。Docker Machine で `create` コマンドを使ってエンジンを作成する場合、 `--engine` で始まるフラグを設定できます。

Docker Machine は、デーモンに対するパラメータを単にセットするだけであり、「依存関係」については設定しませんので、ご注意ください。例えば、デーモンでストレージ・ドライバに `btrfs` を指定する場合は、自分自身で依存関係のインストールと、BTRFS ファイルシステムの作成等が必要です。

```
$ docker-machine create -d virtualbox \
    --engine-label foo=bar \
    --engine-label spam=eggs \
    --engine-storage-driver overlay \
    --engine-insecure-registry registry.myco.com \
```

```
    foobarmachine
```

これはローカルの VirtualBox に仮想マシンを作成するにあたり、ストレージのバックエンドには overlay を使用し、エンジンのラベルとしてキーバリュー・ペアの foo=bar と spam=eggs を指定します。更に、 registry.myco.com にある非安全なレジストリへのイメージ送信・取得を許可します。詳細情報は docker info の出力結果から確認できます。

```
$ eval $(docker-machine env foobarmachine)
$ docker info
Containers: 0
Images: 0
Storage Driver: overlay
...
Name: foobarmachine
...
Labels:
 foo=bar
 spam=eggs
 provider=virtualbox
```

ここでは次のフラグが使えます。

- --engine-insecure-registry : 作成するエンジンが、指定した安全では無いレジストリと通信できるようにする。
- --engine-registry-mirror : 使用するレジストリ・ミラーを指定。
- --engine-label : 作成するエンジン用のラベルを指定。
- --engine-storage-driver : エンジンが使うストレージ・ドライバを指定。

エンジンは複数回のラベル指定（ --label を使用）をサポートしており、Docker Machine で設定できます。

デーモンのフラグを直接指定できるのに加え、Docker Machine は --engine-opt という追加フラグもサポートしています。これは --engine-opt flagname=value の形式で、特別な属性を持つデーモンのオプション指定に使います。例えば、全てのコンテナが DNS サーバに 8.8.8.8 を使うようデーモンに指定したり、常に syslog ログ・ドライバ を使って実行させたりするには、次のように create コマンドを使います。

```
$ docker-machine create -d virtualbox \
    --engine-opt dns=8.8.8.8 \
    --engine-opt log-driver=syslog \
    gdns
```

更に、Docker Machine は --engine-env フラグをサポートしています。これは外部の環境変数を指定するものであり、エンジンに適用するには --engine-env name=value の形式で指定します。例えば、エンジンが example.com をプロキシ・サーバとして使うには、create コマンドで次のように実行します。

```
$ docker-machine create -d virtualbox \
    --engine-env HTTP_PROXY=http://example.com:8080 \
    --engine-env HTTPS_PROXY=https://example.com:8080 \
    --engine-env NO_PROXY=example2.com \
    proxbox
```

マシン作成時に Docker Swarm オプションを指定

先ほどの Docker Engine オプションの設定を指定できるだけではありません。Docker Machine を使えば、Swarm マスタをどのように作成するかも指定できます。 --swarm-strategy フラグを使えば、Docker Swarm が使うべきスケジューリング・ストラテジ（デフォルトは spread ストラテジ ）を指定できます。また前述した `--engine-opt` オプションで指定したように、 `--swarm-opt` オプションで一般的なオプションを設定できますが、違いは `swarm manage` コマンドに対するオプション（マスタ・ノードの起動時に使用）を指定するものです。これらの機能設定を使うことで、パワーユーザであれば heartbeat 間隔の調整や、Swarm のオーバーコミット・リソースの調整に活用できるでしょう。また、 `--swarm-experimental` フラグを使えば Docker Swarm の実験的機能が利用可能になります。

どのようにオプションを設定するか分からない場合は、何も指定しないのがベストな方法です。何も心配しなくても、Docker Machine は適切に初期設定を行います。

作成例：

```
$ docker-machine create -d virtualbox \
    --swarm \
    --swarm-master \
    --swarm-discovery token://<token> \
    --swarm-strategy binpack \
    --swarm-opt heartbeat=5 \
    upbeat
```

こちらは Swarm スケジューリング・ストラテジに「binpack」を指定し（ホストに広く展開するのではなく、できるだけコンテナをホストに集約する設定)、「heartbeat」間隔を 5 秒にします。

作成の事前確認

多くのドライバで、それぞれの場所で実際に作成可能どうか確認する必要があるでしょう（例：VirtualBox がインストールされているかや、指定する API 証明書が有効かどうか)。Docker Machine は「作成の事前確認」（pre-create check）をドライバごとに行えます。

事前確認が成功したら、Docker Machine は通常通り作成手順を進行します。事前確認に失敗したら、Docker Machine のプロセスは終了コード 3 で終了します。つまり、ゼロ以外の終了コードを返す場合は、事前作成に失敗したのが分かります。

エンブ
env

docker コマンドの実行時に、特定のマシンを指し示せるような環境変数を表示します。

```
$ docker-machine env --help

使い方: docker-machine env [オプション] [引数...]

Docker クライアント用の環境変数をセットアップするコマンドを表示

説明:
   引数はマシン名。

オプション:

   --swarm  Docker デーモンの代わりに Swarm の設定を表示
   --shell  環境変数を設定するシェルを指定: [fish, cmd, powershell], デフォルトは sh/bash
   --unset, -u  環境変数の値を指定せずにリセット
   --no-proxy  マシンの IP アドレスに NO_PROXY 環境変数の追加
```

docker-machine env マシン名 を実行したら、サブシェル上で実行可能な export コマンドが表示されます。docker-machine env -u を実行したら、この効果を無効化する unset コマンドを表示します。

```
$ env | grep DOCKER
$ eval "$(docker-machine env dev)"
$ env | grep DOCKER
DOCKER_HOST=tcp://192.168.99.101:2376
DOCKER_CERT_PATH=/Users/nathanleclaire/.docker/machines/.client
DOCKER_TLS_VERIFY=1
DOCKER_MACHINE_NAME=dev
$ # If you run a docker command, now it will run against that host.
$ eval "$(docker-machine env -u)"
$ env | grep DOCKER
$ # The environment variables have been unset.
```

上の出力は bash や zsh シェル上での実行を想定したものです（どのシェルを使っているか分からくても、大抵の場合は bash でしょう）。しかし、Docker Machine がサポートしているシェルはこれだけではありません。どのようなコマンドを使うかは、それぞれの環境にあわせる必要があります。現時点では bash、cmd、powershell、emacs のシステムをサポートしています。

もし fish を使っており、SHELL 環境変数が fish のパスを適切に設定しているのであれば、docker-machine env マシン名 を実行したら、fish を想定した形式で値を表示します。

```
set -x DOCKER_TLS_VERIFY 1;
set -x DOCKER_CERT_PATH "/Users/nathanleclaire/.docker/machine/machines/overlay";
set -x DOCKER_HOST tcp://192.168.99.102:2376;
set -x DOCKER_MACHINE_NAME overlay
# Run this command to configure your shell:
# eval "$(docker-machine env overlay)"
```

docker-machine env コマンドはシェルを自動的に検出します。しかし、もし Windows でパワーシェルや cmd.exe を使う場合であれば、自動検出できません。そのため、docker-machine env に自分で --shell フラグのオプションを上書き指定する必要があります。

