1.)[How to remove old Docker containers](https://stackoverflow.com/questions/17236796/how-to-remove-old-docker-containers)

Remove old containers, Images...

**Also, consider cleaning orphaned docker volumes.** I often find that they consume much more space than old containers and old images.

Good script for removing orphaned docker volumes is available at: [github.com/chadoe/docker-cleanup-volumes](https://github.com/chadoe/docker-cleanup-volumes).

docker ps -a | grep "minute\|hours" | awk '{print $1}' | xargs docker rm

docker ps -a | grep Exited | awk '{print $1}'

**UPDATED** **2017** (NEWEST)

docker container prune

This - **2017** (OLD) way

To remove **ALL STOPPED CONTAINERS**

docker rm $(docker ps -aq)

To remove **ALL CONTAINERS** **(STOPPED AND NON STOPPED)**

docker rm -f $(docker ps -a -q)

The basic steps to stop/remove all containers and images

1. List all the containers

docker ps -aq

1. Stop all running containers

docker stop $(docker ps -aq)

1. Remove all containers

docker rm $(docker ps -aq)

1. Remove all images

docker rmi $(docker images -q)

* -q prints just the container ids (without column headers)
* -f allows you to filter your list of printed containers (in this case we are filtering to only show exited containers)

|  |  |
| --- | --- |
|  | Remove all containers created from a certain image:  docker rm $(docker ps -a | awk '/myimage:mytag/{print $1}') |

# 2.)[How does one remove an image in Docker?](https://stackoverflow.com/questions/17665283/how-does-one-remove-an-image-in-docker)

docker info shows the actual amount of images,

**docker images -a -q | sort | uniq | wc -l** it should be the same result (-a for all images, -q for only ids).

### **Remove single image**

docker rmi image\_name:version/image-id

*For multiple images*

docker rm 70c0e19168cf c2ce80b62174

### **Remove all images**

docker rmi $(docker images -qf "dangling=true")

### **Kill containers and remove them:**

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

Note: Replace *kill* with *stop* for graceful shutdown

### **Remove all images except "my-image"**

Use grep to remove all except my-image and ubuntu

docker rmi $(docker images | grep -v 'ubuntu\|my-image' | awk {'print $3'})

Or (without awk)

docker rmi $(docker images --quiet | grep -v $(docker images --quiet ubuntu:my-image))

**Remove dangling images**   
Dangling images are layers that have no relationship to any tagged images as the Docker images are constituted of multiple images.

1. docker rmi -f $(docker images -f dangling=true -q)
2. **Remove all Docker images**
3. docker rmi -f $(docker images -a -q)

## **Removing Volumes**

To list volumes run docker volume ls

1. **Remove a specific volume**
2. docker volume rm VOLUME\_NAME
3. **Remove dangling volumes**
4. docker volume rm $(docker volume ls -f dangling=true -q)
5. **Remove a container and its volumes**

docker rm -v CONTAINER\_NAME

# 2.1)[How to remove old and unused Docker images](https://stackoverflow.com/questions/32723111/how-to-remove-old-and-unused-docker-images)

When running docker a long time, there are a lot of images in system. How can I remove all unused docker images at once safety to free up the storage?

In addition, I also want to remove images pulled months ago, which have correct TAG

So, I'm not asking for removing untagged images only. I'm searching for a way to remove general unused images, which includes both untagged and other images such as pulled months ago with correct TAG

# 🡪

|  |  |
| --- | --- |
| accepted | Update Sept. 2016 for docker upcoming docker 1.13: [PR 26108](https://github.com/docker/docker/pull/26108) and [commit 86de7c0](https://github.com/docker/docker/commit/86de7c000f5d854051369754ad1769194e8dd5e1) introduce a few new commands to help facilitate visualizing how much space the docker daemon data is taking on disk and allowing for easily cleaning up "unneeded" excess.  [**docker system prune**](https://docs.docker.com/engine/reference/commandline/system_prune/) will delete ALL dangling data (i.e. In order: containers stopped, volumes without containers and images with no containers). Even unused data, with -a option.  You also have:   * [docker container prune](https://docs.docker.com/engine/reference/commandline/container_prune/) * [docker image prune](https://docs.docker.com/engine/reference/commandline/image_prune/) * [docker network prune](https://docs.docker.com/engine/reference/commandline/network_prune/) * [docker volume prune](https://docs.docker.com/engine/reference/commandline/volume_prune/)   For *unused* images, use docker image prune -a (for removing dangling *and* ununsed images). Warning: '*unused*' means "images not referenced by any container": be careful before using -a. |

I usually do:

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

I have an [alias for removing those [dangling images]8](https://github.com/docker/docker/blob/634a848b8e3bdd8aed834559f3b2e0dfc7f5ae3a/man/docker-images.1.md#options): drmi

The dangling=true filter finds unused images

That way, any intermediate image no longer referenced by a labelled image is removed.

I do the same **first** for [exited processes (containers)](https://github.com/VonC/b2d/blob/b010ab51974ac7de6162cdcbff795d7b9e84fd67/.bash_aliases#L21)

alias drmae='docker rm $(docker ps -qa --no-trunc --filter "status=exited")'

[Jess Frazelle (jfrazelle)](https://github.com/jfrazelle) has the [bashrc function](https://github.com/jfrazelle/dotfiles/blob/a7fd3df6ab423e6dd04f27727f653753453db837/.dockerfunc#L8-L11):

dcleanup(){

docker rm -v $(docker ps --filter status=exited -q 2>/dev/null) 2>/dev/null

docker rmi $(docker images --filter dangling=true -q 2>/dev/null) 2>/dev/null

}

* **Delete all Exited Containers**
* docker rm $(docker ps -q -f status=exited)
* **Delete all Stopped Containers**
* docker rm $(docker ps -a -q)
* **Delete All Running and Stopped Containers**
* docker stop $(docker ps -a -q)
* docker rm $(docker ps -a -q)
* **Remove all containers, without any criteria**
* docker container rm $(docker container ps -aq)
* But, in version 1.13 and above, for complete system and cleanup, we can directly user the following command:
* docker system prune
* All unused containers, images, networks and volumes will get deleted. We can also do this using the following commands that clean up the individual components:
* docker container prune
* docker image prune
* docker network prune
* docker volume prune

alias docker-clean=' \

docker container prune -f ; \

docker image prune -f ; \

docker network prune -f ; \

docker volume prune -f '

|  |
| --- |
| Delete stopped (exited) containers:  $ docker ps --no-trunc -aqf "status=exited" | xargs docker rm  Delete unused (dangling) images:  $ docker images --no-trunc -aqf "dangling=true" | xargs docker rmi  If you have exercised ***extreme caution*** with regard to ***irrevocable data loss***, then you can delete unused (dangling) volumes (v1.9 and up):  $ docker volume ls -qf "dangling=true" | xargs docker volume rm  Here they are in a convenient shell alias:  alias docker-clean=' \  docker ps --no-trunc -aqf "status=exited" | xargs docker rm ; \  docker images --no-trunc -aqf "dangling=true" | xargs docker rmi ; \  docker volume ls -qf "dangling=true" | xargs docker volume rm' |

|  |  |
| --- | --- |
|  | 🡪I'm using this command:  export BEFORE\_DATETIME=$(date --date='10 weeks ago' +"%Y-%m-%dT%H:%M:%S.%NZ")  docker images -q | while read IMAGE\_ID; do  export IMAGE\_CTIME=$(docker inspect --format='{{.Created}}' --type=image ${IMAGE\_ID})  if [[ "${BEFORE\_DATETIME}" > "${IMAGE\_CTIME}" ]]; then  echo "Removing ${IMAGE\_ID}, ${BEFORE\_DATETIME} is earlier then ${IMAGE\_CTIME}"  docker rmi -f ${IMAGE\_ID};  fi;  done  This will remove all images that their creation time is greater then 10 weeks ago.  🡪Remove old containers weeks ago.  docker rm $(docker ps -a | grep "weeks" | awk '{ print $1; }')  Remove old images weeks ago. Be careful. This will remove base images which was created weeks ago but which your new images might be using.  docker rmi $(docker images | grep 'weeks' | awk '{ print $3; }') |

# 3.)[Docker - copy file from container to host](https://stackoverflow.com/questions/22049212/docker-copy-file-from-container-to-host)

I'm thinking of using docker to build my dependencies on a CI server, so that I don't have to install all the runtimes and libraries on the agents themselves. To achieve this I would need to copy the build artifacts that are built inside the container back into the host.

In order to copy a file from a container to the host, you can use the command

docker cp <containerId>:/file/path/within/container /host/path/target

🡪docker run --rm -iv${PWD}:/host-volume my-image sh -s <<EOF

chown -v $(id -u):$(id -g) \*.txt

cp -va \*.txt /host-volume

EOF

changed ownership of '/host-volume/bar.txt' to 10335:11111

changed ownership of '/host-volume/qux.txt' to 10335:11111

changed ownership of '/host-volume/foo.txt' to 10335:11111

'bar.txt' -> '/host-volume/bar.txt'

$ ls -n

total 0

-rw-r--r-- 1 10335 11111 0 May 7 18:22 bar.txt

This trick works because the chown invocation within the [heredoc](http://tldp.org/LDP/abs/html/here-docs.html) the takes $(id -u):$(id -g)values from outside the running container; i.e., the docker host.

