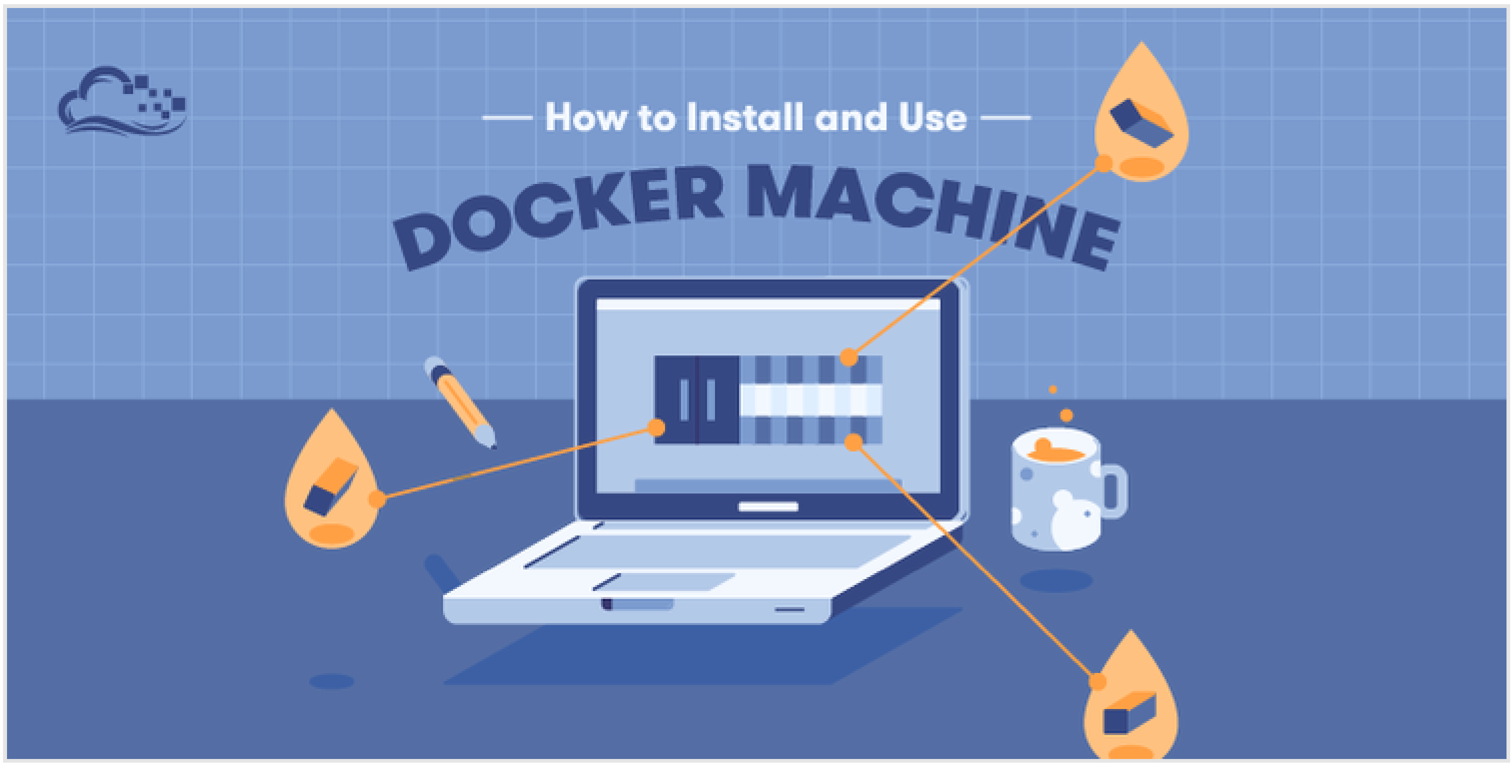

How To Provision and Manage Remote Docker Hosts with Docker Machine on Ubuntu 16.04

15

Updated May 21, 2018 125.8k DOCKER UBUNTU UBUNTU 16.04

By: finid

Not using **Ubuntu 16.04**? Choose a different version:

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 as a matter of course, can be 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 distribution as well as Mac OS X and Windows, in this tutorial, we'll be installing it on your local machine running Ubuntu 16.04 and use it to provision Dockerized DigitalOcean Droplets.

Prerequisites

To follow this tutorial, you will need the following:

- A local machine running Ubuntu 16.04 with Docker installed (see How To Install and Use Docker on Ubuntu 16.04 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 option while generating it, it will have read-write capabilities. To make it easier to use on the command line, be sure to assign the token to a variable as given in that article.