パワーシェルの例：

```
$ docker-machine.exe env --shell powershell dev
$Env:DOCKER_TLS_VERIFY = "1"
$Env:DOCKER_HOST = "tcp://192.168.99.101:2376"
$Env:DOCKER_CERT_PATH = "C:\Users\captain\.docker\machine\machines\dev"
$Env:DOCKER_MACHINE_NAME = "dev"
# Run this command to configure your shell:
# docker-machine.exe env --shell=powershell dev | Invoke-Expression
```

cmd.exe の例：

```
$ docker-machine.exe env --shell cmd dev
set DOCKER_TLS_VERIFY=1
set DOCKER_HOST=tcp://192.168.99.101:2376
set DOCKER_CERT_PATH=C:\Users\captain\.docker\machine\machines\dev
set DOCKER_MACHINE_NAME=dev
# Run this command to configure your shell: copy and paste the above values into your command prompt
```

プロキシを使わずにマシンを作成

env コマンドは --no-proxy フラグをサポートしています。これは、作成するマシンの IP アドレスに NO_PROXY / no_proxy 環境変数を追加します。

インターネットへのアクセスに HTTP プロキシが必要なネットワーク環境では、ローカルの仮想マシン・プロバイダ（例：virtualbox や vmwarefusion）で docker-machine を使うのにこれが役立ちます。

```
$ docker-machine env --no-proxy default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.104:2376"
export DOCKER_CERT_PATH="/Users/databus23/.docker/machine/certs"
export DOCKER_MACHINE_NAME="default"
export NO_PROXY="192.168.99.104"
# Run this command to configure your shell:
# eval "$(docker-machine env default)"
```

また、create 用の設定ドキュメントでは、docker-machine create コマンドで作成時に --engine-env フラグでデーモンの HTTP_PROXY を指定する方法も参考になるでしょう。

help（ヘルプ）

使い方: docker-machine help [引数...]

全てのコマンドを表示するか、１つのコマンドのヘルプを表示

使い方: docker-machine help サブコマンド

ヘルプ・テキストを表示するには、次のように実行します。

```
$ docker-machine help config
使い方: docker-machine config [オプション] [引数...]

マシンに接続する設定を表示

説明:
   引数はマシン名。

オプション:

   --swarm       Docker デーモンの代わりに Swarm の設定を表示
```

inspect（インスペクト）

使い方: docker-machine inspect [オプション] [引数...]

マシンに関する情報を調査

説明:
引数はマシン名。

オプション:
--format, -f go template で指定した出力に整形

デフォルトでは、マシンの情報を JSON 形式で表示します。フォーマットを指定すると、特定のテンプレートを使った結果を表示します。

Go 言語の text/template パッケージのページに、全てのフォーマットに関する情報の説明があります。

text/template 構文の他にも、json や prettyjson で JSON 形式の出力も可能です（ドキュメントは以下をご覧ください。）。

例

全マシンの詳細を表示：

こちらは inspect のデフォルトの使い方です。

```
$ docker-machine inspect dev
{
    "DriverName": "virtualbox",
    "Driver": {
        "MachineName":
"docker-host-128be8d287b2028316c0ad5714b90bcfc11f998056f2f790f7c1f43f3d1e6eda",
        "SSHPort": 55834,
        "Memory": 1024,
        "DiskSize": 20000,
        "Boot2DockerURL": "",
        "IPAddress": "192.168.5.99"
    },
    ...
}
```

マシンの IP アドレスを取得：

JSON の出力結果全体から、適切なフィールドのみを適切に取得できます。

```
$ docker-machine inspect --format='{{.Driver.IPAddress}}' dev
192.168.5.99
```

フォーマットの詳細：

JSON 形式として情報のサブセットが欲しい場合は、テンプレートの中で json ファンクションが使えます。

```
$ docker-machine inspect --format='{{json .Driver}}' dev-fusion
{"Boot2DockerURL":"","CPUS":8,"CPUs":8,"CaCertPath":"/Users/hairyhenderson/.docker/machine/certs/ca.
pem","DiskSize":20000,"IPAddress":"172.16.62.129","ISO":"/Users/hairyhenderson/.docker/machine/machi
nes/dev-fusion/boot2docker-1.5.0-GH747.iso","MachineName":"dev-fusion","Memory":1024,"PrivateKeyPath
":"/Users/hairyhenderson/.docker/machine/certs/ca-key.pem","SSHPort":22,"SSHUser":"docker","SwarmDis
covery":"","SwarmHost":"tcp://0.0.0.0:3376","SwarmMaster":false}
```

json 形式は使い易いのですが、人間にとって非常に読み辛いです。人が読む場合は prettyjson が使えます。

```
$ docker-machine inspect --format='{{prettyjson .Driver}}' dev-fusion
{
    "Boot2DockerURL": "",
    "CPUS": 8,
    "CPUs": 8,
    "CaCertPath": "/Users/hairyhenderson/.docker/machine/certs/ca.pem",
    "DiskSize": 20000,
    "IPAddress": "172.16.62.129",
    "ISO": "/Users/hairyhenderson/.docker/machine/machines/dev-fusion/boot2docker-1.5.0-GH747.iso",
    "MachineName": "dev-fusion",
    "Memory": 1024,
    "PrivateKeyPath": "/Users/hairyhenderson/.docker/machine/certs/ca-key.pem",
    "SSHPort": 22,
    "SSHUser": "docker",
    "SwarmDiscovery": "",
    "SwarmHost": "tcp://0.0.0.0:3376",
    "SwarmMaster": false
}
```

ip

1 つまたは複数マシンの IP アドレスを表示します。

```
$ docker-machine ip dev
192.168.99.104
$ docker-machine ip dev dev2
192.168.99.104
192.168.99.105
```

kill（キル）

使い方: docker-machine kill [引数...]

マシンを KILL (強制停止)する。

説明:
引数は１つまたは複数のマシン名

マシンを kill （強制的に即時停止）します。

```
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL
dev    *        virtualbox   Running   tcp://192.168.99.104:2376
$ docker-machine kill dev
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL
dev    *        virtualbox   Stopped
```

ls

使い方: docker-machine ls [オプション] [引数...]