The benefits over docker cp are:

* you don't have to docker run --name your container before
* you don't have to docker container rm after

|  |
| --- |
| * CAVEAT: When you do this, you may run into problems with the user id of the docker user matching the user id of the current running user. That is, the files in /artifacts will be shown as owned by the user with the UID of the user used inside the docker container. A way around this may be to use the calling user's UID:   docker run -i -v ${PWD}:/working\_dir -w /working\_dir -u $(id -u) \  ubuntu:14.04 sh << COMMANDS  # Since $(id -u) owns /working\_dir, you should be okay running commands here  # and having them work. Then copy stuff into /working\_dir/artifacts .COMMANDS |

|  |  |
| --- | --- |
| -🡪 | If the container is running on any Linux distribution then you can copy the the file from the host to container and vice versa.  For this, you can use Linux scp command to copy a file from one host machine to another host machine. The fact is, you need to know the IP address of the container and a bridge between the host.  A simple example is given here:  Copy a file from a container to the host machine.  scp username\_container@ip\_container:~/Image/xxxx.yzx ~/Desktop/Images  Copy a file from local machine to the container  scp ~/Image/xxxx.yzx username\_container@ip\_container:~/Desktop/Images |

# 4.) [How to get a Docker container's IP address from the host?](https://stackoverflow.com/questions/17157721/how-to-get-a-docker-containers-ip-address-from-the-host)

docker inspect --format "{{ .NetworkSettings.IPAddress }}" containerId

🡪To get the IP address and host port of a container:

docker inspect conatinerId | awk '/IPAddress/ || /HostPort/'

Output:

"HostPort": "4200"

"HostPort": "4200"

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2",

-🡪Show all containers IP addresses:

# docker inspect --format='{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)

# /agitated\_babbage - 172.17.0.3

# /reverent\_brahmagupta - 172.17.0.2

If you are using docker-compose the command will be this:

docker inspect -f '{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)

**-🡪**docker network inspect bridge

Which should then return a Containers section which will display the IP address for that running container.

**-🡪**docker inspect <CONTAINER ID> | grep -w "IPAddress" | awk '{ print $2 }' | head -n 1 | cut -d "," -f1

# 4.1) [How to get the IP address of the docker host from inside a docker container](https://stackoverflow.com/questions/22944631/how-to-get-the-ip-address-of-the-docker-host-from-inside-a-docker-container)

I need to be able to retrieve the IP address the docker hosts and the portmaps from the host to the container, and doing that inside of the container.

🡪/sbin/ip route|awk '/default/ { print $3 }'

As @MichaelNeale noticed, there is no sense to use this method in Dockerfile (except when we need this IP during build time only), because this IP will be hardcoded during build time.

# 5.) [Docker container will automatically stop after “docker run -d”](https://stackoverflow.com/questions/30209776/docker-container-will-automatically-stop-after-docker-run-d)

According to tutorial I read so far, use "docker run -d" will start a container from image, and the container will run in background. This is how it looks like, we can see we already have container id.

root@docker:/home/root# docker run -d centos

605e3928cdddb844526bab691af51d0c9262e0a1fc3d41de3f59be1a58e1bd1d

But if I ran "**docker ps**", nothing was returned.

So I tried "**docker ps -a**", I can see container already exited:

root@docker:/home/root# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

605e3928cddd centos:latest "/bin/bash" 31 minutes ago Exited (0) 31 minutes ago kickass\_swartz

Anything I did wrong? How can I troubleshoot this issue?

-🡪execute command as follows :

docker run -t -d <image-name>

if you want to specify port then command as below:

docker run -t -d -p <port-no> <image-name>

verify the running container using following command:

docker ps

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | I actually like this answer the best, because the most popular answer suggests that you need a command (i.e. tail -f /dev/null). The command is completely optional. The key here is to use -t. I also found -i work in place of -t, or you can also use both -it combined (as the documentation suggests it will run as a shell). |

-🡪Instead of running with docker run -i -t image your-command, using -d is recommended because you can run your container with just one command and you don’t need to detach terminal of container by hitting Ctrl + P + Q.

However, there is a problem with -d option. **Your container immediately stops unless the commands are not running on foreground**.  
Docker requires your command to keep running in the foreground. Otherwise, it thinks that your applications stops and shutdown the container.

The problem is that some application does not run in the foreground. How can we make it easier?

In this situation, you can add tail -f /dev/null to your command.  
By doing this, even if your main command runs in the background, your container doesn’t stop because tail is keep running in the foreground.

So this would work:

docker run -d centos tail -f /dev/null

A docker ps would show the centos container still running.

# 7.) [Correct way to detach from a container without stopping it](https://stackoverflow.com/questions/25267372/correct-way-to-detach-from-a-container-without-stopping-it)

In Docker 1.1.2 (latest), what's the correct way to detach from a container without stopping it?

So for example, if I try:

* docker run -i -t foo /bin/bash or
* docker attach foo (for already running container)

both of which get me to a terminal in the container, how do I exit the container's terminal without stopping it?

exit and CTR+C both stop the container.

-🡪

|  |  |
| --- | --- |
|  | Well Ctrl+C (or Ctrl+\) should detach you from the container but it will kill the container because your main process is a bash.  A little lesson about docker. The container is not a real full functional OS. When you run a container the process you launch take the PID 1 and assume init power. So when that process is terminated the daemon stop the container until a new process is launched (via docker start) (More explanation on the matter <http://phusion.github.io/baseimage-docker/#intro>)  If you want a container that run in detached mode all the time, i suggest you use  docker run -d foo  With an ssh server on the container. (easiest way is to follow the dockerizing openssh tutorial <https://docs.docker.com/engine/examples/running_ssh_service/>)  Or you can just relaunch your container via  docker start foo  (it will be detached by default)  🡪 |
|  | You can accomplish what you want with either:  docker run -t -d <image-name>  or  docker run -i -d <image-name>  or  docker run -it -d <image-name>  The command parameter as suggested by other answers (i.e. tail -f /dev/null) is completely optional, and is NOT required to get your container to stay running in the background.  Also note the Docker documentation suggests that combining -i and -t options will cause it to behave like a shell. |

# -🡪

|  |  |
| --- | --- |
|  | I dug into this and all the answers above are partially right. It all depends on how the container is launched. It comes down to the following when the container was launched:   * was a TTY allocated (-t) * was stdin left open (-i)  **^P^Q does work BUT: you need to specify -t and -i when you launch the container:** [berto@g6]$ docker run -ti -d --name test python:3.6 /bin/bash -c 'while [ 1 ]; do sleep 30; done;'  b26e39632351192a9a1a00ea0c2f3e10729b6d3e22f8e0676d6519e15c08b518  [berto@g6]$ docker attach test  # here I typed ^P^Q  read escape sequence  # i'm back to my prompt  [berto@g6]$ docker kill test; docker rm -v test  test  test **ctrl+c does work BUT: you need to only specify -t when you launch the container:** [berto@g6]$ docker run -t -d --name test python:3.6 /bin/bash -c 'while [ 1 ]; do sleep 30; done;'  018a228c96d6bf2e73cccaefcf656b02753905b9a859f32e60bdf343bcbe834d  [berto@g6]$ docker attach test  ^C  [berto@g6]$ **The third way to detach** There is a way to detach without killing the container though; you need another shell. In summary, running this in another shell detached and left the container running pkill -9 -f 'docker.\*attach':  [berto@g6]$ docker run -d --name test python:3.6 /bin/bash -c 'while [ 1 ]; do sleep 30; done;'  b26e39632351192a9a1a00ea0c2f3e10729b6d3e22f8e0676d6519e15c08b518  [berto@g6]$ docker attach test  # here I typed ^P^Q and doesn't work  ^P  # ctrl+c doesn't work either  ^C  # can't background either  ^Z  # go to another shell and run the `pkill` command above  # i'm back to my prompt  [berto@g6]$  Why? Because you're killing the process that connected you to the container, not the container itself.  -🡪 If you do "docker attach "container id" you get into the container. To exit from the container without stopping the container you need to enter "Ctrl+P+Q" |

# 6.) [How to list containers in Docker](https://stackoverflow.com/questions/16840409/how-to-list-containers-in-docker)

To show only **running containers** use the given command:

docker ps

To show **all containers** use the given command:

docker ps –a

docker ps -aq --no-trunc ((FULL SHA))

To show the **latest created container** (includes all states) use the given command:

docker ps -l

To show **n last created containers** (includes all states) use the given command:

docker ps -n=-1

-n, --last int Show n last created containers (includes all states) (default -1)

-l, --latest Show the latest created container (includes all states)

--no-trunc Don't truncate output

[ansible@devopsserver ~]$ docker ps -n 4

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

9d14779823b3 centos "/bin/bash" 20 minutes ago Up 20 minutes agitated\_babbage

c6cfe791b120 centos "/bin/bash" 28 minutes ago Up 28 minutes reverent\_brahmagupta

03403c5b0c71 centos "/bin/bash" 31 minutes ago Exited (0) 31 minutes ago sad\_bartik

d29b0ca30c3f ubuntu "/bin/bash" 34 minutes ago Exited (0) 34 minutes ago dazzling\_tesla

To display **total file sizes** use the given command:

docker ps –s -🡪Running Conytainers

 docker ps –as -🡪 all containers

# 7.)[Cannot install packages inside docker Ubuntu image](https://stackoverflow.com/questions/27273412/cannot-install-packages-inside-docker-ubuntu-image)

It is because there is no package cache in the image, you need to run:

apt-get -qq update

before installing packages, and if your command is in a Dockerfile, you'll then need:

apt-get -qq -y install curl

-🡪Always combine RUN apt-get update with apt-get install in the same RUN statement, for example

RUN apt-get update && apt-get install -y package-bar

(...)

Using apt-get update alone in a RUN statement causes caching issues and subsequent apt-get install instructions fail.

# 8.) [How to copy docker images from one host to another without via repository?](https://stackoverflow.com/questions/23935141/how-to-copy-docker-images-from-one-host-to-another-without-via-repository)

How to transfer docker image from one machine to another one without using a repository, no matter private or public?

I am used to play and create my own image in virtualbox, and when it is finished, I try to deploy to other machines to have real usage.

Since it is based on own based image (like redhat), it cannot be recreated from Dockerfile.

Are there any simple commands I can use? Or another solution?

**Updated** It seems save/export can achieve similar purpose, see [What is the difference between save and export in Docker?](https://stackoverflow.com/questions/22655867/what-is-the-difference-between-save-and-export-in-docker), and I prefer the save command for my case.

