

Docker Cheat Sheet

Want to improve this cheat sheet? See the [Contributing](#) section!

Table of Contents

- [Why Docker](#)
- [Prerequisites](#)
- [Installation](#)
- [Containers](#)
- [Images](#)
- [Networks](#)
- [Registry and Repository](#)
- [Dockerfile](#)
- [Layers](#)
- [Links](#)
- [Volumes](#)
- [Exposing Ports](#)
- [Best Practices](#)
- [Docker-Compose](#)
- [Security](#)
- [Tips](#)
- [Contributing](#)

Why Docker

"With Docker, developers can build any app in any language using any toolchain. "Dockerized" apps are completely portable and can run anywhere - colleagues' OS X and Windows laptops, QA servers running Ubuntu in the cloud, and production data center VMs running Red Hat.

Developers can get going quickly by starting with one of the 13,000+ apps available on Docker Hub. Docker manages and tracks changes and dependencies, making it easier for sysadmins to understand how the apps that developers build work. And with Docker Hub, developers can automate their build pipeline and share artifacts with collaborators through public or private repositories.

Docker helps developers build and ship higher-quality applications, faster." -- [What is Docker](#)

Prerequisites

I use [Oh My Zsh](#) with the [Docker plugin](#) for autocompletion of docker commands. YMMV.

Linux

The 3.10.x kernel is [the minimum requirement](#) for Docker.

MacOS

10.8 “Mountain Lion” or newer is required.

Windows 10

Hyper-V must be enabled in BIOS VT-D must also be enabled if available (Intel Processors)

Windows Server

Windows Server 2016 is the minimum version required to install docker and docker-compose. Limitations exist on this version, such as multiple virtual networks and linux containers. Windows Server 2019 and later are recommended.

Installation

Linux

Quick and easy install script provided by Docker:

```
curl -sSL https://get.docker.com/ | sh
```

If you're not willing to run a random shell script, please see the [installation](#) instructions for your distribution.

If you are a complete Docker newbie, you should follow the [series of tutorials](#) now.

macOS

Download and install [Docker Community Edition](#). if you have Homebrew-Cask, just type `brew cask install docker`. Or Download and install [Docker Toolbox](#). [Docker For Mac](#) is nice, but it's not quite as finished as the VirtualBox install. [See the comparison](#).

NOTE Docker Toolbox is legacy. You should to use Docker Community Edition, See [Docker Toolbox](#).

Once you've installed Docker Community Edition, click the docker icon in Launchpad. Then start up a container:

```
docker run hello-world
```

That's it, you have a running Docker container.

If you are a complete Docker newbie, you should probably follow the [series of tutorials](#) now.

Windows 10

Instructions to install Docker Desktop for Windows can be found [here](#)

Once insalled, open powershell as administrator

```
#Display the version of docker installed:  
docker version
```

```
##Pull, create, and run 'hello-world' all in one command:  
docker run hello-world
```

To continue with this cheat sheet, right click the Docker icon in the system tray, and go to settings. In order to mount volumes, the C:/ drive will need to be enabled in the settings so that information can be passed into the containers (later described in this article).

To switch between Windows containers and Linux containers, right click the icon in the system tray and click the button to switch container operating system. Doing this will stop the current containers that are running, and make them inaccessible until the container OS is switched back.

Additionally, if you have WSL or WSL2 installed on your desktop, you might want to install the Linux Kernel for Windows. Instructions can be found [here](#). This requires the Windows Subsystem for Linux feature. This will allow for containers to be accessed by WSL operating systems, as well as the efficiency gain from running WSL operating systems in docker. It is also preferred to use [Windows terminal](#) for this.

Windows Server 2016 / 2019

Follow Microsoft's instructions that can be found [here](#)

If using the latest edge version of 2019, be prepared to only work in powershell, as it is only a servercore image (no desktop interface). When starting this machine, it will login and go straight to a powershell window. It is recommended to install text editors and other tools using [Chocolatey](#).

After installing, these commands will work:

```
#Display the version of docker installed:
docker version
```

```
##Pull, create, and run 'hello-world' all in one command:
docker run hello-world
```

Windows Server 2016 is not able to run linux images.

Windows Server Build 2004 is capable of running both linux and windows containers simultaneously through Hyper-V isolation. When running containers, use the `--isolation=hyperv` command, which will isolate the container using a separate kernel instance.

Check Version

It is very important that you always know the current version of Docker you are currently running on at any point in time. This is very helpful because you get to know what features are compatible with what you have running. This is also important because you know what containers to run from the docker store when you are trying to get template containers. That said let see how to know which version of docker we have running currently.

- [docker version](#) shows which version of docker you have running.

Get the server version:

```
$ docker version --format '{{.Server.Version}}'

1.8.0
```

You can also dump raw JSON data:

```
$ docker version --format '{{json .}}'
```

```
{"Client":  
{"Version":"1.8.0","ApiVersion":"1.20","GitCommit":"f5bae0a","GoVersion":"go1.4.  
2","Os":"linux","Arch":"am"}
```

Containers

[Your basic isolated Docker process](#). Containers are to Virtual Machines as threads are to processes. Or you can think of them as chroots on steroids.

Lifecycle

- [docker create](#) creates a container but does not start it.
- [docker rename](#) allows the container to be renamed.
- [docker run](#) creates and starts a container in one operation.
- [docker rm](#) deletes a container.
- [docker update](#) updates a container's resource limits.

Normally if you run a container without options it will start and stop immediately, if you want keep it running you can use the command, `docker run -td container_id` this will use the option `-t` that will allocate a pseudo-TTY session and `-d` that will detach automatically the container (run container in background and print container ID).

If you want a transient container, `docker run --rm` will remove the container after it stops.

If you want to map a directory on the host to a docker container, `docker run -v $HOSTDIR:$DOCKERDIR`. Also see [Volumes](#).

If you want to remove also the volumes associated with the container, the deletion of the container must include the `-v` switch like in `docker rm -v`.

There's also a [logging driver](#) available for individual containers in docker 1.10. To run docker with a custom log driver (i.e., to syslog), use `docker run --log-driver=syslog`.

Another useful option is `docker run --name yourname docker_image` because when you specify the `--name` inside the run command this will allow you to start and stop a container by calling it with the name the you specified when you created it.

Starting and Stopping

- [docker start](#) starts a container so it is running.
- [docker stop](#) stops a running container.
- [docker restart](#) stops and starts a container.
- [docker pause](#) pauses a running container, "freezing" it in place.
- [docker unpause](#) will unpause a running container.
- [docker wait](#) blocks until running container stops.
- [docker kill](#) sends a SIGKILL to a running container.
- [docker attach](#) will connect to a running container.

If you want to detach from a running container, use `Ctrl + p`, `Ctrl + q`. If you want to integrate a container with a [host process manager](#), start the daemon with `-r=false` then use `docker start -a`.

If you want to expose container ports through the host, see the [exposing ports](#) section.

Restart policies on crashed docker instances are [covered here](#).

CPU Constraints

You can limit CPU, either using a percentage of all CPUs, or by using specific cores.

For example, you can tell the [cpu-shares](#) setting. The setting is a bit strange -- 1024 means 100% of the CPU, so if you want the container to take 50% of all CPU cores, you should specify 512. See https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/#_cpu for more:

```
docker run -it -c 512 agileek/cpuset-test
```

You can also only use some CPU cores using [cpuset-cpus](#). See <https://agileek.github.io/docker/2014/08/06/docker-cpuset/> for details and some nice videos:

```
docker run -it --cpuset-cpus=0,4,6 agileek/cpuset-test
```

Note that Docker can still **see** all of the CPUs inside the container -- it just isn't using all of them. See <https://github.com/docker/docker/issues/20770> for more details.

Memory Constraints

You can also set [memory constraints](#) on Docker:

```
docker run -it -m 300M ubuntu:14.04 /bin/bash
```

Capabilities

Linux capabilities can be set by using `cap-add` and `cap-drop`. See <https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities> for details. This should be used for greater security.

To mount a FUSE based filesystem, you need to combine both `--cap-add` and `--device`:

```
docker run --rm -it --cap-add SYS_ADMIN --device /dev/fuse sshfs
```

Give access to a single device:

```
docker run -it --device=/dev/ttyUSB0 debian bash
```

Give access to all devices:

```
docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash
```

More info about privileged containers [here](#).

Info

- [docker ps](#) shows running containers.

- [`docker logs`](#) gets logs from container. (You can use a custom log driver, but logs is only available for `json-file` and `journald` in 1.10).
- [`docker inspect`](#) looks at all the info on a container (including IP address).
- [`docker events`](#) gets events from container.
- [`docker port`](#) shows public facing port of container.
- [`docker top`](#) shows running processes in container.
- [`docker stats`](#) shows containers' resource usage statistics.
- [`docker diff`](#) shows changed files in the container's FS.

`docker ps -a` shows running and stopped containers.

`docker stats --all` shows a list of all containers, default shows just running.

Import / Export

- [`docker cp`](#) copies files or folders between a container and the local filesystem.
- [`docker export`](#) turns container filesystem into tarball archive stream to STDOUT.

Executing Commands

- [`docker exec`](#) to execute a command in container.

To enter a running container, attach a new shell process to a running container called `foo`, use:
`docker exec -it foo /bin/bash`.

Images

Images are just [templates for docker containers](#).

Lifecycle

- [`docker images`](#) shows all images.
- [`docker import`](#) creates an image from a tarball.
- [`docker build`](#) creates image from Dockerfile.
- [`docker commit`](#) creates image from a container, pausing it temporarily if it is running.
- [`docker rmi`](#) removes an image.
- [`docker load`](#) loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- [`docker save`](#) saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

Info

- [`docker history`](#) shows history of image.
- [`docker tag`](#) tags an image to a name (local or registry).

Cleaning up

While you can use the `docker rmi` command to remove specific images, there's a tool called [docker-gc](#) that will safely clean up images that are no longer used by any containers. As of docker 1.13, `docker image prune` is also available for removing unused images. See [Prune](#).

Load/Save image

Load an image from file:

```
docker load < my_image.tar.gz
```

Save an existing image:

```
docker save my_image:my_tag | gzip > my_image.tar.gz
```

Import/Export container

Import a container as an image from file:

```
cat my_container.tar.gz | docker import - my_image:my_tag
```

Export an existing container:

```
docker export my_container | gzip > my_container.tar.gz
```

Difference between loading a saved image and importing an exported container as an image

Loading an image using the `load` command creates a new image including its history.

Importing a container as an image using the `import` command creates a new image excluding the history which results in a smaller image size compared to loading an image.

Networks

Docker has a [networks](#) feature. Docker automatically creates 3 network interfaces when you install it (bridge, host none). A new container is launched into the bridge network by default. To enable communication between multiple containers, you can create a new network and launch containers in it. This enables containers to communicate to each other while being isolated from containers that are not connected to the network. Furthermore, it allows to map container names to their IP addresses. See [working with networks](#) for more details.

Lifecycle

- [docker network create](#) NAME Create a new network (default type: bridge).
- [docker network rm](#) NAME Remove one or more networks by name or identifier. No containers can be connected to the network when deleting it.

Info

- [docker network ls](#) List networks

- [docker network inspect](#) NAME Display detailed information on one or more networks.

Connection

- [docker network connect](#) NETWORK CONTAINER Connect a container to a network
- [docker network disconnect](#) NETWORK CONTAINER Disconnect a container from a network

You can specify a [specific IP address for a container](#):

```
# create a new bridge network with your subnet and gateway for your ip block
docker network create --subnet 203.0.113.0/24 --gateway 203.0.113.254 iptastic

# run a nginx container with a specific ip in that block
$ docker run --rm -it --net iptastic --ip 203.0.113.2 nginx

# curl the ip from any other place (assuming this is a public ip block duh)
$ curl 203.0.113.2
```

Registry & Repository

A repository is a *hosted* collection of tagged images that together create the file system for a container.

A registry is a *host* -- a server that stores repositories and provides an HTTP API for [managing the uploading and downloading of repositories](#).

Docker.com hosts its own [index](#) to a central registry which contains a large number of repositories. Having said that, the central docker registry [does not do a good job of verifying images](#) and should be avoided if you're worried about security.

- [docker login](#) to login to a registry.
- [docker logout](#) to logout from a registry.
- [docker search](#) searches registry for image.
- [docker pull](#) pulls an image from registry to local machine.
- [docker push](#) pushes an image to the registry from local machine.

Run local registry

You can run a local registry by using the [docker distribution](#) project and looking at the [local deploy](#) instructions.

Also see the [mailing list](#).

Dockerfile

[The configuration file](#). Sets up a Docker container when you run `docker build` on it. Vastly preferable to `docker commit`.

Here are some common text editors and their syntax highlighting modules you could use to create Dockerfiles:

- If you use [jEdit](#), I've put up a syntax highlighting module for [Dockerfile](#) you can use.
- [Sublime Text 2](#)
- [Atom](#)
- [Vim](#)
- [Emacs](#)
- [TextMate](#)
- [VS Code](#)
- Also see [Docker meets the IDE](#)

Instructions

- [.dockerignore](#)
- [FROM](#) Sets the Base Image for subsequent instructions.
- [MAINTAINER \(deprecated - use LABEL instead\)](#) Set the Author field of the generated images.
- [RUN](#) execute any commands in a new layer on top of the current image and commit the results.
- [CMD](#) provide defaults for an executing container.
- [EXPOSE](#) informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.
- [ENV](#) sets environment variable.
- [ADD](#) copies new files, directories or remote file to container. Invalidates caches. Avoid ADD and use COPY instead.
- [COPY](#) copies new files or directories to container. By default this copies as root regardless of the USER/WORKDIR settings. Use `--chown=<user>:<group>` to give ownership to another user/group. (Same for ADD.)
- [ENTRYPOINT](#) configures a container that will run as an executable.
- [VOLUME](#) creates a mount point for externally mounted volumes or other containers.
- [USER](#) sets the user name for following RUN / CMD / ENTRYPOINT commands.
- [WORKDIR](#) sets the working directory.
- [ARG](#) defines a build-time variable.
- [ONBUILD](#) adds a trigger instruction when the image is used as the base for another build.
- [STOPSIGNAL](#) sets the system call signal that will be sent to the container to exit.
- [LABEL](#) apply key/value metadata to your images, containers, or daemons.
- [SHELL](#) override default shell is used by docker to run commands.
- [HEALTHCHECK](#) tells docker how to test a container to check that it is still working.

Tutorial

- [Flux7's Dockerfile Tutorial](#)

Examples

- [Examples](#)
- [Best practices for writing Dockerfiles](#)
- [Michael Crosby](#) has some more [Dockerfiles best practices / take 2](#).
- [Building Good Docker Images / Building Better Docker Images](#)

- [Managing Container Configuration with Metadata](#)
- [How to write excellent Dockerfiles](#)

Layers

The versioned filesystem in Docker is based on layers. They're like [git commits or changesets for filesystems](#).

Links

Links are how Docker containers talk to each other [through TCP/IP ports](#). [Atlassian](#) show worked examples. You can also resolve [links by hostname](#).

This has been deprecated to some extent by [user-defined networks](#).

NOTE: If you want containers to ONLY communicate with each other through links, start the docker daemon with `-icc=false` to disable inter process communication.

If you have a container with the name CONTAINER (specified by `docker run --name CONTAINER`) and in the Dockerfile, it has an exposed port:

```
EXPOSE 1337
```

Then if we create another container called LINKED like so:

```
docker run -d --link CONTAINER:ALIAS --name LINKED user/wordpress
```

Then the exposed ports and aliases of CONTAINER will show up in LINKED with the following environment variables:

```
$ALIAS_PORT_1337_TCP_PORT  
$ALIAS_PORT_1337_TCP_ADDR
```

And you can connect to it that way.

To delete links, use `docker rm --link`.

Generally, linking between docker services is a subset of "service discovery", a big problem if you're planning to use Docker at scale in production. Please read [The Docker Ecosystem: Service Discovery and Distributed Configuration Stores](#) for more info.

Volumes

Docker volumes are [free-floating filesystems](#). They don't have to be connected to a particular container. You can use volumes mounted from [data-only containers](#) for portability. As of Docker 1.9.0, Docker has named volumes which replace data-only containers. Consider using named volumes to implement it rather than data containers.

Lifecycle

- [docker volume create](#)
- [docker volume rm](#)

Info

- [docker volume ls](#)
- [docker volume inspect](#)

Volumes are useful in situations where you can't use links (which are TCP/IP only). For instance, if you need to have two docker instances communicate by leaving stuff on the filesystem.

You can mount them in several docker containers at once, using `docker run --volumes-from`.

Because volumes are isolated filesystems, they are often used to store state from computations between transient containers. That is, you can have a stateless and transient container run from a recipe, blow it away, and then have a second instance of the transient container pick up from where the last one left off.

See [advanced volumes](#) for more details. [Container42](#) is also helpful.

You can [map MacOS host directories as docker volumes](#):

```
docker run -v /Users/wsargent/myapp/src:/src
```

You can use remote NFS volumes if you're [feeling brave](#).

You may also consider running data-only containers as described [here](#) to provide some data portability.

Be aware that you can [mount files as volumes](#).

Exposing ports

Exposing incoming ports through the host container is [fiddly but doable](#).

This is done by mapping the container port to the host port (only using localhost interface) using `-p`:

```
docker run -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT --name CONTAINER -t someimage
```

You can tell Docker that the container listens on the specified network ports at runtime by using [EXPOSE](#):

```
EXPOSE <CONTAINERPORT>
```

Note that EXPOSE does not expose the port itself -- only `-p` will do that. To expose the container's port on your localhost's port:

```
iptables -t nat -A DOCKER -p tcp --dport <LOCALHOSTPORT> -j DNAT --to-destination <CONTAINERIP>:<PORT>
```

If you're running Docker in Virtualbox, you then need to forward the port there as well, using [forwarded port](#). Define a range of ports in your Vagrantfile like this so you can dynamically map them:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|  
  ...
```

```
(49000..49900).each do |port|
  config.vm.network :forwarded_port, :host => port, :guest => port
end

...
end
```

If you forget what you mapped the port to on the host container, use `docker port` to show it:

```
docker port CONTAINER $CONTAINERPORT
```

Best Practices

This is where general Docker best practices and war stories go:

- [The Rabbit Hole of Using Docker in Automated Tests](#)
- [Bridget Kromhout](#) has a useful blog post on [running Docker in production](#) at Dramafever.
- There's also a best practices [blog post](#) from Lyst.
- [Building a Development Environment With Docker](#)
- [Discourse in a Docker Container](#)

Docker-Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the [list of features](#).

By using the following command you can start up your application:

```
docker-compose -f <docker-compose-file> up
```

You can also run docker-compose in detached mode using `-d` flag, then you can stop it whenever needed by the following command:

```
docker-compose stop
```

You can bring everything down, removing the containers entirely, with the down command. Pass `--volumes` to also remove the data volume.

Security

This is where security tips about Docker go. The Docker [security](#) page goes into more detail.

First things first: Docker runs as root. If you are in the `docker` group, you effectively [have root access](#). If you expose the docker unix socket to a container, you are giving the container [root access to the host](#).

Docker should not be your only defense. You should secure and harden it.

For an understanding of what containers leave exposed, you should read [Understanding and Hardening Linux Containers](#) by [Aaron Grattafiori](#). This is a complete and comprehensive guide to the issues involved with containers, with a plethora of links and footnotes leading on to yet more

useful content. The security tips following are useful if you've already hardened containers in the past, but are not a substitute for understanding.

Security Tips

For greatest security, you want to run Docker inside a virtual machine. This is straight from the Docker Security Team Lead -- [slides](#) / [notes](#). Then, run with AppArmor / seccomp / SELinux / grsec etc to [limit the container permissions](#). See the [Docker 1.10 security features](#) for more details.

Docker image ids are [sensitive information](#) and should not be exposed to the outside world. Treat them like passwords.

See the [Docker Security Cheat Sheet](#) by [Thomas Sjögren](#): some good stuff about container hardening in there.

Check out the [docker bench security script](#), download the [white papers](#).

Snyk's [10 Docker Image Security Best Practices cheat sheet](#)

You should start off by using a kernel with unstable patches for grsecurity / pax compiled in, such as [Alpine Linux](#). If you are using grsecurity in production, you should spring for [commercial support](#) for the [stable patches](#), same as you would do for RedHat. It's \$200 a month, which is nothing to your devops budget.

Since docker 1.11 you can easily limit the number of active processes running inside a container to prevent fork bombs. This requires a linux kernel ≥ 4.3 with CGROUP_PIDS=y to be in the kernel configuration.