マシン一覧を表示

オプション:

--quiet, -q	静かな（quite）モードを有効化
--filter [--filter option --filter option]	指定した状況に応じてフィルタを出力
--timeout, -t "10"	タイムアウトを秒数で指定。デフォルトは 10 秒
--format, -f	Go テンプレートに一致する内容で表示

タイムアウト

ls コマンドは各ホストに対し、到達性を並列に確認します。対象のホストが 10 秒間応答しなければ、ls コマンドで対象ホストの情報を Timeout 状態として表示します。同様の状況（接続が貧弱、高負荷、あるいはその他トラブルシューティング）では、値を上下させたくなるでしょう。 -t フラグを使い、整数値で秒数を指定できます。

例

```
$ docker-machine ls -t 12
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER   ERRORS
default   -        virtualbox   Running   tcp://192.168.99.100:2376           v1.9.0
```

フィルタリング

フィルタリング・フラグ（ --filter ）の指定は key=value のペア形式です。複数のフィルタを使う場合は、複数のフラグを使います（例： --filter "foo=bar" --filter "bif=baz" ）。

現時点でサポートしているフィルタは次の通りです。

- driver（ドライバ名）
- swarm（swarm のマスタ名）
- state （状態： Running | Paused | Saved | Stopped | Stopping | Starting | Error ）
- name（ドライバが返すマシン名であり、Go 言語形式 の正規表現をサポート）
- label（マシンを--engine-label オプションで作成すると、label=<key>[=<value>] 形式でフィルタできる）

例

```
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER   ERRORS
dev    -        virtualbox   Stopped
foo0   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.9.1
foo1   -        virtualbox   Running   tcp://192.168.99.106:2376           v1.9.1
foo2   *        virtualbox   Running   tcp://192.168.99.107:2376           v1.9.1

$ docker-machine ls --filter name=foo0
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER   ERRORS
foo0   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.9.1

$ docker-machine ls --filter driver=virtualbox --filter state=Stopped
```

```
NAME   ACTIVE   DRIVER       STATE     URL   SWARM   DOCKER   ERRORS
dev    -        virtualbox   Stopped                 v1.9.1

$ docker-machine ls --filter label=com.class.app=foo1 --filter label=com.class.app=foo2
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER   ERRORS
foo1   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.9.1
foo2   *        virtualbox   Running   tcp://192.168.99.107:2376           v1.9.1
```

書式

マシンの結果を分かりやすくするため、書式オプション（ `--format` ）で Go 言語のテンプレートが使えます。以下の Go テンプレートをプレースホルダに指定可能です。

プレースホルダ	説明
.Name	マシン名
.Active	マシンがアクティブか
.ActiveHost	マシンがアクティブな Swarm のホストではないか
.ActiveSwarm	マシンがアクティブな Swarm マスタか
.DriverName	ドライバ名
.State	マシンの状態（実行中、停止中など）
.URL	マシン URL
.Swarm	マシンの Swarm 名
.Error	マシンのエラー
.DockerVersion	Docker デーモンのバージョン
.ResponseTime	ホストの応答時間

`ls` コマンドで `--format` オプションを使うと、テンプレートから自分が必要なデータだけ出力できます。また table 命令を使うと、ヘッダ部分も調整可能です。

以下の例では `Name` と `Driver` のエントリをヘッダ情報無しに表示します。

```
$ docker-machine ls --format "{{.Name}}: {{.DriverName}}"
default: virtualbox
ec2: amazonec2
```

全てのマシン名とドライバを表形式（table format）で表示できます。

```
$ docker-machine ls --format "table {{.Name}} {{.DriverName}}"
NAME      DRIVER
default   virtualbox
ec2       amazonec2
```

regenerate-certs (リジェネレート・サーツ)

使い方: docker-machine regenerate-certs [オプション] [引数...]

マシン用の TLS 証明書を再作成

説明:
引数は 1 つまたは複数のマシン名

オプション:

--force, -f 強制的に再作成し、表示しない

TLS 証明書を再作成し、新しい証明書を使うようにマシンの情報を更新します。
例：

```
$ docker-machine regenerate-certs dev
Regenerate TLS machine certs?  Warning: this is irreversible. (y/n): y
Regenerating TLS certificates
```

リスタート
restart

使い方: docker-machine restart [引数...]

マシンを再起動

説明:
引数は1つまたは複数のマシン名

マシンを再起動します。これは docker-machine stop; docker-machine start の実行と同等です。

```
$ docker-machine restart dev
Waiting for VM to start...
```

rm

マシンを削除（remove）します。ローカル環境から削除するだけでなく、クラウド・プロバイダや仮想化管理プラットフォームからも削除します。

```
$ docker-machine rm --help

使い方: docker-machine rm [オプション] [引数...]

マシンを削除

説明:
   引数は１つまたは複数のマシン名

オプション:

   --force, -f   マシンを削除できなくても、ローカルの設定を削除する。また確認は自動的に `-y` を選択
   -y            削除時の確認で、入力プロンプトを表示せず、自動的に yes を選択
```

例：

```
$ docker-machine ls
NAME   ACTIVE   URL          STATE     URL                         SWARM   DOCKER   ERRORS
bar    -        virtualbox   Running   tcp://192.168.99.101:2376           v1.9.1
baz    -        virtualbox   Running   tcp://192.168.99.103:2376           v1.9.1
foo    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.9.1
qix    -        virtualbox   Running   tcp://192.168.99.102:2376           v1.9.1

$ docker-machine rm baz
About to remove baz
Are you sure? (y/n): y
Successfully removed baz

$ docker-machine ls
NAME   ACTIVE   URL          STATE     URL                         SWARM   DOCKER   ERRORS
bar    -        virtualbox   Running   tcp://192.168.99.101:2376           v1.9.1
foo    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.9.1
qix    -        virtualbox   Running   tcp://192.168.99.102:2376           v1.9.1

$ docker-machine rm bar qix
About to remove bar, qix
Are you sure? (y/n): y
Successfully removed bar
Successfully removed qix

$ docker-machine ls
NAME   ACTIVE   URL          STATE     URL                         SWARM   DOCKER   ERRORS
foo    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.9.1

$ docker-machine rm -y foo
About to remove foo
Successfully removed foo
```

scp

scp を使い、ローカル・ホストからマシンにファイルをコピーします。あるいは、マシンからマシンへ、マシンからローカルホストへコピーします。

引数の表記法は マシン名:/path/to/files（ファイルへのパス）です。対象がホストマシン上であれば、マシン名を指定せずに、パスのみを指定します。

次の例を考えてみます：

```
$ cat foo.txt
cat: foo.txt: No such file or directory
$ docker-machine ssh dev pwd
/home/docker
$ docker-machine ssh dev 'echo A file created remotely! >foo.txt'
$ docker-machine scp dev:/home/docker/foo.txt .
foo.txt                                                   100%   28     0.0KB/s   00:00
$ cat foo.txt
A file created remotely!
```

scp で -r フラグを使うと、再帰的にファイルをコピーします。この機能を docker-machine で使うには -r フラグを使います。

マシンからマシンへファイルを転送する場合は、ローカル・ホスト上のファイルシステムを経由する必要があります（ scp の -3 フラグ）を使います。

ssh

SSH でマシンにログインするか、コマンドを実行します。

ログインするには、 docker-machine ssh マシン名 を実行するだけです。

```
$ docker-machine ssh dev
                        ##         .
                  ## ## ##        ==
               ## ## ## ##       ===
           /""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 1.4.0, build master : 69cf398 - Fri Dec 12 01:39:42 UTC 2014
docker@boot2docker:~$ ls /
Users/   dev/     home/    lib/     mnt/     proc/    run/     sys/     usr/
bin/     etc/     init     linuxrc  opt/     root/    sbin/    tmp      var/
```

docker-machine ssh コマンドは通常の ssh プログラムが動作するように、リモートで直接実行するコマンドを指定することも可能です。

```
$ docker-machine ssh dev free
             total         used         free       shared      buffers
Mem:       1023556       183136       840420            0        30920
-/+ buffers:             152216       871340
Swap:      1212036            0      1212036
```