-🡪 You will need to save the docker image as a tar file:

docker save -o <save image to path> <image name>

Then copy your image to a new system with regular file transfer tools such as cp or scp. After that you will have to load the image into docker:

docker load -i <path to image tar file>

PS: You may need to sudo all commands.

# -🡪

|  |  |
| --- | --- |
|  | I assume you need to save couchdb-cartridge which has a image id 7ebc8510bc2c:  stratos@Dev-PC:~$ docker images  REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE  couchdb-cartridge latest 7ebc8510bc2c 17 hours ago 1.102 GB  192.168.57.30:5042/couchdb-cartridge latest 7ebc8510bc2c 17 hours ago 1.102 GB  ubuntu 14.04 53bf7a53e890 3 days ago 221.3 MB  Save the archiveName image to a tar file. I will use the /media/sf\_docker\_vm/ to save the image.  stratos@Dev-PC:~$ docker save imageID > /media/sf\_docker\_vm/archiveName.tar  docker save awesomesauce:latest > awesomesauce.tar  sudo docker save -o /home/matrix/matrix-data.tar matrix-data  Copy the archiveName.tar file to your new Docker instance using whatever method works in your environment, for example FTP, SCP, etc.  Run the docker load command on your new Docker instance and specify the location of the image tar file.  stratos@Dev-PC:~$ docker load < /media/sf\_docker\_vm/archiveName.tar  sudo docker load -i <path to copied image file>  Finally, run the docker images command to check that the image is now available.  stratos@Dev-PC:~$ docker images  REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE  couchdb-cartridge latest 7ebc8510bc2c 17 hours ago 1.102 GB  192.168.57.30:5042/couchdb-cartridge latest bc8510bc2c 17 hours ago 1.102 GB  ubuntu 14.04 4d2eab1c0b9a 3 days ago 221. |

# 9.) [Where are Docker images stored on the host machine?](https://stackoverflow.com/questions/19234831/where-are-docker-images-stored-on-the-host-machine)

|  |  |
| --- | --- |
|  | I managed to find the containers under directory /var/lib/docker/containers, but I can't find the images.  What are the directories and files under /var/lib/docker? |

# -🡪

The contents of the /var/lib/docker directory vary depending on the [driver Docker is using for storage](https://github.com/docker/docker/blob/990a3e30fa66e7bd3df3c78c873c97c5b1310486/daemon/graphdriver/driver.go#L37-L43).

By default this will be aufs but can fall back to overlay, overlay2, btrfs, devicemapper or zfs depending on your kernel support. In most places this will be aufs but the [RedHats went with devicemapper](http://developerblog.redhat.com/2014/09/30/overview-storage-scalability-docker/).

You can manually set the storage driver with the [-s or --storage-driver=](https://docs.docker.com/engine/reference/commandline/dockerd/#/daemon-storage-driver-option) option to the [Docker daemon](https://docs.docker.com/engine/reference/commandline/dockerd/).

* /var/lib/docker/{driver-name} will contain the driver specific storage for contents of the images.
* /var/lib/docker/graph/<id> now only contains metadata about the image, in the json and layersize files.

In the case of aufs:

* /var/lib/docker/aufs/diff/<id> has the file contents of the images.
* /var/lib/docker/repositories-aufs is a JSON file containing local image information. This can be viewed with the command docker images.

In the case of devicemapper:

* /var/lib/docker/devicemapper/devicemapper/data stores the images
* /var/lib/docker/devicemapper/devicemapper/metadata the metadata
* Note these files are thin provisioned "sparse" files so aren't as big as they seem.

|  |
| --- |
| Actually, Docker images are stored in two files as shown by following command  $ docker info  Data file: /var/lib/docker/devicemapper/devicemapper/data  Metadata file: /var/lib/docker/devicemapper/devicemapper/metadata |

> docker info

...

Docker Root Dir: C:\ProgramData\Docker...

|  |  |
| --- | --- |
|  | The images are stored in /var/lib/docker/graph/<id>/layer.  Note that images are just diffs from the parent image. The parent ID is stored with the image's metadata /var/lib/docker/graph/<id>/json.  When you docker run an image. AUFS will 'merge' all layers into one usable file system. |

# 10.) [I lose my data when the container exits](https://stackoverflow.com/questions/19585028/i-lose-my-data-when-the-container-exits)

# That cannot be called "container exiting", you are just creating a new container.

|  |
| --- |
| There are following ways to persist container data:   1. Docker volumes 2. Docker commit 3. Dockerfile   a) create container from ubuntu image and run a bash terminal.  $ docker run -i -t ubuntu:14.04 /bin/bash  b) Inside the terminal install curl  # apt-get update  # apt-get install curl  c) Exit the container terminal  # exit  d) Take a note of your container id by executing following command :  $ docker ps -a  e) save container as new image  $ docker commit <container\_id> new\_image\_name:tag\_name(optional)  f) verify that you can see your new image with curl installed.  $ docker images  $ docker run -it new\_image\_name:tag\_name bash  # which curl  /usr/bin/curl |
| In addition to [Unferth's answer](https://stackoverflow.com/a/19586345/113938), it is recommended to create a [Dockerfile](http://docs.docker.io/en/latest/use/builder/).  In an empty directory, **create a file called "Dockerfile"** with the following contents.  FROM ubuntu  RUN apt-get install ping  ENTRYPOINT ["ping"]  **Create an image using the Dockerfile**. Let us use a tag so we do not need to remember the hexadecimal image number.  $ docker build -t iman/ping .  And then **run the image** in a container.  $ docker run iman/ping stackoverflow.com |

🡪 My suggestion is to manage docker, with docker compose. Is an easy to way to manage all the docker's containers for your project, you can map the versions and link different containers to work together.

The docs are very simple to understand, better than docker's docs.

[Docker-Compose Docs](https://docs.docker.com/compose/)

🡪 You might want to look at docker volumes if you want to persist the data in your container.

|  |
| --- |
|  |

🡪When you use docker run to start a container, it actually *creates a new container* based on the image you have specified.

Besides the other useful answers here, note that you can restart an existing container after it exited and your changes are still there.

docker start f357e2faab77 # restart it in the background

docker attach f357e2faab77 # reattach the terminal & stdin

# 11.) [Copy directory to other directory at Docker using ADD command](https://stackoverflow.com/questions/26504846/copy-directory-to-other-directory-at-docker-using-add-command)

Have read <http://docs.docker.com/engine/reference/builder/#add> however I met a problem. I want to copy the local directory go to docker /user/local/

I tried:

ADD go /usr/local/

and:

ADD /go/ /usr/local/

also:

RUN chmod 0755 /usr/local/go/src/make.bash

However, I see the following error message /usr/local/go/src/make.bash: No such file or directory but the local go directory does contain make.bash.

🡪 ADD go /usr/local/

will copy the **contents** of your local go directory in the /usr/local/ directory of your docker image.

To copy the go directory itself in /usr/local/ use:

ADD go /usr/local/go

or

COPY go /usr/local/go

# 12.) [Run a Docker Image as a Container](https://stackoverflow.com/questions/18497688/run-a-docker-image-as-a-container)

|  |  |
| --- | --- |
|  | I built a docker image from a dockerfile. I see the image was built successfully, but what do I do with it? Shouldn't it be able to run as a container?   * 🡪To list the docker images   $ docker images   * If your application wants to run in 80 port   $ docker run -d --restart=always -p 80:80 image\_name:version  After you run this images,you can then access the WebDAV instance at http://localhost:8888/webdav. Internally the folder /var/webdav is used as WebDAV root.  You can run this container in following way:  $ docker run -d -e USERNAME=test -e PASSWORD=test -p 8888:80 morrisjobke/webdav 13.) [How to create named and latest tag in Docker?](https://stackoverflow.com/questions/22080706/how-to-create-named-and-latest-tag-in-docker) Supposed I have an image that I want to tag as 0.10.24 (in my case it's an image containing Node.js 0.10.24). I built that image using a Dockerfile and executing docker build and by providing a tag using the -t parameter.  I expect that one day I will have additional versions of that image**, so I will rerun the process, just with another tag name.**  So far, so good. This works great and fine and all is well.  But, and this is where problems start, **I also want to always have the newest image tagged ad latest additionally. So I guess I need to give two names to the very same image.**  How do I do this? Do I really need to re-run docker build on the exact same version again, but this time use another tag, is is there a better option? |

**13.) You can have multiple tags when building the image:**

$ docker build -t whenry/fedora-jboss:latest -t whenry/fedora-jboss:v2.1 .

Reference: <https://docs.docker.com/engine/reference/commandline/build/#tag-image-t>

Here is my bash script

docker build -t ${IMAGE}:${VERSION} .

docker tag ${IMAGE}:${VERSION} ${IMAGE}:latest

You can then remove untagged images if you rebuilt the same version with

docker rmi $(docker images | grep "^<none>" | awk "{print $3}")

[link](http://jimhoskins.com/2013/07/27/remove-untagged-docker-images.html)

or

docker rmi $(docker images | grep "^<none>" | tr -s " " | cut -d' ' -f3 | tr '\n' ' ')

or

**Clean up commands: as of docker 1.13 introduces clean-up commands. To remove all unused containers, images, networks and volumes:**

**docker system prune**

**or individually:**

**docker container prune**

**docker image prune**

**docker network prune**

**docker volume prune**

🡪Just **grep the ID** from docker images:

docker build -t creack/node:latest .

ID="$(docker images | grep 'creak/node' | head -n 1 | awk '{print $3}')"

docker tag "$ID" creack/node:0.10.24

docker tag "$ID" creack/node:latest

Needs no temporary file and gives **full build output**. You still can redirect it to /dev/null or a log file.