```
docker run --pids-limit=64
```

Also available since docker 1.11 is the ability to prevent processes from gaining new privileges. This feature have been in the linux kernel since version 3.5. You can read more about it in [this](#) blog post.

```
docker run --security-opt=no-new-privileges
```

From the [Docker Security Cheat Sheet](#) (it's in PDF which makes it hard to use, so copying below) by [Container Solutions](#):

Turn off interprocess communication with:

```
docker -d --icc=false --iptables
```

Set the container to be read-only:

```
docker run --read-only
```

Verify images with a hashsum:

```
docker pull debian@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Set volumes to be read only:

```
docker run -v $(pwd)/secrets:/secrets:ro debian
```

Define and run a user in your Dockerfile so you don't run as root inside the container:

```
RUN groupadd -r user && useradd -r -g user user
USER user
```

User Namespaces

There's also work on [user namespaces](#) -- it is in 1.10 but is not enabled by default.

To enable user namespaces ("remap the users") in Ubuntu 15.10, [follow the blog example](#).

Security Videos

- [Using Docker Safely](#)
- [Securing your applications using Docker](#)
- [Container security: Do containers actually contain?](#)
- [Linux Containers: Future or Fantasy?](#)

Security Roadmap

The Docker roadmap talks about [seccomp support](#). There is an AppArmor policy generator called [bane](#), and they're working on [security profiles](#).

Tips

Sources:

- [15 Docker Tips in 5 minutes](#)
- [CodeFresh Everyday Hacks Docker](#)

Prune

The new [Data Management Commands](#) have landed as of Docker 1.13:

- `docker system prune`
- `docker volume prune`
- `docker network prune`
- `docker container prune`
- `docker image prune`

df

`docker system df` presents a summary of the space currently used by different docker objects.

Heredoc Docker Container

```
docker build -t htop - << EOF
FROM alpine
RUN apk --no-cache add htop
EOF
```

Last Ids

```
alias dl='docker ps -l -q'
docker run ubuntu echo hello world
docker commit $(dl) helloworld
```

Commit with command (needs Dockerfile)

```
docker commit -run='{"Cmd":["postgres", "-too -many -opts"]}' $(dl) postgres
```

Get IP address

```
docker inspect $(dl) | grep -wm1 IPAddress | cut -d '"' -f 4
```

or with [jq](#) installed:

```
docker inspect $(dl) | jq -r '.[0].NetworkSettings.IPAddress'
```

or using a [go template](#):

```
docker inspect -f '{{ .NetworkSettings.IPAddress }}' <container_name>
```

or when building an image from Dockerfile, when you want to pass in a build argument:

```
DOCKER_HOST_IP=`ifconfig | grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}" | grep -v  
127.0.0.1 | awk '{ print $2 }' | cut -f2 -d: | head -n1`  
echo DOCKER_HOST_IP = $DOCKER_HOST_IP  
docker build \  
  --build-arg ARTIFACTORY_ADDRESS=$DOCKER_HOST_IP  
  -t sometag \  
  some-directory/
```

Get port mapping

```
docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} ->  
{{(index $conf 0).HostPort}} {{end}}' <containername>
```

Find containers by regular expression

```
for i in $(docker ps -a | grep "REGEXP_PATTERN" | cut -f1 -d" "); do echo $i;  
done
```

Get Environment Settings

```
docker run --rm ubuntu env
```

Kill running containers

```
docker kill $(docker ps -q)
```

Delete all containers (force!! running or stopped containers)

```
docker rm -f $(docker ps -qa)
```

Delete old containers

```
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm
```

Delete stopped containers

```
docker rm -v $(docker ps -a -q -f status=exited)
```

Delete containers after stopping

```
docker stop $(docker ps -aq) && docker rm -v $(docker ps -aq)
```

Delete dangling images

```
docker rmi $(docker images -q -f dangling=true)
```

Delete all images

```
docker rmi $(docker images -q)
```

Delete dangling volumes

As of Docker 1.9:

```
docker volume rm $(docker volume ls -q -f dangling=true)
```

In 1.9.0, the filter `dangling=false` does *not* work - it is ignored and will list all volumes.

Show image dependencies

```
docker images -viz | dot -Tpng -o docker.png
```

Slimming down Docker containers

- Cleaning APT in a RUN layer

This should be done in the same layer as other apt commands. Otherwise, the previous layers still persist the original information and your images will still be fat.

```
RUN {apt commands} \  
  && apt-get clean \  
  && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

- Flatten an image

```
ID=$(docker run -d image-name /bin/bash)  
docker export $ID | docker import - flat-image-name
```

- For backup

```
ID=$(docker run -d image-name /bin/bash)  
(docker export $ID | gzip -c > image.tgz)  
gzip -dc image.tgz | docker import - flat-image-name
```

Monitor system resource utilization for running containers

To check the CPU, memory, and network I/O usage of a single container, you can use:

```
docker stats <container>
```

For all containers listed by id:

```
docker stats $(docker ps -q)
```

For all containers listed by name:


```
docker stats $(docker ps --format '{{.Names}}')
```

For all containers listed by image:

```
docker ps -a -f ancestor=ubuntu
```

Remove all untagged images:

```
docker rmi $(docker images | grep "^" | awk '{split($0,a," "); print a[3]}')
```

Remove container by a regular expression:

```
docker ps -a | grep wildfly | awk '{print $1}' | xargs docker rm -f
```

Remove all exited containers:

```
docker rm -f $(docker ps -a | grep Exit | awk '{ print $1 }')
```

Volumes can be files

Be aware that you can mount files as volumes. For example you can inject a configuration file like this:

```
# copy file from container
```

```
docker run --rm httpd cat /usr/local/apache2/conf/httpd.conf > httpd.conf
```

```
# edit file
```

```
vim httpd.conf
```

```
# start container with modified configuration
```

```
docker run --rm -it -v "$PWD/httpd.conf:/usr/local/apache2/conf/httpd.conf:ro" -p "80:80" httpd
```

<https://github.com/wsargent/docker-cheat-sheet>

docker build -t friendlyname . # Create image using this directory's Dockerfile
docker run -p 4000:80 friendlyname # Run "friendlyname" mapping port 4000 to 80
docker run -d -p 4000:80 friendlyname # Same thing, but in detached mode
docker exec -it [container-id] bash # Enter a running container
docker ps # See a list of all running containers
docker stop <hash> # Gracefully stop the specified container
docker ps -a # See a list of all containers, even the ones not running
docker kill <hash> # Force shutdown of the specified container
docker rm <hash> # Remove the specified container from this machine
docker rm \$(docker ps -a -q) # Remove all containers from this machine
docker images -a # Show all images on this machine
docker rmi <imagename> # Remove the specified image from this machine
docker rmi \$(docker images -q) # Remove all images from this machine
docker login # Log in this CLI session using your Docker credentials
docker tag <image> username/repository:tag # Tag <image> for upload to registry
docker push username/repository:tag # Upload tagged image to registry
docker run username/repository:tag # Run image from a registry

docker-compose up
docker-compose up -d
docker-compose down
docker-compose logs

docker stack ls # List all running applications on this Docker host
docker stack deploy -c <composefile> <appname> # Run the specified Compose file
docker stack services <appname> # List the services associated with an app
docker stack ps <appname> # List the running containers associated with an app
docker stack rm <appname> # Tear down an application

docker-machine create --driver virtualbox myvm1 # Create a VM (Mac, Win7, Linux)
docker-machine create -d hyperv --hyperv-virtual-switch "myswitch" myvm1 # Win10
docker-machine env myvm1 # View basic information about your node
docker-machine ssh myvm1 "docker node ls" # List the nodes in your swarm
docker-machine ssh myvm1 "docker node inspect <node ID>" # Inspect a node
docker-machine ssh myvm1 "docker swarm join-token -q worker" # View join token
docker-machine ssh myvm1 # Open an SSH session with the VM; type "exit" to end
docker-machine ssh myvm2 "docker swarm leave" # Make the worker leave the swarm
docker-machine ssh myvm1 "docker swarm leave -f" # Make master leave, kill swarm
docker-machine start myvm1 # Start a VM that is currently not running
docker-machine stop \$(docker-machine ls -q) # Stop all running VMs
docker-machine rm \$(docker-machine ls -q) # Delete all VMs and their disk images
docker-machine scp docker-compose.yml myvm1:~ # Copy file to node's home dir
docker-machine ssh myvm1 "docker stack deploy -c <file> <app>" # Deploy an app

<https://gist.github.com/LeCoupa/f975e330551508ffd598a0d4bc4bed38>

Table of Contents

- [Installation](#)
- [Docker Registries & Repositories](#)
- [Running Containers](#)
- [Starting & Stopping Containers](#)
- [Getting Information about Containers](#)
- [Networking](#)
- [Security](#)
- [Cleaning Docker](#)
- [Docker Swarm](#)
- [Notes](#)

The Ultimate Docker Cheat Sheet

Installation

Linux

For more information, see [here](#)

```
curl -sSL https://get.docker.com/ | sh
```

Mac

For more information, see [here](#)

Use this link to download the dmg.

```
https://download.docker.com/mac/stable/Docker.dmg
```

Open the downloaded file and follow the installation instructions.

Windows

For more information, see [here](#)

Use the msi installer:

```
https://download.docker.com/win/stable/InstallDocker.msi
```

Open the downloaded file and follow the installation instructions.

Docker Registries & Repositories

Login to a Registry

```
docker login
```

```
docker login localhost:8080
```

Logout from a Registry.

```
docker logout
```

```
docker logout localhost:8080
```

Searching an Image

```
docker search nginx
```

```
docker search --filter stars=3 --no-trunc nginx
```

Pulling an Image

```
docker image pull nginx
```

```
docker image pull eon01/nginx localhost:5000/myadmin/nginx
```

Pushing an Image

```
docker image push eon01/nginx
```

```
docker image push eon01/nginx localhost:5000/myadmin/nginx
```

Running Containers

Create and Run a Simple Container

-Start an [ubuntu:latest](#) image

- Bind the port 80 from the **CONTAINER** to port 3000 on the **HOST**
- Mount the current directory to /data on the CONTAINER
- Note: on **windows** you have to change -v \${PWD}:/data to -v "C:\Data":/data

```
docker container run --name infinite -it -p 3000:80 -v ${PWD}:/data  
ubuntu:latest
```

Creating a Container

```
docker container create -t -i eon01/infinite --name infinite
```

Running a Container

```
docker container run -it --name infinite -d eon01/infinite
```

Renaming a Container

```
docker container rename infinite infinity
```

Removing a Container

```
docker container rm infinite
```

A container can be removed only after stopping it using `docker stop` command. To avoid this, add the `-rm` flag while running the container.

Updating a Container

```
docker container update --cpu-shares 512 -m 300M infinite
```

Running a command within a running container

```
docker exec -it infinite sh
```

In the example above, `bash` can replace `sh` as an alternative (if the above is giving an error).

Starting & Stopping Containers

Starting

```
docker container start nginx
```

Stopping

```
docker container stop nginx
```

Restarting

```
docker container restart nginx
```

Pausing

```
docker container pause nginx
```

Unpausing

`docker container unpause nginx`

Blocking a Container

`docker container wait nginx`

Sending a SIGKILL

`docker container kill nginx`

Sending another signal

`docker container kill -s HUP nginx`

Connecting to an Existing Container

`docker container attach nginx`

Getting Information about Containers

From Running Containers

Shortest way:

`docker ps`

Alternative:

`docker container ls`

From All containers.

`docker ps -a`

`docker container ls -a`

Container Logs

`docker logs infinite`

'tail -f' Containers' Logs

`docker container logs infinite -f`

Inspecting Containers

```
docker container inspect infinite
```

```
docker container inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)
```

Containers Events

```
docker system events infinite
```

Public Ports

```
docker container port infinite
```

Running Processes

```
docker container top infinite
```

Container Resource Usage

```
docker container stats infinite
```

Inspecting changes to files or directories on a container's filesystem

```
docker container diff infinite
```

Managing Images

Listing Images

```
docker image ls
```

Building Images

From a Dockerfile in the Current Directory

```
docker build .
```

From a Remote GIT Repository

```
docker build github.com/creack/docker-firefox
```

Instead of Specifying a Context, You Can Pass a Single Dockerfile in the URL or Pipe the File in via STDIN

```
docker build - < Dockerfile
```

```
docker build - < context.tar.gz
```

Building and Tagging

```
docker build -t eon/infinite .
```

Building a Dockerfile while Specifying the Build Context

```
docker build -f myOtherDockerfile .
```

Building from a Remote Dockerfile URI

```
curl example.com/remote/Dockerfile | docker build -f - .
```

Removing an Image

```
docker image rm nginx
```

Loading a Tarred Repository from a File or the Standard Input Stream

```
docker image load < ubuntu.tar.gz
```

```
docker image load --input ubuntu.tar
```

Saving an Image to a Tar Archive

```
docker image save busybox > ubuntu.tar
```

Showing the History of an Image

```
docker image history
```

Creating an Image From a Container

```
docker container commit nginx
```

Tagging an Image

```
docker image tag nginx eon01/nginx
```

Pushing an Image

```
docker image push eon01/nginx
```


Networking

Creating Networks

Creating an Overlay Network

```
docker network create -d overlay MyOverlayNetwork
```

Creating a Bridge Network

```
docker network create -d bridge MyBridgeNetwork
```

Creating a Customized Overlay Network

```
docker network create -d overlay \  
  --subnet=192.168.0.0/16 \  
  --subnet=192.170.0.0/16 \  
  --gateway=192.168.0.100 \  
  --gateway=192.170.0.100 \  
  --ip-range=192.168.1.0/24 \  
  --aux-address="my-router=192.168.1.5" --aux-address="my-switch=192.168.1.6" \  
  --aux-address="my-printer=192.170.1.5" --aux-address="my-nas=192.170.1.6" \  
  MyOverlayNetwork
```

Removing a Network

```
docker network rm MyOverlayNetwork
```

Listing Networks

```
docker network ls
```

Getting Information About a Network

```
docker network inspect MyOverlayNetwork
```

Connecting a Running Container to a Network

```
docker network connect MyOverlayNetwork nginx
```

Connecting a Container to a Network When it Starts

```
docker container run -it -d --network=MyOverlayNetwork nginx
```

Disconnecting a Container from a Network

```
docker network disconnect MyOverlayNetwork nginx
```

Exposing Ports

Using Dockerfile, you can expose a port on the container using:

```
EXPOSE <port_number>
```

You can also map the container port to a host port using:

```
docker run -p $HOST_PORT:$CONTAINER_PORT --name <container_name> -t <image>
```

e.g.

```
docker run -p $HOST_PORT:$CONTAINER_PORT --name infinite -t infinite
```

Security

Guidelines for building secure Docker images

1. Prefer minimal base images
2. Dedicated user on the image as the least privileged user
3. Sign and verify images to mitigate MITM attacks
4. Find, fix and monitor for open source vulnerabilities
5. Don't leak sensitive information to docker images
6. Use fixed tags for immutability
7. Use COPY instead of ADD
8. Use labels for metadata
9. Use multi-stage builds for small secure images
10. Use a linter

You can find more information on Snyk's [10 Docker Image Security Best Practices](#) blog post.

Cleaning Docker

Removing a Running Container

```
docker container rm nginx
```

Removing a Container and its Volume

```
docker container rm -v nginx
```

Removing all Exited Containers

```
docker container rm $(docker container ls -a -f status=exited -q)
```

Removing All Stopped Containers

```
docker container rm $(docker container ls -a -q)
```

Removing a Docker Image

```
docker image rm nginx
```

Removing Dangling Images

```
docker image rm $(docker image ls -f dangling=true -q)
```

Removing all Images

```
docker image rm $(docker image ls -a -q)
```

Removing all Untagged Images

```
docker image rm -f $(docker image ls | grep "^<none>" | awk "{print $3}")
```

Stopping & Removing all Containers

```
docker container stop $(docker container ls -a -q) && docker container rm $(docker container ls -a -q)
```

Removing Dangling Volumes

```
docker volume rm $(docker volume ls -f dangling=true -q)
```

Removing all unused (containers, images, networks and volumes)

```
docker system prune -f
```

Clean all

```
docker system prune -a
```

Docker Swarm

Installing Docker Swarm

```
curl -ssl https://get.docker.com | bash
```

Initializing the Swarm

```
docker swarm init --advertise-addr 192.168.10.1
```

Getting a Worker to Join the Swarm

```
docker swarm join-token worker
```

Getting a Manager to Join the Swarm

```
docker swarm join-token manager
```

Listing Services

```
docker service ls
```

Listing nodes

```
docker node ls
```

Creating a Service

```
docker service create --name vote -p 8080:80 instavote/vote
```

Listing Swarm Tasks

```
docker service ps
```

Scaling a Service

```
docker service scale vote=3
```

Updating a Service

```
docker service update --image instavote/vote:movies vote
```

```
docker service update --force --update-parallelism 1 --update-delay 30s nginx
```

```
docker service update --update-parallelism 5--update-delay 2s --image instavote/vote:indent vote
```

