Uninstalling minikube

```
$ minikube stop
Stopping "minikube" in hyperkit ...
"minikube" stopped.

$ minikube delete
Deleting "minikube" in hyperkit ...
The "minikube" cluster has been deleted.
Successfully deleted profile "minikube"
```

Deleting minikube

Don’t remember how, but I ended up with two Kubernetes installs on my mcfly desktop with macOS: the one that came with Docker Desktop for macOS and the minikube variety that I must have downloaded and installed in the past.

What is minikube?

minikube is a local Kubernetes environment for testing and development purposes. It spins up a lightweight virtual machine (will work even on a modest laptop) and runs an entire Kubernetes cluster in it.

Kubernetes (I haven’t written about it on Unix Tutorial yet) is an open-source system for managing containerized applications – deploying, scaling and failing them over using cluster architecture.
Deleting minikube

It’s great that minikube has support for such scenarios, so I just stopped it and invoked delete command like this:

greys@mcfly:~ $ minikube stop
  Stopping "minikube" in hyperkit ...
  "minikube" stopped.

greys@mcfly:~ $ minikube delete
  Deleting "minikube" in hyperkit ...
  The "minikube" cluster has been deleted.
  Successfully deleted profile "minikube"

To be sure things aren’t left behind, I also deleted the minikube configurartion directory:

and even the binary symlink itself:

Docker-desktop variety of Kubernetes is now the only one left, so I can continue my experiments and will publish more in the coming days:
See Also

- Docker
- Install Docker with brew
- Docker Desktop vs Docker Machine

Hello, World in podman
Turns out it’s not that easy to install Docker CE in **RHEL 8** just yet. Well, maybe there’s no immediate need since RHEL 8 comes with its own containerization stack based on **podman**?

**Hello, World in podman**

**podman** provides comprehensive compatibility with **docker** command, most non-Docker specific options are supported.

If you are familiar with **docker** command syntax, give it a try by just replacing docker with **podman** command.

Let’s do the hello world exercise:

greys@redhat:~ $ **podman run hello-world**
Trying to pull registry.redhat.io/hello-world:latest...Failed
Trying to pull quay.io/hello-world:latest...Failed
Trying to pull docker.io/hello-world:latest...Getting image source signatures
Copying blob 1b930d010525: 977 B / 977 B
[==================================] 0s
Copying config fce289e99eb9: 1.47 KiB / 1.47 KiB
[================================] 0s
Writing manifest to image destination
Storing signatures

Hello from Docker!

This message shows that your installation appears to be working correctly.
To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

As you can see, **podman** searches in Red Hat and Quay image repositories before moving on to Docker registry, but finally gets the hello-world image there.

**Run Ubuntu image in podman**

And if we want to follow Docker’s advice and try running the Ubuntu Docker image, we’ll replace

```
docker run -it ubuntu bash
```

with

```
podman run -it ubuntu bash
```

… It just works:
`podman` will definitely read up and post more about `podman` and containerization in `Red Hat` in the following weeks.
One of the first things I wanted to try on my fresh Linux Mint 19.1 install was Docker. This post captures the steps I’ve taken to get Docker CE (community edition) version installed on my Linux-based Dell XPS laptop.

Docker in Linux Mint

Since Linux Mint is based on Ubuntu, you shouldn’t have any issues getting Docker installed from the official repository of Docker.io. I have followed instructions from here: Docker Docs – Install Docker CE in Ubuntu.

I followed instruction pretty much word for word, except the part where you’re adding a new apt repository to your Linux Mint.

In the procedure fully documented below, I have changed this command:
This is because Linux Mint has its own version names and they don’t match Ubuntu ones. But because I know that Linux Mint 19.1 is based on Ubuntu Bionic, I just made the replacement as shown above.

**Install Docker CE in Linux Mint 19.1**

Here are the full steps for the installation.