🡪#!/bin/bash

VERSION=1.0.0

IMAGE=company/image

ID=$(docker build -t ${IMAGE} . | tail -1 | sed 's/.\*Successfully built \(.\*\)$/\1/')

docker tag ${ID} ${IMAGE}:${VERSION}

docker tag -f ${ID} ${IMAGE}:latest

🡪ID=$(docker build -t creack/node .) doesn't work for me since ID will contain the output from the build.

SO I'm using this small BASH script:

#!/bin/bash

set -o pipefail

IMAGE=...your image name...

VERSION=...the version...

docker build -t ${IMAGE}:${VERSION} . | tee build.log || exit 1

ID=$(tail -1 build.log | awk '{print $3;}')

docker tag $ID ${IMAGE}:latest

docker images | grep ${IMAGE}

docker run --rm ${IMAGE}:latest /opt/java7/bin/java -version

# 14.) [Running nano in docker container](https://stackoverflow.com/questions/27826241/running-nano-in-docker-container)

I open an interactive shell into a docker container like so

sudo docker exec -t -i {container\_name} bash

So far so good but trying to run nano results in:

Error opening terminal: unknown.

🡪alias nano='export TERM=xterm && nano'

docker exec -it id\_container bash

apt-get update

apt-get install nano

export TERM=xterm

🡪You can add

ENV TERM xterm

to your Dockerfile if you will use the editor regularly. We have that setting in our base container, since we're constantly debugging things with vi/emacs.

# Basically I needed to add the ENV TERM xterm before the RUN command.

# 15.) [Docker how to change repository name or rename image?](https://stackoverflow.com/questions/25211198/docker-how-to-change-repository-name-or-rename-image)

I'm trying to change repository name of the image:

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

server latest d583c3ac45fd 26 minutes ago 685.5 MB

Hence I want to change the name server to something like myname/server:

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

myname/server latest d583c3ac45fd 26 minutes ago 685.5 MB

# 🡪What is the difference between REPOSITORY and TAG?

# TAG is typically used for the version of the repository—e.g., repository mariadb with tag 10.1–

docker tag server:latest myname/server:latest

or docker tag d583c3ac45fd myname/server:latest

Tags are just human-readable aliases for the full image name (d583c3ac45fd...).

So you can have as many of them associated with the same image as you like. If you don't like the old name you can remove it after you've retagged it:

docker rmi server

That will just remove the alias/tag. Since d583c3ac45fd has other names, the actual image won't be deleted.

# 16.) [How do I assign a port mapping to an existing Docker container?](https://stackoverflow.com/questions/19335444/how-do-i-assign-a-port-mapping-to-an-existing-docker-container)

I'm not sure if I've misunderstood something here, but it seems like it's only possible to set port mappings by creating a new container from an image. Is there a way to assign a port mapping to an existing Docker container?

🡪

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | Using iptables may work like this answer [Exposing a Port on a Live Docker Container](http://stackoverflow.com/questions/19897743/exposing-a-port-on-a-live-docker-container) |

🡪port forwardings can be specified ONLY with 'docker run' command.  
Other commands, 'docker start' does not have -p option and 'docker port' only displays current forwardings.

To add port forwardings, I always follow these steps,

1. [stop](https://docs.docker.com/engine/reference/commandline/stop/) running container
2. docker stop test01
3. [commit](https://docs.docker.com/engine/reference/commandline/commit/) the container
4. docker commit test01 test02

**NOTE:** The above, test02 is a new image that I'm constructing from the test01 container.

1. re-[run](https://docs.docker.com/engine/reference/commandline/run/) from the commited image
2. docker run -p 8080:8080 -td test02

Where the first 8080 is the local port and the second 8080 is the container port.

--- What if I want to keep the test01 name?

@user69715 - as was described in Luca's answer - stackoverflow.com/a/36189165/33204

you can keep the existing of a container by creating an image of it, deleting the current container, test01, and then recreate it using the previously saved image.

|  |
| --- |
| In Fujimoto Youichi's example test01 is a container, whereas test02 is an image.  Before doing docker run you can remove the original container and then assign the container the same name again:  $ docker stop container01  $ docker commit container01 image01  $ docker rm container01  $ docker run -d -P --name container01 image01  (Using -P to expose ports to random ports rather than manually assigning). |

🡪You can change the port mapping by directly editing the hostconfig.json file at /var/lib/docker/containers/[hash\_of\_the\_container]/hostconfig.json

You can determine the [hash\_of\_the\_container] via the docker inspect <container\_name>command and the value of the "Id" field is the hash.

1) stop the container

2) change the file

3) restart your docker engine (to flush/clear config caches)

4) start the container

So you don't need to create an image with this approach. You can also change the restart flag here.

Perfect... There was no need to create an image with this approach.

It's important to stop container, stop docker engine and change both hostconfig.json and config.v2.json to make this work.

# 17.) [Clone private git repo with dockerfile](https://stackoverflow.com/questions/23391839/clone-private-git-repo-with-dockerfile)

I have copied this code from what seems to be various working dockerfiles around, here is mine:

FROM ubuntu

MAINTAINER Luke Crooks "luke@pumalo.org"

# Update aptitude with new repo

RUN apt-get update

# Install software

RUN apt-get install -y git python-virtualenv

# Make ssh dir

RUN mkdir /root/.ssh/

# Copy over private key, and set permissions

ADD id\_rsa /root/.ssh/id\_rsa

RUN chmod 700 /root/.ssh/id\_rsa

RUN chown -R root:root /root/.ssh

# Create known\_hosts

RUN touch /root/.ssh/known\_hosts

# Remove host checking

RUN echo "Host bitbucket.org\n\tStrictHostKeyChecking no\n" >> /root/.ssh/config

# Clone the conf files into the docker container

RUN git clone git@bitbucket.org:Pumalo/docker-conf.git /home/docker-conf

This gives me the error

Step 10 : RUN git clone git@bitbucket.org:Pumalo/docker-conf.git /home/docker-conf

---> Running in 0d244d812a54

Cloning into '/home/docker-conf'...

Warning: Permanently added 'bitbucket.org,131.103.20.167' (RSA) to the list of known hosts.

Permission denied (publickey).

fatal: Could not read from remote repository.

Please make sure you have the correct access rights

and the repository exists.

2014/04/30 16:07:28 The command [/bin/sh -c git clone git@bitbucket.org:Pumalo/docker-conf.git /home/docker-conf] returned a non-zero code: 128

This is my first time using dockerfiles, but from what I have read (and taken from working configs) I cannot see why this doesn't work.

My id\_rsa is in the same folder as my dockerfile and is a copy of my local key which can clone this repo no problem.

Edit:

In my dockerfile I can add:

RUN cat /root/.ssh/id\_rsa

And it prints out the correct key, so I know its being copied correctly.

I have also tried to do as noah advised and ran:

RUN echo "Host bitbucket.org\n\tIdentityFile /root/.ssh/id\_rsa\n\tStrictHostKeyChecking no" >> /etc/ssh/ssh\_config

This sadly also doesn't work.

-->My key was password protected which was causing the problem, a working file is now listed below (for help of future googlers)

FROM ubuntu

MAINTAINER Luke Crooks "luke@pumalo.org"

# Update aptitude with new repo

RUN apt-get update

# Install software

RUN apt-get install -y git

# Make ssh dir

RUN mkdir /root/.ssh/

# Copy over private key, and set permissions

ADD id\_rsa /root/.ssh/id\_rsa

# Create known\_hosts

RUN touch /root/.ssh/known\_hosts

# Add bitbuckets key

RUN ssh-keyscan bitbucket.org >> /root/.ssh/known\_hosts

# Clone the conf files into the docker container

RUN git clone git@bitbucket.org:User/repo.git

🡪

|  |
| --- |
| You should create new SSH key set for that Docker image, as you probably don't want to embed there your own private key. To make it work, you'll have to add that key to deployment keys in your git repository. Here's complete recipe:   1. Generate ssh keys with ssh-keygen -q -t rsa -N '' -f repo-key which will give you repo-key and repo-key.pub files. 2. Add repo-key.pub to your repository deployment keys. On GitHub, go to [your repository] -> Settings -> Deploy keys 3. Add something like this to your Dockerfile: 4. ADD repo-key / 5. RUN \ 6. chmod 600 /repo-key && \ 7. echo "IdentityFile /repo-key" >> /etc/ssh/ssh\_config && \ 8. echo -e "StrictHostKeyChecking no" >> /etc/ssh/ssh\_config && \ 9. // your git clone commands here...   Note that above switches off StrictHostKeyChecking, so you don't need .ssh/known\_hosts. Although I probably like more the solution with ssh-keyscan in one of the answers above. |

🡪For bitbucket repository, generate App Password (Access Management -> App Password) with read access to the repo and project. Then the command that you should use is:

git clone https://username:generated\_password@bitbucket.org/reponame/projectname.git.

# 18.) [Using SSH keys inside docker container](https://stackoverflow.com/questions/18136389/using-ssh-keys-inside-docker-container)

Alright, so essentially I have an app that executes various fun stuff with Git (like running git clone & git push) and I'm trying to docker-ize it.

I'm running into an issue though where I need to be able to add an SSH key to the container for the container 'user' to use.

Is there a good way to do this? I tried copying it into /root/.ssh/, changing $HOME, creating a git ssh wrapper, and still no luck.

Here is the Dockerfile for reference:

#DOCKER-VERSION 0.3.4

from ubuntu:12.04

RUN apt-get update

RUN apt-get install python-software-properties python g++ make git-core openssh-server -y

RUN add-apt-repository ppa:chris-lea/node.js

RUN echo "deb http://archive.ubuntu.com/ubuntu precise universe" >> /etc/apt/sources.list

RUN apt-get update

RUN apt-get install nodejs -y

ADD . /src

ADD ../../home/ubuntu/.ssh/id\_rsa /root/.ssh/id\_rsa

RUN cd /src; npm install

EXPOSE 808:808

CMD [ "node", "/src/app.js"]

app.js runs the git commands like git pull

🡪This line is a problem:

ADD ../../home/ubuntu/.ssh/id\_rsa /root/.ssh/id\_rsa

When specifying the files you want to copy into the image you can only use relative paths - relative to the directory where your Dockerfile is. So you should instead use:

ADD id\_rsa /root/.ssh/id\_rsa

And put the id\_rsa file into the same directory where your Dockerfile is.