```
docker service update --limit-cpu 2 nginx
```

```
docker service update --replicas=5 nginx
```

Notes

This work was first published in [Painless Docker Course](#)

<https://github.com/eon01/DockerCheatSheet>

docker-cheat-sheet

Quick reference guide for Docker commands - not at all exhaustive, just a cheat sheet.

Building/Running

task **command**

Monitoring/Removing Images

task	command
list images	<code>docker images</code>
remove specific image(s)	<code>docker rmi <image_id> <image_id> ...</code>
remove dangling images	<code>docker image prune</code>
remove all unused images	<code>docker image prune --all</code>
remove all images	<code>docker rmi -f \$(docker images -q)</code>

Monitoring/Killing Containers

task	command	notes
see running containers	<code>docker ps</code>	like ps in bash!
see most recently launched container	<code>docker ps -l</code>	- l for last
see all containers	<code>docker ps -a</code>	- a for all
see hash of running containers	<code>docker ps -q</code>	- q for hash
see hash of most recent container	<code>docker ps -ql</code>	mix -q and -l for hash of last
see hash of all containers	<code>docker ps -aq</code>	ditto for -a and -q
see running processes	<code>docker top id container</code>	use id container
kill all running containers	<code>docker kill \$(docker ps -q)</code>	kill only stops running containers
kill most recent container	<code>docker kill \$(docker ps -ql)</code>	
remove all exited containers	<code>docker container prune</code>	
remove all containers	<code>docker rm -f \$(docker ps -qa)</code>	- f forces rm to kill and remove
remove old containers	<code>docker ps -a grep 'weeks ago' awk '{print \$1}' xargs docker rm</code>	removes containers that are created weeks ago

<https://github.com/brainhack101/docker-cheat-sheet>

Docker Cheat Sheet

Docker is a tool which helps developers build and ship high quality applications, faster, anywhere.

Why Docker

With Docker, developers can build any app in any language using any toolchain. Dockerized apps are completely portable and can run anywhere.

Developers can get going by just spinning any container out of list on Docker Hub. Docker manages and tracks changes and dependencies, making it easier for sysadmins to understand how the apps that developers build work. And with Docker Hub, developers can automate their build pipeline and share artifacts with collaborators through public or private repositories.

If you are a complete Docker newbie, you should probably follow the [series of tutorials](#) now.

What we are going to learn?

- [Installation](#)
- [Containers](#)
- [Images](#)
- [Networks](#)
- [Dockerfile](#)
- [Layers](#)
- [Volumes](#)
- [Links](#)
- [Exposing Ports](#)
- [Best Practices](#)
- [Security](#)
- [Tips](#)

Installation

Linux

Quick and easy install script provided by Docker:

```
curl -sSL https://get.docker.com/ | sh
```

If you're not willing to run a random shell script, please see the [installation](#) instructions for your distribution.

Mac OS X

Download and install [Docker For Mac](#)

Windows

Use the [msi](#) installer

Then start up a container:

```
docker run hello-world
```

That's it, you have a running Docker container.

If you are a complete Docker newbie, you should probably follow the [series of tutorials](#) now.

Containers

Docker implements a high-level API to provide lightweight containers that run processes in [isolation](#).

Lifecycle

- [docker create](#) creates a container but does not start it.
- [docker rename](#) allows the container to be renamed.
- [docker run](#) creates and starts a container in one operation.
- [docker rm](#) deletes a container.
- [docker update](#) updates a container's resource limits.

Normally if you run a container without options it will start and stop immediately, if you want keep it running you can use the command, `docker run -td container_id` this will use the option `-t` that will allocate a pseudo-TTY session and `-d` that will detach automatically the container (run container in background and print container ID)

If you want a transient container, `docker run --rm` will remove the container after it stops.

Another useful option is `docker run --name customname docker_image` because when you specify the `--name` inside the run command this will allow you to start and stop a container by calling it with the name the you specified when you created it.

Starting and Stopping

- [docker start](#) starts a container so it is running.
- [docker stop](#) stops a running container.
- [docker restart](#) stops and starts a container.
- [docker pause](#) pauses a running container, "freezing" it in place.
- [docker unpause](#) will unpause a running container.
- [docker wait](#) blocks until running container stops.
- [docker kill](#) sends a SIGKILL to a running container.
- [docker attach](#) will connect to a running container.

If you want to integrate a container with a [host process manager](#), start the daemon with `-r=false` then use `docker start -a`.

If you want to expose container ports through the host, see the [exposing ports](#) section.

Restart policies on crashed docker instances are [covered here](#).

Info

- [docker ps](#) shows running containers.
- [docker logs](#) gets logs from container. (You can use a custom log driver, but logs is only available for `json-file` and `journald` in 1.10).
- [docker inspect](#) looks at all the info on a container (including IP address).
- [docker events](#) gets events from container.
- [docker port](#) shows public facing port of container.
- [docker top](#) shows running processes in container.
- [docker stats](#) shows containers' resource usage statistics.
- [docker diff](#) shows changed files in the container's FS.

`docker ps -a` shows running and stopped containers.

`docker stats --all` shows a running list of containers.

Import / Export

- [docker cp](#) copies files or folders between a container and the local filesystem.
- [docker export](#) turns container filesystem into tarball archive stream to STDOUT.

Executing Commands

- [docker exec](#) to execute a command in container.

To enter a running container, attach a new shell process to a running container called `foo`, use:
`docker exec -it foo /bin/bash`.

Images

Images are just [templates for docker containers](#).

Lifecycle

- [docker images](#) shows all images.
- [docker import](#) creates an image from a tarball.
- [docker build](#) creates image from Dockerfile.
- [docker commit](#) creates image from a container, pausing it temporarily if it is running.
- [docker rmi](#) removes an image.
- [docker load](#) loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- [docker save](#) saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

Info

- [docker history](#) shows history of image.
- [docker tag](#) tags an image to a name (local or registry).

Cleaning up

While you can use the `docker rmi` command to remove specific images, there's a tool called [docker-gc](#) that will clean up images that are no longer used by any containers in a safe manner.

Load/Save image

Load an image from file:

```
docker load < my_image.tar.gz
```

Save an existing image:

```
docker save my_image:my_tag | gzip > my_image.tar.gz
```

Import/Export container

Import a container as an image from file:

```
cat my_container.tar.gz | docker import - my_image:my_tag
```

Export an existing container:

```
docker export my_container | gzip > my_container.tar.gz
```

Difference between loading a saved image and importing an exported container as an image

Loading an image using the `load` command creates a new image including its history.

Importing a container as an image using the `import` command creates a new image excluding the history which results in a smaller image size compared to loading an image.

Networks

Docker has a [networks](#) to configure docker containers to talk to each other without using ports. See [working with networks](#) for more details.

Lifecycle

- [docker network create](#)
- [docker network rm](#)

Info

- [docker network ls](#)
- [docker network inspect](#)

Connection

- [docker network connect](#)
- [docker network disconnect](#)

You can specify a [specific IP address for a container](#):

```
# create a new bridge network with your subnet and gateway for your ip block
docker network create --subnet 203.0.113.0/24 --gateway 203.0.113.254 iptastic

# run a nginx container with a specific ip in that block
$ docker run --rm -it --net iptastic --ip 203.0.113.2 nginx

# curl the ip from any other place (assuming this is a public ip block duh)
$ curl 203.0.113.2
```

Registry & Repository

A repository is a *hosted* collection of tagged images that together create the file system for a container.

A registry is a *host* -- a server that stores repositories and provides an HTTP API for [managing the uploading and downloading of repositories](#).

Docker.com hosts its own [index](#) to a central registry which contains a large number of repositories. Having said that, the central docker registry [does not do a good job of verifying images](#) and should be avoided if you're worried about security.

- [docker login](#) to login to a registry.
- [docker logout](#) to logout from a registry.
- [docker search](#) searches registry for image.
- [docker pull](#) pulls an image from registry to local machine.
- [docker push](#) pushes an image to the registry from local machine.

Run local registry

You can run a local registry by using the [docker distribution](#) project and looking at the [local deploy](#) instructions.

Also see the [mailing list](#).

Dockerfile

[The configuration file](#). Sets up a Docker container when you run `docker build` on it. Vastly preferable to `docker commit`.

Here are some common text editors and their syntax highlighting modules you could use to create Dockerfiles:

- If you use [jEdit](#), I've put up a syntax highlighting module for [Dockerfile](#) you can use.
- [Sublime Text 2](#)
- [Atom](#)
- [Vim](#)
- [Emacs](#)
- [TextMate](#)
- [VS Code](#)
- Also see [Docker meets the IDE](#)

Instructions

- [.dockerignore](#)
- [FROM](#) Sets the Base Image for subsequent instructions.
- [MAINTAINER \(deprecated - use LABEL instead\)](#) Set the Author field of the generated images.
- [RUN](#) execute any commands in a new layer on top of the current image and commit the results.
- [CMD](#) provide defaults for an executing container.
- [EXPOSE](#) informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.
- [ENV](#) sets environment variable.
- [ADD](#) copies new files, directories or remote file to container. Invalidates caches. Avoid ADD and use COPY instead.
- [COPY](#) copies new files or directories to container. Note that this only copies as root, so you have to chown manually regardless of your USER / WORKDIR setting. See <https://github.com/moby/moby/issues/30110>
- [ENTRYPOINT](#) configures a container that will run as an executable.
- [VOLUME](#) creates a mount point for externally mounted volumes or other containers.
- [USER](#) sets the user name for following RUN / CMD / ENTRYPOINT commands.
- [WORKDIR](#) sets the working directory.
- [ARG](#) defines a build-time variable.
- [ONBUILD](#) adds a trigger instruction when the image is used as the base for another build.
- [STOPSIGNAL](#) sets the system call signal that will be sent to the container to exit.
- [LABEL](#) apply key/value metadata to your images, containers, or daemons.

Examples

- [Examples](#)
- [Best practices for writing Dockerfiles](#)
- [Michael Crosby](#) has some more [Dockerfiles best practices](#) / [take 2](#).
- [Building Good Docker Images](#) / [Building Better Docker Images](#)
- [Managing Container Configuration with Metadata](#)

Layers

The versioned filesystem in Docker is based on layers. They're like [git commits or changesets for filesystems](#).

Volumes

Docker volumes are [free-floating filesystems](#). They don't have to be connected to a particular container. You should use volumes mounted from [data-only containers](#) for portability.

Lifecycle

- [docker volume create](#)
- [docker volume rm](#)

Info

- [docker volume ls](#)
- [docker volume inspect](#)

Volumes are useful in situations where you can't use links (which are TCP/IP only). For instance, if you need to have two docker instances communicate by leaving stuff on the filesystem.

You can mount them in several docker containers at once, using `docker run --volumes-from`.

Because volumes are isolated filesystems, they are often used to store state from computations between transient containers. That is, you can have a stateless and transient container run from a recipe, blow it away, and then have a second instance of the transient container pick up from where the last one left off.

See [advanced volumes](#) for more details. Container42 is [also helpful](#).

You can [map MacOS host directories as docker volumes](#):

```
docker run -v /Users/wsargent/myapp/src:/src
```

You can use remote NFS volumes if you're [feeling brave](#).

You may also consider running data-only containers as described [here](#) to provide some data portability.

Links

Links are how Docker containers talk to each other [through TCP/IP ports](#). [Linking into Redis](#) and [Atlassian](#) show worked examples. You can also resolve [links by hostname](#).

This has been deprecated to some extent by [user-defined networks](#).

NOTE: If you want containers to ONLY communicate with each other through links, start the docker daemon with `-icc=false` to disable inter process communication.

If you have a container with the name CONTAINER (specified by `docker run --name CONTAINER`) and in the Dockerfile, it has an exposed port:

```
EXPOSE 1337
```

Then if we create another container called LINKED like so:

```
docker run -d --link CONTAINER:ALIAS --name LINKED user/wordpress
```

Then the exposed ports and aliases of CONTAINER will show up in LINKED with the following environment variables:

```
$ALIAS_PORT_1337_TCP_PORT  
$ALIAS_PORT_1337_TCP_ADDR
```

And you can connect to it that way.

To delete links, use `docker rm --link`.

Generally, linking between docker services is a subset of "service discovery", a big problem if you're planning to use Docker at scale in production. Please read [The Docker Ecosystem: Service Discovery and Distributed Configuration Stores](#) for more info.

Exposing ports

Exposing incoming ports through the host container is [fiddly but doable](#).

This is done by mapping the container port to the host port (only using localhost interface) using `-p`:

```
docker run -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT --name CONTAINER -t someimage
```

You can tell Docker that the container listens on the specified network ports at runtime by using [EXPOSE](#):

```
EXPOSE <CONTAINERPORT>
```

Note that EXPOSE does not expose the port itself -- only `-p` will do that. To expose the container's port on your localhost's port:

```
iptables -t nat -A DOCKER -p tcp --dport <LOCALHOSTPORT> -j DNAT --to-destination <CONTAINERIP>:<PORT>
```

If you're running Docker in Virtualbox, you then need to forward the port there as well, using [forwarded port](#). Define a range of ports in your Vagrantfile like this so you can dynamically map them:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  ...

  (49000..49900).each do |port|
    config.vm.network :forwarded_port, :host => port, :guest => port
  end

  ...
end
```

If you forget what you mapped the port to on the host container, use `docker port` to show it:

```
docker port CONTAINER $CONTAINERPORT
```

Best Practices

This is where general Docker best practices and war stories go:

- [The Rabbit Hole of Using Docker in Automated Tests](#)
- [Bridget Kromhout](#) has a useful blog post on [running Docker in production](#) at Dramafever.
- There's also a best practices [blog post](#) from Lyst.
- [A Docker Dev Environment in 24 Hours!](#)
- [Building a Development Environment With Docker](#)
- [Discourse in a Docker Container](#)

Security

This is where security tips about Docker go. The Docker [security](#) page goes into more detail.

First things first: Docker runs as root. If you are in the `docker` group, you effectively [have root access](#). If you expose the docker unix socket to a container, you are giving the container [root access to the host](#).

Docker should not be your only defense. You should secure and harden it.

For an understanding of what containers leave exposed, you should read is [Understanding and Hardening Linux Containers](#) by [Aaron Grattafiori](#). This is a complete and comprehensive guide to the issues involved with containers, with a plethora of links and footnotes leading on to yet more useful content. The security tips following are useful if you've already hardened containers in the past, but are not a substitute for understanding.

Security Tips

For greatest security, you want to run Docker inside a virtual machine. This is straight from the Docker Security Team Lead -- [slides](#) / [notes](#). Then, run with AppArmor / seccomp / SELinux / grsec etc to [limit the container permissions](#). See the [Docker 1.10 security features](#) for more details.

Docker image ids are [sensitive information](#) and should not be exposed to the outside world. Treat them like passwords.

See the [Docker Security Cheat Sheet](#) by [Thomas Sjögren](#): some good stuff about container hardening in there.

since docker 1.11 you can easily limit the number of active processes running inside a container to prevent fork bombs. This requires a linux kernel ≥ 4.3 with `CGROUP_PIDS=y` to be in the kernel configuration.

```
docker run --pids-limit=64
```

Also available since docker 1.11 is the ability to prevent processes from gaining new privileges. This feature have been in the linux kernel since version 3.5. You can read more about it in [this](#) blog post.

```
docker run --security-opt=no-new-privileges
```

From the [Docker Security Cheat Sheet](#) (it's in PDF which makes it hard to use, so copying below) by [Container Solutions](#):

Turn off interprocess communication with:

```
docker -d --icc=false --iptables
```

Set the container to be read-only:

```
docker run --read-only
```

Verify images with a hashsum:

```
docker pull debian@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Set volumes to be read only:

```
docker run -v $(pwd)/secrets:/secrets:ro debian
```


Tips

- [15 Docker Tips in 5 minutes](#)
- [CodeFresh Everyday Hacks Docker](#)

<https://github.com/glitterd/docker-cheat-sheet>

docker-cheat-sheet

Quick reference guide for Docker commands

Click  if you like the project. Pull Requests are highly appreciated. Follow me [@SudheerJonna](#) for technical updates.

Downloading PDF/Epub formats

You can download the PDF and Epub version of this repository from the latest run on the [actions](#) [tab](#).

Table of Contents

No.	Questions
1	What is docker?
2	Why docker?
3	Installation
4	Registries and Repositories
5	Create,Run,Update and Delete containers
6	Start and stop containers
7	Networks
8	Cleanup commands
9	Utility commands
10	Docker Hub
11	Dockerfile
12	Docker Compose
13	Docker Swarm

What is docker?

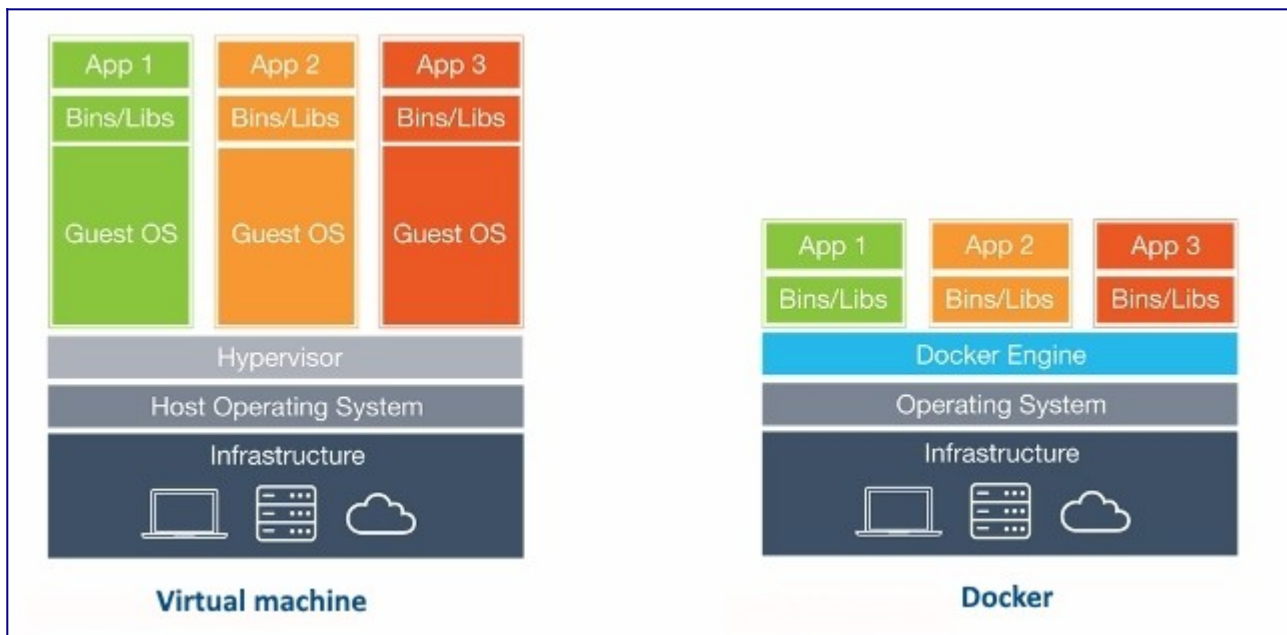
Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.



[Back to Top](#)

Why docker?

Docker is useful to automate the deployment of applications inside a software containers, which makes the applications easy to ship and run virtually anywhere (i.e, platform independent). The Docker container processes run on the host kernel, unlike VM which runs processes in guest kernel.



[↑ Back to Top](#)

Installation

The docker desktop downloads are available for windows, mac and linux distributions.

Windows

It supports for Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later) editions. You need to follow the below steps for installation.

1. Download docker desktop for windows from <https://docs.docker.com/docker-for-windows/install/>
2. Double-click Docker Desktop Installer.exe to run the installer.
3. Make sure Enable Hyper-V Windows Features option is selected

Mac

1. Download docker desktop for mac from <https://docs.docker.com/docker-for-mac/install/>
2. Double-click Docker.dmg to open the installer and drag it to the Applications folder.
3. Double-click Docker.app in the Applications folder to start Docker.

Linux

You can install from a package easily

1. Go to <https://download.docker.com/linux/ubuntu/dists/>, choose your Ubuntu version and then go to pool/stable/ to get .deb file
2. Install Docker Engine by referring the downloaded location of the Docker package.