**Installing Additional Packages**

greys@xps:/$ sudo apt-get install \
> apt-transport-https \
> ca-certificates \
> curl \
> gnupg-agent \
> software-properties-common

Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20180409).
The following additional packages will be installed:
dirmngr gnupg gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server gpgconf
gpgsm gpgv libcurl4
Suggested packages:
tor parcimonie xloadimage scdaemon
The following NEW packages will be installed:
apt-transport-https gnupg-agent software-properties-common
The following packages will be upgraded:
Add Ubuntu version of Docker Repository To Our System

greys@xps:/$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
greys@xps:/$ sudo apt-key fingerprint 0EBFCD88
pub rsa4096 2017-02-22 [SCEA]
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid [ unknown] Docker Release (CE deb) <docker@docker.com>
sub rsa4096 2017-02-22 [S]
greys@xps:/$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
greys@xps:/$ sudo apt-get update
Hit:1 http://archive.canonical.com/ubuntu bionic InRelease
Get:2 http://security.ubuntu.com/ubuntu  bionic-security InRelease [88.7 kB]
Hit:3 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:4  http://archive.ubuntu.com/ubuntu  bionic-updates InRelease
Get:5 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Hit:6 https://download.docker.com/linux/ubuntu  bionic InRelease
Ign:7 http://packages.linuxmint.com tessa InRelease
Hit:8 http://packages.linuxmint.com tessa Release
Fetched 163 kB in 1s (239 kB/s)
Reading package lists... Done
greys@xps:$ sudo apt-get install docker-ce docker-ce-cli
containerd.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
Recommended packages:
aufs-tools cgroupfs-mount | cgroup-lite git pigz
The following NEW packages will be installed:
containerd.io docker-ce docker-ce-cli
0 upgraded, 3 newly installed, 0 to remove and 243 not
upgraded.
Need to get 50.4 MB of archives.
After this operation, 242 MB of additional disk space will be
used.
Get:1 https://download.docker.com/linux/ubuntu bionic/stable
amd64 containerd.io amd64 1.2.2-3 [19.9 MB]
Get:2 https://download.docker.com/linux/ubuntu bionic/stable
amd64 docker-ce-cli amd64 5:18.09.2~3-0~ubuntu-bionic [13.2
MB]
Get:3 https://download.docker.com/linux/ubuntu bionic/stable
amd64 docker-ce amd64 5:18.09.2~3-0~ubuntu-bionic [17.4 MB]
Fetched 50.4 MB in 5s (10.8 MB/s)
Selecting previously unselected package containerd.io.
(Reading database ... 259022 files and directories currently
installed.)
Preparing to unpack .../containerd.io_1.2.2-3_amd64.deb ...
Unpacking containerd.io (1.2.2-3) ...
Selecting previously unselected package docker-ce-cli.
Preparing to unpack .../docker-ce-cli_5%3a18.09.2~3-0~ubuntu-
bionic_amd64.deb ...
Unpacking docker-ce-cli (5:18.09.2~3-0~ubuntu-bionic) ...
Selecting previously unselected package docker-ce.
Preparing to unpack .../docker-ce_5%3a18.09.2~3-0~ubuntu-
bionic_amd64.deb ...
Unpacking docker-ce (5:18.09.2~3-0~ubuntu-bionic) ...
Setting up containerd.io (1.2.2-3) ...
Created symlink /etc/systemd/system/multi-
user.target.wants/containerd.service →
/lib/systemd/system/containerd.service.
Processing triggers for ureadahead (0.100.0-20) ...
ureadahead will be reprofiled on next reboot
Processing triggers for systemd (237-3ubuntu10.9) ...
Listing Docker Containers

There shouldn’t be any:

greys@xps:/$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

Checking Docker Version

greys@xps:/$ docker --version
Docker version 18.09.2, build 6247962

Installing hello-world with Docker

greys@xps:/$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

That’s it!

See Also

- Docker
- Linux Mint
- Ubuntu
- List Docker Containers
- Restart All Stopped Docker Containers
Docker Inspect

As soon as you get familiar with listing Docker containers and starting containers or stopping containers, you’ll probably get curious about learning a little more. Docker has a great command for obtaining low level configuration details about containers: docker inspect.