🡪Turns out when using Ubuntu, the ssh\_config isn't correct. You need to add

RUN echo " IdentityFile ~/.ssh/id\_rsa" >> /etc/ssh/ssh\_config

to your Dockerfile in order to get it to recognize your ssh key.

🡪Note: only use this approach for images that are private and will always be! The ssh key remains stored within the image, even if you remove the key in a layer command after adding it (see comments in [this post](https://stackoverflow.com/a/23411161/933245)).

In my case this is ok, so this is what I am using:

# Setup for ssh onto github

RUN mkdir -p /root/.ssh

ADD id\_rsa /root/.ssh/id\_rsa

RUN chmod 700 /root/.ssh/id\_rsa

RUN echo "Host github.com\n\tStrictHostKeyChecking no\n" >> /root/.ssh/config

**18.)**[How to force docker for clean build of an image?](https://stackoverflow.com/questions/35594987/how-to-force-docker-for-clean-build-of-an-image)

|  |  |
| --- | --- |
|  | I have build docker image from docker file using below command.  $ docker build -t u12\_core -f u12\_core .  when I am trying to rebuild it with same command, it's using build cache like  Step 1 : FROM ubuntu:12.04  ---> eb965dfb09d2  Step 2 : MAINTAINER Pavan Gupta <pavan.gupta@gmail.com>  ---> Using cache  ---> 4354ccf9dcd8  Step 3 : RUN apt-get update  ---> Using cache  ---> bcbca2fcf204  Step 4 : RUN apt-get install -y openjdk-7-jdk  ---> Using cache  ---> 103f1a261d44  Step 5 : RUN apt-get install -y openssh-server  ---> Using cache  ---> dde41f8d0904  Step 6 : RUN apt-get install -y git-core  ---> Using cache  ---> 9be002f08b6a  Step 7 : RUN apt-get install -y build-essential  ---> Using cache  ---> a752fd73a698  Step 8 : RUN apt-get install -y logrotate  ---> Using cache  ---> 93bca09b509d  Step 9 : RUN apt-get install -y lsb-release  ---> Using cache  ---> fd4d10cf18bc  Step 10 : RUN mkdir /var/run/sshd  ---> Using cache  ---> 63b4ecc39ff0  Step 11 : RUN echo 'root:root' | chpasswd  ---> Using cache  ---> 9532e31518a6  Step 12 : RUN sed -i 's/PermitRootLogin without-password/PermitRootLogin yes/' /etc/ssh/sshd\_config  ---> Using cache  ---> 47d1660bd544  Step 13 : RUN sed 's@session\s\*required\s\*pam\_loginuid.so@session optional pam\_loginuid.so@g' -i /etc/pam.d/sshd  ---> Using cache  ---> d1f97f1c52f7  Step 14 : RUN wget -O aerospike.tgz 'http://aerospike.com/download/server/latest/artifact/ubuntu12'  ---> Using cache  ---> bd7dde7a98b9  Step 15 : RUN tar -xvf aerospike.tgz  ---> Using cache  ---> 54adaa09921f  Step 16 : RUN dpkg -i aerospike-server-community-\*/\*.deb  ---> Using cache  ---> 11aba013eea5  Step 17 : EXPOSE 22 3000 3001 3002 3003  ---> Using cache  ---> e33aaa78a931  Step 18 : CMD /usr/sbin/sshd -D  ---> Using cache  ---> 25f5fe70fa84  Successfully built 25f5fe70fa84  cache shows that aerospike is installed however I don't find it inside containers spawn from this image, so want to rebuild this image without using cache. how can I force docker to rebuild clean image without cache? |

🡪There's a --no-cache option.

docker build --no-cache -t u12\_core -f u12\_core .

|  |  |
| --- | --- |
|  | The command docker build --no-cache . solved our similar problem.  Our dockerfile was:  RUN apt-get update  RUN apt-get -y install php5-fpm  But should have been:  RUN apt-get update && apt-get -y install php5-fpm  To prevent caching the update and install separately. |

# 19.) [How to continue a docker which is exited](https://stackoverflow.com/questions/21928691/how-to-continue-a-docker-which-is-exited)

--docker start `docker ps -a -q --filter "name=elas"`

docker start -a -i `docker ps -q -l`  
-a attach to container

--docker start $(docker ps -a -q --filter "status=exited")

This will start all container which are in exited state.

🡪You can restart an existing container after it exited and your changes are still there.

docker start `docker ps -q -l` # restart it in the background

docker attach `docker ps -q -l` # reattach the terminal & stdin

**20.)** [**docker run -> name is already in use by container**](https://stackoverflow.com/questions/31697828/docker-run-name-is-already-in-use-by-container)

Running the docker registry with below command always throws an error:

dev:tmp me$ docker run \

-d --name registry-v1 \

-e SETTINGS\_FLAVOR=local \

-e STORAGE\_PATH=/registry \

-e SEARCH\_BACKEND=sqlalchemy \

-e LOGLEVEL=DEBUG \

-p 5000:5000 \

registry:0.9.1

Error response from daemon: Conflict. The name "registry-v1" is already in use by container f9e5798a82e0. You have to delete (or rename) that container to be able to reuse that name.

How to prevent this error ?

# 🡪

|  |  |
| --- | --- |
|  | You are getting this error because registry-v1 was used by another container in the past...even though that container may have exited i.e (currently not in use).  You have 2 options to fix this...   1. Remove previous container using that name, with the command docker rm $(docker ps -aq --filter name=registry-v1)   OR   1. Rename current container to a different name i.e change this portion --name registry-v1 to something like --name registry-v1-second-project-name |

🡪I got confused by this also. There are two commands relevant here:

**docker run** Run a command in a **new** container

**docker start** Start one or more stopped container

# 21.) [Docker command not found even though installed with apt-get](https://stackoverflow.com/questions/30379381/docker-command-not-found-even-though-installed-with-apt-get)

 I tried installing Docker using ubuntu 14.0LTS virtualbox

sudo apt get install docker

However, when I try running docker it gives me the following error

The program 'docker' is currently not installed. You can install it by typing: sudo apt-get install docker

Why is ubuntu not seeing docker?

🡪sudo apt-get install docker

is a different library on ubuntu.

Use sudo apt-get install docker.io to install the correct docker.

The Ubuntu package docker actually refers to a GUI application, not the beloved DevOps tool we've come out to look for.

**22.)** [Can I run multiple programs in a Docker container?](https://stackoverflow.com/questions/19948149/can-i-run-multiple-programs-in-a-docker-container)

I'm trying to wrap my head around Docker from the point of deploying an application which is intended to run on the users on desktop. My application is simply a flask web application and mongo database. Normally I would install both in a VM and, forward a host port to the guest web app. I'd like to give Docker a try but I'm not sure how I'm meant to use more than one program. The documentations says there can only be only ENTRYPOINT so how can I have Mongo and my flask application. Or do they need to be in separate contains, in which case how do they talk to each other and how does this make distributing the app easy?

# 🡪

|  |  |
| --- | --- |
|  | There can be only one ENTRYPOINT, but that target is usually a script that launches as many programs that are needed. You can additionally use for example [Supervisord](http://supervisord.org/) or similar to take care of launching multiple services inside single container. [This is an example of a docker container running mysql, apache and wordpress within a single container](https://github.com/jbfink/docker-wordpress).  Say, You have one database that is used by a single web application. Then it is probably easier to run both in a single container.  If You have a shared database that is used by more than one application, then it would be better to run the database in its own container and the applications each in their own containers.  There are at least two possibilities how the applications can communicate with each other when they are running in different containers:   1. Use exposed IP ports and connect via them. 2. Recent docker versions [support linking](http://docs.docker.com/userguide/dockerlinks/#container-linking). |

🡪Docker suggests supervisord: <https://docs.docker.com/engine/admin/using_supervisord/>

Or you can use Honcho: <https://blog.codeship.com/using-honcho-create-multi-process-docker-container/>

 I agree with the other answers that using two containers is preferable, but if you have your heart set on bunding multiple services in a single container you can use something like supervisord.

in [Hipache](https://github.com/dotcloud/hipache) for instance, the included Dockerfile runs supervisord, and the file supervisord.conf specifies for both hipache and redis-server to be run.

# 23.) [Docker can't connect to docker daemon](https://stackoverflow.com/questions/21871479/docker-cant-connect-to-docker-daemon)

After I update my Docker version to 0.8.0, I get an error message while entering sudo docker version:

Client version: 0.8.0

Go version (client): go1.2

Git commit (client): cc3a8c8

2014/02/19 12:54:16 Can't connect to docker daemon. Is 'docker -d' running on this host?

And I've followed the instructions and entered command sudo docker -d, and I got this:

[/var/lib/docker|2462000b] +job initserver()

[/var/lib/docker|2462000b.initserver()] Creating server

open /var/lib/docker/aufs/layers/cf2414da53f9bcfaa48bc3d58360d7f1cfd3784e4fe51fbef95197709dfc285d: no such file or directory[/var/lib/docker|2462000b] -job initserver() = ERR (1)

2014/02/19 12:55:57 initserver: open /var/lib/docker/aufs/layers/cf2414da53f9bcfaa48bc3d58360d7f1cfd3784e4fe51fbef95197709dfc285d: no such file or directory

How do I solve the problem?