```
$ sudo dpkg -i /path/to/package.deb
```

3. Verify the Docker Engine by running the hello-world image to check correct installation.

```
$ sudo docker run hello-world
```

 [Back to Top](#)

Registries and Repositories

Registry:

Docker Registry is a service that stores your docker images. It could be hosted by a third party, as public or private registry. Some of the examples are,

- Docker Hub,
- Quay,
- Google Container Registry,
- AWS Container Registry

Repository:

A Docker Repository is a collection of related images with same name which have different tags. These tags are alphanumeric identifiers (like 1.0 or latest) attached to images within a repository.

For example, if you want to pull golang image using `docker pull golang:latest` command, it will download the image tagged latest within the `golang` repository from the Docker Hub registry. The tags appeared on dockerhub as below,

Login

Login to a registry

```
> docker login [OPTIONS] [SERVER]
```

```
[OPTIONS]:  
-u/--username username  
-p/--password password
```

Example:

1. `docker login localhost:8080` // Login to a registry on your localhost
2. `docker login`

Logout

Logout from a registry

```
> docker logout [SERVER]
```

Example:

```
docker logout localhost:8080 // Logout from a registry on your localhost
```

Search image

Search for an image in registry

```
docker search [OPTIONS] TERM
```

Example:

```
docker search golang  
docker search --filter stars=3 --no-trunc golang
```

Pull image

This command pulls an image or a repository from a registry to local machine

```
docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Example:

```
docker image pull golang:latest
```

Push image

This command pushes an image to the registry from local machine.

```
docker image push [OPTIONS] NAME[:TAG]
```

```
docker image push golang:latest
```



[Back to Top](#)

Create,Run,Update and Delete containers

Create

Create a new container

```
docker container create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Example:

```
docker container create -t -i sudheerj/golang --name golang
```

Rename

Rename a container

```
docker container rename CONTAINER NEW_NAME
```

Example:

```
docker container rename golang golanguage
```

```
docker container rename golanguage golang
```

Run

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Example:

```
docker container run -it --name golang -d sudheerj/golang
```

You can also run a command inside container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Example:

```
docker exec -it golang sh // Or use bash command if sh is failed
```

Update

Update configuration of one or more containers

```
docker container update [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container update --memory "1g" --cpuset-cpu "1" golang // update the
golang to use 1g of memory and only use cpu core 1
```

Remove

Remove one or more containers

```
docker container rm [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container rm golang
```

```
docker rm $(docker ps -q -f status=exited) // Remove all the stopped containers
```



[Back to Top](#)

Start and stop containers

Start

Start one or more stopped containers

```
docker container start [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container start golang
```

Stop

Stop one or more running containers

```
docker container stop [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container stop golang
```

```
docker stop $(docker ps -a -q) // To stop all the containers
```

Restart

Restart one or more containers and processes running inside the container/containers.

```
docker container restart [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container restart golang
```

Pause

Pause all processes within one or more containers

```
docker container pause CONTAINER [CONTAINER...]
```

Example:

```
docker container pause golang
```

Unpause/Resume

Unpause all processes within one or more containers

```
docker container unpause CONTAINER [CONTAINER...]
```

Example:

```
docker container unpause golang
```

Kill

Kill one or more running containers

```
docker container kill [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
docker container kill golang
```

Wait

Block until one or more containers stop and print their exit codes after that

```
docker container wait CONTAINER [CONTAINER...]
```

Example:

```
docker container wait golang
```



[Back to Top](#)

Networks

Docker provides network commands connect containers to each other and to other non-Docker workloads. The usage of network commands would be `docker network COMMAND`

List networks

List down available networks

```
docker network ls
```

Connect a container to network

You can connect a container by name or by ID to any network. Once it connected, the container can communicate with other containers in the same network.

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

Example:

```
docker network connect multi-host-network container1
```

Disconnect a container from a network

You can disconnect a container by name or by ID from any network.

```
docker network disconnect [OPTIONS] NETWORK CONTAINER
```

Example:

```
docker network disconnect multi-host-network container1
```

Remove one or more networks

Removes one or more networks by name or identifier. Remember, you must first disconnect any containers connected to it before removing it.

```
docker network rm NETWORK [NETWORK...]
```

Example:

```
docker network rm my-network
```

Create network

It is possible to create a network in Docker before launching containers

```
docker network create [OPTIONS] NETWORK
```

Example:

```
sudo docker network create --driver bridge some_network
```

The above command will output the long ID for the new network.

Inspect network

You can see more details on the network associated with Docker using network inspect command.

```
docker network inspect networkname
```

Example:

```
docker network inspect bridge
```

Cleanup commands

You may need to cleanup resources (containers, volumes, images, networks) regularly.

Remove all unused resources

```
docker system prune
```

Images

```
$ docker images
```

```
$ docker rmi $(docker images --filter "dangling=true" -q --no-trunc)
```

```
$ docker images | grep "none"
```

```
$ docker rmi $(docker images | grep "none" | awk '/ / { print $3 }')
```

Containers

```
$ docker ps
```

```
$ docker ps -a
```

```
$ docker rm $(docker ps -qa --no-trunc --filter "status=exited")
```

Volumes

```
$ docker volume rm $(docker volume ls -qf dangling=true)
```

```
$ docker volume ls -qf dangling=true | xargs -r docker volume rm
```

Networks

```
$ docker network ls
```

```
$ docker network ls | grep "bridge"
```

```
$ docker network rm $(docker network ls | grep "bridge" | awk '/ / { print $1 }')
```

 [Back to Top](#)

Utility commands

 [Back to Top](#)

Docker Hub

Docker Hub is a cloud-based repository provided by Docker to test, store and distribute container images which can be accessed either privately or publicly.

From

It initializes a new image and sets the Base Image for subsequent instructions. It must be a first non-comment instruction in the Dockerfile.

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

Note: Both `tag` and `digest` are optional. If you omit either of them, the builder assumes a latest by default.

Dockerfile

Dockerfile is a text document that contains set of commands and instructions which will be executed in a sequence in the docker environment for building a new docker image.

FROM

This command Sets the Base Image for subsequent instructions

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

Example:
FROM ubuntu:18.04

RUN

RUN instruction allows you to install your application and packages required for it. It executes any commands on top of the current image and creates a new layer by committing the results. It is quite common to have multiple RUN instructions in a Dockerfile.

It has two forms

1. Shell Form: RUN

```
RUN npm start
```

2. Exec form RUN ["", "", ""]

```
RUN [ "npm", "start" ]
```

ENTRYPOINT

An ENTRYPOINT allows you to configure a container that will run as an executable. It is used to run when container starts.

Exec Form:
ENTRYPOINT ["executable", "param1", "param2"]
Shell Form:
ENTRYPOINT command param1 param2

Example:
FROM alpine:3.5
ENTRYPOINT ["/bin/echo", "Print ENTRYPOINT instruction of Exec Form"]

If an image has an ENTRYPOINT and pass an argument to it while running the container, it won't override the existing entrypoint but it just appends what you passed with the entrypoint. To override the existing ENTRYPOINT, you should use `-entrypoint` flag for the running container.

Let's see the behavior with the above dockerfile,

Build image:
docker build -t entrypointImage .

Run the image:
docker container run entrypointImage // Print ENTRYPOINT instruction of Exec Form

Override entrypoint:
docker run --entrypoint "/bin/echo" entrypointImage "Override ENTRYPOINT instruction" // Override ENTRYPOINT instruction

CMD

CMD instruction is used to set a default command, which will be executed only when you run a container without specifying a command. But if the docker container runs with a command, the default command will be ignored.

The CMD instruction has three forms,

1. Exec form:
CMD ["executable", "param1", "param2"]
2. Default params to ENTRYPOINT:
CMD ["param1", "param2"]
3. Shell form:
CMD command param1 param2

The main purpose of the CMD command is to launch the required software in a container. For example, running an executable .exe file or a Bash terminal as soon as the container starts.

Remember, if docker runs with executable and parameters then CMD instruction will be overridden (Unlike ENTRYPOINT).

docker run executable parameters

Note: There should only be one CMD command in your Dockerfile. Otherwise only the last instance of CMD will be executed.

COPY

The COPY instruction copies new files or directories from source and adds them to the destination filesystem of the container.

```
COPY [--chown=<user>:<group>] <src>... <dest>
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

Example:

```
COPY test.txt /absoluteDir/
COPY tes? /absoluteDir/ // Copies all files or directories starting with test to
destination container
```

The path must be relative to the source directory that is being built. Whereas is an absolute path, or a path relative to WORKDIR.

ADD

The ADD instruction copies new files, directories or remote file URLs from source and adds them to the filesystem of the image at the destination path. The functionality is similar to COPY command and supports two forms of usage,

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

Example:

```
ADD test.txt /absoluteDir/
ADD tes? /absoluteDir/ // Copies all files or directories starting with test to
destination container
```

ADD commands provides additional features such as downloading remote resources, extracting TAR files etc.

1. Download an external file and copy to the destination
ADD http://source.file/url /destination/path

2. Copies compressed files and extract the content in the destination
ADD source.file.tar.gz /temp

ENV

The ENV instruction sets the environment variable to the value . It has two forms,

1. The first form, ENV <key> <value>, will set a single variable to a value.
2. The second form, ENV <key>=<value> . . . , allows for multiple variables to be set at one time.

```
ENV <key> <value>
ENV <key>=<value> [<key>=<value> ...]
```

Example:

```
ENV name="John Doe" age=40
ENV name John Doe
ENV age 40
```

EXPOSE

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. i.e, It helps in inter-container communication. You can specify whether the port listens on TCP or UDP, and the default is TCP.

```
EXPOSE <port> [<port>/<protocol>...]
```

Example:

```
EXPOSE 80/udp
EXPOSE 80/tcp
```

But if you want to bind the port of the container with the host machine on which the container is running, use -p option of `docker run` command.

```
docker run -p <HOST_PORT>:<CONTAINER_PORT> IMAGE_NAME
```

Example:

```
docker run -p 80:80/udp myDocker
```

WORKDIR

The WORKDIR command is used to define the working directory of a Docker container at any given time for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.

```
WORKDIR /path/to/workdir
```

Example:

```
WORKDIR /c
WORKDIR d
WORKDIR e
RUN pwd // /c/d/e
```

LABEL

The LABEL instruction adds metadata as key-value pairs to an image. Labels included in base or parent images (images in the FROM line) are inherited by your image.

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

Example:

```
LABEL version="1.0"
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

You can view an image's labels using the `docker image inspect --format=''` myimage command. The output would be as below,

```
{
  "version": "1.0",
  "multi.label1": "value1",
  "multi.label2": "value2",
  "other": "value3"
}
```

MAINTAINER

The MAINTAINER instruction sets the Author field of the generated images.

```
MAINTAINER <name>
```

Example:

```
MAINTAINER John
```

This command is deprecated status now and the recommended usage is with LABEL command

```
LABEL maintainer="John"
```

VOLUME

The VOLUME instruction creates a mount point with the specified name and mounted volumes from native host or other containers.

```
VOLUME ["/data"]
```

Example:

```
FROM ubuntu  
RUN mkdir /test  
VOLUME /test
```

Docker Compose

Docker compose(or compose) is a tool for defining and running multi-container Docker applications.

Docker Swarm

Docker Swarm(or swarm) is an open-source tool used to cluster and orchestrate Docker containers.

<https://github.com/sudheerj/docker-cheat-sheet>

Docker Cheatsheet

This cheatsheet provides a collection of commonly used docker commands.

Getting Started

The getting started guide on Docker has detailed instructions for setting up [Docker](#).

After setup is complete, run the following commands to verify the success of installation:

PLEASE NOTE POST INSTALLATION STEPS BELOW IF YOU HAVE TO PREPEND SUDO TO EVERY COMMAND

- **docker version** - provides full description of docker version
- **docker info** - display system wide information
- **docker -v** - provides a short description of docker version
- **docker run hello-world** - pull hello-world container from registry and run it

Have a look at the [free training](#) offered by Docker.

Have a look at the [repository](#) of images offered by Docker.

Optional Post Installation Steps

To create the docker group and add your user:

- Create the docker group
`sudo groupadd docker`
 - Add your user to the docker group
`sudo usermod -aG docker $USER`
 - Log out and log back in so that your group membership is re-evaluated.
 - Verify that you can run docker commands without sudo.
-

Docker Commands

Get docker info

General

Command	Description
<code>docker version</code>	provides full description of docker version
<code>docker -v</code>	provides a short description of docker version
<code>docker info</code>	display system wide information
<code>docker info --format '{{.DriverStatus}}'</code>	display 'DriverStatus' fragment from docker information

Command	Description
<code>docker info --format '{{json .DriverStatus}}'</code>	display 'DriverStatus' fragment from docker information in JSON format

Manage Images

Command	Description
<code>docker image ls</code>	shows all local images
<code>docker image ls --filter 'reference=ubuntu:16.04'</code>	show images filtered by name and tag
<code>docker image pull [image-name]</code>	pull specified image from registry
<code>docker image rm [image-name]</code>	remove image for specified <i>image-name</i>
<code>docker image rm [image-id]</code>	remove image for specified <i>image-id</i>
<code>docker image prune</code>	remove unused images

Search Images

Command	Description
<code>docker search [image-name] --filter "is-official=true"</code>	find only official images having <i>[image-name]</i>
<code>docker search [image-name] -- filter "stars=1000"</code>	find only images having specified <i>[image-name]</i> and 1000 or more stars

Manage Containers

Display Container Information

Command	Description
<code>docker container ls</code>	show all running containers
<code>docker container ls -a</code>	show all containers regardless of state
<code>docker container ls --filter "status=exited" --filter "ancestor=ubuntu"</code>	show all container instances of the ubuntu image that have exited
<code>docker container inspect [container-name]</code>	display detailed information about specified container
<code>docker container inspect --format '{{.NetworkSettings.IPAddress}}' [container-name]</code>	display detailed information about specified container using specified format
<code>docker container inspect --format '{{json .NetworkSettings}}' [container-name]</code>	display detailed information about specified container using specified format

Run Container

Command	Description
<code>docker container run [image-name]</code>	run container based on specified image
<code>docker container run --rm [image-name]</code>	run container based on specified image and immediately remove it once it stops
<code>docker container run --name fuzzy-box [image-name]</code>	assign name and run container based on specified image

Remove Container

Command	Description
<code>docker container rm [container-name]</code>	remove specified container
<code>docker container rm \$(docker container ls --filter</code>	remove all containers whose id's are

Command	Description
"status=exited" --filter "ancestor=ubuntu" -q)	returned from '\$(...)' list

Manage Volumes

Display Volume Information

Command	Description
docker volume ls	show all volumes
docker volume ls --filter "dangling=true"	display all volumes not referenced by any containers
docker volume inspect [volume-name]	display detailed information on <i>[volume-name]</i>

Remove Volumes

Command	Description
docker volume rm [volume-name]	remove specified volume
docker volume rm \$(docker volume ls --filter "dangling=true" -q)	remove all volumes having an id equal to any of the id's returned from '\$(...)' list

Running containers

Run hello-world container

```
docker run hello-world
```

Run an [Alpine Linux](#) container (a lightweight linux distribution)

- Find image and display brief summary

```
docker search alpine --filter=stars=1000 --no-trunc
```
- Fetch alpine image from Docker registry and save it

```
docker pull alpine
```
- Display list of local images

```
docker image ls
```
- List container contents using *listing* format

```
docker run alpine ls -l
```
- Print message from container

```
docker run alpine echo "Hello from Alpine!"
```
- Running the run command with the -it flags attaches container to an interactive tty

```
docker run -it alpine bin/sh
```

Run MongoDB

Run MongoDB Using Named Volume

To run a new MongoDB container, execute the following command from the CLI:

```
docker run --rm --name mongo-dev -v mongo-dev-db:/data/db -d mongo
```

CLI Command	Description
--rm	remove container when stopped
--name mongo-dev	give container a custom name
-v mongo-dev-db:/data/db	map the container volume 'data/db' to a custom name 'mongo-dev-db'
-d mongo	run mongo container as a daemon in the background

Run MongoDB Using Bind Mount

```
cd
mkdir -p mongoddb/data/db
docker run --rm --name mongo-dev -v ~/mongoddb/data/db:/data/db -d mongo
```

CLI Command	Description
--rm	remove container when stopped
--name mongo-dev	give container a custom name
-v ~/mongoddb/data/db:/data/db	map the container volume 'data/db' to a bind mount '~/mongoddb/data/db'
-d mongo	run mongo container as a daemon in the background

Access MongoDB

Connect to MongoDB

There are 2 steps to accessing the MongoDB shell.

1. Firstly, access the MongoDB container shell by executing the following command:

```
docker exec -it mongo-dev bash
```

This will open an interactive shell (bash) on the MongoDB container.

2. Secondly, once inside the container shell, access the MongoDB shell by executing the following command:

```
mongo localhost
```

Show Databases

Once connected to MongoDB shell, run the following command to show a list of databases:

```
show dbs
```

Create New Database

Create a new database and collection

```
use test
db.messages.insert({"message": "Hello World!"})
db.messages.find()
```

Creating Images

Create custom [Alpine Linux](#) image with GIT setup

1. Create project folder with empty Dockerfile

```
cd ~
mkdir -p projects/docker/alpine-git
cd !$
touch Dockerfile
```

2. Create Dockerfile

```
FROM alpine:latest

LABEL author="codesaucer" \
      description="Official Alpine image with GIT and VIM installed"

ENV WORKING_DIRECTORY=/projects

RUN apk update && apk add git vim

RUN mkdir $WORKING_DIRECTORY

WORKDIR $WORKING_DIRECTORY
```

3. Build Dockerfile

```
docker image build -t [docker-username]/alpine-git
docker run --rm -it [docker-username]/alpine-git /bin/sh
```

4. Push Dockerfile

```
docker login
docker push [docker-username]/alpine-git:latest
```

<https://github.com/drminnaar/cheatsheets/blob/master/docker-cheatsheet.md>

Docker Cheat Sheet

- [Helpful Tools](#)
- [Installation](#)
- [Containers](#)
- [Images](#)
- [Registry and Repository](#)
- [Dockerfile](#)
- [Layers](#)
- [Links](#)
- [Volumes](#)
- [Exposing Ports](#)
- [Best Practices](#)
- [Security](#)
- [Tips](#)

Helpful Tools

I use [PyCharm Docker plugin](#) for scripts Docker builds. It's my thing. Use whatever IDE you like.

Linux

The 3.10.x kernel is [the minimum requirement](#) for Docker.

MacOS

10.8 “Mountain Lion” or newer is required.

Installation

Linux

Quick and easy install script provided by Docker:

