Docker Machine

<< Back to Docker main

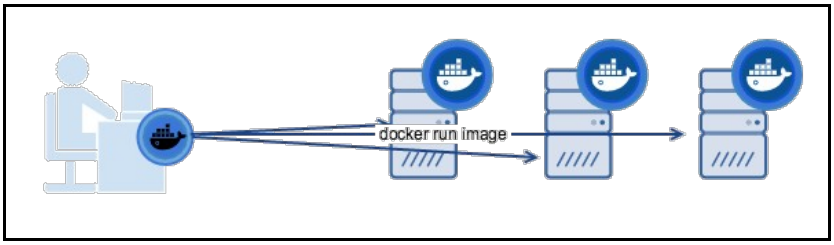
Contents

Manage Virtual Hosts with docker-machine

Introduction

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. You can use Machine to create Docker hosts on your local Mac or Windows box, on your company network, in your data center, or on cloud providers like Azure, AWS, or Digital Ocean.

Using docker-machine commands, you can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.



When people say ?Docker? they typically mean Docker Engine, the client-server application made up of the Docker daemon, a REST API that specifies interfaces for interacting with the daemon, and a command line interface (CLI) client that talks to the daemon (through the REST API wrapper). Docker Engine accepts docker commands from the CLI, such as docker run <image>, docker ps to list running containers, docker image ls to list images, and so on.

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). The Dockerized hosts themselves can be thought of, and are sometimes referred to as, managed ?machines?.

Source: https://docs.docker.com/machine/overview/

Hypervisor drivers

What is a driver

Machine can be created with the **docker-machine create** command. Most simple usage:

```
docker-machine create -d <hybervisor driver name> <driver options> <machine name>
```

The default value of the driver parameter is "virtualbox".

docker-machine can create and manage virtual hosts on the local machine and on remote clouds. Always the chosen driver determines where and how the virtual machine will be created. The guest operating system that is being installed on the new machine is also determined by the hypervisor driver. E.g. with the "virtualbox" driver you can create machines locally using the boot2docker as the guest OS.

The driver also determines the virtual network types and interfaces types that are created inside the virtual machine. E.g. the KVM driver creates two virtual networks (bridges), one host-global and one host-private network.

In the **docker-machine create** command, the available driver options are also determined by the driver. You always has to check the available options at the vandor of driver. For cloud drivers typical options are the remote url, the login name and the password. Some driver allows to change the guest OS, the CPU number or the default memory.

Here is a complete list of the currently available drivers:
https://github.com/docker/docker.github.io/blob/master/machine/AVAILABLE_DRIVER_PLUGINS.md

KVM driver

KVM driver home page: https://github.com/dhiltgen/docker-machine-kvm

Minimum Parameters:

- --driver kvm
- --kvm-network: The name of the kvm virtual (public) network that we would like to use. If this is not set, the new machine will be connected to the **"default"** KVM virtual network.

Images:
By default docker-machine-kvm uses a boot2docker.iso as guest os for the kvm hypervisior. It's also possible to use every guest os image that is derived from boot2docker.iso as well. For using another image use the --kvm-boot2docker-url parameter.

Dual Network:

- **eth1** - A host private network called **docker-machines** is automatically created to ensure we always have connectivity to the VMs.
  - The docker-machine ip command will always return this IP address which is only accessible from your local system.
  - That network is to be used for the swarm management, node to node communication (as this data is very sensitive), so this ip has to be provided in the **--advertise-addr** parameter in the "**docker swarm init**" or in the "**docker swarm join**" command.
- **eth0** - You can specify any libvirt named network. If you don't specify one, the "default" named network will be used.
  - Typically this would be your "public" network accessible from external systems
  - To retrieve the IP address of this network, you can run a command like the following:

```
docker-machine ssh mymachinename "ip -one -4 addr show dev eth0|cut -f7 -d' '"
```

Driver Parameters:

- --kvm-cpu-count Sets the used CPU Cores for the KVM Machine. Defaults to 1 .
- --kvm-disk-size Sets the kvm machine Disk size in MB. Defaults to 20000 .
- --kvm-memory Sets the Memory of the kvm machine in MB. Defaults to 1024.
- --kvm-network Sets the Network of the kvm machinee which it should connect to. Defaults to default.
- --kvm-boot2docker-url Sets the url from which host the image is loaded. By default it's not set.
- --kvm-cache-mode Sets the caching mode of the kvm machine. Defaults to default.
- --kvm-io-mode-url Sets the disk io mode of the kvm machine. Defaults to threads.