Step 1 — Installing Docker Machine on Your Local Computer

In this step, we'll work through the process of installing Docker Machine on your local computer running Ubuntu 16.04.

To download and install the Docker Machine binary, type:

```
$ wget https://github.com/docker/machine/releases/download/v0.14.0/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
```

The output should be similar to

```
Output
docker-machine version 0.14.0, build 89b8332
```

Step 2 — Installing Additional Docker Machine Scripts

There are three bash scripts in the Docker Machine GitHub repository designed to facilitate the usage of the `docker` and `docker-machine` commands. They provide command completion and bash-prompt customization.

In this step, we'll install these three scripts on your local machine. They will be downloaded and installed into the `/etc/bash_completion.d` directory.

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

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

To complete the installation of the above file, you'll have to set a custom value for the `PS1` variable in your `.bashrc` file. To do so, open it using `nano` (`PS1` is a special shell variable used to modify the bash command prompt):

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

Within that file, there are three lines that begin with **PS1**. They should be 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 so that they read:

~/.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$(__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 easy to switch between Docker Machines. To download it, type:

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

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-m
```

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. Command completion for the `docker` and `docker-machine` commands should now be working.

Step 3 — Provisioning a Dockerized Host Using Docker Machine

Now that you have Docker and Docker Machine running on your local machine, you can now 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 a bash variable using:

```
$ 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 is necessary if you want to the value to persist across terminal sessions.

Open that file with `nano`:

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

Add a line similar to this anywhere:

~/.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 (at a minimum) the driver, the API token (or the variable that evaluates to it), and a unique name for the machine. To create your first machine, type:

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

Partial output as the machine is being created follows. In this output, the name of the machine is `ubuntu1604-docker`:

```
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: c
```

An SSH key pair is created for the new host so that `docker-machine` can access it remotely. The Droplet is provisioned with the desired operating system, and Docker is installed on the system. When the command is complete, your Docker Droplet is up and running.

To see the newly create machine from the command line, type:

```
$ docker-machine ls
```

The output should be similar to this:

```
Output
NAME                ACTIVE      DRIVER          STATE       URL                          SWARM    DOCKER
ubuntu1604-docker     -       digitalocean    Running    tcp://203.0.113.71:2376              v18.05.0-ce
```

Step 4 — Specifying the Base OS 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 DigitalOcean API 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.

Step 5 — Executing Additional Docker Machine Commands

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

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

```
$ docker-machine inspect machine-name
```

The output should include lines like these. 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": "ubuntu1604-docker",
        "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 machine-name
```

The output should 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 machine-name
```

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

```
$ docker-machine stop machine-name
```

Verify that it is stopped.

```
$ docker-machine ls
```

The status of the machine has changed:

```
Ouput
NAME                 ACTIVE   DRIVER         STATE     URL                        SWARM   DOCKER    E
ubuntu1604-docker                 digitalocean   Timeout
```

To start it again:

```
$ docker-machine start machine-name
```

Verify that it is started:

```
$ docker-machine ls
```

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

```
Ouput
NAME                 ACTIVE   DRIVER         STATE     URL                        SWARM   DOCKER    E

ubuntu1604-docker        -        digitalocean   Running   tcp://203.0.113.71:2376          v18.05.
```

Then you may remove it using:

```
$ docker-machine rm machine-name
```

The Droplet is deleted along with the SSH key created for it by `docker-machine.` Now, when you list the Dockerized hosts, you shouldn't see the one you just deleted:

```
$ docker-machine ls
```

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 machine-name apt-get update
```

You can even apply available updates using:

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

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

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

Besides using the `ssh` subcommand to execute commands on the remote Docker host, you can also use it to log into the machine itself. It's as easy as typing:

```
$ docker-machine ssh machine-name
```

Your command prompt will change to reflect the fact that you're logged into the remote host:

```
Output
root@machine-name#
```

To exit from the remote host, type:

```
exit
```

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. 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 the above commands, your bash prompt should 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:~ [machine-name]$
```

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

If a host is active on the terminal that the `docker-machine ls` command is run, the asterisk under the **ACTIVE** column shows that it is the active one.

```
Output
NAME                ACTIVE          DRIVER         STATE     URL                        SWARM   DOCK
ubuntu1604-docker   *         digitalocean   Running   tcp://203.0.113.71:2376           v18.05.0-ce
```

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

```
docker-machine use -u
```

You will be returned to the prompt for your local system.

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 machine-name
```

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

```
$ 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.