```
curl -sSL https://get.docker.com/ | sh
```

If you're not willing to run a random shell script, please see the [installation](#) instructions for your distribution.

If you are a complete Docker newbie, you should follow the [series of tutorials](#) now.

Mac OS X

Download and install [Docker Toolbox](#). If that doesn't work, see the [installation instructions](#).

Once you've installed Docker Toolbox, install a VM with Docker Machine using the VirtualBox provider:

```
docker-machine create --driver=virtualbox default
docker-machine ls
eval "$(docker-machine env default)"
```

Then start up a container:

```
docker run hello-world
```

That's it, you have a running Docker container.

If you are a complete Docker newbie, you should probably follow the [series of tutorials](#) now.

Containers

[Your basic isolated Docker process](#). Containers are to Virtual Machines as threads are to processes. Or you can think of them as chroots on steroids.

Lifecycle

- [docker create](#) creates a container but does not start it.
- [docker run](#) creates and starts a container in one operation.
- [docker stop](#) stops it.
- [docker start](#) will start it again.
- [docker restart](#) restarts a container.
- [docker rm](#) deletes a container.
- [docker kill](#) sends a SIGKILL to a container.
- [docker attach](#) will connect to a running container.
- [docker wait](#) blocks until container stops.

If you want to run and then interact with a container, `docker start`, then spawn a shell as described in [Executing Commands](#).

If you want a transient container, `docker run --rm` will remove the container after it stops.

If you want to remove also the volumes associated with the container, the deletion of the container must include the `-v` switch like in `docker rm -v`.

If you want to poke around in an image, `docker run -t -i <myimage> <myshell>` to open a tty.

If you want to poke around in a running container, `docker exec -t -i <mycontainer> <myshell>` to open a tty.

If you want to map a directory on the host to a docker container, `docker run -v $HOSTDIR:$DOCKERDIR`. Also see [Volumes](#).

If you want to integrate a container with a [host process manager](#), start the daemon with `-r=false` then use `docker start -a`.

If you want to expose container ports through the host, see the [exposing ports](#) section.

Restart policies on crashed docker instances are [covered here](#).

Info

- [docker ps](#) shows running containers.

- [docker logs](#) gets logs from container.
- [docker inspect](#) looks at all the info on a container (including IP address).
- [docker events](#) gets events from container.
- [docker port](#) shows public facing port of container.
- [docker top](#) shows running processes in container.
- [docker stats](#) shows containers' resource usage statistics.
- [docker diff](#) shows changed files in the container's FS.

`docker ps -a` shows running and stopped containers.

Import / Export

- [docker cp](#) copies files or folders between a container and the local filesystem..
- [docker export](#) turns container filesystem into tarball archive stream to STDOUT.

Executing Commands

- [docker exec](#) to execute a command in container.

To enter a running container, attach a new shell process to a running container called foo, use:
`docker exec -it foo /bin/bash`.

Images

Images are just [templates for docker containers](#).

Lifecycle

- [docker images](#) shows all images.
- [docker import](#) creates an image from a tarball.
- [docker build](#) creates image from Dockerfile.
- [docker commit](#) creates image from a container.
- [docker rmi](#) removes an image.
- [docker load](#) loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- [docker save](#) saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

Info

- [docker history](#) shows history of image.
- [docker tag](#) tags an image to a name (local or registry).

Registry & Repository

A repository is a *hosted* collection of tagged images that together create the file system for a container.

A registry is a *host* -- a server that stores repositories and provides an HTTP API for [managing the uploading and downloading of repositories](#).

Docker.com hosts its own [index](#) to a central registry which contains a large number of repositories. Having said that, the central docker registry [does not do a good job of verifying images](#) and should be avoided if you're worried about security.

- [docker login](#) to login to a registry.
- [docker search](#) searches registry for image.
- [docker pull](#) pulls an image from registry to local machine.
- [docker push](#) pushes an image to the registry from local machine.

Run local registry

[Registry implementation](#) has an official image for basic setup that can be launched with [docker run -p 5000:5000 registry](#) Note that this installation does not have any authorization controls. You may use option `-P -p 127.0.0.1:5000:5000` to limit connections to localhost only. In order to push to this repository tag image with `repositoryHostName:5000/imageName` then push this tag.

Dockerfile

[The configuration file](#). Sets up a Docker container when you run `docker build` on it. Vastly preferable to `docker commit`. Check out the [tools section](#) for additional editors.

Instructions

- [.dockerignore](#)
- [FROM](#)
- [MAINTAINER](#)
- [RUN](#)
- [CMD](#)
- [EXPOSE](#)
- [ENV](#)
- [ADD](#)
- [COPY](#)
- [ENTRYPOINT](#)
- [VOLUME](#)
- [USER](#)
- [WORKDIR](#)
- [ONBUILD](#)

Tutorial

- [Flux7's Dockerfile Tutorial](#)

Layers

The versioned filesystem in Docker is based on layers. They're like [git commits or changesets for filesystems](#).

Note that if you're using [aufs](#) as your filesystem, Docker does not always remove data volumes containers layers when you delete a container! See [PR 8484](#) for more details.

Links

Links are how Docker containers talk to each other [through TCP/IP ports](#). [Linking into Redis](#) and [Atlassian](#) show worked examples. You can also (in 0.11) resolve [links by hostname](#).

NOTE: If you want containers to ONLY communicate with each other through links, start the docker daemon with `-icc=false` to disable inter process communication.

If you have a container with the name CONTAINER (specified by `docker run --name CONTAINER`) and in the Dockerfile, it has an exposed port:

```
EXPOSE 1337
```

Then if we create another container called LINKED like so:

```
docker run -d --link CONTAINER:ALIAS --name LINKED user/wordpress
```

Then the exposed ports and aliases of CONTAINER will show up in LINKED with the following environment variables:

```
$ALIAS_PORT_1337_TCP_PORT  
$ALIAS_PORT_1337_TCP_ADDR
```

And you can connect to it that way.

To delete links, use `docker rm --link .`

If you want to link across docker hosts then you should look at [Swarm](#). This [link on stackoverflow](#) provides some good information on different patterns for linking containers across docker hosts.

Volumes

Docker volumes are [free-floating filesystems](#). They don't have to be connected to a particular container. You should use volumes mounted from [data-only containers](#) for portability.

Volumes are useful in situations where you can't use links (which are TCP/IP only). For instance, if you need to have two docker instances communicate by leaving stuff on the filesystem.

You can mount them in several docker containers at once, using `docker run --volumes-from`.

Because volumes are isolated filesystems, they are often used to store state from computations between transient containers. That is, you can have a stateless and transient container run from a recipe, blow it away, and then have a second instance of the transient container pick up from where the last one left off.

See [advanced volumes](#) for more details. Container42 is [also helpful](#).

For an easy way to clean abandoned volumes, see [docker-cleanup-volumes](#)

As of 1.3, you can [map MacOS host directories as docker volumes](#) through boot2docker:

```
docker run -v /Users/wsargent/myapp/src:/src
```

You can also use remote NFS volumes if you're [feeling brave](#).

You may also consider running data-only containers as described [here](#) to provide some data portability.

Exposing ports

Exposing incoming ports through the host container is [fiddly but doable](#).

The fastest way is to map the container port to the host port (only using localhost interface) using `-p`:

```
docker run -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT --name CONTAINER -t someimage
```

If you don't want to use the `-p` option on the command line, you can persist port forwarding by using [EXPOSE](#):

```
EXPOSE <CONTAINERPORT>
```

If you're running Docker in Virtualbox, you then need to forward the port there as well, using [forwarded port](#). It can be useful to define something in Vagrantfile to expose a range of ports so that you can dynamically map them:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  ...

  (49000..49900).each do |port|
    config.vm.network :forwarded_port, :host => port, :guest => port
  end

  ...
end
```

If you forget what you mapped the port to on the host container, use `docker port` to show it:

```
docker port CONTAINER $CONTAINERPORT
```

Examples

- [Examples](#)
- [Best practices for writing Dockerfiles](#)
- [Michael Crosby](#) has some more [Dockerfiles best practices](#) / [take 2](#).

Best Practices

This is where general Docker best practices and war stories go:

- [The Rabbit Hole of Using Docker in Automated Tests](#)

- [Bridget Kromhout](#) has a useful blog post on [running Docker in production](#) at Dramafever.
- There's also a best practices [blog post](#) from Lyst.
- [A Docker Dev Environment in 24 Hours!](#)
- [Building a Development Environment With Docker](#)
- [Discourse in a Docker Container](#)

Security

This is where security tips about Docker go.

If you are in the `docker` group, you effectively [have root access](#).

Likewise, if you expose the docker unix socket to a container, you are giving the container [root access to the host](#).

Docker image ids are [sensitive information](#) and should not be exposed to the outside world. Treat them like passwords.

See the [Docker Security Cheat Sheet](#) by [Thomas Sjögren](#).

From the [Docker Security Cheat Sheet](#) (it's in PDF which makes it hard to use, so copying below) by [Container Solutions](#):

Turn off interprocess communication with:

```
docker -d --icc=false --iptables
```

Set the container to be read-only:

```
docker run --read-only
```

Verify images with a hashsum:

```
docker pull debian@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Set volumes to be read only:

```
docker run -v $(pwd)/secrets:/secrets:ro debian
```

Set memory and CPU sharing:

```
docker -c 512 -mem 512m
```

Define and run a user in your Dockerfile so you don't run as root inside the container:

```
RUN groupadd -r user && useradd -r -g user user
USER user
```

Tips

Sources:

- [15 Docker Tips in 5 minutes](#)

Last Ids

```
alias dl='docker ps -l -q'
docker run ubuntu echo hello world
docker commit `dl` helloworld
```

Commit with command (needs Dockerfile)

```
docker commit -run='{"Cmd":["postgres", "-too -many -opts"]}' `dl` postgres
```

Get IP address

```
docker inspect `dl` | grep IPAddress | cut -d '"' -f 4
```

or

```
wget http://stedolan.github.io/jq/download/source/jq-1.3.tar.gz
tar xzvf jq-1.3.tar.gz
cd jq-1.3
./configure && make && sudo make install
docker inspect `dl` | jq -r '.[0].NetworkSettings.IPAddress'
```

or using a [go template](#)

```
docker inspect -f '{{.NetworkSettings.IPAddress}}' <container_name>
```

Get port mapping

```
docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' <containername>
```

Find containers by regular expression

```
for i in $(docker ps -a | grep "REGEXP_PATTERN" | cut -f1 -d" "); do echo $i; done`
```

Get Environment Settings

```
docker run --rm ubuntu env
```

Kill running containers

```
docker kill $(docker ps -q)
```

Delete old containers

```
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm
```

Delete stopped containers

```
docker rm -v `docker ps -a -q -f status=exited`
```

Delete dangling images

```
docker rmi $(docker images -q -f dangling=true)
```


Delete all images

```
docker rmi $(docker images -q)
```

Show image dependencies

```
docker images -viz | dot -Tpng -o docker.png
```

Slimming down Docker containers [Intercity Blog](#)

- Cleaning APT

```
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

- Flatten an image

```
ID=$(docker run -d image-name /bin/bash)
docker export $ID | docker import - flat-image-name
```

- For backup

```
ID=$(docker run -d image-name /bin/bash)
(docker export $ID | gzip -c > image.tgz)
gzip -dc image.tgz | docker import - flat-image-name
```

Monitor system resource utilization for running containers

To check the CPU, memory and network i/o usage, you can use:

```
docker stats <container>
```

for a single container or

```
docker stats $(docker ps -q)
```

to monitor all containers on the docker host.

<https://github.com/therandomsecurityguy/docker-cheat-sheet>

docker-cheat-sheet

Best way to clean docker safely:

```
$ docker system prune -a --volumes
```

Remove all container Forcefully:

```
docker rm --force $(docker ps -a -q)
# or
docker container prune -f
```

Remove all Networks:

```
docker network prune -f
# or
docker network rm $(docker network ls | grep "bridge" | awk '/ / { print $1 }')
```

Remove all dangling volumes:

```
docker volume rm $(docker volume ls -qf dangling=true)
```

Remove only dangling images:

```
docker rmi $(docker images -f dangling=true -q)
```

Remove all images:

```
docker rmi $(docker images -f -a -q)
```

Remove images with pattern:

```
docker images -a | grep "pattern" | awk '{print $3}' | xargs docker rmi
```

Remove all stopped containers:

```
docker rm $(docker ps -a -q)
# or
docker container prune
```

Docker Run and remove (my-container) container when stopped:

```
docker run --rm -it my-docker
```

Additional guides

[Remote Docker setup \(over TCP port\)](#)

[Install Docker using snap](#)

[Installing dokcer-compose \(linux\)](#)

Other Cheat sheets

[Git](#)

[SSH](#)

[Docker Compose](#)

<https://github.com/WSMathias/docker-cheat-sheet>

Docker Cheat Sheet

List images

```
$ docker images
```

Remove image

```
$ docker rmi <IMAGE_ID>
```

List containers

```
$ docker ps # running containers only  
$ docker ps -a # all containers
```

Remove container

```
$ docker rm <CONTAINER_ID>  
$ docker rm `docker ps -aq` # clean them all
```

Stop container

```
$ docker stop <CONTAINER_ID>
```

Fetch logs of a container

```
$ docker logs <CONTAINER_ID> # last line only  
$ docker logs <CONTAINER_ID> -f # continue streaming
```

Bash or SSH into a running container in background mode

```
$ docker exec -it <CONTAINER_ID> bash  
$ exit
```

Build an run

```
$ docker build -t <IMAGE_NAME> .  
$ docker run -i -p <HOST_PORT>:<CONTAINER_PORT> -d <IMAGE_NAME>
```

<https://gist.github.com/martinbuberl/5de2effea36e5a2c2d5c6ef394d13467>

Docker Commands, Help & Tips

Show commands & management commands

```
$ docker
```

Docker version info

```
$ docker version
```

Show info like number of containers, etc

```
$ docker info
```

WORKING WITH CONTAINERS

Create an run a container in foreground

```
$ docker container run -it -p 80:80 nginx
```

Create an run a container in background

```
$ docker container run -d -p 80:80 nginx
```

Shorthand

```
$ docker container run -d -p 80:80 nginx
```

Naming Containers

```
$ docker container run -d -p 80:80 --name nginx-server nginx
```

TIP: WHAT RUN DID

- Looked for image called nginx in image cache
- If not found in cache, it looks to the default image repo on Dockerhub
- Pulled it down (latest version), stored in the image cache
- Started it in a new container
- We specified to take port 80- on the host and forward to port 80 on the container
- We could do "\$ docker container run --publish 8000:80 --detach nginx" to use port 8000
- We can specify versions like "nginx:1.09"

List running containers

```
$ docker container ls
```

OR

```
$ docker ps
```

List all containers (Even if not running)

```
$ docker container ls -a
```

Stop container

```
$ docker container stop [ID]
```

Stop all running containers

```
$ docker stop $(docker ps -aq)
```

Remove container (Can not remove running containers, must stop first)

```
$ docker container rm [ID]
```

To remove a running container use force(-f)

```
$ docker container rm -f [ID]
```

Remove multiple containers

```
$ docker container rm [ID] [ID] [ID]
```

Remove all containers

```
$ docker rm $(docker ps -aq)
```

Get logs (Use name or ID)

```
$ docker container logs [NAME]
```

List processes running in container

```
$ docker container top [NAME]
```

TIP: ABOUT CONTAINERS

Docker containers are often compared to virtual machines but they are actually just processes running on your host os. In Windows/Mac, Docker runs in a mini-VM so to see the processes you'll need to connect directly to that. On Linux however you can run "ps aux" and see the processes directly

IMAGE COMMANDS

List the images we have pulled

```
$ docker image ls
```

We can also just pull down images

```
$ docker pull [IMAGE]
```

Remove image

```
$ docker image rm [IMAGE]
```

Remove all images

```
$ docker rmi $(docker images -a -q)
```

TIP: ABOUT IMAGES

- Images are app binaries and dependencies with meta data about the image data and how to run the image
- Images are not a complete OS. No kernel, kernel modules (drivers)
- Host provides the kernel, big difference between VM

Some sample container creation

NGINX:

```
$ docker container run -d -p 80:80 --name nginx nginx (-p 80:80 is optional as it runs on 80 by default)
```

APACHE:

```
$ docker container run -d -p 8080:80 --name apache httpd
```

MONGODB:

```
$ docker container run -d -p 27017:27017 --name mongo mongo
```

MYSQL:

```
$ docker container run -d -p 3306:3306 --name mysql --env MYSQL_ROOT_PASSWORD=123456 mysql
```

CONTAINER INFO

View info on container

```
$ docker container inspect [NAME]
```

Specific property (--format)

```
$ docker container inspect --format '{{.NetworkSettings.IPAddress}}' [NAME]
```

Performance stats (cpu, mem, network, disk, etc)

```
$ docker container stats [NAME]
```

ACCESSING CONTAINERS

Create new nginx container and bash into

```
$ docker container run -it --name [NAME] nginx bash
```

- i = interactive Keep STDIN open if not attached
- t = tty - Open prompt

For Git Bash, use "winpty"

```
$ winpty docker container run -it --name [NAME] nginx bash
```

Run/Create Ubuntu container

```
$ docker container run -it --name ubuntu ubuntu
```

(no bash because ubuntu uses bash by default)

You can also make it so when you exit the container does not stay by using the -rm flag

```
$ docker container run --rm -it --name [NAME] ubuntu
```

Access an already created container, start with -ai

```
$ docker container start -ai ubuntu
```

Use exec to edit config, etc

```
$ docker container exec -it mysql bash
```

Alpine is a very small Linux distro good for docker

```
$ docker container run -it alpine sh
```

(use sh because it does not include bash) (alpine uses apk for its package manager - can install bash if you want)

NETWORKING

"bridge" or "docker0" is the default network

Get port

```
$ docker container port [NAME]
```

List networks

```
$ docker network ls
```

Inspect network

```
$ docker network inspect [NETWORK_NAME]  
("bridge" is default)
```

Create network

```
$ docker network create [NETWORK_NAME]
```

Create container on network

```
$ docker container run -d --name [NAME] --network [NETWORK_NAME] nginx
```

Connect existing container to network

```
$ docker network connect [NETWORK_NAME] [CONTAINER_NAME]
```

Disconnect container from network

```
$ docker network disconnect [NETWORK_NAME] [CONTAINER_NAME]
```

Detach network from container

```
$ docker network disconnect
```

IMAGE TAGGING & PUSHING TO DOCKERHUB

tags are labels that point to an image ID

```
$ docker image ls
```

You'll see that each image has a tag

Retag existing image

```
$ docker image tag nginx btraversy/nginx
```

Upload to dockerhub

```
$ docker image push bradtraversy/nginx
```

If denied, do

```
$ docker login
```

Add tag to new image

```
$ docker image tag bradtraversy/nginx bradtraversy/nginx:testing
```

DOCKERFILE PARTS

- FROM - The OS used. Common is alpine, debian, ubuntu
- ENV - Environment variables
- RUN - Run commands/shell scripts, etc
- EXPOSE - Ports to expose
- CMD - Final command run when you launch a new container from image
- WORKDIR - Sets working directory (also could use 'RUN cd /some/path')

- COPY # Copies files from host to container

Build image from dockerfile (reponame can be whatever)

From the same directory as Dockerfile

```
$ docker image build -t [REPONAME] .
```