Install softwares

First we have to install the docker-machine app itself:

```
base=https://github.com/docker/machine/releases/download/v0.14.0 &&
  curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
  sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Secondly we have to install the hypervisor driver for the docker-machine to be able to create, manage Virtual Machines running on the hypervisor. As we are going to use the KVM hypervisor, we have to install the "docker-machine-driver-kvm" driver:

Warning
Fontos, hogy a 10-es verziójú KVM driver ami a CentOS7-hez való nem kompatibilis a Fedora 26-al. Én végül továbbra is a 7-es verziót használom, ez továbbra is jó

```
# curl -Lo docker-machine-driver-kvm \
  https://github.com/dhiltgen/docker-machine-kvm/releases/download/v0.7.0/docker-machine-driver-kvm \
  && chmod +x docker-machine-driver-kvm \
  && sudo mv docker-machine-driver-kvm /usr/local/bin
```

We suppose that KVM and the libvirt is already installed on the system.

Tip
If you want to use VirtualBox as your hypervisor, no extra steps are needed, as its docker-machine driver is included in the docker-machine app

Available 3rd party drivers:
https://github.com/docker/docker.github.io/blob/master/machine/AVAILABLE_DRIVER_PLUGINS.md

Create machines with KVM

Create the machine

Before a new machine can be created with the docker-machine command, the proper KVM virtual network must be created.

See How to create KVM networks for details.

```
# docker-machine create -d kvm --kvm-network "docker-network" manager

Running pre-create checks...
```

```
Creating machine...
(manager) Copying /root/.docker/machine/cache/boot2docker.iso to /root/.docker/machine/machines/manager/boot2docker.iso...
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 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 Engine running on this virtual machine, run: docker-machine env manager
```

Tip
The machine is created under **/USER_HOME/.docker/machine/machines/<machine_name>** directory

If the new VM was created with virtualbox driver, the VirtualBox graphical management interface must be started with the same user, that the VM was created with, and the VirtualBox will discover the new VM automatically

Warning
A 18.9-es boot2docker nem kompatibilis a Fedora-s docker-machine-al, mert valamiért nem nem érzékeli, hogy az SSH már elindult (Waiting for SSH to be available...). Töltsük le a boot2docker_v18.06.1, és azzal már m?ködik. A --kvm-boot2docker-url-el kell megadni a letöltött ISO-t.

```
# docker-machine create -d kvm --kvm-network "docker-network" --kvm-boot2docker-url="/home/adam/Downloads/boot2docker/boot2docker-v18.05.0-ce

# docker-machine create -d kvm --kvm-network "docker-network" --kvm-boot2docker-url="/root/.docker/machine/cache/boot2docker_v18.06.1-ce.iso"
```

Use different OS

Az operációs rendszert es?sorban a driver típusa határozza meg. A KVM driver-hez alapértelmezetten a boot2docker OS tartozik. Azonban bármilyen más boot2docker alapú oprendszert is használhatunk. Az Amazon AWS driver használata esetén Ubuntu lesz az OS.

KVM driver esetén ha nem az alapértelmezett OS-t akarjuk használni, akkor a --kvm-boot2docker-url kapcsolóval kell megadni a .ISO helyét. Én a rancherOS-t próbáltam ki és m?ködött. Innen lehet letölteni: https://github.com/rancher/os (közvetlen link: https://releases.rancher.com/os/v1.5.0/rancheros.iso)

```
docker-machine create -d kvm --kvm-boot2docker-url "/home/adam/Downloads/rancheros.iso" --kvm-network "docker-network" manager1
```

Check what was created

Interfaces on the host

```
# ifconfig
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.105  netmask 255.255.255.0  broadcast 192.168.0.255
        ....
virbr1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.42.1  netmask 255.255.255.0  broadcast 192.168.42.255
        ...
virbrDocker: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.123.1  netmask 255.255.255.0  broadcast 192.168.123.255
        inet6 2001:db8:ca2:2::1  prefixlen 64  scopeid 0x0<global>
        ...
```

On the host, upon the regular interfaces, we can see the two bridges for the two virtual networks:

- **virbrDocker**: That is the virtual network that we created in libvirt. This is connected to the host network with NAT. We assigned these IP addresses, when we defined the network.
- **virbr1**: That is the host-only virtual network that was created out-of-the-box. This one has no internet access.

Interface the new VM

You can log in to the newly created VM with the **docker-machine ssh <machine_name>** command On the newly created docker ready VM, four interfaces were created.