Docker Inspect features

I find the following to be most useful when using docker inspect to look at one of the running containers:

- long form container ID and timestamp when it was created
- current status (really useful even if container is stopped – will show you why)
- Docker image info
- Filesystem binds and volume info, mounts
- environment variables – this is where a lot of usernames/passwords will be found – automation comes at a cost to security
- command line parameters passed into the container
- network configuration: IP address and gateway, secondary addresses for IPv4 and IPv6
Docker Inspect output

Here’s a fragment of the docker inspect output for a MariaDB container on one of my hosts:

```
root@centos:~ # docker inspect c74010b06784
[
{
  "Id": "c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c",
  "Created": "2019-02-04T12:41:34.958272492Z",
  "Path": "docker-entrypoint.sh",
  "Args": [
    "--character-set-server=utf8",
    "--collation-server=utf8_bin",
    "--transaction-isolation=READ-COMMITTED"
  ],
  "State": {
    "Status": "running",
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "Dead": false,
    "Pid": 21868,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2019-02-26T22:30:33.060964954Z",
    "FinishedAt": "2019-02-26T21:43:52.895816071Z"
  },
  "Image": "sha256:901583bfdf5a129ba68b033c989dab7f10d2a9235c3a2093ad16f9ac979ac9f9",
  "ResolvConfPath": "/var/lib/docker/containers/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c/resolv.conf",
  "HostnamePath": "/var/lib/docker/containers/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c/hostname",
  "HostsPath": "/var/lib/docker/containers/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c/hosts"
}
```

"/var/lib/docker/containers/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c/hosts",
"LogPath":
"/var/lib/docker/containers/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c/c74010b06784ee04976239fb6dc08062b6f5f9ea6ce3e67bd2c876365986910c-json.log",
"Name": "/db",
"RestartCount": 0,
"Driver": "devicemapper",
"MountLabel": "",
"ProcessLabel": "",
"AppArmorProfile": "",
"ExecIDs": null,
"HostConfig": {
"Binds": [
"/storage/docker/db/datadir:/var/lib/mysql",
"/storage/docker/db/conf.d:/etc/mysql/conf.d"
],
"ContainerIDFile": "",
"LogConfig": {
"Type": "json-file",
"Config": {}
},
"NetworkMode": "confluencenet",
"PortBindings": {},
"RestartPolicy": {
"Name": "no",
"MaximumRetryCount": 0
},
...
Restart Stopped Containers in Docker

Sometimes an issue on one of your servers may interrupt your Docker based development and stop all the containers that you haven’t fully configured to be auto-started just yet. In such cases, it will be useful for you to know how to find stopped containers and restart them all using a single command.

List Stopped Containers in Docker

Using the filtering functionality of the docker ps command, we can quickly get all the necessary information for the stopped containers:

```
root@xps:~# docker ps -a -f status=exited
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
014a746dbb9d wordpress "docker-entrypoint.s..." 21 hours ago Exited (0) 21 hours ago romantic_fermi
080cf6412ac4 hello-world "/hello" 3 days ago Exited (0) 3 days ago modest_mestorf
```

Since we want to restart of these containers, we’ll probably
need to pass their docker container IDs to another command, like docker start.

Hence the command above should be run with the -q parameter, which skips all the non-essential info and only returns the list of docker containers:

```
root@xps:~# docker ps -a -q -f status=exited
014a746dbb9d
080cf6412ac4
```

## Restart all the Stopped Containers in Docker

Now all we have left to do is pass the above command to the docker start, like shown below. One by one, all the container IDs will appear as Docker restarts them:

```
root@xps:~# docker start $(docker ps -a -q -f status=exited)
014a746dbb9d
080cf6412ac4
```

Sure enough, when we do docker ps now, we should see these containers:

```
root@xps:~# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
9e7115e34496 wordpress "docker-entrypoint.s..." 19 hours ago Up
19 hours 127.0.0.1:80->80/tcp, 127.0.0.1:443->443/tcp
wordpress
014a746dbb9d wordpress "docker-entrypoint.s..." 21 hours ago Up
2 seconds 80/tcp romantic_fermi
c397a72fbd58 mariadb:latest "docker-entrypoint.s..." 21 hours ago Up
21 hours 3306/tcp db
```

I can see the 014a746dbb9d container, but the other one is not running. Want to know why? It’s because this was a Hello, world Docker container – it’s not mean to stay running in background. Instead, it shows Hello, world and exits. It’s usually run like this:
root@xps:~# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

That's it for today. Enjoy!