TIP: CACHE & ORDER

- If you re-run the build, it will be quick because everything is cached.
- If you change one line and re-run, that line and everything after will not be cached
- Keep things that change the most toward the bottom of the Dockerfile

EXTENDING DOCKERFILE

Custom Dockerfile for html page with nginx

```
FROM nginx:latest # Extends nginx so everything included in that image is included here
```

```
WORKDIR /usr/share/nginx/html
```

```
COPY index.html index.html
```

Build image from Dockerfile

```
$ docker image build -t nginx-website
```

Running it

```
$ docker container run -p 80:80 --rm nginx-website
```

Tag and push to Dockerhub

```
$ docker image tag nginx-website:latest btraversy/nginx-website:latest
```

```
$ docker image push bradtraversy/nginx-website
```

VOLUMES

Volume - Makes special location outside of container UFS. Used for databases

Bind Mount -Link container path to host path

Check volumes

```
$ docker volume ls
```

Cleanup unused volumes

```
$ docker volume prune
```

Pull down mysql image to test

```
$ docker pull mysql
```

Inspect and see volume

```
$ docker image inspect mysql
```

Run container

```
$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
```

Inspect and see volume in container

```
$ docker container inspect mysql
```

TIP: Mounts

- You will also see the volume under mounts
- Container gets its own unique location on the host to store that data
- Source: xxx is where it lives on the host

Check volumes

```
$ docker volume ls
```

There is no way to tell volumes apart for instance with 2 mysql containers, so we used named volumes

Named volumes (Add -v command)(the name here is mysql-db which could be anything)

```
$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var/lib/mysql mysql
```

Inspect new named volume

```
docker volume inspect mysql-db
```

BIND MOUNTS

- Can not use in Dockerfile, specified at run time (uses -v as well)
- ... run -v /Users/brad/stuff:/path/container (mac/linux)
- ... run -v //c:/Users/brad/stuff:/path/container (windows)

TIP: Instead of typing out local path, for working directory use \$(pwd):/path/container - On windows may not work unless you are in your users folder

Run and be able to edit index.html file (local dir should have the Dockerfile and the index.html)

```
$ docker container run -p 80:80 -v $(pwd):/usr/share/nginx/html nginx
```

Go into the container and check

```
$ docker container exec -it nginx bash
$ cd /usr/share/nginx/html
$ ls -al
```

You could create a file in the container and it will exist on the host as well

```
$ touch test.txt
```

DOCKER COMPOSE

- Configure relationships between containers
- Save our docker container run settings in easy to read file
- 2 Parts: YAML File (docker.compose.yml) + CLI tool (docker-compose)

1. docker.compose.yml - Describes solutions for

- containers
- networks
- volumes

2. docker-compose CLI - used for local dev/test automation with YAML files

Sample compose file (From Bret Fishers course)

```
version: '2'

# same as
# docker run -p 80:4000 -v $(pwd):/site bretfisher/jekyll-serve

services:
  jekyll:
    image: bretfisher/jekyll-serve
    volumes:
      - ./site
    ports:
      - '80:4000'
```

To run

```
docker-compose up
```

You can run in background with

```
docker-compose up -d
```

To cleanup

```
docker-compose down
```



Sphinxs commented

Awesome, thanks (:



panyaxbo commented

really helpful



porte22 commented

thank you!!



rbsrafa commented

Great job!



ashifiqbal2911 commented

Awesome



VisionsOfDrifting commented

Thank you for the tutorials!



sumitmanchanda01 commented

Thanks Brad !!



ohtkenneth commented

Fantastic tutorial and documentation. Thank you!



TechRancher commented

Thanks Brad very helpful!



zzCOMzz commented

Thanks Brad, ^^



wildcod commented

thanks



AnthD commented

cheers



iskandarsaleh commented

nice bro...thanks a lot..!



tripu commented

Thanks for this, and for the video tutorials!



sami93 commented

Thanks



pratapbunga commented

Ah, life-saver!! Thank you Brad ¹⁰⁰



petercr commented

Awesome thanks so much Brad 😎👍



devtechgen commented

Thanks a million



RyzorBent commented

Great stuff! :)



clissmancamacho commented

Thanks Brad! You Are Amazing!



Alhamou commented

thank you Brad !! شكراً استاذ براد



indefinitus commented

Thanks!! 🎉🎉🎉



rmfranciacastillo commented

This is awesome! Thanks Brad! Really appreciate your work



sinovic commented

Thank you for the tutorials!



mlounnaci commented

Great job! Thanks



devatsrs commented

Great job thanks



Nejini commented

Saw your video on youtube and started working on docker! Thanks for the video and this help sheet! :)



BrianLe1507 commented

You save my time. And your video about docker is easy to understand too . Thank so much, Brad.



DX9807 commented

Thanks Brad, this is awesome



sae13 commented



imran-ib commented

Thank You....



seksitha commented

Thank you!!!



ahmedsadman commented •

In windows, the bind mount can be done like:

```
docker container run -d -p 8080:80 -v  
f:/test:/usr/share/nginx/html --name nginx-website nginx
```



Awais-cb commented

Awwwwsome :D



efefregene commented

Salute



thearabbit commented

I have problem with Volume mongod.conf

```
docker run --name mongodb -v /data/db:/data/db -v /etc/mongo:/etc/mongo -p 27017:27017 -
```



naveenrawat51 commented

Thank you Brad



manucho007 commented •

To mount bind using Docker Toolbox on Windows 10 HomeEdition:

```
docker run -d --name nginx-website -v //d/dockercode://usr/share/nginx/html -p 8080:80 nginx
```



moeinrahimi commented

thanks brad



nikidev commented

Thank you Brad !



bruno-silva5 commented

Thank you a lot, Brad!



ZZhilong commented

Thank you Brad till now--2020



edvdbl commented

Thank you!



ablascoa commented

thank you!



dracorlll commented

thank you!!



Saspian commented

Thank you brad!



lifeeric commented

Updated Version

Docker Commands, Help & Tips

Show commands & management commands

```
$ docker
```

Docker version info

```
$ docker version
```

Show info like number of containers, etc

```
$ docker info
```

WORKING WITH CONTAINERS

Create and run a container in foreground

```
$ docker container run -it -p 80:80 nginx
```

Create and run a container in background

```
$ docker container run -d -p 80:80 nginx
```

Shorthand

```
$ docker container run -d -p 80:80 nginx
```

Naming Containers

```
$ docker container run -d -p 80:80 --name nginx-server nginx
```

TIP: WHAT RUN DID

- Looked for image called nginx in image cache
- If not found in cache, it looks to the default image repo on Dockerhub
- Pulled it down (latest version), stored in the image cache
- Started it in a new container
- We specified to take port 80- on the host and forward to port 80 on the container
- We could do "\$ docker container run --publish 8000:80 --detach nginx" to use port 8000
- We can specify versions like "nginx:1.09"

List running containers

```
$ docker container ls
```

OR

```
$ docker ps
```

List all containers (Even if not running)

```
$ docker container ls -a
```

Stop container

```
$ docker container stop [ID]
```

Stop all running containers

```
$ docker stop $(docker ps -aq)
```

Remove container (Can not remove running containers, must stop first)

```
$ docker container rm [ID]
```

To remove a running container use force(-f)

```
$ docker container rm -f [ID]
```

Remove multiple containers

```
$ docker container rm [ID] [ID] [ID]
```

Remove all containers

```
$ docker rm $(docker ps -aq)
```

Get logs (Use name or ID)

```
$ docker container logs [NAME]
```

List processes running in container

```
$ docker container top [NAME]
```

TIP: ABOUT CONTAINERS

Docker containers are often compared to virtual machines but they are actually just processes running on your host. On a Mac, Docker runs in a mini-VM so to see the processes you'll need to connect directly to that. On Linux however you can use "sudo docker ps" and see the processes directly

IMAGE COMMANDS

List the images we have pulled

```
$ docker image ls
```

We can also just pull down images

```
$ docker pull [IMAGE]
```

Remove image

```
$ docker image rm [IMAGE]
```

Remove all images

```
$ docker rmi $(docker images -a -q)
```

TIP: ABOUT IMAGES

- Images are app binaries and dependencies with meta data about the image data and how to run the image
- Images are not a complete OS. No kernel, kernel modules (drivers)
- Host provides the kernel, big difference between VM

Some sample container creation

NGINX:

```
$ docker container run -d -p 80:80 --name nginx nginx (-p 80:80 is optional as it runs on
```

APACHE:

```
$ docker container run -d -p 8080:80 --name apache httpd
```

MONGODB:

```
$ docker container run -d -p 27017:27017 --name mongo mongo
```

MYSQL:

```
$ docker container run -d -p 3306:3306 --name mysql --env MYSQL_ROOT_PASSWORD=123456 mysql
```

CONTAINER INFO

View info on container

```
$ docker container inspect [NAME]
```

Specific property (--format)

```
$ docker container inspect --format '{{ .NetworkSettings.IPAddress }}' [NAME]
```

Performance stats (cpu, mem, network, disk, etc)

```
$ docker container stats [NAME]
```

ACCESSING CONTAINERS

Create new nginx container and bash into

```
$ docker container run -it --name [NAME] nginx bash
```

- i = interactive Keep STDIN open if not attached
- t = tty - Open prompt

For Git Bash, use "winpty"

```
$ winpty docker container run -it --name [NAME] nginx bash
```

Run/Create Ubuntu container

```
$ docker container run -it --name ubuntu ubuntu
```

(no bash because ubuntu uses bash by default)

You can also make it so when you exit the container does not stay by using the -rm flag

```
$ docker container run --rm -it --name [NAME] ubuntu
```

Access an already created container, start with -ai

```
$ docker container start -ai ubuntu
```

Use exec to edit config, etc

```
$ docker container exec -it mysql bash
```

Alpine is a very small Linux distro good for docker

```
$ docker container run -it alpine sh
```

(use sh because it does not include bash)

(alpine uses apk for its package manager - can install bash if you want)

NETWORKING

"bridge" or "docker0" is the default network

Get port

```
$ docker container port [NAME]
```

List networks

```
$ docker network ls
```

Inspect network

```
$ docker network inspect [NETWORK_NAME]  
("bridge" is default)
```


Create network

```
$ docker network create [NETWORK_NAME]
```

or

```
$ docker network create --driver bridge [NETWORK_NAME]
```

Link to container, add to network

```
$ docker run -d --net=[NETWORK_NAME] --name mongodb mongo
```

Create container on network

```
$ docker container run -d --name [NAME] --network [NETWORK_NAME] nginx
```

Connect existing container to network

```
$ docker network connect [NETWORK_NAME] [CONTAINER_NAME]
```

Disconnect container from network

```
$ docker network disconnect [NETWORK_NAME] [CONTAINER_NAME]
```

Detach network from container

```
$ docker network disconnect
```

Delete/Remove network

To remove the network by name or id, multiple can be deleted:

```
$ docker network rm [NETWORK_NAME] [NETWORK_NAME]
```

IMAGE TAGGING & PUSHING TO DOCKERHUB

tags are labels that point to an image ID

```
$ docker image ls
```

You'll see that each image has a tag

Retag existing image

```
$ docker image tag nginx btraversy/nginx
```

Upload to dockerhub

```
$ docker image push btraversy/nginx
```

If denied, do

```
$ docker login
```

Add tag to new image

```
$ docker image tag bradtraversy/nginx bradtraversy/nginx:testing
```

DOCKERFILE PARTS

- FROM - The os used. Common is alpine, debian, ubuntu
- ENV - Environment variables
- RUN - Run commands/shell scripts, etc
- EXPOSE - Ports to expose
- CMD - Final command run when you launch a new container from image
- WORKDIR - Sets working directory (also could use 'RUN cd /some/path')
- COPY # Copies files from host to container

Build image from dockerfile (reponame can be whatever)

From the same directory as Dockerfile

```
$ docker image build -t [REPONAME] .
```

Benchmarking builds

```
$ DOCKER_BUILDKIT=1 docker image build -t [REPONAME] .
```

TIP: CACHE & ORDER

- If you re-run the build, it will be quick because everything is cached.
- If you change one line and re-run, that line and everything after will not be cached
- Keep things that change the most toward the bottom of the Dockerfile

EXTENDING DOCKERFILE

Custom Dockerfile for html page with nginx

```
FROM nginx:latest # Extends nginx so everything included in that image is included here
WORKDIR /usr/share/nginx/html
COPY index.html index.html
```

Build image from Dockerfile

```
$ docker image build -t nginx-website
```

Running it

```
$ docker container run -p 80:80 --rm nginx-website
```

Tag and push to Dockerhub

```
$ docker image tag nginx-website:latest btraversy/nginx-website:latest
```

```
$ docker image push bradtraversy/nginx-website
```

VOLUMES

Volume - Makes special location outside of container UFS. Used for databases

Bind Mount -Link container path to host path

Check volumes

```
$ docker volume ls
```

Cleanup unused volumes

```
$ docker volume prune
```

Pull down mysql image to test

```
$ docker pull mysql
```

Inspect and see volume

```
$ docker image inspect mysql
```

Run container

```
$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
```

Inspect and see volume in container

```
$ docker container inspect mysql
```

TIP: Mounts

- You will also see the volume under mounts
- Container gets its own unique location on the host to store that data
- Source: xxx is where it lives on the host

Check volumes

```
$ docker volume ls
```

There is no way to tell volumes apart for instance with 2 mysql containers, so we used named volumes

Named volumes (Add -v command)(the name here is mysql-db which could be anything)

```
$ docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var
```

Inspect new named volume

```
docker volume inspect mysql-db
```

BIND MOUNTS

- Can not use in Dockerfile, specified at run time (uses -v as well)
- ... run -v /Users/brad/stuff:/path/container (mac/linux)
- ... run -v //c/Users/brad/stuff:/path/container (windows)

TIP: Instead of typing out local path, for working directory use \$(pwd):/path/container - On windows make sure you are in your users folder

Run and be able to edit index.html file (local dir should have the Dockerfile and the index.html)

```
$ docker container run -p 80:80 -v $(pwd):/usr/share/nginx/html nginx
```

Go into the container and check

```
$ docker container exec -it nginx bash
$ cd /usr/share/nginx/html
$ ls -al
```

You could create a file in the container and it will exist on the host as well

```
$ touch test.txt
```

Link

```
$ docker run -d --name my-postgres postgres
```

now Link it with dot net

```
$ docker run -d -p 5000:5000 --link my-postgres:postgres btree/dotnet
```

DOCKER COMPOSE

- Configure relationships between containers
- Save our docker container run settings in easy to read file
- 2 Parts: YAML File (docker.compose.yml) + CLI tool (docker-compose)

1. docker.compose.yml - Describes solutions for

- containers
- networks
- volumes

2. docker-compose CLI - used for local dev/test automation with YAML files

Sample compose file (From Bret Fishers course)

```
version: '2'

# same as
# docker run -p 80:4000 -v $(pwd):/site bretfisher/jekyll-serve

services:
  jekyll:
    image: bretfisher/jekyll-serve
    volumes:
      - ./site
    ports:
      - '80:4000'
```

Example 2

```
version: '3'
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - '80:3000'
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
    ports:
      - '27017:27017'
```

To run

docker-compose up

You can run in background with

docker-compose up -d

To cleanup

docker-compose down

<https://gist.github.com/bradtraversy/89fad226dc058a41b596d586022a9bd3>

Things I wish I knew before I started using Docker

Commands

Init docker on mac os

```
boot2docker init
```

Restart docker vm on mac os

```
boot2docker up
```

Build new image from base dir

```
docker build -t hookio .
```

Run image you just built

```
docker run -p 49160:9999 hookio
```

Run same image in interactive mode

```
docker run -p 49160:9999 -i hookio
```

Curl running instance

```
curl $(boot2docker ip):49160
```

List all running containers

```
docker ps
```

Stop a container

```
docker stop <id>
```

Stop all containers

```
docker stop $(docker ps -a -q)
```

Delete all containers

```
docker rm $(docker ps -a -q)
```

Delete all images

```
docker rmi $(docker images -q)
```

Build docker-compose.yml file

```
docker-compose build
```

Start docker-compose.yml file

```
docker-compose up
```

Create bash shell to running instance by ID

```
docker exec -i -t 665b4a1e17b6 bash #by ID
```

Gotchas

- virtual box gives the instance its own network ip (probably in 192.168.*.* range)
- knowing this ip address may be important to your application
- if you crash the docker vm and restart, you'll have to re-export ENV variables (read output of `boot2docker up`)
- all configuration paths in your application should be based to `__dirname` (if not already)
- don't forget to run `docker-compose build` before `docker-compose up`

<https://gist.github.com/Marak/63f437e3946805f6473a>

Cheat Sheets¶

- [Cheat Sheets](#)
- [eBooks](#)
- [Documentation Browser for Software Developers](#)
 - [Dash for MacOS \(paid\)](#)
 - [Velocity \(Windows, paid\)](#)
 - [Zeal \(Windows, Linux, Free\)](#)
- [Kubernetes Knowledge Hubs and Glossary](#)
- [Kubectl Cheat Sheets](#)
- [Docker Cheat Sheets](#)
- [Git and GitHub Cheat Sheets](#)
 - [Git Flow Cheat Sheets](#)
 - [Sourcetree Cheat Sheet](#)
 - [GitKraken Git Cheat](#)
- [Ansible Cheat Sheets](#)
- [Packer and Terraform Cheat Sheets](#)
- [Linux Command Cheat Sheets](#)
 - [SSH Cheat Sheets](#)
- [OpenShift Cheat Sheets](#)
 - [Debezium Cheat Sheets](#)
- [Kubernetes Operator Cheat Sheets](#)
- [Kubernetes POD Cheat Sheets](#)
- [Buildah Cheat Sheets](#)
- [Helm Cheat Sheets](#)
- [Maven Cheat Sheets](#)
- [Gradle Cheat Sheets](#)
- [Jenkins Cheat Sheets](#)
- [JMeter Cheat Sheets](#)
- [Quarkus Cheat Sheets](#)
- [Markdown Cheat Sheets](#)
- [Machine Learning](#)
- [TypeScript](#)
- [Jupyter](#)
- [Postgres](#)
- [MariaDB and MySQL](#)
- [MongoDB](#)

Cheat Sheets

- [wizardzines.com](#) programming zines by Julia Evans
- [lzone.de/cheat-sheet](#)
- [cheatography.com](#)
- [LeCoupa/awesome-cheatsheets](#)
- [detailyang/awesome-cheatsheet](#)

- [Red Hat Developer cheat sheets](#) Browse through our collection of cheat sheets to help you develop with Red Hat products, which you can download for free as a Red Hat Developer member. You'll find handy guides on a range of the latest developer tools and technologies, including Kubernetes, microservices, containers, and more.
- blog.jromanmartin.io: ActiveMQ, Kafka, Strimzi and CodeReady Containers

eBooks

- [Red Hat Developer eBooks](#) Browse through our collection of eBooks to help you develop with Red Hat products, which you can download for free as a Red Hat Developer member. You'll find handy books on a range of the latest developer tools and technologies, including Kubernetes, microservices, containers, and more.
- [Transformation takes practice](#) Our experts understand this: When it comes to your unique business challenges, one size does not fit all. We can guide you through exercises and tools, like the ones within the Open Practice Library, that are right for where you are, right now.