```
# docker-machine ssh manager
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \_____ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.05.0-ce, build HEAD : b5d6989 - Thu May 10 16:35:28 UTC 2018
```

```
Docker version 18.05.0-ce, build f150324
```

Check the interfaces of the new VM:

```
docker@manager:~$ ifconfig
docker0   inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          ...
eth0      inet addr:192.168.123.195  Bcast:192.168.123.255  Mask:255.255.255.0
          ...
eth1      inet addr:192.168.42.118  Bcast:192.168.42.255  Mask:255.255.255.0
```

- **eth0**:192.168.123.195 - Interface to the new virtual network (docker-network) created by us. this network is connected to the host network,so it has public internet access as well. (the network was created with "**forward mode=nat**", see KVM#Add new network for details)
- **eth1**:192.168.42.118 - This connect to the dynamically created host-only virtual network. Just for VM-to-VM communication
- **docker0**:172.17.0.1 - This VM is ment to host docker container, so the docker daemon was already installed and started on it. Form docker point of view, this VM is also a (docker) host, and therefore the docker daemon created the default virtual bridge, that the containers will be connected to unless it is specified implicitly otherwise during container creation.

Inspect the new VM with the **docker-machine inspect** command

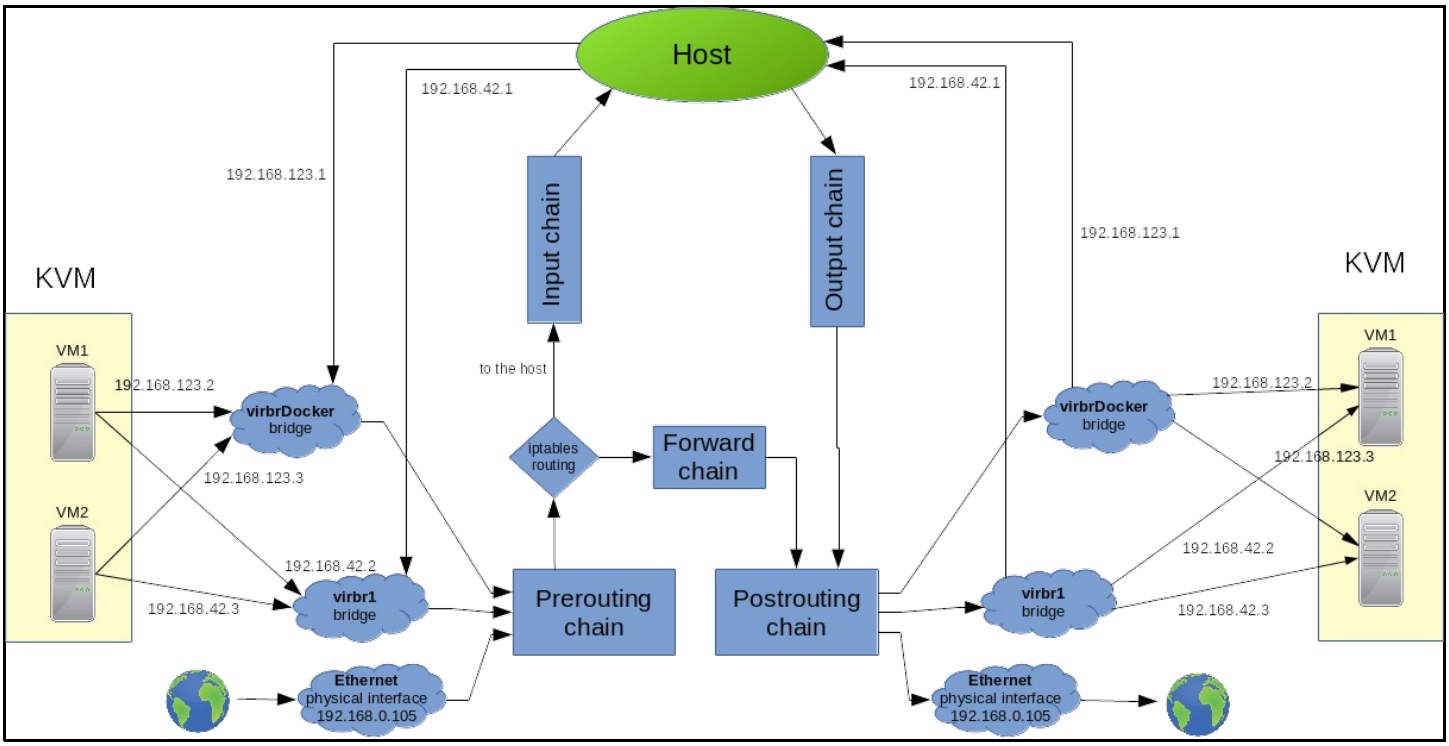
```
# docker-machine inspect manager
{
    "ConfigVersion": 3,
    "Driver": {
        ....
        "CPU": 1,
        "Network": "docker-network",
        "PrivateNetwork": "docker-machines",
        "ISO": "/root/.docker/machine/machines/manager/boot2docker.iso",
        "...
    },
    "DriverName": "kvm",
    "HostOptions": {
       ....
        },
        "SwarmOptions": {
            "IsSwarm": false,
            ...
        },
        "AuthOptions": {
           ....
        }
    },
    "Name": "manager"
}
```

Routing table

All the packages that ment to go to the docker VMs are routed to the bridges

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
...
192.168.42.0    0.0.0.0         255.255.255.0   U     0      0        0 virbr1  <<<<this
192.168.123.0   0.0.0.0         255.255.255.0   U     0      0        0 virbrDocker  <<<this
```