## **Linux**

|  |  |
| --- | --- |
|  | [Manage Docker as a non-root user](https://docs.docker.com/engine/installation/linux/ubuntulinux/#/install-the-latest-version)  1) Create Docker Group  sudo groupadd docker  2) Make user belong to docker group to get the group's privileges.  sudo usermod -aG docker $USER  Check whether the DOCKER\_HOST environment variable is set for your shell.  env | grep DOCKER\_HOST  If it exists,  unset DOCKER\_HOST  Then this should work:  docker run hello-world |

-🡪

1. Add the user to the **docker** group.

sudo usermod -aG docker $(whoami)

1. Log out and log back in to ensure docker runs with correct permissions.
2. Start docker.

sudo service docker start

🡪If all the other solutions above don't work you can try checking the ownership of /var/run/docker.sock:

ls -l /var/run/docker.sock

If you're not the owner then change ownership with the command:

sudo chown \*your-username\* /var/run/docker.sock

Then you can go ahead and try executing the Docker commands hassle-free :D

🡪Restart the daemon and configure your environment:

docker-machine regenerate-certs default

docker-machine restart

docker-machine kill default

docker-machine ls

docker-machine rm -f default

docker-machine create -d virtualbox default

docker-machine env

Finally,

eval $(docker-machine env)

To test the daemon is running:

docker ps -a or docker-machine ls. This will list all containers.

🡪

|  |  |
| --- | --- |
|  | I had the same problem running Docker 1.10 on Ubuntu 14.04 and none of the given answers worked. For me, the fix was to specify the storage driver when running the Docker daemon.  sudo docker daemon --storage-driver=devicemapper  🡪 Try to change the Docker configuration file, docker or docker-network in /etc/sysconfig:  (... ~ v1.17) **docker file:** OPTIONS= -H fd://  or (v1.18): **docker-network file:** DOCKER\_NETWORK\_OPTIONS= -H unix:///var/run/docker.sock |

🡪  Have similar problem. I had to logout and login again to shell because I have just installed Docker and following command didn't show in my environment.

export DOCKER\_HOST=127.0.0.1:4243 >> ~/.bashrc

# 24.) [Docker - Enter Running Container with new TTY](https://stackoverflow.com/questions/20932357/docker-enter-running-container-with-new-tty)

docker exec -it [container-id] bash

# 25.) [Is it possible to start a shell session in a running container (without ssh)](https://stackoverflow.com/questions/17903705/is-it-possible-to-start-a-shell-session-in-a-running-container-without-ssh)

# 26.) [Cannot download Docker images behind a proxy](https://stackoverflow.com/questions/23111631/cannot-download-docker-images-behind-a-proxy)

I installed docker on my ubuntu 13.10 and when I type in my console:

sudo docker pull busybox

I get the following error:

Pulling repository busybox

2014/04/16 09:37:07 Get https://index.docker.io/v1/repositories/busybox/images: dial tcp: lookup index.docker.io on 127.0.1.1:53: no answer from server

🡪On RHEL6.6 only this works (note the use of export):

**/etc/sysconfig/docker**

export http\_proxy="http://myproxy.example.com:8080"

export https\_proxy="http://myproxy.example.com:8080"

NOTE: Both can use the http protocol.)

Thanks to <https://crondev.com/running-docker-behind-proxy/>

# 27.) [Permission denied on accessing host directory in docker](https://stackoverflow.com/questions/24288616/permission-denied-on-accessing-host-directory-in-docker)

In short: I am trying to mount a host directory in Docker, but then I can not access it from within the container, even if the access permissions look good.

The details:

I am doing

sudo docker run -i -v /data1/Downloads:/Downloads ubuntu bash

and then

ls -al

It gives me:

total 8892

drwxrwxr-x. 18 1000 1000 12288 Jun 16 11:40 Downloads

and a lot more lines like that (I think this is the relevant portion).

If I do

cd /Downloads

ls

the result is

ls: cannot open directory .: Permission denied

The host is Fedora 20, with Docker 1.0.0 and go1.2.2.

Any ideas what is going wrong?

🡪 I resolved that issue by using a data container, this also has the advantage of isolating the data from the application layer. You could run it like this:

docker run --volumes-from=<container-data-name> ubuntu

🡪 Try docker volume create.

mkdir -p /data1/Downloads

docker volume create --driver local --name hello --opt type=none --opt device=/data1/Downloads --opt o=uid=root,gid=root --opt o=bind

docker run -i -v hello:/Downloads ubuntu bash

🡪 running the container as privileged:

sudo docker run --privileged=true -i -v /data1/Downloads:/Downloads ubuntu bash

Another option (that I have not tried) would be to create a privileged container and then create non-privileged containers inside of it.

🡪 It is an selinux issue.

You can temporarily issue

su -c "setenforce 0"

on the host to access or else add an selinux rule by running

chcon -Rt svirt\_sandbox\_file\_t /path/to/volume

# 28.) [From inside of a Docker container, how do I connect to the localhost of the machine?](https://stackoverflow.com/questions/24319662/from-inside-of-a-docker-container-how-do-i-connect-to-the-localhost-of-the-mach)

So I have a Nginx running inside a docker container, I have a mysql running on localhost, I want to connect to the MySql from within my Nginx. The MySql is running on localhost and not exposing a port to the outside world, so its bound on localhost, not bound on the ip address of the machine.

Is there any way to connect to this MySql or any other program on localhost from within this docker container?

# --🡪Note on docker container networking modes

Docker offers [different networking modes](https://docs.docker.com/engine/reference/run/#network-settings) when running containers. Depending on the mode you choose you would connect to your MySQL database running on the docker host differently.

docker run --net="bridge" (default)

Docker creates a bridge named docker0 by default. Both the docker host and the docker containers have an IP address on that bridge.

on the Docker host, type sudo ip addr show docker0 you will have an output looking like:

[vagrant@docker:~] $ sudo ip addr show docker0

4: docker0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff

inet 172.17.42.1/16 scope global docker0

valid\_lft forever preferred\_lft forever

inet6 fe80::5484:7aff:fefe:9799/64 scope link

valid\_lft forever preferred\_lft forever

So here my docker host has the IP address 172.17.42.1 on the docker0 network interface.

Now start a new container and get a shell on it: docker run --rm -it ubuntu:trusty bash -il and within the container type ip addr show eth0 to discover how its main network interface is set up:

root@e77f6a1b3740:/# ip addr show eth0

863: eth0: <BROADCAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 66:32:13:f0:f1:e3 brd ff:ff:ff:ff:ff:ff

inet 172.17.1.192/16 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::6432:13ff:fef0:f1e3/64 scope link

valid\_lft forever preferred\_lft forever

Here my container has the IP address 172.17.1.192. Now look at the routing table:

root@e77f6a1b3740:/# route

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface

default 172.17.42.1 0.0.0.0 UG 0 0 0 eth0

172.17.0.0 \* 255.255.0.0 U 0 0 0 eth0

So the IP Address of the docker host 172.17.42.1 is set as the default route and is accessible from your container.

root@e77f6a1b3740:/# ping 172.17.42.1

PING 172.17.42.1 (172.17.42.1) 56(84) bytes of data.

64 bytes from 172.17.42.1: icmp\_seq=1 ttl=64 time=0.070 ms

64 bytes from 172.17.42.1: icmp\_seq=2 ttl=64 time=0.201 ms

64 bytes from 172.17.42.1: icmp\_seq=3 ttl=64 time=0.116 ms

docker run --net="host"

Alternatively you can run a docker container with [network settings set to host](http://docs.docker.com/engine/reference/run/#network-host). Such a container will share the network stack with the docker host and from the container point of view, localhost(or 127.0.0.1) will refer to the docker host.

Be aware that any port opened in your docker container would be opened on the docker host. And this without requiring the [-p or -P docker run option](https://docs.docker.com/engine/reference/run/#expose-incoming-ports).

IP config on my docker host:

[vagrant@docker:~] $ ip addr show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:98:dc:aa brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fe98:dcaa/64 scope link

valid\_lft forever preferred\_lft forever

and from a docker container in **host** mode:

[vagrant@docker:~] $ docker run --rm -it --net=host ubuntu:trusty ip addr show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 08:00:27:98:dc:aa brd ff:ff:ff:ff:ff:ff

inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::a00:27ff:fe98:dcaa/64 scope link

valid\_lft forever preferred\_lft forever

As you can see both the docker host and docker container share the exact same network interface and as such have the same IP address.

**Connecting to MySQL from containers**

bridge mode

To access MySQL running on the docker host from containers in *bridge mode*, you need to make sure the MySQL service is listening for connections on the 172.17.42.1 IP address.

To do so, make sure you have either bind-address = 172.17.42.1 or bind-address = 0.0.0.0 in your MySQL config file (my.cnf).

If you need to set an environment variable with the IP address of the gateway, you can run the following code in a container :

export DOCKER\_HOST\_IP=$(route -n | awk '/UG[ \t]/{print $2}')

then in your application, use the DOCKER\_HOST\_IP environment variable to open the connection to MySQL.

**Note:** if you use bind-address = 0.0.0.0 your MySQL server will listen for connections on all network interfaces. That means your MySQL server could be reached from the Internet ; make sure to setup firewall rules accordingly.

**Note 2:** if you use bind-address = 172.17.42.1 your MySQL server won't listen for connections made to 127.0.0.1. Processes running on the docker host that would want to connect to MySQL would have to use the 172.17.42.1 IP address.

host mode

To access MySQL running on the docker host from containers in *host mode*, you can keep bind-address = 127.0.0.1 in your MySQL configuration and all you need to do is to connect to 127.0.0.1 from your containers:

[vagrant@docker:~] $ docker run --rm -it --net=host mysql mysql -h 127.0.0.1 -uroot -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 36

Server version: 5.5.41-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

**note:** Do use mysql -h 127.0.0.1 and not mysql -h localhost; otherwise the MySQL client would try to connect using a unix socket.