See Also

- Docker
- Docker – Stop Containers
- Docker – List containers
- Migrate Docker container
- Remove Unused Volumes in Docker
- Inspect Docker containers
Remove Unused Volumes in Docker

You can list unused volumes using the filtering option of docker volume ls command. Once these are identified, it’s easy enough to remove such volumes altogether.

List dangling volumes in Docker

Relying on the dangling flag of a volume object, we can list all the dangling volumes as shown below:

```
root@s5:~ # docker volume ls -q -f "dangling=true"
7b8baf0d804862c1a5daf8b5770b7d21ce7051d429815412e15b5ded95cb1d36
15a2e1552ab9ac37d7ab4f65d168894c3ea7d798000e90d8ab6426f5fecf42f3
81c68768733ae6f6ec5918f42f9ba9c39f7e7181298a7c9a70
92cf29ef5e3d42ab77d6467aee7a201f2aa04d92f369ff6d809b84f217398565
```
Excellent! Now let’s remove these volumes.
Remove unused (dangling) volumes in Docker

We’ll use the docker volume ls output as the list of command line options for the docker volume rm command.

As always, we’ll get confirmation about each volume that’s just been removed:

```
root@s5:~ # docker volume rm $(docker volume ls -q -f "dangling=true")
7b8baf0d804862c1a5daf8b5770b7d21ce7051d429815412e15b5ded95cb1d36
15a2e1552ab9ac37d7ab4f65d168894c3ea7d798000e90d8ab6426f5fece42f3
81c68768733ae6fcee19d256e20859aa6772875448126c7655c8d455c0067fd
b3
83f9e08b9e23f4218189fe115b96c88ec59142f9b9a9c39f7e7181298a7c9a70
92cf29ef5e3d42ab77d6467aae7a201f2a04d92f369ff6d6b09b84f2173985
65
307f5debc78882b6b46695c8f7a66298e0f90b8b5b89833d6096775a1eed8
43
3013b332c97c9ff9e344ac48aa1d850075db83ab68d177e340c4d357834812
df
3910ca56fae2a73b546ebd9e8e665bea9770287f5a86a8c217b03e731843e9
b7
5577dfc4792afad0bca22cd73193bdf5148d0440e3a4bdec3d8cfee5ccc
0d
374764091f95d2a998045b81ad0bc860cddf8474500e4069df27b400c6c43
fe
a754d3eda4c754952ffdbd62d311d1ae7db80ff94721a2df56a5092c53bb2e
10
a4031051a268838724a1d7ca2cd5306c345074d2893556a7a16c1399c96820
85
afa4d8b0cfd547f27246156cd6337f9a38d856ee54e998b501c6e99b62bb19
0b
bf2c587c94688e358ebf622ce892a77f684242c0bf4d638e2e4937bab29b99
b5
c5d5c25dcfc4e001fccc7a4d72e2f9506512f9b2720ad6d1d10c85c47aa1da
```
And just to be sure, let's re-run the command that lists unused Docker volumes:

```
root@s5:~ # docker volume ls -q -f "dangling=true"
root@s5:~ #
```

As you can see, there’s nothing returned now — which means all the volumes were indeed removed.

**See Also**

- Official Docker website
- [Docker @ Unix Tutorial](#)
- [Migrate Docker container to another server](#)
- [Stop Docker Containers](#)
- [List Docker Containers](#)
Migrate Docker container to new server

There are many ways of migrating Docker containers to a different server, today I’ll show you one of the possible approaches.

**IMPORTANT:** it’s a beginner’s tutorial for copying basic Docker containers (no external dependencies like additional networks or storage volumes).

If you have filesystem volumes attached to your original Docker container, this procedure will not be enough. I’ll publish a more advanced tutorial soon – stay tuned.

This is a simple enough procedure. Steps 1, 2 and 3 should be done on the old server, Steps 4, 5 and 6 should be done on the new server. All you need is root access on both servers and a way to transfer images between the two servers (**scp** for instance).