IPtables modifications




```
# iptables-save | grep virbr
```

Switches:

- -o, --out-interface name
- -i, --input-interface name
- -s, source IP address
- -d, destination IP address
- -p, Sets the IP protocol for the rule
- -j, jump to the given target/chain

DNS and DCHP packages from the Virtual Bridges are allowed to be sent to the host machine.

```
-A INPUT -i virbr1 -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i virbr1 -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -i virbr1 -p udp -m udp --dport 67 -j ACCEPT
-A INPUT -i virbr1 -p tcp -m tcp --dport 67 -j ACCEPT
-A INPUT -i virbrDocker -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -i virbrDocker -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -i virbrDocker -p udp -m udp --dport 67 -j ACCEPT
-A INPUT -i virbrDocker -p tcp -m tcp --dport 67 -j ACCEPT
```

The host machine is allowed to send DHCP packages to the virtual bridges in order to configure them.

```
-A OUTPUT -o virbr1 -p udp -m udp --dport 68 -j ACCEPT
-A OUTPUT -o virbrDocker -p udp -m udp --dport 68 -j ACCEPT
```

The bridge **virbrDocker** can send packages anywhere (first line) and can receive packages back if the connections was previously established (second line)

```
-A FORWARD -s 192.168.123.0/24 -i virbrDocker -j ACCEPT
-A FORWARD -d 192.168.123.0/24 -o virbrDocker -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Both bridges can send packages to any host on the same network (because it was created with "**forward mode=nat**", see KVM#Add new network for details). Everything else is rejected that was sent to or form the bridges (last four lines)

```
-A FORWARD -i virbrDocker -o virbrDocker -j ACCEPT
-A FORWARD -i virbr1 -o virbr1 -j ACCEPT

#If not accepted above, we reject everything from the two bridges
-A FORWARD -o virbrDocker -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -i virbrDocker -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -o virbr1 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -i virbr1 -j REJECT --reject-with icmp-port-unreachable
```

The bridge **virbrDocker** can send packages to the outside world. (MASQUERADE is a special SNAT target, where the destination IP doesn't have to be specified. SNAT replaces the source IP address of the package with the public IP address of our system) Last two lines: The bridge can't send anything to the multicast and to the broadcast addresses.

```
-A POSTROUTING -s 192.168.123.0/24 ! -d 192.168.123.0/24 -p tcp -j MASQUERADE --to-ports 1024-65535
-A POSTROUTING -s 192.168.123.0/24 ! -d 192.168.123.0/24 -p udp -j MASQUERADE --to-ports 1024-65535
-A POSTROUTING -s 192.168.123.0/24 ! -d 192.168.123.0/24 -j MASQUERADE
```

```
-A POSTROUTING -s 192.168.123.0/24 -d 224.0.0.0/24 -j RETURN
-A POSTROUTING -s 192.168.123.0/24 -d 255.255.255.255/32 -j RETURN
```

Manage machines

List/Inspect

Whit the **ls** subcommand we can list all the docker-machine managed hosts.