コマンドに対するフラグも、同様に動作します。

```
$ docker-machine ssh dev df -h
Filesystem                Size      Used Available Use% Mounted on
rootfs                  899.6M     85.9M    813.7M  10% /
tmpfs                   899.6M     85.9M    813.7M  10% /
tmpfs                   499.8M         0    499.8M   0% /dev/shm
/dev/sda1                18.2G     58.2M     17.2G   0% /mnt/sda1
cgroup                  499.8M         0    499.8M   0% /sys/fs/cgroup
/dev/sda1                18.2G     58.2M     17.2G   0%
/mnt/sda1/var/lib/docker/aufs
```

「外部の」SSH タイプを使う場合の詳細については、次のセクションで説明します。 ssh バイナリを実行するコマンドには（Docker Machine によって作成されるコマンドと衝突しない限り）追加オプションを設定できます。例えば、次のコマンドは default マシンから自分のホスト・コンピュータ上の localhost に対して、ポート 8080 に転送するコマンドです。

```
$ docker-machine ssh default -L 8080:localhost:8080
```

異なる種類の SSH

Docker Machine が呼び出される時、伝統的な ssh バイナリがローカルにあるかどうか確認し、SSH コマンドを使う必要があれば、これを使おうとします。使う場面とは、create 時のような内部処理か、利用者が必要な時に明示するかどうかに拘わりません。外部の ssh バイナリがローカルに存在しなければ、Go 言語内蔵の crypto/ssh[*1] を使おうとします。これは伝統的な UNIX ツールが利用できない環境で便利になるでしょう。例えば、Windows 上で Docker Machine を使う時に、msysgit がインストールされていなくても使えます。

多くの場合、実装の詳細について考慮する必要はありません。Docker Machine は難しい設定をしなくても利用できます。しかし、Go ネイティブ・バージョンを敢えて使おうとする場合は、グローバルなコマンドライン上のフラグ、または、環境変数の指定が必要です。

```
$ docker-machine --native-ssh ssh dev
```

２つの手法は様々なパターンを伴いますので、何らかの問題や一貫性が無い場合があれば、ご報告ください。

*1 https://godoc.org/golang.org/x/crypto/ssh

スタート
start

使い方: docker-machine start [引数...]

マシンを起動

説明:
引数は1つまたは複数のマシン名

マシンを起動します。

```
$ docker-machine start dev
Starting VM...
```

ステータス
status

使い方: docker-machine status [引数...]

マシンの状態を取得

説明:
引数はマシン名

マシンのステータス（状態）を取得します。

```
$ docker-machine status dev
Running
```

ストップ
stop

使い方: docker-machine stop [引数...]

マシンを丁寧に（Gracefully）に停止

説明:
引数は１つまたは複数のマシン名

マシンを丁寧に（gracefully）停止します。

```
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL
dev    *        virtualbox   Running   tcp://192.168.99.104:2376
$ docker-machine stop dev
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL
dev    *        virtualbox   Stopped
```

アップグレード upgrade

マシンを Docker の最新バージョンにアップグレードします。どのようにアップグレードをするかは、インスタンス作成に用いたディストリビューションに依存します。

たとえば、動作するオペレーティング・システムを Ubuntu としてマシンを起動している場合、Docker Machine は Ubuntu マシンのパッケージを管理しているとみなすため、`sudo apt-get upgrade docker-engine` のようなコマンドを実行します。

```
$ docker-machine upgrade default
Stopping machine to do the upgrade...
Upgrading machine default...
Downloading latest boot2docker release to /home/username/.docker/machine/cache/boot2docker.iso...
Starting machine back up...
Waiting for VM to start...
```

url

ホストの URL を取得します。

```
$ docker-machine url dev
tcp://192.168.99.109:2376
```

5 章

Machine ドライバ・リファレンス

5.1 サポートしているドライバ

- Amazon Web Services
- Microsoft Azure
- Digital Ocean
- exoscale
- Google Compute Engine
- 汎用(generic)ドライバ
- Microsoft Hyper-V
- OpenStack
- rackspace
- IBM SoftLayer
- Oracle VirtualBox
- vm-cloud
- vm-fusion
- vsphere

5.2 ドライバのオプションとデフォルト OS

ローカル・ネットワーク・プロバイダ、あるいはリモート環境、そして Amazon Web Services のようなクラウド・プロバイダ上において、Docker Machine がコンテナをプロビジョニングする場合は、プロバイダに対応したドライバと、ベースとなるオペレーティング・システムの両方を定義する必要があります。10 を越えるサポート済みドライバと、その他のプロバイダ上にマシンを追加する generic （ジェネリック）ドライバがあります。

各ドライバは、各プロバイダ固有のオプション群を持っています。これらのオプションはマシンに対する情報を提供するものです。たとえば、接続時に必要となる認証情報（credential)、ポート番号などがあります。

例えば、Azure マシンを作成するには、ポータル上でサブスクリプション ID を取得してから、`docker-machine create` を次のように実行します。

```
$ docker-machine create -d azure \
  --azure-subscription-id="SUB_ID" --azure-subscription-cert="mycert.pem" A-VERY-UNIQUE-NAME
```

プロバイダの一覧や、各プロバイダで利用できるオプション一覧を確認するには、Docker Machine ドライバ・リファレンスをご覧ください。

プロバイダに加え、ベース・オペレーティング・システムごとに固有のオプションを指定できます。しかし、Docker machine はローカル・リモートの各プロバイダに対するデフォルト指定を持っているため、オプション指定は任意です。VirtualBox、Fusion、Hyper-V 等のようなローカル・プロバイダでは、デフォルトのベース・オペレーティング・システムは Boot2Docker です。クラウド・プロバイダ向けのベース・オペレーティング・システムは、プロバイダが提供している最新の Ubuntu LTS です。

オペレーティング・システム	バージョン	メモ
Boot2Docker	1.5+	ローカル用のデフォルト
Ubuntu	12.04+	リモート用のデフォルト
RancherOS	0.3+	
Debian	8.0+	実験的(experimental)
Red Hat Enterprise Linux	7.0+	実験的(experimental)
CentOS	7.0+	実験的(experimental)
Fedora	21+	実験的(experimental)

リモート・プロバイダ上で異なったベース・オペレーティング・システムを使うには、プロバイダのイメージ・フラグと利用可能なイメージの指定が必要になります。例えば、DigitalOcean で `debian-8-x64` イメージを指定するには、 `--digitalocean-image=debian-8-x64` フラグが必要です。