If the command executed successfully, 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 should be able to list the images on that host:

```
docker images
```

The output should include the Nginx image you just used, plus others you downloaded before:

```
Output
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              ae513a47849c        3 weeks ago         109MB
```

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

```
$ docker ps
```

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

```
Output
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
4284f9d25548        nginx               "nginx -g 'daemon of…"   20 minutes ago      Up 20 minutes
```

To exit the prompt for the remote host, type. This will close the terminal as well:

```
$ exit
```

> **Note:** 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.

Step 9 — 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 automatically to a Docker account on Bugsnag. If you're not comfortable with this, you can disable the reporting by creating an empty file called `no-error-report` in your installation `.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

This has been an introduction to installing and using Docker Machine to provision multiple Docker Droplets remotely from one local system. Now you should be able to quickly provision as many Dockerized hosts on your DigitalOcean account as you need.

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

By: finid

Upvote (15) Subscribe Share

Editor:
Tammy Fox

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

How To Use Traefik as a Reverse Proxy for Docker Containers on Debian 9

20 Comments

Leave a comment...

Log In to Comment

dekkermichelle1 *June 22, 2016*

0 Hi,

I followed your tutorial. Have some issues with the install as certain parts didn't want to create correctly. I love the fact that I can create docker machines from my ubuntu PC. So if I want to install WordPress for example, do I have to install docker compose? I am new to this. All the info out there is overwhelming to say the least.

---

maxtr *June 26, 2016*

1 docker-machine: 'use' is not a docker-machine command. See 'docker-machine --help'.

---

maxtr *June 26, 2016*

0 [deleted]

---

anthonyrobertson *October 6, 2017*

0 I also ran into this issue, but it's because I had skipped Step 2 which installs some additional wrapper/helper commands (including 'use').

The native equivalent is:

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

Confirm the active host with:

```
docker-machine active
```

And to 'unset':

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

---

sistemas289256 *September 27, 2016*

0 I'm trying to use a diferent user, other than default which is root (for Ubuntu). I'm using --digitalocean-ssh-user=username command line option, but it seems not to be working. I think that it could be related to a cloud-config (userdata) issue... but I have no examples on setting it using docker-machine with digitalocean's driver. Could you help me?

---

selimbaygin *November 6, 2016*

0 Is it possible to use Alpine Linux for container base?

flammable *January 19, 2017*

0 Hello,

My question may sound stupid, but still... At step 8 there is a *'http://machine-ip:8080'* line. Where and how do I find that machine-ip?

billwright *February 20, 2017*

0 I think step 5 tells you this:

> docker-machine ip machine-name

billwright *February 20, 2017*

0 Thanks for this great tutorial.

I have a problem, probably of my own making. I started a docker machine on digital ocean with an API token, which I generated, but I didn't save the token, stupidly. The DigitalOcean UI doesn't seem to allow the displaying of this token, so I created a new one. But I can't seem to get rid of the docker machine I created on DigitalOcean, at least as far as my local docker-machine is concerned. I think this is because I'm no longer using the token that I used to start the machine. I've deleted the machine via DigitalOcean's UI, but I still get this on my machine:

Bills-New-iMac:~ bwright $ docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
consul - digitalocean Running tcp://104.236.28.74:2376 v1.13.1
docker-app-machine - digitalocean Error Unknown GET https://api.digitalocean.com/v2/droplets/40085932: 401 Unable to authenticate you.
Bills-New-iMac:~ bwright $ docker-machine kill docker-app-machine
Killing "docker-app-machine"...
POST https://api.digitalocean.com/v2/droplets/40085932/actions: 401 Unable to authenticate you.
Bills-New-iMac:~ bwright $ docker-machine rm docker-app-machine
About to remove docker-app-machine
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Error removing host "docker-app-machine": DELETE https://api.digitalocean.com/v2/account/keys/6462501: 401 Unable to authenticate you.
Bills-New-iMac:~ bwright $ docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
consul - digitalocean Running tcp://104.236.28.74:2376 v1.13.1
docker-app-machine - digitalocean Error Unknown GET https://api.digitalocean.com/v2/droplets/40085932: 401 Unable to authenticate you.
Bills-New-iMac:~ bwright $

Any thoughts on how I can clean up the mess I made?

callumjhays *April 6, 2017*

1 This guide appears to be out of date. Not sure if there are any newer ones on this site, but following it and running the command:

```
docker-machine create --driver digitalocean --digitalocean-access-token $DOTOKEN ubuntu1604-
```

resulted in

```
Error creating machine: Error in driver during machine creation: POST https://api.digitaloce
```