```
# docker-machine ls
NAME      ACTIVE   DRIVER   STATE     URL                         SWARM   DOCKER        ERRORS
manager   -        kvm      Running   tcp://192.168.42.118:2376           v18.05.0-ce
```

- NAME: name of the created machine
- ACTIVE: from the Docker client point of view, the active virtual host can be managed with the docker and with the docker-compose commands form the local host, as we would executed these commands on the remote virtual host. There can be always a single active machine that is marked with an asterisk '*' in the ls output.
- DRIVER:
- STATE:
- URL: The IP address of the virtual host.
- SWARM: Megmutatja, hogy az adott VM tagja egy egy swarm classic cluster-nek, és ha igen, akkor master vagy agent.

Note
A **SWARM** oszlop kizárólag a swarm classic cluster-ekre vonatkozik, és ott is csak azokra a VM-ekre, amik a --swarm kapcsolóval lettek létrehozva.
Lásd: Docker Swarm Classic

With the **inspect <machine name>** subcommand we can get very detailed information about a specific machine:

```
# docker-machine inspect manager
{
    "ConfigVersion": 3,
    "Driver": {
        "IPAddress": "",
        "MachineName": "manager",
        "SSHUser": "docker",
        "SSHPort": 22,
        "SSHKeyPath": "",
        "StorePath": "/root/.docker/machine",
        "SwarmMaster": false,
        "SwarmHost": "tcp://0.0.0.0:3376",
        "SwarmDiscovery": "",
        "Memory": 1024,
        "DiskSize": 20000,
        "CPU": 1,
        "Network": "docker-network",
        "PrivateNetwork": "docker-machines",
        "ISO": "/root/.docker/machine/machines/manager/boot2docker.iso",
        ...
    },
    "DriverName": "kvm",
    "HostOptions": {
         ...
       ...
    "Name": "manager"
}
```

Set Active machine

With our local docker client we can connect to the docker daemon of any of the virtual hosts. That virtual host that we can managed locally is called "active" host. From Docker client point of view, the active virtual host can be managed with the **docker** and with the **docker-compose** commands form the local host, as we executed these commands on the (remote) virtual host. We can make any docker-machine managed virtual host active with the '**docker-machine env <machine name>**' command. Docker gets connection information from environment variables. With this command we can redirect our docker CLI. Run this command in the host.

```
# docker-machine env manager
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.42.118:2376"
export DOCKER_CERT_PATH="/root/.docker/machine/machines/manager"
export DOCKER_MACHINE_NAME="manager"
# Run this command to configure your shell:
# eval $(docker-machine env manager)
```

As the output of the env command suggests, you have to run the **eval** command in that shell that you want to use to managed the active virtual host.

```
# eval $(docker-machine env manager)
```

Now, in the same shell, run the **ls** command again. The machine 'manager' will be marked with the asterisk in the ACTIVE column.

```
# docker-machine ls
NAME      ACTIVE   DRIVER   STATE     URL                         SWARM   DOCKER        ERRORS
manager   *        kvm      Running   tcp://192.168.42.118:2376           v18.05.0-ce
```

In the same shell on the host machine, create a docker container:

```
# docker run -d -i -t --name container1 ubuntu /bin/bash
9dfda56f7739831b0d19c8acd95748b1c93f6c6bb82d2aa87cfb10ecee0e4f28
```

Now we will log on to the virtual host with the **ssh <machine name>** command.

```
# docker-machine ssh manager
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.05.0-ce, build HEAD : b5d6989 - Thu May 10 16:35:28 UTC 2018
Docker version 18.05.0-ce, build f150324
docker@manager:~$
```

List the available docker containers. We should see there the newly created **container1**. The **docker run** command was executed in the host, bat was run in the remote, virtual host.

```
docker@manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
9dfda56f7739        ubuntu              "/bin/bash"         5 minutes ago       Up 5 minutes                            container2
```

Running the **docker ps** command on the host, should give the same result.

Unset active machine

The active machine can be unset with the "**--unset**" switch. Once the active docker machine was unset, the docker client will manage the local docker daemon again.

```
# docker-machine env --unset
unset DOCKER_TLS_VERIFY
unset DOCKER_HOST
unset DOCKER_CERT_PATH
unset DOCKER_MACHINE_NAME
# Run this command to configure your shell:
# eval $(docker-machine env --unset)
```

As the output suggest, we have to run the eval command again with the **--unset** switch to clear the shell. Alternatively you can just start a new shell.

```
# eval $(docker-machine env --unset)
```

Now lets run the **ps** command again. As the docker client now connected to the local docker daemon, we shouldn't see **container1** anymore in the list.

```
# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
66e9cfbbc947        busybox             "sh"                30 hours ago        Up 4 minutes                            critcon
```

Libvirt

List all the KVM hosts:

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     manager                        running
 -     worker1                        shut off
 -     worker2                        shut off
 -     worker3                        shut off
```

Set the VM to autostart at system boot

```
# virsh autostart manager
Domain manager marked as autostarted



# virsh start worker3
Domain worker3 started
```