Documentation Browser for Software Developers

Dash for MacOS (paid)

- [Dash for MacOS](#) Dash gives your Mac instant offline access to 200+ API documentation sets.
- [Dash Cheat Sheets](#)

Velocity (Windows, paid)

- [Velocity](#) Velocity gives your Windows desktop offline access to over 150 API documentation sets

Zeal (Windows, Linux, Free)

- [Zeal](#) Zeal is an offline documentation browser for software developers.

Kubernetes Knowledge Hubs and Glossary

- k8sref.io Kubernetes Reference
- [Kubernetes Research. Research documents on node instance types, managed services, ingress controllers, CNIs, etc.](#) A research hub to collect all knowledge around Kubernetes. Those are in-depth reports and comparisons designed to drive your decisions. Should you use GKE, AKS, EKS? How many nodes? What instance type?
- [Kubernetes Glossary](#)
- Kubernetes Reference: dev-k8sref-io.web.app Imports paths are not always easy to find for a resource. Get some help from this doc.

Kubectrl Cheat Sheets

- developers.redhat.com: Kubernetes Cheat Sheet
- kubernetes.io

- [linuxacademy](#)
- [fabric8 - kubectl](#)
- [intellipaat.com](#)
- [dzone: kubectl commands cheat sheet](#)
- [jimmysong.io: kubectl cheat sheet](#)
- [cheatsheet.dennyzhang.com: kubectl kubernetes free cheat sheet](#)
- [opensource.com: 9 kubectl commands sysadmins need to know](#) Keep these 9 critical kubectl commands handy to help you with troubleshooting and managing your Kubernetes cluster administration.
- [bluematador.com: kubectl cheatsheet](#)
- [dockerlabs.collabnix.com: Cheatsheet - Kubectl](#)
- [medium: Awesome Kubernetes Command-Line Hacks](#) Tips for you to kubectl like a pro
- [akhilsharma.work: kubectl Get Resource - Short Names](#)

Docker Cheat Sheets

- [dockerlux.github.io: Docker Cheat Sheet](#)
- [Dzone: docker cheat sheet](#)
- [developers.redhat.com: Containers Cheat Sheet](#)
- [github.com: Docker cheat Sheet](#)
- [linuxhero.tk: Docker command cheat sheet](#)
- [docker.com: Docker Cheat Sheet](#)

Git and GitHub Cheat Sheets

- [guides.github.com](#)
- [dev.to: Git & Github Cheatsheet](#)
- [git-scm.com: Git reference](#)
- [zeroturnaround.com: Git cheat sheet](#)
- [ndpsoftware.com: Interactive git cheat sheet](#)
- [The awesome git cheat sheet](#)
- [developers.redhat.com: Git cheat sheet](#)
- [atlassian.com: Git cheat sheet](#)
- [github.github.com/training-kit: Git cheat sheet](#)
- [education.github.com: Git cheat sheet](#)
- [dzone.com: refcard - getting started with git](#)
- [git-tower.com: Git cheat sheet](#)
- [rogerdudler.github.io: git - the simple guide](#) Just a simple guide for getting started with git. no deep shit ;)
 - [rogerdudler.github.io: git cheat sheet pdf](#)

Git Flow Cheat Sheets

- [Git-flow cheatsheet](#)

Sourcetree Cheat Sheet

- [Sourcetree Cheat Sheet](#)

GitKraken Git Cheat

- [GitKraken Git Cheat](#)

Ansible Cheat Sheets

- [Ansible Roles Explained | Cheat Sheet](#)
- [edureka.co: Ansible Cheat Sheet – A DevOps Quick Start Guide](#)
- [intellipaat.com: Ansible Basic Cheat Sheet](#)
- [mrupalmeiras: Ansible Cheat Sheet](#)
- [google.com/site/mrupalmeiras: Ansible Cheat Sheet](#)
- [Ansible k8s cheat sheet](#) The Ansible k8s module enables you to manage Kubernetes objects with Ansible playbooks.

Packer and Terraform Cheat Sheets

- [Packer Cheatsheet](#)
- [dzone: Terraform Cheat Sheet](#)
- [terraform.io: Terraform Commands](#)
- [github.com/scraly: Terraform Cheat sheet](#)
- [lzone.de: Terraform Cheat Sheet](#)
- [thedevelopsblog.co.uk: Terraform Cheat Sheet](#)
- [terraform-infraestructura.readthedocs.io: comandos](#)
- [acloudguru.com: The Ultimate Terraform Cheatsheet](#)

Linux Command Cheat Sheets

- [linuxide.com: Linux Commands Cheat Sheet](#)
- [commandlinefu.com](#)
- [opensource.com: 10 cheat sheets for Linux sysadmins](#)
- [curl cheat sheet for Linux and Unix users](#)
- [NetworkManager CLI cheatsheet](#)

SSH Cheat Sheets

- [ssh cheat sheet](#)
- [lzone.de: ssh cheat sheet](#)
- [pentestmonkey.net: ssh cheat sheet](#)
- [The SSH Commands Cheat Sheet for Linux SysAdmins / Users](#)
- [opensource.com: Learn advanced SSH commands with this cheat sheet](#)

OpenShift Cheat Sheets

- [mastertheboss.com: OpenShift Cheat Sheet](#)
- [developers.redhat.com: Red Hat OpenShift Container Platform Cheat Sheet](#)
- [github.com: Openshift cheat sheet 1](#)
- [gist.github.com: Openshift cheat sheet 2](#)
- [github.com: openshift cheat sheet 3](#)

- [monodot.co.uk: openshift cheat sheet 4](https://monodot.co.uk/openshift-cheat-sheet-4)
- openshift.tips

Debezium Cheat Sheets

- [developers.redhat.com: Debezium on OpenShift Cheat Sheet](https://developers.redhat.com/articles/2018/08/28/debezium-on-openshift-cheat-sheet) Debezium is a distributed open-source platform for change data capture. Start it up, point it at your databases, and your apps can start responding to all of the inserts, updates, and deletes that other apps commit to your databases. Debezium is durable and fast, so your apps can respond quickly and never miss an event, even when things go wrong. This cheat sheet covers how to deploy/create/run/update a Debezium Connector on OpenShift.

Kubernetes Operator Cheat Sheets

- [developers.redhat.com: Writing a Kubernetes Operator in Java using Quarkus - Cheat Sheet](https://developers.redhat.com/articles/2018/08/28/writing-a-kubernetes-operator-in-java-using-quarkus)
—

Kubernetes POD Cheat Sheets

- jimmysong.io/kubernetes-handbook/concepts/pod.html
- [https://dev.to/aurelievache: Understanding Kubernetes: part 1 – Pods](https://dev.to/aurelievache/understanding-kubernetes-part-1-pods)
- [garba.org: Kubernetes Pod Life Cycle Cheat Sheet](https://garba.org/kubernetes-pod-life-cycle-cheat-sheet)

Gradle Cheat Sheets

- [polyglotdeveloper.com: Gradle Cheat Sheet](https://polyglotdeveloper.com/Gradle-Cheat-Sheet)
- [eta-lang.org: Gradle Cheat Sheet](https://eta-lang.org/Gradle-Cheat-Sheet)
- [mingliang.me: Gradle Cheat Sheet](https://mingliang.me/Gradle-Cheat-Sheet)
- [rratliff.com: Gradle Cheat Sheet](https://rratliff.com/Gradle-Cheat-Sheet)
- [github.com/jahe: Gradle Cheat Sheet](https://github.com/jahe/Gradle-Cheat-Sheet)
- [github.com/jiffle: Gradle Cheat Sheet](https://github.com/jiffle/Gradle-Cheat-Sheet)

Jenkins Cheat Sheets

- [edureka.co: Jenkins Cheat Sheet](https://edureka.co/Jenkins-Cheat-Sheet)
 - [Jenkins Cheat Sheet](#)
- [medium: Jenkins Cheat Sheet](#)
- [cheatography.com: Jenkins Cheat Sheet](https://cheatography.com/Jenkins-Cheat-Sheet)

JMeter Cheat Sheets

- [Dzone Refcard: Getting Started with Apache JMeter](#)
- [Groovy Templates Cheat Sheet for JMeter](#) Need help with your Groovy templates? Check out this cheat sheet to help you get started with scripting in Apache JMeter.
- [JMeter Web Application Testing Cheatsheet](#)
- [CheatSheet for JMeter time Function Calls](#)
- [martkos-it.co.uk: JMeter Cheat Sheet](https://martkos-it.co.uk/JMeter-Cheat-Sheet) This jmeter cheat sheet provides gentle reminders of the usage of jmeter gui/non-gui. It includes installation/execution, plugins, shortcut keys and functions and variables.
 - [jmeter-testing-cheat-sheet-v10.pdf](#)
- [Cheat Sheet for Regular Expression in Jmeter](#)

Quarkus Cheat Sheets

- [Quarkus Cheat-Sheet](#)

Markdown Cheat Sheets

- markdownguide.org
 - [Markdown Cheat Sheet 1](#)
- [guides.github.com: Markdown Cheat Sheet 2](https://guides.github.com/Markdown-Cheat-Sheet-2)
- [Markdown Cheat Sheet 3](#)
- [Markdown Cheat Sheet 4](#)

Machine Learning

- [Machine Learning Glossary](#)

TypeScript

- [React+TypeScript Cheatsheets](#)

Jupyter

- [opensource.com: JupyterLab cheat sheet](#)

Postgres

- [postgrescheatsheet.com](#)

MariaDB and MySQL

- [opensource.com: MariaDB and MySQL cheat sheet](#)

MongoDB

- [developer.mongodb.com: MongoDB Cheat Sheet](#)

<https://redhatspain.com/cheatsheets/>

Docker Cheat Sheet



bogotobogo.com site search:

Container run

1. **docker create**: creates a writable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to STDOUT. This is similar to **docker run -d** except the container is never started. We can then use the **docker start <container_id>** command to start the container at any point.

```
$ docker create -t -i fedora bash
...
dff32a272ad4c94cd51419ee4f53c371e3c3a8cfb448a469634d4420e139d118

$ docker start -a -i dff32a272ad4c
[root@dff32a272ad4 /]#
```

i: interactive, keep STDIN open even if not attached

t: Allocate a pseudo-TTY

a: Attach container's STDOUT and STDERR and forward all signals to the process.

2. **docker rename** allows the container to be renamed.

```
$ docker rename my_container my_new_container
```

3. **docker run** creates and starts a container in one operation.

```
$ docker run -it ubuntu-ssh-k /bin/bash
```

4. **docker rm** deletes a container.


```
$ docker rm myfedora
```

5. **docker update** updates a container's resource limits. Example : updating multiple resource configurations for multiple containers:

```
$ docker update --cpu-shares 512 -m 300M dff32a272ad4 happy_kare
dff32a272ad4
happy_kare
```

Container info

1. **docker ps** shows running containers.

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
e2481c1bad5d	ubuntu-ssh-k:latest	"/bin/bash"	10 hours ago
Up 10 hours		hopeful_carson	

2. **docker logs** gets logs from container. (We can use a custom log driver, but logs is only available for json-file and journald in 1.10)

```
$ docker logs 839f66a78983
```

3. **docker inspect** looks at all the info on a container (including IP address). To get an IP address of a running container:

```
$ docker inspect --format '{{.NetworkSettings.IPAddress }}' $(docker ps -q)
172.17.0.5
```

4. **docker events** gets events from container.
5. **docker port** shows public facing port of container.
6. **docker top** shows running processes in container.
7. **docker stats** shows containers' resource usage statistics.
8. **docker diff** shows changed files in the container's FS.

Container start/stop

1. **docker start** starts a container so it is running.
2. **docker stop** stops a running container.
3. **docker restart** stops and starts a container.
4. **docker pause** pauses a running container, "freezing" it in place.
5. **docker unpause** will unpause a running container.
6. **docker wait** blocks until running container stops.
7. **docker kill** sends a SIGKILL to a running container.

8. **docker attach** will connect to a running container.

Or we can use the following to get tty:

```
$ docker exec -i -t my-nginx-1 /bin/bash
```

Note that we used **exec** to run a command in a "running" container!

Image create/remove

1. **docker images** : shows all images.
2. **docker import** : creates an image from a tarball.
3. **docker build** : creates image from Dockerfile.
4. **docker commit** : creates image from a container, by default, the container being committed and its processes will be paused while the image is committed. This reduces the likelihood of encountering data corruption during the process of creating the commit. As an example, let's install nginx to the ubuntu container:

```
$ docker run -it ubuntu /bin/bash
root@bf5d24821dfa:/# apt-get update
root@bf5d24821dfa:/# apt-get install nginx
```

While the container is running, on another terminal, we can commit the change to another image:

```
$ docker ps
CONTAINER ID      IMAGE      COMMAND
bf5d24821dfa      ubuntu     "/bin/bash"

$ docker commit bf5d24821dfa ubuntu-nginx

$ docker images
REPOSITORY      TAG      IMAGE ID
ubuntu-nginx     latest   91cc8b745f5e

$ docker run -it ubuntu-nginx
root@9644a814e95a:/#

$ docker ps
CONTAINER ID      IMAGE      COMMAND
9644a814e95a      ubuntu-nginx  "/bin/bash"
bf5d24821dfa      ubuntu     "/bin/bash"
```

Note the the new container dit not start "nginx" server. So, we can **commit** a container with new **CMD** and **EXPOSE** instructions:

```
root@9644a814e95a:/# exit
```

```
$ docker commit -m 'added nginx start' --change='CMD ["nginx", "-g daemon off;"]' -c "EXPOSE 80" 9644a814e95a ubuntu-nginx:version2
```

1. -m 'added nginx start' creates a comment for this commit.
2. --change='CMD ...' is changing the CMD command, which is what the image will run when it is first started up. In this example, we are telling the image to run nginx in the foreground. Most base os images have CMD set to bash so we can interact with the os when attaching.
3. We used "EXPOSE" to get the port exposed to the host.
4. "9644a814e95a" is the name of the container we want to commit from.
5. "ubuntu-nginx:version2" is our name for the new image. We appended version to the name of the container.

We can now view our new image with the following command:

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID
ubuntu-nginx	version2	61918a284221

Run the new image:

```
$ docker run -idt ubuntu-nginx:version2
```

```
$ ps aux | grep -v grep |grep nginx
```

```
root      12546  0.0  0.2 124972  9560 pts/25   Ss+  10:21   0:00 nginx:
master process nginx -g daemon off;
www-data  12565  0.0  0.0 125332  3040 pts/25   S+   10:21   0:00 nginx:
worker process
www-data  12566  0.0  0.0 125332  3040 pts/25   S+   10:21   0:00 nginx:
worker process
```

```
$ docker inspect --format '{{.NetworkSettings.IPAddress}}' $(docker ps -q)
172.17.0.2
```

We can see the index page:



5. **docker rmi** : removes an image
6. **docker load** : loads an image from a tar archive as STDIN, including images and tags .
7. **docker save** : saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions.

Container info

Getting Docker Container's IP Address from host machine:

```
$ docker inspect --format '{{.NetworkSettings.IPAddress}}' $(docker ps -q)
172.17.0.2
172.17.0.3
```

Image info

1. **docker history** shows history of image.
2. **docker tag** tags an image to a name (local or registry).

Network create/remove/info/connection

1. **docker network create**
2. **docker network rm**
3. **docker network ls**
4. **docker network inspect**
5. **docker network connect**
6. **docker network disconnect**

Docker Repo

1. **docker login** to login to a registry. (see docker push)

```
$ docker login --username=dockerbogo
Password:
Login Succeeded
```

2. **docker logout** to logout from a registry.

```
$ docker logout
Removing login credentials for https://index.docker.io/v1/
```

3. **docker search** searches registry for image.

```
$ docker search mysql
NAME                DESCRIPTION
STARS               OFFICIAL    AUTOMATED
mysql               MySQL is a widely used, open-source relati...
2797                [OK]
mysql/mysql-server  Optimized MySQL Server Docker images. Crea...
182                 [OK]
...
```

4. **docker pull** pulls an image from registry to local machine.

```
$ docker pull ubuntu
latest: Pulling from ubuntu
20ee58809289: Pull complete
f905badeb558: Pull complete
119df6bf2a3a: Pull complete
94d6eea646bc: Pull complete
bb4eabee84bf: Pull complete
Digest:
sha256:85af8b61adffea165e84e47e0034923ec237754a208501fce5dbeecbb197062c
Status: Downloaded newer image for ubuntu:latest
```

Docker images can consist of multiple layers. In the example above, the image consists of five layers.

