How To Provision and Manage Remote Docker Hosts with Docker Machine on Ubuntu 18.04

Posted October 2, 2018 6.3k DOCKER UBUNTU 18.04

By: finid By: Brian Hogan

Not using **Ubuntu 18.04**? Choose a different version:

CentOS 7 >

Ubuntu 16.04 >

Automated: Docker request

Introduction

Docker Machine is a tool that makes it easy to provision and manage multiple Docker hosts remotely from your personal computer. Such servers are commonly referred to as Dockerized hosts and are used to run Docker containers.

While Docker Machine can be installed on a local or a remote system, the most common approach is to install it on your local computer (native installation or virtual machine) and use it to provision Dockerized remote servers.

Though Docker Machine can be installed on most Linux distributions as well as macOS and Windows, in this tutorial, you'll install it on your local machine running Ubuntu 18.04 and use it to provision Dockerized DigitalOcean Droplets. If you don't have a local Ubuntu 18.04 machine, you can follow these instructions on any Ubuntu 18.04 server.

Prerequisites

To follow this tutorial, you will need the following:

- A local machine or server running Ubuntu 18.04 with Docker installed. See How To Install and Use Docker on Ubuntu 18.04 for instructions.
- A DigitalOcean API token. If you don't have one, generate it using this guide. When you generate a token, be sure that it has read-write scope. That is the default, so if you do not change any options while generating it, it will have read-write capabilities.

Step 1 — Installing Docker Machine

In order to use Docker Machine, you must first install it locally. On Ubuntu, this means downloading a handful of scripts from the official Docker repository on GitHub.

To download and install the Docker Machine binary, type:

```
$ wget https://github.com/docker/machine/releases/download/v0.15.0/docker-machine-$(uname -s)-$(unam
```

The name of the file should be `docker-machine-Linux-x86_64`. Rename it to `docker-machine` to make it easier to work with:

```
$ mv docker-machine-Linux-x86_64 docker-machine
```

Make it executable:

```
$ chmod +x docker-machine
```

Move or copy it to the `/usr/local/bin` directory so that it will be available as a system command:

```
$ sudo mv docker-machine /usr/local/bin
```

Check the version, which will indicate that it's properly installed:

```
$ docker-machine version
```

You'll see output similar to this, displaying the version number and build:

```
Output
docker-machine version 0.15.0, build b48dc28d
```

Docker Machine is installed. Let's install some additional helper tools to make Docker Machine easier to work with.

Step 2 — Installing Additional Docker Machine Scripts

There are three Bash scripts in the Docker Machine GitHub repository you can install to make working with the `docker` and `docker-machine` commands easier. When installed, these scripts provide command completion and prompt customization.

In this step, you'll install these three scripts into the `/etc/bash_completion.d` directory on your local machine by downloading them directly from the Docker Machine GitHub repository.

> **Note**: Before downloading and installing a script from the internet in a system-wide location, you should inspect the script's contents first by viewing the source URL in your browser.

The first script allows you to see the active machine in your prompt. This comes in handy when you are working with and switching between multiple Dockerized machines. The script is called `docker-machine-prompt.bash`. Download it

```
$ sudo wget https://raw.githubusercontent.com/docker/machine/master/contrib/completion/bash/docker-ma
```

To complete the installation of this file, you'll have to modify the value for the `PS1` variable in your `.bashrc` file. The `PS1` variable is a special shell variable used to modify the Bash command prompt. Open `~/.bashrc` in your editor:

```
$ nano ~/.bashrc
```

Within that file, there are three lines that begin with `PS1`. They should look just like these:

~/.bashrc

```
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]
...
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
...
PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
```

For each line, insert `$(__docker_machine_ps1 " [%s]")` near the end, as shown in the following example:

~/.bashrc

```
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[0
```



```
m\]
```

```
...

PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w$(__docker_machine_ps1 " [%s]")\$ '

...

PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$(__docker_machine_ps1 " [%s]")$PS1"
```

Save and close the file.

The second script is called `docker-machine-wrapper.bash`. It adds a `use` subcommand to the `docker-machine` command, making it significantly easier to switch between Docker hosts. To download it, type:

```
$ sudo wget https://raw.githubusercontent.com/docker/machine/master/contrib/completion/bash/docker-ma
```

The third script is called `docker-machine.bash`. It adds bash completion for `docker-machine` commands. Download it using:

```
$ sudo wget https://raw.githubusercontent.com/docker/machine/master/contrib/completion/bash/docker-ma
```

To apply the changes you've made so far, close, then reopen your terminal. If you're logged into the machine via SSH, exit the session and log in again, and you'll have command completion for the `docker` and `docker-machine` commands.