プロバイダ用のベース・イメージを変更する時、SSH ユーザの変更も必要になる場合があります。例えば、EC2 上のデフォルト Red Hat AMI の SSH ユーザは `ec2-user` ですので、 `--amazonec2-ssh-user ec2-user` と指定する必要があります。

5.3 Amazon Web Services

Amazon Web Services 上にマシンを作成します。

認証ファイルの設定

AWS 認証情報ファイル

amazonec2 ドライバを使う前に、認証情報の設定が必要です。

認証情報を設定する 1 つの方法は、Amazon　AWS 用の認証情報をファイル `~/.aws/credentials` に置くことです。次のような書式です。

```
[default]
aws_access_key_id = AKID1234567890
aws_secret_access_key = MY-SECRET-KEY
```

Mac や Linux ディストリビューションでは、AWS コマンドライン・インターフェース（ `aws cli` ）をターミナルにインストールし、`aws configure` コマンドで認証情報ファイルを作成します。

これが最も簡単にマシンを作成できる方法です。

```
$ docker-machine create --driver amazonec2 aws01
```

コマンドラインのフラグ

他には、コマンドライン上で`--amazonec2-access-key` と`--amazonec2-secret-key` フラグを使う方法があります。

```
$ docker-machine create --driver amazonec2 --amazonec2-access-key AKI******* --amazonec2-secret-key
8T93C*******  aws01
```

環境変数

環境変数も使えます。

```
$ export AWS_ACCESS_KEY_ID=AKID1234567890
$ export AWS_SECRET_ACCESS_KEY=MY-SECRET-KEY
$ docker-machine create --driver amazonec2 aws01
```

オプション

- `--amazonec2-access-key` : **必須** 自分の Amazon Web Services API 用のアクセス・キー緒です。
- `--amazonec2-secret-key` : **必須** 自分の Amazon Web Services API 用のシークレット・アクセスキーです。
- `--amazonec2-session-token` : 自分の Amazon Web Services API 用のセッション・トークンです。
- `--amazonec2-ami` : インスタンスに使用する AMI ID です。
- `--amazonec2-region` : インスタンスを起動するリージョンです。
- `--amazonec2-vpc-id` : **必須** 起動したインスタンスを置く VPC ID です。
- `--amazonec2-zone` : インスタンスを置く AWS ゾーンです（例： a, b, c, d, e のいずれか)。
- `--amazonec2-subnet-id` : AWS VPC サブネット ID です。
- `--amazonec2-security-group` : AWS VPC セキュリティ・グループ名です。
- `--amazonec2-tags` ： AWS タグをキーバリューのペアで指定します（カンマ区切りです。例：

key1,value1,key2,value2)。

- --amazonec2-instance-type : 実行するインスタンス・タイプです。
- --amazonec2-device-name : 実行するデバイス名です。
- --amazonec2-root-size : インスタンスのルート・ディスク容量（単位：GB)。
- --amazonec2-volume-type : インスタンスにアタッチする Amazon EBS の種類を指定。
- --amazonec2-iam-instance-profile : インスタンスのプロフィールに使われる AWS IAM ロール名。
- --amazonec2-ssh-user : SSH ログイン・ユーザ名です。ここには AMI が使うデフォルトの SSH ユーザを一致する必要があります。
- --amazonec2-request-spot-instance : スポット・インスタンスを使用。
- --amazonec2-spot-price : スポット・インスタンスの bid 価格（単位：ドル）。--amazonec2-request-spot-instance フラグが必要です。
- --amazonec2-use-private-addressce : docker-machine の通信にプライベート IP アドレスを使います。ですがパブリックな IP アドレスも作成されます。
- --amazonec2-private-address-only : プライベート・アドレスのみ使います。
- --amazonec2-monitoring : CloudWatch モニタリングを有効化します。
- --amazonec2-user-ebs-optimized-instance : EBS 最適化インスタンスを作成します。インスタンス・タイプが対応している必要があります。
- --amazonec2-ssh-keypath : インスタンス用のプライベート・キーに使うファイルのパスを指定します。対応する公開鍵の拡張子は .pub になっている必要があります。

環境変数とデフォルト値は以下の通りです。

コマンドライン・オプション	環境変数	デフォルト値
--amazonec2-access-key	AWS_ACCESS_KEY_ID	
--amazonec2-secret-key	AWS_SECRET_ACCESS_KEY	
--amazonec2-session-token	AWS_SESSION_TOKEN	
--amazonec2-ami	AWS_AMI	ami-5f709f34
--amazonec2-region	AWS_DEFAULT_REGION	us-east-1
--amazonec2-vpc-id	AWS_VPC_ID	
--amazonec2-vpc-id	AWS_VPC_ID	
--amazonec2-zone	AWS_ZONE	a
--amazonec2-subnet-id	AWS_SUBNET_ID	
--amazonec2-security-group	AWS_SECURITY_GROUP	docker-machine
--amazonec2-instance-type	AWS_INSTANCE_TYPE	t2.micro
--amazonec2-device-name	AWS_DEVICE_NAME	/dev/sda
--amazonec2-root-size	AWS_ROOT_SIZE	16
--amazonec2-volume-type	AWS_VOLUME_TYPE	gp2
--amazonec2-iam-instance-profile	AWS_INSTANCE_PROFILE	
--amazonec2-ssh-user	AWS_SSH_USER	ubuntu
--amazonec2-request-spot-instance		false
--amazonec2-spot-price		0.50
--amazonec2-user-private-address		false
--amazonec2-private-address-only		false
--amazonec2-monitoring		false
--amazonec2-use-ebs-optimized-instance		false
--amazonec2-ssh-keypath	AWS_SSH_KEYPATH	

デフォルト AMI

デフォルトでは、Amazon EC2 ドライバは Ubuntu 15.10 LTS の daily イメージを使います。

リージョン	AMI ID
ap-northeast-1	ami-b36d4edd
ap-southeast-1	ami-1069af73
ap-southeast-2	ami-1d336a7e
cn-north-1	ami-79eb2214
eu-west-1	ami-8aa67cf9
eu-central-1	ami-ab0210c7
sa-east-1	ami-185de774
us-west-1	ami-26d5af4c
us-west-1	ami-9cbcd2fc
us-west-2	ami-16b1a077
us-gov-west-1	ami-b0bad893

セキュリティ・グループ

セキュリティ・グループが作成され、ホストに関連付けられるのでご注意ください。セキュリティ・グループは以下のインバウンド通信を許可します。

- ssh (22/tcp)
- docker (2376/tcp)
- swarm (3376/tcp) ノードが Swarm マスタの場合のみです

これポート以外にポートを開くには、 `--amazonec2-security-group` フラグを使って自分でセキュリティ・グループを指定し、ポートが開かれたか確認します。特定のアプリケーションが必要とするポートを開きたい場合は、AWS コンソールで設定を調整ください。