**Step 1: Stop Docker container**

I’m hoping to transfer the database container called db (container id c745794419a9 below):
root@oldserver:/ # docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
1b8b1657736e datadog/agent:latest "/init" 9 months ago Up 26 hours (healthy) 8125/udp, 8126/tcp dd-agent
c745794419a9 mariadb:latest "docker-entrypoint sàng..." 9 months ago Up 29 minutes 3306/tcp db
32cd3e477546 nginx:latest "nginx -g 'daemon of..." 12 months ago Up 26 hours 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp nginx

Let’s stop the container:

root@oldserver:/ # docker stop db
db

...then make sure it’s down:

root@oldserver:/ # docker ps --all | grep c745794419a9
c745794419a9 mariadb:latest "docker-entrypoint sàng..." 9 months ago Exited (0) About an hour ago db

**Step 2. Commit Docker container to image**

root@oldserver:/ # docker commit c745794419a9
sha256:9d07849ed7c73f8fecd1e5e3e2aedc3592eea6b02f239fa6efba903f1a1ef835

**Step 3: Save Docker image to a file**

root@oldserver:/ # docker save 9d07849ed7c73f8fecd1e5e3e2aedc3592eea6b02f239fa6efba903f1a1ef835 > s5-db.tar

**Step 4: Transfer Docker image file**

**Step 5: Load Docker image from a**
On the new server, we docker load the image. Note how it is the same image ID:

```bash
root@newserver:/ # cat s5-db.tar | docker load
4bcdff70da2:  Loading layer [==================================================>]
129.3MB/129.3MB
ae12d30e1dfc:  Loading layer [==================================================>]
345.1kB/345.1kB
7a065b613dee:  Loading layer [==========================================>
3.178MB/3.178MB
cb2872ddbc2c:  Loading layer [==================================================>]
1.536kB/1.536kB
328a5e02ea3f:  Loading layer [==================================================>]
15.05MB/15.05MB
736f4a72442b:  Loading layer [==================================================>]
25.6kB/25.6kB
3fbb3db5b99e:  Loading layer [==================================================>]
5.12kB/5.12kB
fbf207c08d17:  Loading layer [==================================================>]
5.12kB/5.12kB
c61ded92b25c:  Loading layer [==================================================>]
257MB/257MB
74569dcf2238:  Loading layer [==================================================>]
8.704kB/8.704kB
b954e0840314:  Loading layer [==================================================>]
1.536kB/1.536kB
9b819b273348:  Loading layer [==================================================>]
```
Step 6: Start Docker container

Now let’s start a Docker container from this image:

```bash
root@newserver:/ # docker run -d --name db-new
9d07849ed7c73f8fecd1e5e3e2aedc3592eea6b02f239fa6efba903f1a1ef835
1ca6041d6e1e6c661234e24b16c0d23b0a302586f8628809020d5469e3acd405
```

As you can see, it’s running now:

```bash
root@newserver:/ # docker ps | grep db-new
1ca6041d6e1e 9d07849ed7c7 "docker-entrypoint..." 5 seconds ago
Up 3 seconds
```

See Also

- [Docker – Stop All Containers](#)
- [Docker – List Containers](#)
- [Docker](#)

---

Unix Tutorial Digest — February 4th, 2019
Thanks to the super busy January, this is actually the first Unix Tutorial digest of 2019! Lots of news and so many things to follow up and test now!
As always, Please get in touch if you want to suggest a useful link for the next digest.