Let's test things out by creating a new Docker host with Docker Machine.

Step 3 — Provisioning a Dockerized Host Using Docker Machine

Now that you have Docker and Docker Machine running on your local machine, you can provision a Dockerized Droplet on your DigitalOcean account using Docker Machine's `docker-machine create` command. If you've not done so already, assign your DigitalOcean API token to an environment variable:

```
$ export DOTOKEN=your-api-token
```

> **NOTE:** This tutorial uses DOTOKEN as the bash variable for the DO API token. The variable name does not have to be DOTOKEN, and it does not have to be in all caps.

To make the variable permanent, put it in your `~/.bashrc` file. This step is optional, but it want to the value to persist across shell sessions.

Open that file with `nano`:

```
$ nano ~/.bashrc
```

Add this line to the file:

~/.bashrc

```
export DOTOKEN=your-api-token
```

To activate the variable in the current terminal session, type:

```
$ source ~/.bashrc
```

To call the `docker-machine create` command successfully you must specify the *driver* you wish to use, as well as a machine name. The driver is the adapter for the infrastructure you're going to create. There are drivers for cloud infrastructure providers, as well as drivers for various virtualization platforms.

We'll use the `digitalocean` driver. Depending on the driver you select, you'll need to provide additional options to create a machine. The `digitalocean` driver requires the API token (or the variable that evaluates to it) as its argument, along with the name for the machine you want to create.

To create your first machine, type this command to create a DigitalOcean Droplet called `docker-01`:

```
$ docker-machine create --driver digitalocean --digitalocean-access-token $DOTOKEN docker-01
```

You'll see this output as Docker Machine creates the Droplet:

```
Output
 ...
Installing Docker...
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 your Docker Client to the Docker Engine running on this virtual machine, run: d
```

Docker Machine creates an SSH key pair for the new host so it can access the server remotely. The Droplet is provisioned with an operating system and Docker is installed. When the command is complete, your Docker Droplet is up and running.

To see the newly-created machine from the command line, type:

```
$ docker-machine ls
```

The output will be similar to this, indicating that the new Docker host is running:

```
Output
NAME        ACTIVE   DRIVER         STATE     URL                         SWARM   DOCKER        ERROR
docker-01   -        digitalocean   Running   tcp://209.97.155.178:2376           v18.06.1-ce
```

Now let's look at how to specify the operating system when we create a machine.

Step 4 — Specifying the Base OS and Droplet Options When Creating a Dockerized Host

By default, the base operating system used when creating a Dockerized host with Docker Machine is *supposed* to be the latest Ubuntu LTS. However, at the time of this publication, the `docker-machine create` command is still using Ubuntu 16.04 LTS as the base operating system, even though Ubuntu 18.04 is the latest LTS edition. So if you need to run Ubuntu 18.04 on a recently-provisioned machine, you'll have to specify Ubuntu along with the desired version by passing the `--digitalocean-image` flag to the `docker-machine create` command.

For example, to create a machine using Ubuntu 18.04, type:

```
$ docker-machine create --driver digitalocean --digitalocean-image ubuntu-18-04-x64 --digitalocean-a
```

You're not limited to a version of Ubuntu. You can create a machine using any operating system supported on DigitalOcean. For example, to create a machine using Debian 8, type:

```
$ docker-machine create --driver digitalocean --digitalocean-image debian-8-x64 --digitalocean-acces
```

To provision a Dockerized host using CentOS 7 as the base OS, specify `centos-7-0-x86` as the image name, like so:

```
$ docker-machine create --driver digitalocean --digitalocean-image centos-7-0-x64 --digitalocean-acc
```

The base operating system is not the only choice you have. You can also specify the size of the Droplet. By default, it is the smallest Droplet, which has 1 GB of RAM, a single CPU, and a 25 GB SSD.

Find the size of the Droplet you want to use by looking up the corresponding slug in the documentation.