VPC ID

コマンドを実行する前に、自分のデフォルト VPC を確認します。時々、デフォルトの VPC がなかったり、あるいはデフォルトの VPC を使いたくない場合があるでしょう。VPC を指定するには `--amazonec2-vpc-id` フラグを使います。

VPC ID を確認するには：

1. AWS コンソールにログインします。
2. `Services` -> `VPC` -> `VPC` -> 自分の `VPC` に移動します。
3. VPC 列から使用する VPC ID を選びます。
4. `Services` -> `VPC` -> `Subnets` に移動します。Availability Zones 列を確認し、ゾーン `a` が存在しているのと、自分の VPC ID と一致していることを確認します。

例えば、`us-east-1a` にはアベイラビリティ・ゾーン `a` が存在しています。もし `a` ゾーンが表示されなければ、マシンを作成するために、新しいサブネットを作成するか別のゾーンを指定します。

マシン・インスタンスを作成するには、 `--driver amazonec2` と 3 つの必須パラメータを指定します。

```
$ docker-machine create --driver amazonec2 \
          --amazonec2-access-key AKI******* \
          --amazonec2-secret-key 8T93C********* \
          --amazonec2-vpc-id vpc-****** \
          aws01
```

この例では、VPC ID が a アベイラビリティ・ゾーンに存在しているものと想定されます。 a ゾーン以外を指定するには、 --amazonec2-zone フラグを使います。例えば、 --amazonec2-zone c は us-east1-c を表しています。

VPC の接続性

Machine は SSH を使い EC2 インスタンス上にセットアップします。その時、インスタンスに直接接続できるようにする必要があります。

フラグ --amazonec2-private-address-only を使うときは、VPC の内部ネットワーク内で新しいインスタンスを作成できるようにする必要があります（例：社内の VPN から VPC の接続、VPC 内の VPN インスタンス、VPC 内で Docker Machine インスタンスを使う）。

VPC セットアップ

VPC の設定はこのドキュメントの範囲外ですが、トラブルシューティングの始めのステップとして、AWS VPC ユーザガイド[*1] のガイダンスから、NAT の利用に関する情報をご覧ください。インターネットに接続するためのセットアップに関する、全ての手順が書かれています。

カスタム AMI と SSH ユーザ名

デフォルト AMI 用のデフォルト SSH ユーザ名は ubuntu です。

カスタム AMI が異なった SSH ユーザ名を使っている場合、この SSH ユーザ名の設定を変更する必要があります。

--amazonec2-ami で指定した AMI が必要とする SSH ユーザ名を --amazonec2-ssh-user で指定します。

*1 http://docs.aws.amazon.com/ja_jp/AmazonVPC/latest/UserGuide/VPC_Scenario2.html

5.4 Digital Ocean

Digital Ocean 上に Docker マシンを作成します。

Digital Ocean のコントロール・パネルにある「Apps & API」から、パーソナル・アクセス・トークンを作成する必要があります。それから `docker-machine create` に `--digitalocean-access-token` オプションで渡します。

```
$ docker-machine create --driver digitalocean \
  --digitalocean-access-token=aa9399a2175a93b17b1c86c807e08d3fc4b79876545432a629602f61cf6ccd6b \
  test-this
```

オプション：

`--digitalocean-access-token` : **必須** Digital Ocean API 用のパーソナル・アクセス・トークン。

`--digitalocean-image` : Digital Ocean イメージ用の名前。

`--digitalocean-region` : ドロップレットを作成するリージョンの指定です。詳細一覧は Region API[*1] を参照。

`--digitalocean-size` : Digital Ocean ドロップレットのサイズ（ デフォルトの `2gb` 以上より大きいもの ）。

--digitalocean-ipv6 : ドロップレットで IPv6 を有効化。

`--digitalocean-private-networking` : ドロップレットのプライベート・ネットワーク対応を有効化。

`--digitalocean-backups` : ドロップレットのバックアップを有効化。

`--digitalocean-userdata` : ドロップレット用のユーザ・データを含むファイルのパス。

`--digitalocean-ssh-user` : SSH ユーザ名。

`--digitalocean-ssh-port` : SSH ポート番号。

`--digitalocean-ssh-key-fingerprint` : 新しい SSH 鍵を作らず、既存の鍵を使います。詳細は SSH Keys のページ[*2] をご覧ください。

DigitalOcean ドライバは、`ubuntu-15-10-x64` をデフォルトのイメージとして使います。

利用可能な環境変数とデフォルト値は以下の通りです。

コマンドライン・オプション	環境変数	デフォルト値
--digitalocean-access-token	DIGITALOCEAN_ACCESS_TOKEN	
--digitalocean-image	DIGITALOCEAN_IMAGE	ubuntu-15-10-x64
--digitalocean-region	DIGITALOCEAN_REGION	nyc3
--digitalocean-size	DIGITALOCEAN_SIZE	512mb
--digitalocean-ipv6	DIGITALOCEAN_IPV6	false
--digitalocean-private-networking	DIGITALOCEAN_PRIVATE_NETWORKING	false
--digitalocean-backups	DIGITALOCEAN_BACKUPS	false
--digitalocean-userdata	DIGITALOCEAN_USERDATA	
--digitalocean-ssh-user	DIGITALOCEAN_SSH_USER	root
--digitalocean-ssh-port	DIGITALOCEAN_SSH_PORT	22
--digitalocean-ssh-key-fingerprint	DIGITALOCEAN_SSH_KEY_FINGERPRINT	

*1 https://developers.digitalocean.com/documentation/v2/#regions

*2 https://developers.digitalocean.com/documentation/v2/#ssh-keys

5.5 generic（汎用）ドライバ

既存の仮想マシン/ホストを、SSH 経由で Docker Machine が扱えるマシンにします。

これが役立つのは、Docker　Machine のプロバイダがサポートされていない場合や、既存の Docker ホスト環境を Docker Machine で管理できるよう移行する場合です。

作成時、ドライバは以下の処理を行います。

- ホスト上で docker が動いていなければ、自動的にインストールを行う。
- ホスト上のパッケージを更新します（ `apt-get update` `yum update ...` ）
- docker デーモンを安全にする証明書を生成する。
- docker デーモンを再起動するため、実行中のコンテナは全て停止される。
- ホスト名がマシン名に対応します。

例

マシン・インスタンスを作成するには、`--driver generic` を指定します。また、ホストの IP アドレスまたは DNS 名と、ホストに接続できるようにするための SSH 秘密鍵も指定します。

```
$ docker-machine create \
  --driver generic \
  --generic-ip-address=203.0.113.81 \
  --generic-ssh-key=~/.ssh/id_rsa \
  vm
```

パスワードで保護された SSH 鍵

SSH 認証情報の指定（ `--generic-ssh-key` フラグを使う ）がなければ、SSH エージェントは（実行中であれば）訊ねます。つまり、パスワードで保護された SSH 鍵を簡単に使えるようにします。