5. **docker push** pushes an image to the registry from local machine. First, we'll install nginx on top of ubuntu, and then commit it. After login to DockerHub, we'll tag the image and push it:

```
$ docker run -it ubuntu:latest /bin/bash

root@846346c3482f:/# apt-get update
root@846346c3482f:/# apt-get install nginx

$ docker commit 846346c3482f ubuntu-with-nginx

$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-with-nginx	latest	de412f7faba5	16 minutes ago	212 MB

```
$ docker login --username=dockerbogo
```

Password:

Login Succeeded

```
$ docker tag de412f7faba5 dockerbogo/ubuntu-with-nginx:myfirstpush
```

```
$ docker push dockerbogo/ubuntu-with-nginx:myfirstpush
```

The push refers to a repository [docker.io/dockerbogo/ubuntu-with-nginx]

44f391d61c42: Pushed

73e5d2de6e3e: Mounted from library/ubuntu

08f405d988e4: Mounted from library/ubuntu

511ddc11cf68: Mounted from library/ubuntu

a1a54d352248: Mounted from library/ubuntu

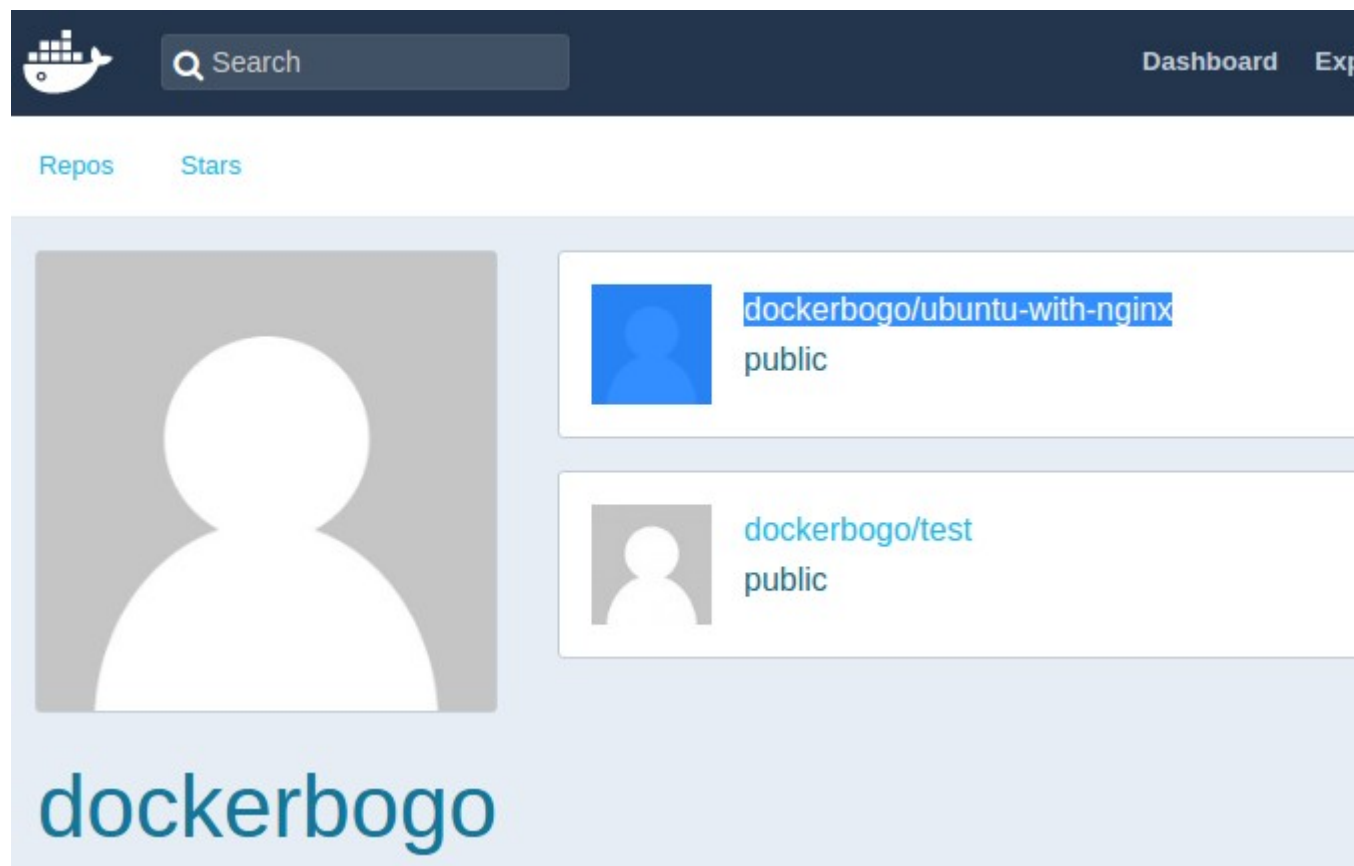
9d3227c1793b: Mounted from library/ubuntu

myfirstpush: digest:

sha256:a24d914b8e3a013987314054b2c2409e99c00384342b25ee7dba4c51cb1ea49a

size: 1569

We can check if the push was successful:



Refs

<https://docs.docker.com/engine/reference/commandline/docker/>

Docker CheatSheet

Share the Love



Installation

Linux

```
curl -sSL https://get.docker.com/ | sh
```

Mac

Use this link to download the dmg.

<https://download.docker.com/mac/stable/Docker.dmg>

Windows

Use the msi installer:

<https://download.docker.com/win/stable/InstallDocker.msi>

Docker Registries & Repositories

Login to a Registry

```
docker login
```

```
docker login localhost:8080
```

Logout from a Registry.

```
docker logout
```

```
docker logout localhost:8080
```

Searching an Image

```
docker search nginx
```

```
docker search nginx --stars=3 --no-trunc busybox
```

Pulling an Image

```
docker pull nginx
```

```
docker pull eon01/nginx localhost:5000/myadmin/nginx
```

Pushing an Image

```
docker push eon01/nginx
```

```
docker push eon01/nginx localhost:5000/myadmin/nginx
```

Running Containers

Creating a Container

```
docker create -t -i eon01/infinite --name infinite
```


Running a Container

```
docker run -it --name infinite -d eon01/infinite
```

Renaming a Container

```
docker rename infinite infinity
```

Removing a Container

```
docker rm infinite
```

Updating a Container

```
docker update --cpu-shares 512 -m 300M infinite
```

Starting & Stopping Containers

Starting

```
docker start nginx
```

Stopping

```
docker stop nginx
```

Restarting

```
docker restart nginx
```

Pausing

```
docker pause nginx
```

Unpausing

```
docker unpause nginx
```

Blocking a Container

```
docker wait nginx
```

Sending a SIGKILL

```
docker kill nginx
```

Connecting to an Existing Container

```
docker attach nginx
```

Getting Information about Containers

Running Containers

```
docker ps
```

```
docker ps -a
```

Container Logs

```
docker logs infinite
```

Inspecting Containers

```
docker inspect infinite
```

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)
```

Containers Events

```
docker events infinite
```

Public Ports

```
docker port infinite
```

Running Processes

```
docker top infinite
```

Container Resource Usage

```
docker stats infinite
```

Inspecting changes to files or directories on a container's filesystem

```
docker diff infinite
```

Manipulating Images

Listing Images

```
docker images
```

Building Images

```
docker build .
```

```
docker build github.com/creack/docker-firefox
```

```
docker build - < Dockerfile
```

```
docker build - < context.tar.gz
```

```
docker build -t eon/infinite .
```

```
docker build -f myOtherDockerfile .
```

```
curl example.com/remote/Dockerfile | docker build -f - .
```

Removing an Image

```
docker rmi nginx
```

Loading a Tarred Repository from a File or the Standard Input Stream

```
docker load < ubuntu.tar.gz
```

```
docker load --input ubuntu.tar
```

Save an Image to a Tar Archive

```
docker save busybox > ubuntu.tar
```

Showing the History of an Image

```
docker history
```

Creating an Image From a Container

```
docker commit nginx
```

Tagging an Image

```
docker tag nginx eon01/nginx
```

Pushing an Image

```
docker push eon01/nginx
```

Networking

Creating Networks

```
docker network create -d overlay MyOverlayNetwork
```

```
docker network create -d bridge MyBridgeNetwork
```

```
docker network create -d overlay \
  --subnet=192.168.0.0/16 \
  --subnet=192.170.0.0/16 \
  --gateway=192.168.0.100 \
  --gateway=192.170.0.100 \
  --ip-range=192.168.1.0/24 \
  --aux-address="my-router=192.168.1.5" --aux-address="my-switch=192.168.1.6" \
  --aux-address="my-printer=192.170.1.5" --aux-address="my-nas=192.170.1.6" \
  MyOverlayNetwork
```

Removing a Network

```
docker network rm MyOverlayNetwork
```

Listing Networks

```
docker network ls
```

Getting Information About a Network

```
docker network inspect MyOverlayNetwork
```

Connecting a Running Container to a Network

```
docker network connect MyOverlayNetwork nginx
```

Connecting a Container to a Network When it Starts

```
docker run -it -d --network=MyOverlayNetwork nginx
```

Disconnecting a Container from a Network

```
docker network disconnect MyOverlayNetwork nginx
```

Cleaning Docker

Removing a Running Container

```
docker rm nginx
```

Removing a Container and its Volume

```
docker rm -v nginx
```

Removing all Exited Containers

```
docker rm $(docker ps -a -f status=exited -q)
```

Removing All Stopped Containers

```
docker rm $(docker ps -a -q)
```

Removing a Docker Image

```
docker rmi nginx
```

Removing Dangling Images

```
docker rmi $(docker images -f dangling=true -q)
```

Removing all Images

```
docker rmi $(docker images -a -q)
```

Removing all untagged images

```
docker rmi -f $(docker images | grep "^<none>" | awk "{print $3}")
```

Stopping & Removing all Containers

```
docker stop $(docker ps -a -q) && docker rm $(docker ps -a -q)
```

Removing Dangling Volumes

```
docker volume rm $(docker volume ls -f dangling=true -q)
```

Docker Swarm

Installing Docker Swarm

```
curl -ssl https://get.docker.com | bash
```

Initializing the Swarm

```
docker swarm init --advertise-addr 192.168.10.1
```

Getting a Worker to Join the Swarm

```
docker swarm join-token worker
```

Getting a Manager to Join the Swarm

```
docker swarm join-token manager
```

Listing Services

```
docker service ls
```

Listing nodes

```
docker node ls
```

Creating a Service

```
docker service create --name vote -p 8080:80 instavote/vote
```

Listing Swarm Tasks

```
docker service ps
```

Scaling a Service

```
docker service scale vote=3
```

Updating a Service

```
docker service update --image instavote/vote:movies vote
```

```
docker service update --force --update-parallelism 1 --update-delay 30s nginx
```

```
docker service update --update-parallelism 5--update-delay 2s --image instavote/  
vote:indent vote
```

```
docker service update --limit-cpu 2 nginx
```

```
docker service update --replicas=5 nginx
```

<http://dockercheatsheet.painlessdocker.com/>

The Ultimate Docker Cheat Sheet

Complete Docker CLI



Cheatsheet for Docker

Run a new Container

Start a new Container from an Image

```
docker run IMAGE  
docker run nginx
```

...and assign it a name

```
docker run --name CONTAINER IMAGE  
docker run --name web nginx
```

...and map a port

```
docker run -p HOSTPORT:CONTAINERPORT IMAGE  
docker run -p 8080:80 nginx
```

...and map all ports

```
docker run -P IMAGE  
docker run -P nginx
```

...and start container in background

```
docker run -d IMAGE  
docker run -d nginx
```

...and assign it a hostname

```
docker run --hostname HOSTNAME IMAGE  
docker run --hostname srv nginx
```

...and add a dns entry

```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container

```
docker run -v HOSTDIR:TARGETDIR IMAGE  
docker run -v ~/.:/usr/share/nginx/html nginx
```

...but change the entrypoint

```
docker run -it --entrypoint EXECUTABLE IMAGE  
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers

```
docker ps
```

Show a list of all containers

```
docker ps -a
```

Delete a container

```
docker rm CONTAINER  
docker rm web
```

Delete a running container

```
docker rm -f CONTAINER  
docker rm -f web
```

Delete stopped containers

```
docker container prune
```

Stop a running container

```
docker stop CONTAINER  
docker stop web
```

Start a stopped container

```
docker start CONTAINER  
docker start web
```

Copy a file from a container to the host

```
docker cp CONTAINER:SOURCE TARGET  
docker cp web:/index.html index.html
```

Copy a file from the host to a container

```
docker cp TARGET CONTAINER:SOURCE  
docker cp index.html web:/index.html
```

Start a shell inside a running container

```
docker exec -it CONTAINER EXECUTABLE  
docker exec -it web bash
```

Rename a container

```
docker rename OLD_NAME NEW_NAME  
docker rename 096 web
```

Create an image out of container

```
docker commit CONTAINER  
docker commit web
```

Container Management CLIs

Container management co

command	desc
<code>docker create image [command]</code> <code>docker run image [command]</code>	creat = cr
<code>docker start container...</code> <code>docker stop container...</code> <code>docker kill container...</code> <code>docker restart container...</code>	start grace kill (S = st
<code>docker pause container...</code> <code>docker unpause container...</code>	suspe resum
<code>docker rm [-f³] container...</code>	destr

²send SIGTERM to the main process + SIGKILL 1

³-f allows removing running containers (= `docker`

Inspecting The Container

Inspecting the container

command	description
<code>docker ps</code>	list running containers
<code>docker ps -a</code>	list all containers
<code>docker logs [-f⁶] container</code>	show container logs (streaming)
<code>docker top container [ps options]</code>	list the processes running inside the container
<code>docker diff container</code>	show the changes to the container's filesystem
<code>docker inspect container...</code>	show detailed information about the container (in json)

Interacting with Container

Interacting with the cont

command	descripti
<code>docker attach container</code>	attach to (stdin/std
<code>docker cp container:path hostpath </code> <code>docker cp hostpath - container:path</code>	copy files copy files
<code>docker export container</code>	export th the conta
<code>docker exec container args...</code>	run a con container
<code>docker wait container</code>	wait until and retur
<code>docker commit container image</code>	commit a (snapshot

Image Management Commands

Image management commands

command	description
<code>docker images</code> <code>docker history image</code> <code>docker inspect image...</code>	list all local images show the history of an image (list of ancestor images) show low-level details about an image (in json format)
<code>docker tag image tag</code>	tag an image
<code>docker commit container image</code> <code>docker import url - [tag]</code>	create an image from a container create an image from a tar archive
<code>docker rmi image...</code>	delete images

Image Transfer Commands

Image transfer commands

Using the registry API

<code>docker pull repo[:tag]...</code>	pull an image/
<code>docker push repo[:tag]...</code>	push an image
<code>docker search text</code>	search an image
<code>docker login ...</code>	login to a registry
<code>docker logout ...</code>	logout from a registry

Manual transfer

<code>docker save repo[:tag]...</code>	export an image
<code>docker load</code>	load images from a tar file
<code>docker-ssh¹⁰ ...</code>	proposed script for between two docker hosts

Builder Main Commands

Builder main commands

command	description
FROM <i>image scratch</i>	base image for the build
MAINTAINER <i>email</i>	name of the maintainer
COPY <i>path dst</i>	copy <i>path</i> from the context into the container at <i>dst</i>
ADD <i>src dst</i>	same as COPY but understands wildcards and accepts http urls
RUN <i>args. . .</i>	run an arbitrary command inside the container
USER <i>name</i>	set the default username
WORKDIR <i>path</i>	set the default working directory
CMD <i>args. . .</i>	set the default command
ENV <i>name value</i>	set an environment variable

The Docker CLI

Manage images

docker build

```
docker build [options] .  
  -t "app/container_name"    # name
```

Create an image from a Dockerfile.

docker run

```
docker run [options] IMAGE  
  # see `docker create` for options
```

Run a command in an image.

Manage containers

docker create

```
docker create [options] IMAGE  
-a, --attach                # attach stdout/err  
-i, --interactive           # attach stdin (interactive)  
-t, --tty                   # pseudo-tty  
    --name NAME              # name your image  
-p, --publish 5000:5000     # port map  
    --expose 5432            # expose a port to linked containers  
-P, --publish-all          # publish all ports  
    --link container:alias  # linking  
-v, --volume `pwd`:/app     # mount (absolute paths needed)  
-e, --env NAME=hello        # env vars
```

Example

```
$ docker create --name app_redis_1 \  
  --expose 6379 \  
  redis:3.0.2
```

Create a container from an image.

docker exec

```
docker exec [options] CONTAINER COMMAND  
-d, --detach                # run in background  
-i, --interactive           # stdin  
-t, --tty                   # interactive
```

Example

```
$ docker exec app_web_1 tail logs/development.log  
$ docker exec -t -i app_web_1 rails c
```


Run commands in a container.

docker start

```
docker start [options] CONTAINER
-a, --attach          # attach stdout/err
-i, --interactive     # attach stdin
```

```
docker stop [options] CONTAINER
```

Start/stop a container.

docker ps

```
$ docker ps
$ docker ps -a
$ docker kill $ID
```

Manage containers using ps/kill.

Images

docker images

```
$ docker images
REPOSITORY    TAG       ID
ubuntu        12.10     b750fe78269d
me/myapp      latest    7b2431a8d968
```

```
$ docker images -a  # also show intermediate
```

Manages images.

docker rmi

```
docker rmi b750fe78269d
```

Deletes images.

Also see

- [Getting Started](#) (*docker.io*)

Dockerfile

Inheritance

```
FROM ruby:2.2.2
```

Variables

```
ENV APP_HOME /myapp
RUN mkdir $APP_HOME
```

Initialization

```
RUN bundle install
```

```
WORKDIR /myapp
```

```
VOLUME ["/data"]  
# Specification for mount point
```

```
ADD file.xyz /file.xyz  
COPY --chown=user:group host_file.xyz /path/container_file.xyz
```

Onbuild

```
ONBUILD RUN bundle install  
# when used with another file
```

Commands

```
EXPOSE 5900  
CMD ["bundle", "exec", "rails", "server"]
```

Entrypoint

```
ENTRYPOINT ["executable", "param1", "param2"]  
ENTRYPOINT command param1 param2
```

Configures a container that will run as an executable.

```
ENTRYPOINT exec top -b
```

This will use shell processing to substitute shell variables, and will ignore any CMD or docker run command line arguments.

Metadata

```
LABEL version="1.0"
```

```
LABEL "com.example.vendor"="ACME Incorporated"  
LABEL com.example.label-with-value="foo"
```

```
LABEL description="This text illustrates \  
that label-values can span multiple lines."
```

See also

- <https://docs.docker.com/engine/reference/builder/>

docker-compose

Basic example

```
# docker-compose.yml  
version: '2'
```

```
services:
  web:
    build: .
    # build from Dockerfile
    context: ./Path
    dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./code
  redis:
    image: redis
```

Commands

```
docker-compose start
docker-compose stop
```

```
docker-compose pause
docker-compose unpause
```

```
docker-compose ps
docker-compose up
docker-compose down
```

Reference

Building

```
web:
  # build from Dockerfile
  build: .

  # build from custom Dockerfile
  build:
    context: ./dir
    dockerfile: Dockerfile.dev

  # build from image
  image: ubuntu
  image: ubuntu:14.04
  image: tutum/influxdb
  image: example-registry:4000/postgresql
  image: a4bc65fd
```

Ports

```
ports:
  - "3000"
  - "8000:80" # guest:host
```

```
# expose ports to linked services (not to host)
expose: ["3000"]
```

Commands

```
# command to execute
```

```
command: bundle exec thin -p 3000
command: [bundle, exec, thin, -p, 3000]

# override the entrypoint
entrypoint: /app/start.sh
entrypoint: [php, -d, vendor/bin/phpunit]
```

Environment variables

```
# environment vars
environment:
  RACK_ENV: development
environment:
  - RACK_ENV=development

# environment vars from file
env_file: .env
env_file: [.env, .development.env]
```

Dependencies

```
# makes the `db` service available as the hostname `database`
# (implies depends_on)
links:
  - db:database
  - redis

# make sure `db` is alive before starting
depends_on:
  - db
```

Other options

```
# make this service extend another
extends:
  file: common.yml # optional
  service: webapp

volumes:
  - /var/lib/mysql
  - ./_data:/var/lib/mysql
```

Advanced features

Labels

```
services:
  web:
    labels:
      com.example.description: "Accounting web app"
```

DNS servers

```
services:
  web:
    dns: 8.8.8.8
    dns:
```

- 8.8.8.8
- 8.8.4.4

Devices

```
services:
  web:
    devices:
      - "/dev/ttyUSB0:/dev/ttyUSB0"
```

External links

```
services:
  web:
    external_links:
      - redis_1
      - project_db_1:mysql
```

Hosts

```
services:
  web:
    extra_hosts:
      - "somehost:192.168.1.100"
```

services

To view list of all the services running in swarm

```
docker service ls
```

To see all running services

```
docker stack services stack_name
```

to see all services logs

```
docker service logs stack_name service_name
```

To scale services quickly across qualified node

```
docker service scale stack_name_service_name=replicas
```

clean up

To clean or prune unused (dangling) images

```
docker image prune
```

To remove all images which are not in use containers , add -a

```
docker image prune -a
```

To Purge your entire system

```
docker system prune
```

To leave swarm

```
docker swarm leave
```

To remove swarm (deletes all volume data and database info)

```
docker stack rm stack_name
```

To kill all running containers

```
docker kill $(docekr ps -q )
```

Contributor -

[Sangam biradar](#) - Docker Community Leader

<https://dockerlabs.collabnix.com/docker/cheatsheet/>