# 29.) [How to generate a Dockerfile from an image?](https://stackoverflow.com/questions/19104847/how-to-generate-a-dockerfile-from-an-image)

Is it possible to generate a Dockerfile from an image? I want to know for two reasons:

1. I can download images from the repository but would like to see the recipe that generated them.
2. I like the idea of saving snapshots, but once I am done it would be nice to have a structured format to review what was done.

--🡪To understand how a docker image was built, use the docker history --no-trunc command.

You can build a docker file from an image, but it will not contain everything you would want to fully understand how the image was generated. Reasonably what you can extract is the MAINTAINER, ENV, EXPOSE, VOLUME, WORKDIR, ENTRYPOINT, CMD, and ONBUILD parts of the dockerfile.

The following script should work for you:

#!/bin/bash

docker history --no-trunc "$1" | \

sed -n -e 's,.\*/bin/sh -c #(nop) \(MAINTAINER .\*[^ ]\) \*0 B,\1,p' | \

head -1

docker inspect --format='{{range $e := .Config.Env}}

ENV {{$e}}

{{end}}{{range $e,$v := .Config.ExposedPorts}}

EXPOSE {{$e}}

{{end}}{{range $e,$v := .Config.Volumes}}

VOLUME {{$e}}

{{end}}{{with .Config.User}}USER {{.}}{{end}}

{{with .Config.WorkingDir}}WORKDIR {{.}}{{end}}

{{with .Config.Entrypoint}}ENTRYPOINT {{json .}}{{end}}

{{with .Config.Cmd}}CMD {{json .}}{{end}}

{{with .Config.OnBuild}}ONBUILD {{json .}}{{end}}' "$1"

I use this as part of a script to rebuild running containers as images:<https://github.com/docbill/docker-scripts/blob/master/docker-rebase>

The Dockerfile is mainly useful if you want to be able to repackage an image.

The thing to keep in mind, is a docker image can actually just be the tar backup of a real or virtual machine. I have made several docker images this way. Even the build history shows me importing a huge tar file as the first step in creating the image...

-🡪How to generate a Dockerfile from an image?

You can.

First way

$ docker pull centurylink/dockerfile-from-image

$ alias dfimage="docker run -v /var/run/docker.sock:/var/run/docker.sock --rm centurylink/dockerfile-from-image"

$ dfimage --help

Usage: dockerfile-from-image.rb [options] <image\_id>

-f, --full-tree Generate Dockerfile for all parent layers

-h, --help Show this message

Here is the example to generate the Dockerfile from an exist image selenium/node-firefox-debug

core@core-01 ~ $ docker pull centurylink/dockerfile-from-image

core@core-01 ~ $ alias dfimage="docker run -v /var/run/docker.sock:/var/run/docker.sock --rm centurylink/dockerfile-from-image"

core@core-01 ~ $ dfimage selenium/node-firefox-debug

ADD file:b43bf069650bac07b66289f35bfdaf474b6b45cac843230a69391a3ee342a273 in /

RUN echo '#!/bin/sh' > /usr/sbin/policy-rc.d && echo 'exit 101' >> /usr/sbin/policy-rc.d && chmod +x /usr/sbin/policy-rc.d && dpkg-divert --local --rename --add /sbin/initctl && cp -a /usr/sbin/policy-rc.d /sbin/initctl && sed -i 's/^exit.\*/exit 0/' /sbin/initctl && echo 'force-unsafe-io' > /etc/dpkg/dpkg.cfg.d/docker-apt-speedup && echo 'DPkg::Post-Invoke { "rm -f /var/cache/apt/archives/\*.deb /var/cache/apt/archives/partial/\*.deb /var/cache/apt/\*.bin || true"; };' > /etc/apt/apt.conf.d/docker-clean && echo 'APT::Update::Post-Invoke { "rm -f /var/cache/apt/archives/\*.deb /var/cache/apt/archives/partial/\*.deb /var/cache/apt/\*.bin || true"; };' >> /etc/apt/apt.conf.d/docker-clean && echo 'Dir::Cache::pkgcache ""; Dir::Cache::srcpkgcache "";' >> /etc/apt/apt.conf.d/docker-clean && echo 'Acquire::Languages "none";' > /etc/apt/apt.conf.d/docker-no-languages && echo 'Acquire::GzipIndexes "true"; Acquire::CompressionTypes::Order:: "gz";' > /etc/apt/apt.conf.d/docker-gzip-indexes

RUN sed -i 's/^#\s\*\(deb.\*universe\)$/\1/g' /etc/apt/sources.list

CMD ["/bin/bash"]

MAINTAINER Selenium <selenium-developers@googlegroups.com>

RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main universe\n" > /etc/apt/sources.list && echo "deb http://archive.ubuntu.com/ubuntu trusty-updates main universe\n" >> /etc/apt/sources.list

RUN apt-get update -qqy && apt-get -qqy --no-install-recommends install ca-certificates openjdk-7-jre-headless unzip wget && rm -rf /var/lib/apt/lists/\* && sed -i 's/\/dev\/urandom/\/dev\/.\/urandom/' ./usr/lib/jvm/java-7-openjdk-amd64/jre/lib/security/java.security

RUN mkdir -p /opt/selenium && wget --no-verbose http://selenium-release.storage.googleapis.com/2.46/selenium-server-standalone-2.46.0.jar -O /opt/selenium/selenium-server-standalone.jar

RUN sudo useradd seluser --shell /bin/bash --create-home && sudo usermod -a -G sudo seluser && echo 'ALL ALL = (ALL) NOPASSWD: ALL' >> /etc/sudoers && echo 'seluser:secret' | chpasswd

MAINTAINER Selenium <selenium-developers@googlegroups.com>

ENV DEBIAN\_FRONTEND=noninteractive

ENV DEBCONF\_NONINTERACTIVE\_SEEN=true

ENV TZ=US/Pacific

RUN echo "US/Pacific" | sudo tee /etc/timezone && dpkg-reconfigure --frontend noninteractive tzdata

RUN apt-get update -qqy && apt-get -qqy install xvfb && rm -rf /var/lib/apt/lists/\*

COPY file:335d2f6f9bfe311d2b38034ceab3b2ae2a1e07b9b203b330cac9857d6e17c148 in /opt/bin/entry\_point.sh

RUN chmod +x /opt/bin/entry\_point.sh

ENV SCREEN\_WIDTH=1360

ENV SCREEN\_HEIGHT=1020

ENV SCREEN\_DEPTH=24

ENV DISPLAY=:99.0

USER [seluser]

CMD ["/opt/bin/entry\_point.sh"]

MAINTAINER Selenium <selenium-developers@googlegroups.com>

USER [root]

RUN apt-get update -qqy && apt-get -qqy --no-install-recommends install firefox && rm -rf /var/lib/apt/lists/\*

COPY file:52a2a815e3bb6b85c5adfbceaabb5665b63f63ef0fb0e3f774624ee399415f84 in /opt/selenium/config.json

USER [seluser]

MAINTAINER Selenium <selenium-developers@googlegroups.com>

USER [root]

RUN apt-get update -qqy && apt-get -qqy install x11vnc && rm -rf /var/lib/apt/lists/\* && mkdir -p ~/.vnc && x11vnc -storepasswd secret ~/.vnc/passwd

ENV LANGUAGE=en\_US.UTF-8

ENV LANG=en\_US.UTF-8

RUN locale-gen en\_US.UTF-8 && dpkg-reconfigure --frontend noninteractive locales && apt-get update -qqy && apt-get -qqy --no-install-recommends install language-pack-en && rm -rf /var/lib/apt/lists/\*

RUN apt-get update -qqy && apt-get -qqy --no-install-recommends install fonts-ipafont-gothic xfonts-100dpi xfonts-75dpi xfonts-cyrillic xfonts-scalable && rm -rf /var/lib/apt/lists/\*

RUN apt-get update -qqy && apt-get -qqy install fluxbox && rm -rf /var/lib/apt/lists/\*

COPY file:90e3a7f757c3df44d541b59234ad4ca996f799455eb8d426218619b244ebba68 in /opt/bin/entry\_point.sh

RUN chmod +x /opt/bin/entry\_point.sh

EXPOSE 5900/tcp

Another way, which you needn't pull the image to local and no command need be run.

Use above image as sample, you can get Dockerfile commands via below url:

<https://imagelayers.io/?images=selenium%2Fnode-firefox-debug:latest>

Wait for a while, there will be two windows, the up window lists the layers, the down window lists the command in Dockerfile.

The URL format is:

https://imagelayers.io/?images=<USER>%2F<IMAGE>:<TAG>

In face, imagelayers.io is built by Centurylink

# [What is the difference between save and export in Docker?](https://stackoverflow.com/questions/22655867/what-is-the-difference-between-save-and-export-in-docker)

I am playing around with [Docker](https://www.docker.io/) for a couple of days and I already made some images (which was really fun!). Now I want to persist my work and came to the save and export commands, but I don't fully understand them.