ただしサポートされているのは、外部の SSH クライアント、ここでは `ssh` バイナリが実行できるデフォルトの挙動が扱える場合のみです。ネイティブ・クライアントを使う場合（ `--native-ssh` ）は、まだ SSH エージェントの利用をサポートしていませんのでご注意ください。

```
$ docker-machine create \
  --driver generic \
  --generic-ip-address=203.0.113.81 \
  other
```

sudo 権限

ホストに SSH 接続できるユーザを指定するには `--generic-ssh-user` フラグを使います。この時、指定したユーザはパスワード入力なしに `sudo` を実行する権限が必要です。もしそのようになっていなければ、 `sudoers` ファイルを編集し、そのユーザに `NOPASSWD` としての調整が必要になります。詳しくはドキュメント[*1] をご覧ください。

*1 https://help.ubuntu.com/community/Sudoers

オプション

- `--generic-engine-port` ： Docker デーモンが使うポート番号（メモ：このフラグは boot2docker では機能しません）
- `--generic-ip-address` : **必須** ホストの IP アドレス
- `--generic-ssh-key` : SSH ユーザのプライベート鍵のパス
- `--generic-ssh-user` : 接続に使う SSH ユーザ名
- `--generic-ssh-port` : SSH に使うポート番号

Docker Machine がサポートしているベース・オペレーティング・システムを使う必要があります。

利用可能な環境変数とデフォルト値は以下の通りです。

コマンドライン・オプション	環境変数	デフォルト値
--generic-engine-port	GENERIC_ENGINE_PORT	2376
--generic-ip-address	GENERIC_IP_ADDRESS	—
--generic-ssh-key	GENERIC_SSH_KEY	（ssh-agent に従う）
--generic-ssh-user	GENERIC_SSH_USER	root
--generic-ssh-port	GENERIC_SSH_PORT	22

5.6 IBM SoftLayer

SoftLayer 上にマシンを作成します。

SoftLayer コントロール・パネルで API を生成する必要があります。API Key を取得するにはドキュメント[*1] をご覧ください。 。

```
$ docker-machine create --driver softlayer --softlayer-user=user --softlayer-api-key=KEY
--softlayer-domain=domain vm
```

オプション：

- `--softlayer-memory` : ホストのメモリを MB 単位で指定。
- `--softlayer-disk-size` : `0` の値を指定すると、SoftLayer のデフォルトを使用。
- `--softlayer-user` : **必須** SoftLayer アカウントのユーザ名であり、API キーと一致する必要がある。
- `--softlayer-api-key` : **必須** ユーザ・アカウント用の API キー。
- `--softlayer-region` : SoftLayer のリージョン。
- `--softlayer-cpu` : マシンで使う CPU 数。
- `--softlayer-hostname` : マシンのホスト名。
- `--softlayer-domain` : **必須** マシンのドメイン名。
- `--softlayer-api-endpoint` : SoftLayer API エンドポイントの変更。
- `--softlayer-hourly-billing` : 時間単位（hourly）課金を指定すべき。そうしなければ、月単位で課金される。
- `--softlayer-local-disk` : SoftLayer SAN のかわりに、ローカル・マシンを使う。
- `--softlayer-private-net-only` : パブリック・ネットワークを無効化する。
- `--softlayer-image` : 使用する OS イメージ。
- `--softlayer-public-vlan-id` : パブリック VLAN ID 。
- `--softlayer-private-vlan-id` : プライベート VLAN ID 。

SoftLayer ドライバは、デフォルトで `UBUNTU_LATEST` イメージ・タイプを使います。

利用可能な環境変数とデフォルト値は以下の通りです。

コマンドライン・オプション	環境変数	デフォルト値
`--softlayer-memory`	`SOFTLAYER_MEMORY`	`1024`
`--softlayer-disk-size`	`SOFTLAYER_DISK_SIZE`	`0`
`--softlayer-user`	`SOFTLAYER_USER`	
`--softlayer-api-key`	`SOFTLAYER_API_KEY`	
`--softlayer-region`	`SOFTLAYER_REGION`	`dal01`
`--softlayer-cpu`	`SOFTLAYER_CPU`	`1`
`--softlayer-hostname`	`SOFTLAYER_HOSTNAME`	`docker`
`--softlayer-domain`	`SOFTLAYER_DOMAIN`	
`--softlayer-api-endpoint`	`SOFTLAYER_API_ENDPOINT`	`api.softlayer.com/rest/v3`
`--softlayer-hourly-billing`	`SOFTLAYER_HOURLY_BILLING`	`false`
`--softlayer-local-disk`	`SOFTLAYER_LOCAL_DISK`	`false`
`--softlayer-private-net-only`	`SOFTLAYER_PRIVATE_NET`	`false`

*1 http://knowledgelayer.softlayer.com/procedure/retrieve-your-api-key

--softlayer-image	SOFTLAYER_IMAGE	UBUNTU_LATEST
--softlayer-public-vlan-id	SOFTLAYER_PUBLIC_VLAN_ID	0
--softlayer-private-vlan-id	SOFTLAYER_PRIVATE_VLAN_IDT	0

5.7 Microsoft Azure

Microsoft Azure 上にマシンを作成します。

証明書（cert）を使ってサブスクリプションを作成する必用があります。これらのコマンドを実行し、問い合わせに回答します。

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
$ openssl pkcs12 -export -out mycert.pfx -in mycert.pem -name "My Certificate"
$ openssl x509 -inform pem -in mycert.pem -outform der -out mycert.cer
```

Azure ポータルに移動し、「Settings」ページに移動します（左スライド・バーの下の方にリンクがあります。スクロールが必要かもしれません)。それから「Management Certificates」で `mycert.cer` からアップロードします。

ポータルから自分のサブスクリプション ID を取得し、次のように `docker-machine create` の詳細を実行します。