To fix it, I followed the official Docker guide on DigitalOcean usage with Ubuntu here, using the command:

```
curl -L https://github.com/docker/machine/releases/download/v0.10.0/docker-machine-`uname -s
  chmod +x /tmp/docker-machine &&
  sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

ceroso398 *March 25, 2018*

0 This worked like a charm. Thanks!

borislemke *April 29, 2017*

0 Thanks for writing this post. Does any one have any tips on resizing a machine provisioned by docker-machine? What are the steps to correctly vertical scale a droplet and making sure it will re-attach to the swarm after resizing?

checkos *May 18, 2017*

0 Is it possible to enable private networking while creating a new host?

checkos *March 27, 2018*

0 Figured it out. Posting here in case someone else needs it:

```
docker-machine create --driver digitalocean --digitalocean-size 1gb --digitalocean-regior
```

ronysheer *June 3, 2017*

0 What is the remote machines initial firewall configuration? Can it be strengthened?

christiangiacomi *July 25, 2017*

0 Thank you for the tutorial, it's very easy to follow and works perfectly.

The only question I have is, which are the outgoing firewall ports that need to be opened up to be able to run 'docker run...' from behind the digital ocean cloud firewall?

thanks in advance :)

ceroso398 *March 25, 2018*

0 I followed the steps but it didn't work at first, I got this when running the `docker-machine create` command:

```
Error creating machine: Error in driver during machine creation: POST https://api.digitaloce
```

So I created the machine specifying the ubuntu image explicitly:

```
docker-machine create --driver digitalocean --digitalocean-image ubuntu-16-04-x64 --digitalo
```

I got past the error to get this new output:

```
Running pre-create checks...
Creating machine...
(ubuntu1604-docker) Creating SSH key...
(ubuntu1604-docker) Creating Digital Ocean droplet...
(ubuntu1604-docker) Waiting for IP address to be assigned to the Droplet...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with ubuntu(systemd)...
Installing Docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Error creating machine: Error running provisioning: ssh command error:
command : sudo systemctl -f start docker
err     : exit status 1
output  : Job for docker.service failed because the control process exited with error code.
```

According to this thread this is apparently related to a version mismatch problem so I provisioned the machined using the command specified in the thread:

```
docker-machine create --driver digitalocean --digitalocean-image ubuntu-16-04-x64 --digitalo
```

After this, the machine got successfully created.

mikkel418725b8766e1a77ce54 *April 9, 2018*

0 Thanks great article..but how do you connect to your swarm from yet another machine ? i would imagine something like eval $(docker-machine env production-manager-1) but the new machine has no knowledge of what production-manager-1 is.

tannerchung *April 10, 2018*

0 The new machine should be provisioned with docker-machine already so you can just go into the server you just created via `docker-machine ssh` : https://docs.docker.com/machine/reference/ssh/

For your case `docker-machine ssh production-manager-1`

Or if you ssh into the server you just created and you should be able to do `docker-machine ls` : https://docs.docker.com/machine/reference/ls/ and see all the machines on the swarm.

mikkel418725b8766e1a77ce54 *April 10, 2018*

0 Thanks but I'm not sure that answers my question. Say I created the swarm from my Ubuntu workstation #1 and want to manage it from workstation #2, who has nothing to do with that swarm at all. Then what ?

kobenauf *June 2, 2018*

0 When I run docker-machine create, I get an error. Here's output with debug turned on. The machine-name is 'docker-registry'.
Command

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

Error

```
checking docker daemon
(docker-registry) Calling .GetSSHHostname
(docker-registry) Calling .GetSSHPort
(docker-registry) Calling .GetSSHKeyPath
(docker-registry) Calling .GetSSHKeyPath
(docker-registry) Calling .GetSSHUsername
Using SSH client type: external
Using SSH private key: /home/kobenauf/.docker/machine/machines/docker-registry/id_rsa (-rw--
&{[-F /dev/null -o PasswordAuthentication=no -o StrictHostKeyChecking=no -o UserKnownHostsFi
3.66.142 -o IdentitiesOnly=yes -i /home/kobenauf/.docker/machine/machines/docker-registry/id
About to run SSH command:
sudo docker version
SSH cmd err, output: exit status 1: sudo: docker: command not found

Error getting SSH command to check if the daemon is up: ssh command error:
command : sudo docker version
err     : exit status 1
output  : sudo: docker: command not found
```

Earlier in the output I noticed:

```
About to run SSH command:
if ! type docker; then curl -sSL https://get.docker.com | sh -; fi
SSH cmd err, output: <nil>: bash: line 0: type: docker: not found
```

When I run just `curl -sSL https://get.docker.com` it tells me the certificate has expired. It seems that is causing docker not to install and the rest to fail.