Unix and Linux News

- Linux kernel 4.20
- Ubuntu Core 18.10 got released – sounds like a pretty cool idea for IoT things. I must try it on one of my Raspberry Pi systems, but it seems it won’t work on Raspberry Pi Model 1.
- I seem to have forgotten Red Hat Enterprise Linux 8 beta in my previous digest
- Bash 5.0 got released! Not just any software, but one of the most popular Unix shells!
Software News

- LetsEncrypt (my SSL provider of choice for all the self-hosted elements) reported 2018 progress and LetsEncrypt 2019 plans – 150M websites are using them now, impressive!
- In mid January, 4 issues with SCP in OpenSSH, Putty and WinSCP were discovered. Hope you were not affected!
- Were you affected by the DNS Flag Day on February 1st? Great initiative to highlight dependency on DNS and to steer users towards better implementations of it.
- Wine 4.0 got released, quickly followed up by Wine Staging 4.0.
- Kodi 18 platform got released, followed by LibreELEC 9.0.0 based on it

Interesting and Useful

- Scraping TripAdvisor: text mining and sentiment analysis – can’t believe how simple and readable the Python code is for a task of such complexity

Unix Tutorial articles

I made some New Years’ resolutions, one of them is a dramatically improved commitment to updating Unix Tutorial. As the result, I published more content in January 2019, than in the previous 9 years!

- Happy New Year 2019!
- ISO to USB in MacOS
- Test TCP connectivity with curl
- VirtualBox 6.0
- How To: Test Disk I/O with dd
- How To Use 7zip
- SSH port
- GitHub: Private Repositories are Free Now
- Docker – List Containers
- HW Virtualization
- Bash 5.0
- Unix Epoch
- Linux: List All Users
- How To: Mount ISO image in Linux
- pcs STONITH
- How To: Restart MySQL
- screenFetch in RHEL 8
- How To: Disable SElinux
- Docker: Stop All Containers
- How To: Confirm Current Kernel Boot Commands in Linux
- SElinux Status
- How To: Generate SSH key
- Unix Diff
- SElinux: Advanced sestatus usage
- List RHEL subscriptions
- YUM: list and install software groups
- tmux: basic configuration
- Show network errors with netstat

That’s it for today!

See Also

- Unix Tutorial Digest
- Unix commands
- Basic Unix commands
- Advanced Unix commands
Now and then, especially when working on a development environment, you need to stop multiple Docker containers. Quite often, you need to stop all of the currently running containers. I’m going to show you one of the possible ways.

Docker: Stop All Containers

You need to use a container name or container ID with the docker stop command.

For example, I have an nginx load balancer container:

```
root@s5:~ # docker ps -f name=nginx
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
32cd3e477546 nginx:latest "nginx -g 'daemon of…" 11 months ago Up About a minute 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp nginx
```

Based on this output, I can stop my nginx container like this:

```
root@s5:~ # docker stop nginx
nginx
```

... or like that:

```
root@s5:~ # docker stop 32cd3e477546
```
Docker: Stop Multiple Containers

Since I also have a MariaDB container named db, I might need to stop it together with nginx.

Here’s the info on the db container:

```
root@s5:~ # docker ps -f name=db
CONTAINER ID  IMAGE            COMMAND                  CREATED             STATUS              PORTS NAMES
510972d55d8c  mariadb:latest   "docker-entrypoint.s..." 9 months ago Up 4 seconds 3306/tcp db
```

If I ever decide to stop both nginx and db together, I can do it like this:

```
root@s5:~ # docker stop nginx db
nginx
db
```

Docker: Stop All Containers

As you can see from previous examples, `docker stop` simply takes a list of containers to stop. If there’s more than one container, just use space as a delimiter between container names or IDs.

This also allows us to use a clever shell expansion trick: you can some other command, and pass its output to the docker stop container.

For instance, this shows us the list of all the IDs for currently running Docker containers:

```
root@s5:~ # docker ps -q
510972d55d8c
1b8b1657736e
c745794419a9
32cd3e477546
```
What we can do now is pass the result of this command as the parameter for the docker stop command:

root@s5:~ # docker stop $(docker ps -q)
510972d55d8c
1b8b1657736e
c745794419a9
32cd3e477546

And just to check, running docker ps now won’t show any running containers:

root@s5:~ # docker ps -q

**IMPORTANT**: make sure you double-check what you’re doing! Specifically, run docker ps -q, compare it to docker ps, this kind of thing. Because once containers stopped you may not have an easy way to generate the list of same containers to restart.