[What is the relationship between the docker host OS and the container base image OS?](https://stackoverflow.com/questions/18786209/what-is-the-relationship-between-the-docker-host-os-and-the-container-base-image)

kernel is still the kernel of the host, you will not have any specific kernel module/patches provided by the distribution.

|  |  |
| --- | --- |
|  | **Docker manage images that are the file system representation. You can install any linux distribution or simply put binaries.**   * **----**save will fetch an image : for a VM or a physical server, that would be the installation .ISO image or disk. The base operating system.   It will pack the layers and metadata of all the chain required to build the image. You can then load this "saved" images chain into another docker instance and create containers from these images.   * export will fetch the whole container : like a snapshot of a regular VM. Saves the OS of course, but also any change you made, any data file written during the container life. This one is more like a traditional backup.   It will give you a flat .tar archive containing the filesystem of your container.  Edit: as my explanation may still lead to confusion, I think that it is important to understand that one of these commands works with containers, while the other works with images.   * An **image** has to be considered as 'dead' or immutable, starting 0 or 1000 containers from it won't alter a single byte. That's why I made a comparison with a system install ISO earlier. It's maybe even closer to a live-CD. * A **container** "boots" the image and adds an additional layer on top of it. This layer stores any change on the container (created/changed/removed files...).   **-🡪**Docker **save** Produces a tar file repo which contains all parent layers, and all tags + versions, or specified repo:tag, for each argument provided from **image**.  Docker **export** Produces specified file(can be tar or tgz) with flat contents without contents of specified volumes from **Container**.  docker **save** need to use on docker image while docker **export** need to use on container(just like running image)  Save Usage  docker save [OPTIONS] IMAGE [IMAGE...]  Save an image(s) to a tar archive (streamed to STDOUT by default)  --help=false Print usage -o, --output="" Write to a file, instead of STDOUT  export Usage  docker export [OPTIONS] CONTAINER  Export the contents of a container's filesystem as a tar archive  --help=false Print usage -o, --output="" Write to a file, instead of STDOUT  **-🡪**There are two main differences between save and export commands.   1. Save command saves whole image with history and metadata but export command exports only files structure (without history and metadata). So the exported tar file will be smaller then the saved one. 2. When you use exported file system for creating a new image then this new image will not contains any USER, EXPOSE, RUN etc. commands from your Dockerfile. Only file structure will be transferred. So when you are using mentioned keywords in your Dockerfile then you cannot use export command for transferring image to another machine - you need always use save command.  1.)[what's the difference between a container and an image?](https://stackoverflow.com/questions/21498832/in-docker-whats-the-difference-between-a-container-and-an-image) An instance of an image is called a container. You have an image, which is a set of layers as you describe. If you start this image, you have a running container of this image. You can have many running containers of the same image..   * <https://docs.docker.com/engine/reference/glossary/#image>   A container is an active (or inactive if exited) stateful instantiation of an image.   * <https://docs.docker.com/engine/reference/glossary/#container>   the image is the recipe, the container is the cake ;-) you can make as many cakes as you like with a given recipe.  +------------+ docker build +--------------+ docker run -dt +-----------+ docker exec -it +------+  | Dockerfile | --------------> | Image | ---------------> | Container | -----------------> | Bash |  +------------+ +--------------+ +-----------+ +------+  ^  | docker pull  |  +--------------+  | Registry |  +--------------+  **Dockerfile** > (Build) > **Image** > (Run) > **Container**.   * **Dockerfile**: contains a set of docker instructions that provisions your operating system the way you like, and installs/configure all your software's. * **Image**: compiled Dockerfile. Saves you time from rebuilding the Dockerfile every time you need to run a container. And it's a way to hide your provision code. * **Container**: the virtual operating system itself, you can ssh into it and run any commands you wish, as if it's a real environment. You can run 1000+ containers from the same Image.   Images depict the state of a container at every point in time. So the basic workflow is:   1. create an image 2. start a container 3. make changes to the container 4. save the container back as an image   🡪An image is an immutable template for creating a container. It's easier to understand the difference between an image and container by considering what happens to an image to turn it into a container.  The Docker engine takes the image and adds a read-write filesystem on top, then initialises various settings. These settings include network options (IP, port, etc.), name, ID, and any resource limits (CPU, memory). If the Docker engine has been asked to run the container it will also initialise a process inside it. A container can be stopped and restarted, in which case it will retain all settings and filesystem changes (but will lose anything in memory and all processes will be restarted). For this reason a stopped or exited container is not the same as an image.  🡪Using an object-oriented programming analogy, the difference between a Docker image and a Docker container is the same as that of the difference between a class and an object. An object is the runtime instance of a class. Similarly, a container is the runtime instance of an image.  An object gets created only once when it is instantiated. Similarly, a container can be running or stopped. Containers are created out of an image, though this might not always be the case. |

**🡪**Images are frozen immutable snapshots of live containers. Containers are running (or stopped) instances of some image.

Start with the base image called 'ubuntu'. Let's run bash interactively within the ubuntu image and create a file. We'll use the -i and -t flags to give us an interactive bash shell.

$ docker run -i -t ubuntu /bin/bash

root@48cff2e9be75:/# ls

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

root@48cff2e9be75:/# cat > foo

This is a really important file!!!!

root@48cff2e9be75:/# exit

Don't expect that file to stick around when you exit and restart the image. You're restarting from exactly the same defined state as you started in before, not where you left off.

$ docker run -i -t ubuntu /bin/bash

root@abf181be4379:/# ls

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

root@abf181be4379:/# exit

But, the container, now no longer running, has state and can be saved (committed) to an image.

$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

abf181be4379 ubuntu:14.04 /bin/bash 17 seconds ago Exited (0) 12 seconds ago elegant\_ardinghelli

48cff2e9be75 ubuntu:14.04 /bin/bash About a minute ago Exited (0) 50 seconds ago determined\_pare

...

Let's create an image from container ID 48cff2e9be75 where we created our file:

$ docker commit 48cff2e9be75 ubuntu-foo

d0e4ae9a911d0243e95556e229c8e0873b623eeed4c7816268db090dfdd149c2

Now, we have a new image with our really important file:

$ docker run ubuntu-foo /bin/cat foo

This is a really important file!!!!

Try the command docker images. You should see your new image ubuntu-foo listed along with the ubuntu standard image we started with.

🡪 I couldn't understand the concept of Image and Layer inspite of reading all the questions here and then eventually stumbled upon this [excellent documentation from docker](https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#container-size-on-disk) (duh!).

The example there is really the key to understand the whole concept. It is a lengthy post, so I am summarising the key points that need to be really grasped to get clarity.

* **Image**: A Docker image is built up from a series of **read only** layers
* **Layer**: Each layer represents an instruction in the image’s Dockerfile.

Example: The below Dockerfile contains four commands, each of which creates a layer.

FROM ubuntu:15.04

COPY . /app

RUN make /app

CMD python /app/app.py

**Importantly**, Each layer is only a set of differences from the layer before it.

* **Container**. When you create a new container, *you add a new writable layer on top of the underlying layers*. This layer is often called the “container layer”. All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this *thin* writable container layer.

Hence, the The major difference between a container and an image is the **top writable layer**. All wr

rites to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged.

**Understanding Images and Containers from a Size on Disk Perspective**

To view the approximate size of a running container, you can use the docker ps -s command. You get size and virtual size as two of the outputs

* Size: the amount of data (on disk) that is used for the writable layer of each container
* Virtual Size: the amount of data used for the read-only image data used by the container. Multiple containers may share some or all read-only image data. **Hence these are not additive. i.e. you can't add all the virtual sizes to calculate how much size on disk is used by the Image**

**Another important concept is the Copy on Write Strategy** If a file or directory exists in a lower layer within the image, and another layer (including the writable layer) needs read access to it, it just uses the existing file. The first time another layer needs to modify the file (when building the image or running the container), the file is copied into that layer and modified

**How-is-docker-different-from-a-normal-virtual-machine:**

One important difference is that **VMs use a separate kernel to run the OS**. That's the reason it is heavy and takes time to boot, consuming more system resources.

1. **In Docker, the containers share the kernel** with the host; hence, it is lightweight and can start and stop quickly.

|  |  |
| --- | --- |
|  | **1. Snapshot process is faster in Docker than VMs**  We generally start with a base image, and then make our changes, and commit those changes using docker, and it creates an image. This image contains only the differences from the base. When we want to run our image, we also need the base, and it layers our image on top of the base using a layered file system. File system merges the different layers together and we get what we want, and we just need to run it. Since docker typically builds on top of ready-made images from a registry, we rarely have to "snapshot" the whole OS ourself. This ability of Dockers to snapshot the OS into a common image also makes it easy to deploy on other docker hosts.  **2. Startup time is less for Docker than VMs**  A virtual machine usually takes minutes to start, but containers takes seconds, and sometime even less than a second.  **4. Docker images have more portability**  Docker images are composed of layers. When we pull or transfer an image, only the layers we haven’t yet in cache are retrieved. That means that if we use multiple images based on the same base Operating System, the base layer is created or retrieved only once. VM images doesn't have this flexibility.  **5. Docker provides versioning of images**  We can use the docker commit command. We can specify two flags: -m and -a. The -m flag allows us to specify a commit message, much like we would with a commit on a version control system:  $ sudo docker commit -m "Added json gem" -a "Kate Smith"  0b2616b0e5a8 ouruser/sinatra:v2  4f177bd27a9ff0f6dc2a830403925b5360bfe0b93d476f7fc3231110e7f71b1c  **6. Docker images do not have states**  In Docker terminology, a read-only Layer is called an image. An image never changes. Since Docker uses a Union File System, the processes think the whole file system is mounted read-write. But all the changes go to the top-most writeable layer, and underneath, the original file in the read-only image is unchanged. Since images don't change, images do not have state.  **7. VMs are hardware-centric and docker containers are application-centric**  Let's say we have a container image that is 1GB in size. If we wanted to use a Full VM, we would need to have 1GB times x number of VMs you want. In docker container we can share the bulk of the 1GB and if you have 1000 containers we still might only have a little over 1GB of space for the containers OS, assuming they are all running the same OS image.  **8. Supported image formats**  Docker images:   * bare. The image does not have a container or metadata envelope. * ovf. The OVF container format. * aki. An Amazon kernel image. * ari. An Amazon ramdisk image. * ami. An Amazon machine image.   VM images:   * raw. An unstructured disk image format; if you have a file without an extension it is possibly a raw format * vhd. The VHD disk format, a common disk format used by virtual machine monitors from VMware, Xen, Microsoft, VirtualBox, and others * vmdk. Common disk format supported by many common virtual machine monitors * vdi. Supported by VirtualBox virtual machine monitor and the QEMU emulator * iso. An archive format for the data contents of an optical disc, such as CD-ROM. * qcow2. Supported by the QEMU emulator that can expand dynamically and supports Copy on Write * aki. An Amazon kernel image. * ari. An Amazon ramdisk image. * ami. An Amazon machine image. |