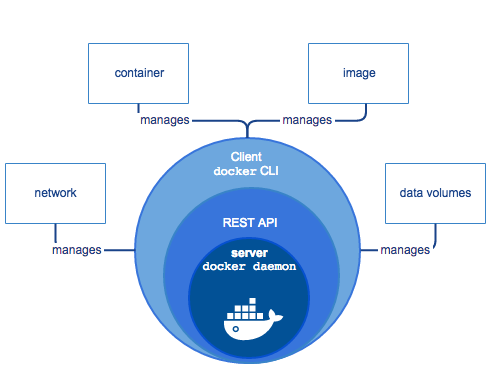
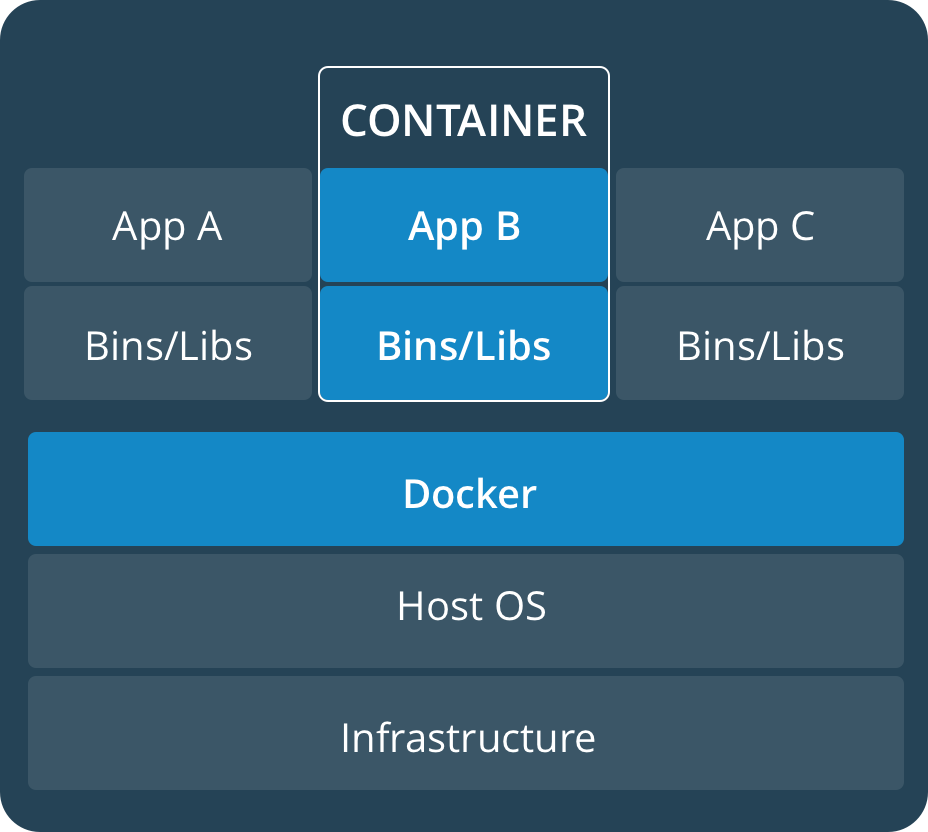
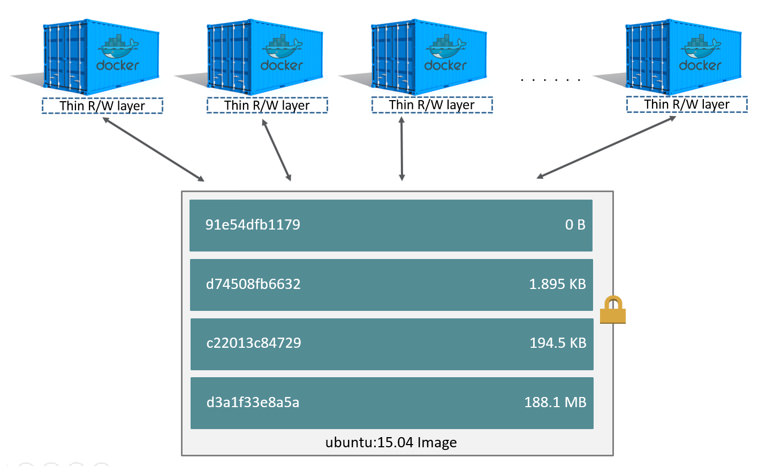
**Docker Deep Dive**

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

Containers doesn’t have an OS to run the applications like Physical servers/VMs, then? It uses the lib/binr of the OS to run the application which makes you container lightweight.