In my case, I’m just specifying them manually as the parameters for docker start:

root@s5:~ # docker start 510972d55d8c 1b8b1657736e
c745794419a9 32cd3e477546
510972d55d8c
1b8b1657736e
c745794419a9
32cd3e477546

That’s it for today! Hope you enjoyed this quick how-to, let me know if you have any questions, **Docker** and whatnot!

**See Also**

- [Docker Software](#)
- [List containers in Docker](#)
- [Restart Stopped Docker containers](#)
- [Docker Inspect](#)
- [Remove Unused Docker volumes](#)
Docker – List Containers

If you’re just getting started with Docker containers, you may be a bit confused how there doesn’t seem to be a command called “list” to show the containers available on your system. There is indeed no such command, but listing functionality is certainly there.

List currently running Docker containers

You need the docker ps command – it lists containers in a readable table:

```
root@dcs:~ # docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
1b8b1657736e datadog/agent:latest "/init" 8 months ago Up 8 months (healthy) 8125/udp, 8126/tcp dd-agent
c745794419a9 mariadb:latest "docker-entrypoint.s..." 8 months ago Up 8 months 3306/tcp db
32cd3e477546 nginx:latest "nginx -g 'daemon of..." 11 months ago Up 4 months 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp nginx
```
Executed without any command line options, `docker ps` shows only the active containers – the ones running at this very moment.

**List all the Docker containers**

In case you experience some trouble with one of the containers, where the Docker container will start and immediately go offline, your `docker ps` won’t help – by the time you run it the container will disappear from the list.

This is where you need to use the `--all` command line option:

```
root@dcs:~ # docker ps --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f4b4ae616898 wordpress "docker-entrypoint.s..." 7 months ago Exited (0) 7 months ago competent_johnson
1b8b1657736e datadog/agent:latest "/init" 8 months ago Up 8 months (healthy) 8125/udp, 8126/tcp dd-agent
c745794419a9 mariadb:latest "docker-entrypoint.s..." 8 months ago Up 8 months 3306/tcp db
4c82fa3d5d1c mariadb:latest "docker-entrypoint.s..." 9 months ago Exited (1) 9 months ago mysql
78fd23e82bba confluence:latest "/sbin/tini -- /entr..." 11 months ago Exited (143) 10 months ago wiki_https
73c9ca67c77b confluence:latest "/sbin/tini -- /entr..." 11 months ago Exited (143) 11 months ago wiki
56728d0f1ab5 mariadb:latest "docker-entrypoint.s..." 11 months ago Exited (0) 10 months ago mariadb
32cd3e477546 nginx:latest "nginx -g 'daemon of..." 11 months ago Up 4 months 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp nginx
496b0d371a70 hello-world "/hello" 11 months ago Exited (0) 11 months ago stoic_brattain
```

List containers filtered by a specified criteria

You’ll soon realise that on a busy Docker host you probably need to apply some filter when listing containers. This functionality allows you to filter lists by many common Docker container properties.
For example, this is how we can list just the containers with a specific name:

```
root@dcs:~ # docker ps -f name=nginx
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
32cd3e477546 nginx:latest "nginx -g 'daemon of…" 11 months ago
Up 4 months 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp nginx
```

And this is how you can show just the Docker containers with “exited” status:

```
root@dcr:~ # docker ps -f status=exited
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f4b4ae616898 wordpress "docker-entrypoint.s…" 7 months ago
Exited (0) 7 months ago competent_johnson
4c82fa3d5d1c mariadb:latest "docker-entrypoint.s…" 9 months ago
Exited (1) 9 months ago mysql
78fd23e82bba confluence:latest "/sbin/tini -- /entr…" 11 months ago
Exited (143) 10 months ago wiki_https
73c9ca67c77b confluence:latest "/sbin/tini -- /entr…" 11 months ago
Exited (143) 11 months ago wiki
56728d0f1ab5 mariadb:latest "docker-entrypoint.s…" 11 months ago
Exited (0) 10 months ago mariadb
496b0d371a70 hello-world "/hello" 11 months ago
Exited (0) 11 months ago stoic_brattain
```

That’s it for today! Let me know if you’re using Docker and whether you need help with anything!

**See Also**

- [Official Docker page](#)
- [docker ps documentation](#)