# Docker Machine Basics

**Objective:** Introduce Docker Machine basics and advances cloud host options.<br>

**Preparation:** Open two terminal windows, an internet browser, and optional create an online Ditigal Ocean account. Also, navigate to the appropriate lab folder.<br>

**Outcome:** Create an container in a VM from out host, and then create cloud host from our host, using Ditigal Ocean.<br>

**Data Files:** None.<br>

Docker Machine is a tool for provisioning and managing your Dockerized hosts (hosts with Docker Engine on them). Typically, you install Docker Machine on your local system. Docker Machine has its own command line client docker-machine and the Docker Engine client, docker. You can use Machine to install Docker Engine on one or more virtual systems. These virtual systems can be local (as when you use Machine to install and run Docker Engine in VirtualBox on Mac or Windows) or remote (as when you use Machine to provision Dockerized hosts on cloud providers).

## Step 1. Installation<br>

1. Check first to see if docker-machine is already installed:

```
$ docker-machine -v
```

Output:

```
docker-machine version X.X.X, build XXXXX  // This is wha
t you would see if it is installed.
```

2. Confirm that the Docker binary has already been installed.

```
$ docker -v
```

3. Download the Docker Machine binary and extract it to your PATH.

**MacOS:**

```
$ curl -L https://github.com/docker/machine/releases/down
load/v0.10.0/docker-machine-`uname -s`-`uname -m` >/usr/l
ocal/bin/docker-machine && chmod +x /usr/local/bin/docker
-machine
```

**Linux:**

```
$ curl -L https://github.com/docker/machine/releases/down
load/v0.10.0/docker-machine-`uname -s`-`uname -m` >/tmp/d
ocker-machine && chmod +x /tmp/docker-machine && \
sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

**Windows:**

```
$ if [[ ! -d "$HOME/bin" ]]; then mkdir -p "$HOME/bin"; f
i && \
curl -L https://github.com/docker/machine/releases/download/v0.10.0/docker-machine-Windows-x86_64.exe > "$HOME/bin/docker-machine.exe" && \
  chmod +x "$HOME/bin/docker-machine.exe"
```

3. Check the installation by displaying the Machine version:

```
$ docker-machine version
```

Output:

```
$ docker-machine -v
```

Output

```
docker-machine version X.X.X, build XXXXX
```

## Step 2. Create a machine<br>

For this lab, we will use docker machine on Mac system, and create a docker host with virtualbox driver.

1. Before we start, we can use `ls` command to check if there is any machine already in our host.

```
$ docker-machine ls
```

2. Now, create a machine called default.

```
$ docker-machine create -d virtualbox default
```

Output:

```
Running pre-create checks...
Creating machine...
(default) Copying /Users/penxiao/.docker/machine/cache/bo
ot2docker.iso to /Users/penxiao/.docker/machine/machines/
default/boot2docker.iso...
(default) Creating VirtualBox VM...
(default) Creating SSH key...
(default) Starting the VM...
(default) Check network to re-create if needed...
(default) Waiting for an IP...
Waiting for machine to be running, this may take a few mi
nutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
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 your Docker Client to the Docker En
gine running on this virtual machine, run: docker-machine
 env default
```

3. Now, let’s list of current Docker Machines again, like before:

```
$ docker-machine ls
```

Output:

```
NAME      ACTIVE   DRIVER       STATE     URL
             SWARM   DOCKER    ERRORS
default   -        virtualbox   Running   tcp://192.168.9
9.100:2376           v1.12.3
```

## Step 2. How to use the docker host<br>

There are two ways to access the docker host, both are demonstrated below. First, we will ssh into the docker host directly and play with docker inside use docker client on localhost (outside the docker host) to access the docker engine inside the docker host. Then, we will set out environment variables, so that we can use our host machine to run our VM.

1. SSH into the docker host

```
$ docker-machine ssh default
```

Output:

```
                       ##         .
                 ## ## ##        ==
              ## ## ## ## ##    ===
          /"""""""""""""""""\___/ ===
     ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
          \______ o           __/
            \    \         __/
             \____\_______/
_                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ _
_
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '_
_|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 1.12.3, build HEAD : 7fc7575 - Thu Oc
t 27 17:23:17 UTC 2016
Docker version 1.12.3, build 6b644ec
```

2. List our current docker conainters:

```
# docker ps
```

Output:

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

3. Let’s run ‘hello-world’ and remove it upon completion, then exit:

```
# docker run --rm hello-world
```

Output:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd6779851241
1de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest
```

```
# exit
```

4. Now, let’s connect our Docker client with our remote Docker engine on the VM. This way we can run commands from our host. Get the environment commands for your new VM.

```
$ docker-machine env default
```

Output:

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/penxiao/.docker/machine/m
achines/default"
export DOCKER_MACHINE_NAME="default"
...
```

5. Now, connect your docker client CLI to the new machine. But follow along below to get a before and after we run eval “$(docker-machine env default)” on localhost:

```
$ docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID
   CREATED             SIZE
ubuntu              14.04               aae2b63c4946
   5 days ago          188 MB
mongo               2.6                 1999482cb0a5
   6 weeks ago         391 MB
python              2.7                 6b494b5f019c
   3 months ago        676.1 MB
tutum/nginx         latest              a2e9b71ed366
   8 months ago        206.1 MB
```

> *NOTE: Images output on the host will appear differently, according to what images you have currently stored in your local repo.*

Now, we'll actually set our environment variables:

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

And now, that's see what's happened:

```
$ docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID
   CREATED             SIZE
hello-world         latest              c54a2cc56cbb
   5 months ago        1.848 kB
```

This sets environment variables for the current shell that the Docker client will read which specify the TLS settings. You need to do this each time you open a new shell or restart your machine. You can now run Docker commands on this host.

### Create a Machine in the Cloud<br>

## Step 3. Create Digital Ocean

## Account<br>

> *NOTE: Docker Machine will still work as described below, but Docker Cloud supercedes Machine for this purpose.*

Follow along with this example to create a Dockerized Digital Ocean Droplet (cloud host).

1. Create a Digital Ocean account-If you have not done so already, go to Digital Ocean, create an account, and log in.

2. Generate a personal access token-To generate your access token:

3. Go to the Digital Ocean administrator console and click API in the header.

4. Click API in Digital Ocean console

5. Click Generate New Token to get to the token generator.

6. Generate token-Give the token a clever name (e.g. “machine”), make sure the Write (Optional) checkbox is checked, and click Generate Token.

7. Name and generate token-Grab (copy to clipboard) the generated big long hex string and store it somewhere safe.

8. Copy and save personal access token-This is the personal access token you’ll use in the next step to create your cloud server.

## Step 4. Use Machine to Create the Droplet

Run docker-machine create with the digitalocean driver and pass your key to the --digitalocean-access-token flag, along with a name for the new cloud server.

1. For this example, we’ll call our new Droplet “docker-sandbox”.

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

Output:

```
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 assig
ned to the Droplet...
Waiting for machine to be running, this may take a few mi
nutes...
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
```

When the Droplet is created, Docker generates a unique SSH key and stores it on your local system in `~/.docker/machines`. Initially, this is used to provision the host. Later, it's used under the hood to access the Droplet directly with the `docker-machine` ssh command. Docker Engine is installed on the cloud server and the daemon is configured to accept remote connections over TCP using TLS for authentication.

2. Go to the Digital Ocean console to view the new Droplet.

3. At the command terminal, run docker-machine ls.

```
 $ docker-machine ls
```

Output:

```
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
```

The new docker-sandbox machine is running, and it is the active host as indicated by the asterisk (`*`). When you create a new machine, your command shell automatically connects to it. If for some reason your new machine is not the active host, you'll need to run `docker-machine env docker-sandbox`, followed by `eval $(docker-machine env docker-sandbox)` to connect to it.

## Step 5. Run Docker Commands on the Droplet

Run some docker-machine commands to inspect the remote host.

1. For example, `docker-machine ip <machine>` gets the host IP address and `docker-machine inspect <machine>` lists all the details.

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

Output:

```
  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/machin
e/machines/docker-sandbox/id_rsa",
     "StorePath": "/Users/samanthastevens/.docker/machine
",
     "SwarmMaster": false,
     "SwarmHost": "tcp://0.0.0.0:3376",
     "SwarmDiscovery": "",
     ...
```

2. Verify Docker Engine is installed correctly by running `docker` commands.

Start with something basic like `docker run hello-world`, or for a more interesting test, run a Dockerized webserver on your new remote machine.

In this example, the `-p` option is used to expose port 80 from the `nginx` container and make it accessible on port 8000 of the `docker-sandbox` host.

```
 $ 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
```

In a web browser, go to `http://<host_ip>:8000` to bring up the webserver home page. You got the `<host_ip>` from the output of the `docker-machine ip <machine>` command you ran in a previous step. Use the port you exposed in the docker run command.

## Step 7. Use Machine to remove the Droplet

To remove a host and all of its containers and images, first stop the machine, then use docker-machine rm:

```
$ docker-machine stop docker-sandbox
$ docker-machine rm docker-sandbox
```

Output:

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

```
$ docker-machine ls
```

Output:

```
NAME      ACTIVE   DRIVER       STATE     URL
             SWARM
default   *        virtualbox   Running   tcp:////xxx.xxx
.xx.xxx:xxxx
```

If you monitor the Digital Ocean console while you run these commands, you will see it update first to reflect that the Droplet was stopped, and then removed.

If you create a host with Docker Machine, but remove it through the cloud provider console, Machine will lose track of the server status. So please use the `docker-machine rm` command for hosts you create with `docker-machine create`.

## Conclusion

In the lesson we learn to create Docker Machines. By combining Docker Engine with Docker Machine, plus VirtualBox, we where able to work within a VM, from our host. We created an image using Docker Machine and then explored it. Then, we learned how we might spin up a cloud host, using Docker Machine.