```
$ docker-machine create -d azure \
  --azure-subscription-id="SUB_ID" --azure-subscription-cert="mycert.pem" A-VERY-UNIQUE-NAME
```

Azure ドライバは、デフォルトで `b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-15_10-amd64-server-20151116.1-en-us-30GB` イメージを使います。なお、このイメージは中国リージョンでは利用できません。中国リージョンは `b549f4301d0b4295b8e76ceb65df47d4__Ubuntu-15_10-amd64-server-20151116.1-en-us-30GB` をご利用ください。

OS を確実に更新するためには、仮想マシンに SSH でログインし、再起動する必要があるでしょう。

オプション：

- `--azure-docker-port` : Docker デーモンのポート番号。
- `--azure-image` : Azure イメージ名です。詳細は How to: Get the Windows Azure Image Name[*1] を参照。
- `--azure-location` : マシン・インスタンスの場所。
- `--azure-password` : Azure パスワード。
- `--azure-publish-settings-file` : Azure 設定ファイル。詳細は How to: Download and Import Publish Settings and Subscription Information [*2] を参照。
- `--azure-size` : Azure ディスク容量。
- `--azure-ssh-port` : Azure SSH ポート番号。
- `--azure-subscription-id` : **必須** Azure サブスクリプション ID（GUID は **d255d8d7-5af0-4f5c-8a3e-1545044b861e** のようなものです）.
- `--azure-subscription-cert` : **必須** Azure サブスクリプション証明書（cert)。
- `--azure-username` : Azure ログイン・ユーザ名。

利用可能な環境変数とデフォルト値は以下の通りです。

*1 https://msdn.microsoft.com/en-us/library/dn135249%28v=nav.70%29.aspx

*2 https://msdn.microsoft.com/en-us/library/dn385850%28v=nav.70%29.aspx

コマンドライン・オプション	環境変数	デフォルト値
--azure-docker-port		2376
--azure-image	AZURE_IMAGE	Ubuntu 15.10 x64
--azure-location	AZURE_LOCATION	West US
--azure-password		
--azure-publish-settings-file	AZURE_PUBLISH_SETTINGS_FILE	
--azure-size	AZURE_SIZE	Small
--azure-ssh-port		22
--azure-subscription-cert	AZURE_SUBSCRIPTION_CERT	
--azure-subscription-id	AZURE_SUBSCRIPTION_ID	
--azure-username		ubuntu

5.8 Oracle VirtualBox

VirtualBox を使い、ローカルにマシンを作成します。このドライバを使うには、ホスト上に VirtualBox 5 以上のインストールが必要です。VirtualBox 4 の場合は動作するかもしれませんが、警告が出ます。それよりも古いバージョンは実行できません。

```
$ docker-machine create --driver=virtualbox vbox-test
```

完全に新しいマシンを作成するか、Boot2Docker 仮想マシンにあるデータを変換して仮想マシンに取り込めます。Boot2Docker 仮想マシンを変換するには、以下のコマンドを実行します。

```
$ docker-machine create -d virtualbox --virtualbox-import-boot2docker-vm boot2docker-vm b2d
```

オプション：

- `--virtualbox-memory` : ホストのメモリ容量を MB 単位で指定。
- `--virtualbox-cpu-count` : 作成する仮想マシンが使う CPU 数。デフォルトは CPU 1 つ。
- `--virtualbox-disk-size` : ホストのディスク容量を MB 単位で指定。
- `--virtualbox-host-dns-resolver` : ホス DN レゾルバの使用（Boolean 値で、デフォルトは false)。
- `--virtualbox-boot2docker-url` : boot2docker イメージの URL を指定。デフォルトは利用可能な最新バージョン。
- `--virtualbox-import-boot2docker-vm` : 取り込む Boot2Docker 仮想マシンの名前。
- `--virtualbox-hostonly-cidr` : ホストオンリー・アダプタの CIDR 。
- `--virtualbox-hostonly-nictype` : ホストオンリー・ネットワーク・アダプタのタイプを指定。値は `82540EM` (Intel PRO/1000)、 `Am79C973` (PCnet-FAST III) 、`virtio-net` 準仮想化ネットワーク・アダプタのいずれか。
- `--virtualbox-hostonly-nicpromisc` : ホスト・オンリー・ネットワーク・アダプタのプロミスキャス・モードを指定。オプションは deny、allow-vms、allow-all のいずれか。
- `--virtualbox-no-share` : ホーム・ディレクトリのマウントを無効化。
- `--virtualbox-no-dns-proxy` : 全ての DNS リクエストをホスト側にプロキシしない（ Boolean 値で、デフォルトは false)。
- `--virtualbox-no-vtx-check` : 仮想マシンを起動する前にハードウェア仮想化が利用可能かどうかを確認。

`--virtualbox-boot2docker-url` フラグには、いくつかの異なった使い方があります。デフォルトでは、フラグに値を何も指定しなければ、Docker Machine はローカルの boot2docker ISO を探します。もしローカル上に見つかれば、マシン作成用の ISO として用いられます。もし見つからない場合は、boot2docker/boot2docker[*1] にある最新の ISO イメージをダウンロードし、ローカルに保存してから使います。つまり、ローカルに「キャッシュされた」boot2docker ISO を更新したい場合は、`docker-machine upgrade` を実行しなくてはいけません。

これはデフォルトの挙動（ `--virtualbox-boot2docker-url=""` を指定 ）ですが、オプションで ISO を `http://` や `file://` プロトコルで指定することもサポートされています。 `file://` はローカルに置かれている ISO イメージのパスを探します。例えば、 `--virtualbox-boot2docker-url file://$HOME/Downloads/rc.iso` を指定すると、リリース候補のダウンロード済み ISO を確認します。あるいは、 `http://` 形式を使い、インターネットの ISO を直接指定できます。

*1 https://github.com/boot2docker/boot2docker

ホスト・オンリー・アダプタをカスタマイズするには、 --virtualbox-hostonly-cidr フラグを使えます。ここでホスト IP を指定すると、Machine は VirtualBox DHCP サーバ・アドレスを計算（ .1 〜 .25 までのサブネット上の、ランダムな IP）するので、指定したホスト IP と衝突しないようにします。また、Machine は自動的に最小 .100 〜最大 .254 までの間で DHCP を指定します。たとえば、CIDR 192.168.24.1/24 を指定すると、DHCP サーバは 192.168.24.2-25 になり、IP アドレスの範囲は最小 192.168.24.100 から最大 192.168.24.254 となります。

利用可能な環境変数とデフォルト値は以下の通りです。

コマンドライン・オプション	環境変数	デフォルト値
--virtualbox-memory	VIRTUALBOX_MEMORY_SIZE	1024
--virtualbox-cpu-count	VIRTUALBOX_CPU_COUNT	1
--virtualbox-disk-size	VIRTUALBOX_DISK_SIZE	20000
--virtualbox-host-dns-resolver	VIRTUALBOX_HOST_DNS_RESOLVER	false
--virtualbox-boot2docker-url	VIRTUALBOX_BOOT2DOCKER_URL	最新の boot2docker url
--virtualbox-import-boot2docker-vm	VIRTUALBOX_BOOT2DOCKER_IMPORT_VM	boot2docker-vm
--virtualbox-hostonly-cidr	VIRTUALBOX_HOSTONLY_CIDR	192.168.99.1/24
--virtualbox-hostonly-nictype	VIRTUALBOX_HOSTONLY_NIC_TYPE	82540EM
--virtualbox-hostonly-nicpromisc	VIRTUALBOX_HOSTONLY_NIC_PROMISC	deny
--virtualbox-no-share	VIRTUALBOX_NO_SHARE	false
--virtualbox-no-dns-prox	VIRTUALBOX_NO_DNS_PROXY	false
--virtualbox-no-vtx-check	VIRTUALBOX_NO_VTX_CHECK	false

]既知の問題

Vboxfs は longstanding bug[*1] により、キャッシュされたファイル内容を提供するため sendfile(2)[*2] を引き起こします。

これにより、nginx のようなウェブ・サーバが共有ボリュームから静的ファイルを読み込むとき、問題を引き起こしがちです。開発環境では、サーバの設定で sendfile を無効化するのが良いでしょう。

*1 https://www.virtualbox.org/ticket/9069

*2 http://linux.die.net/man/2/sendfile