For example, to provision a machine with 2 GB of RAM, two CPUs, and a 60 GB SSD, use the slug `s-2vcpu-2gb`:

```
$ docker-machine create --driver digitalocean --digitalocean-size s-2vcpu-2gb --digitalocean-access-
```

To see all the flags specific to creating a Docker Machine using the DigitalOcean driver, type:

```
$ docker-machine create --driver digitalocean -h
```

Tip: If you refresh the Droplet page of your DigitalOcean dashboard, you will see the new machines you created using the `docker-machine` command.

Now let's explore some of the other Docker Machine commands.

Step 5 — Executing Additional Docker Machine Commands

You've seen how to provision a Dockerized host using the `create` subcommand, and how to list the hosts available to Docker Machine using the `ls` subcommand. In this step, you'll learn a few more useful subcommands.

To obtain detailed information about a Dockerized host, use the `inspect` subcommand, like so:

```
$ docker-machine inspect docker-01
```

The output includes lines like the ones in the following output. The `Image` line reveals the version of the Linux distribution used and the `Size` line indicates the size slug:

```
Output
...
{
    "ConfigVersion": 3,
    "Driver": {
        "IPAddress": "203.0.113.71",
        "MachineName": "docker-01",
        "SSHUser": "root",
        "SSHPort": 22,
        ...
        "Image": "ubuntu-16-04-x64",
        "Size": "s-1vcpu-1gb",
        ...
    },

---
```

To print the connection configuration for a host, type:

```
$ docker-machine config docker-01
```

The output will be similar to this:

```
Output
--tlsverify
--tlscacert="/home/kamit/.docker/machine/certs/ca.pem"
--tlscert="/home/kamit/.docker/machine/certs/cert.pem"
--tlskey="/home/kamit/.docker/machine/certs/key.pem"
-H=tcp://203.0.113.71:2376
```

The last line in the output of the `docker-machine config` command reveals the IP address of the host, but you can also get that piece of information by typing:

```
$ docker-machine ip docker-01
```

If you need to power down a remote host, you can use `docker-machine` to stop it:

```
$ docker-machine stop docker-01
```

Verify that it is stopped:

```
$ docker-machine ls
```

The output shows that the status of the machine has changed:

```
Ouput
NAME        ACTIVE   DRIVER         STATE     URL   SWARM   DOCKER    ERRORS
docker-01   -        digitalocean   Stopped                 Unknown
```

To start it again, use the `start` subcommand:

```
$ docker-machine start docker-01
```

Then review its status again:

```
$ docker-machine ls
```

You will see that the `STATE` is now set `Running` for the host:

```
Ouput
NAME                ACTIVE   DRIVER         STATE     URL                         SWARM   DOCKER     ER
docker-01           -        digitalocean   Running   tcp://203.0.113.71:2376             v18.06.1-ce
```

Next let's look at how to interact with the remote host using SSH.

Step 6 — Executing Commands on a Dockerized Host via SSH

At this point, you've been getting information about your machines, but you can do more than that. For example, you can execute native Linux commands on a Docker host by using the `ssh` subcommand of `docker-machine` from your local system. This section explains how to perform `ssh` commands via `docker-machine` as well as how to open an SSH session to a Dockerized host.

Assuming that you've provisioned a machine with Ubuntu as the operating system, execute the following command from your local system to update the package database on the Docker host:

```
$ docker-machine ssh docker-01 apt-get update
```

You can even apply available updates using:

```
$ docker-machine ssh docker-01 apt-get upgrade
```

Not sure what kernel your remote Docker host is using? Type the following:

```
$ docker-machine ssh docker-01 uname -r
```

Finally, you can log in to the remote host with the `docker machine ssh` command:

```
docker-machine ssh docker-01
```

You'll be logged in as the **root** user and you'll see something similar to the following:

```
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
```

```
    http://www.ubuntu.com/business/services/cloud

14 packages can be updated.
10 updates are security updates.
```

Log out by typing `exit` to return to your local machine.

Next, we'll direct Docker's commands at our remote host.

Step 7 — Activating a Dockerized Host

Activating a Docker host connects your local Docker client to that system, which makes it possible to run normal `docker` commands on the remote system.

First, use Docker Machine to create a new Docker host called `docker-ubuntu` using Ubuntu 18.04:

```
$ docker-machine create --driver digitalocean --digitalocean-image ubuntu-18-04-x64 --digitalocean-a
```

To activate a Docker host, type the following command:

```
$ eval $(docker-machine env machine-name)
```

Alternatively, you can activate it by using this command:

```
$ docker-machine use machine-name
```

Tip When working with multiple Docker hosts, the `docker-machine use` command is the easiest method of switching from one to the other.

After typing any of these commands, your prompt will change to indicate that your Docker client is pointing to the remote Docker host. It will take this form. The name of the host will be at the end of the prompt:

```
username@localmachine:~ [docker-01]$
```

Now any `docker` command you type at this command prompt will be executed on that remote host.

Execute `docker-machine ls` again:

```
[docker-01]$ docker-machine ls
```

You'll see an asterisk under the `ACTIVE` column for `docker-01`:

Output

```
NAME                ACTIVE   DRIVER         STATE     URL                         SWARM   DOCKER     E
docker-01           *        digitalocean   Running   tcp://203.0.113.71:2376             v18.06.1-ce
```

To exit from the remote Docker host, type the following:

```
[docker-01]$ docker-machine use -u
```

Your prompt will no longer show the active host.

Now let's create containers on the remote machine.

Step 8 — Creating Docker Containers on a Remote Dockerized Host

So far, you have provisioned a Dockerized Droplet on your DigitalOcean account and you've activated it — that is, your Docker client is pointing to it. The next logical step is to spin up containers on it. As an example, let's try running the official Nginx container.

Use `docker-machine use` to select your remote machine:

```
$ docker-machine use docker-01
```

Now execute this command to run an Nginx container on that machine:

```
[docker-01]$ docker run -d -p 8080:80 --name httpserver nginx
```

In this command, we're mapping port `80` in the Nginx container to port `8080` on the Dockerized host so that we can access the default Nginx page from anywhere.

Once the container builds, you will be able to access the default Nginx page by pointing your web browser to `http://docker_machine_ip:8080`.

While the Docker host is still activated (as seen by its name in the prompt), you can list the images on that host:

```
[docker-01]$ docker images
```

The output includes the Nginx image you just used:

Output

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              71c43202b8ac        3 hours ago         109MB
```

You can also list the active or running containers on the host:

```
[docker-01]$ docker ps
```

If the Nginx container you ran in this step is the only active container, the output will look like this:

```
Output
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
 PORTS                    NAMES
d3064c237372        nginx               "nginx -g 'daemon of…"   About a minute ago   Up About a minu
 0.0.0.0:8080->80/tcp   httpserver
```

If you intend to create containers on a remote machine, your Docker client must be pointing to it — that is, it must be the active machine in the terminal that you're using. Otherwise you'll be creating the container on your local machine. Again, let your command prompt be your guide.

Docker Machine can create and manage remote hosts, and it can also remove them.

Step 9 – Removing Docker Hosts

You can use Docker Machine to remove a Docker host you've created. Use the `docker-machine rm` command to remove the `docker-01` host you created:

```
$ docker-machine rm docker-01
```

The Droplet is deleted along with the SSH key created for it. List the hosts again:

```
$ docker-machine ls
```

This time, you won't see the `docker-01` host listed in the output. And if you've only created one host, you won't see any output at all.

Be sure to execute the command `docker-machine use -u` to point your local Docker daemon back to your local machine.

Step 10 — Disabling Crash Reporting (Optional)

By default, whenever an attempt to provision a Dockerized host using Docker Machine fails, or Docker Machine crashes, some diagnostic information is sent to a Docker account on Bugsnag.

comfortable with this, you can disable the reporting by creating an empty file called `no-error-report` in your local computer's `.docker/machine` directory.

To create the file, type:

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

Check the file for error messages if provisioning fails or Docker Machine crashes.

Conclusion

You've installed Docker Machine and used it to provision multiple Docker hosts on DigitalOcean remotely from your local system. From here you should be able to provision as many Dockerized hosts on your DigitalOcean account as you need.

For more on Docker Machine, visit the official documentation page. The three Bash scripts downloaded in this tutorial are hosted on this GitHub page.

By: finid By: Brian Hogan

Related Tutorials

How To Install Docker Compose on Ubuntu 18.04

How To Remove Docker Images, Containers, and Volumes

Naming Docker Containers: 3 Tips for Beginners

How to Manually Set Up a Prisma Server on Ubuntu 18.04

SCROLL TO TOP

0 Comments

Leave a comment...

Log In to Comment