****

****

## Architecture Overview

Docker architecture:

* Client-server architecture
* Client talks to the Docker daemon
* The Docker daemon handles:
  + Building Running
  + Distributing
* Both communicate using a REST API:
  + UNIX sockets
  + Network interface

The Docker daemon (dockerd):

* Listens for Docker API requests and manages Docker objects:
  + Images
  + Containers
  + Networks
  + Volumes

The Docker client (docker):

* Is how users interact with Docker
* Sends commands to dockerd

Docker registries:

* Stores Docker images
* Public registry such as Docker Hub
* Let you run your own private registry

Docker objects:

* Images:
  + Read-only template with instructions for creating a Docker container
  + Image is based on another image
  + Create your own images
  + Use images published to a registry
  + Use a Dockerfile to build images
* Containers:
  + Runnable instance of an image
  + Connect a container to networks
  + Attach storage
  + Create a new image based on its current state
  + Isolated from other containers and the host machine
* Services
  + Scale containers across multiple Docker daemons
  + Docker Swarm
  + Define the desired state
  + Service is load-balanced

Docker Swarm:

* Multiple Docker daemons (Master and Workers)
* The daemons all communicate using the Docker API
* Supported in Docker 1.12 and higher

## Running Containers

**docker container run -it --name <NAME> <IMAGE>:<TAG>**

Creating a container:

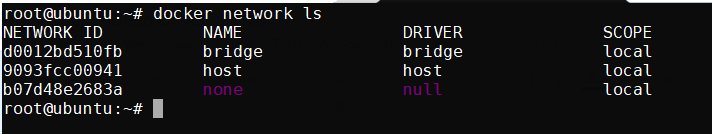
* CLI use for executing a command
* Docker client uses the appropriate API payload
* POSTs to the correct API endpoint
* Docker deamon receives instructions
* Docker deamon calls containerd to start a new container
* Docker daemon uses gRPC (a CRUD style API)
* containerd creates an OCI bundle from the Docker image
* Tells runc to create a container using the OCI bundle
* runc interfaces with the OS kernel to get the constructs needed to create a container
  + This includes namespaces, cgroups, etc.
* Container process starts as a child process
* runc exits once the container starts
* Process is complete, and container is running

**Docker network:**

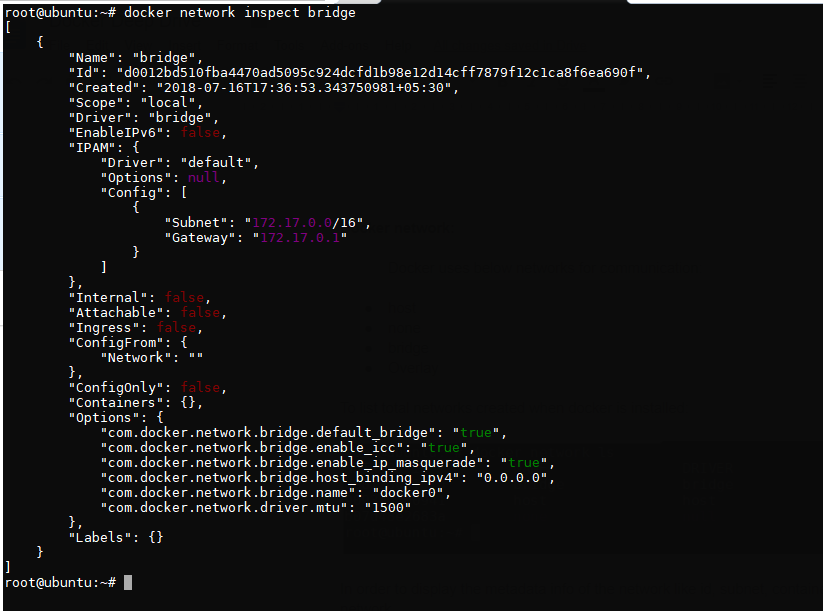
Docker uses below networks for communication:

* host
* none
* bridge
* Overlay

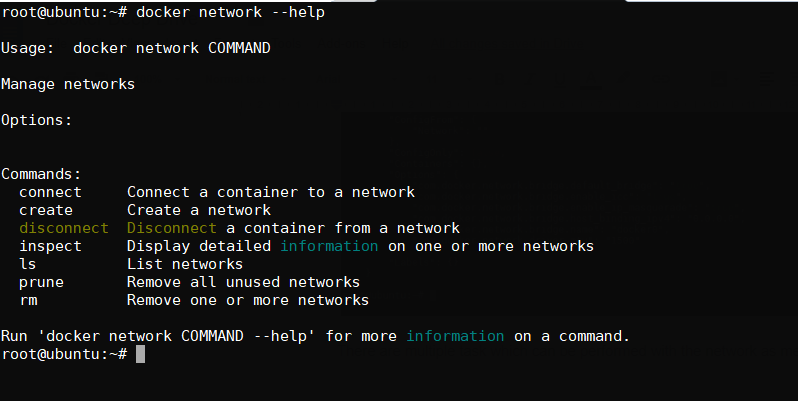
To list total networks created when docker is installed:



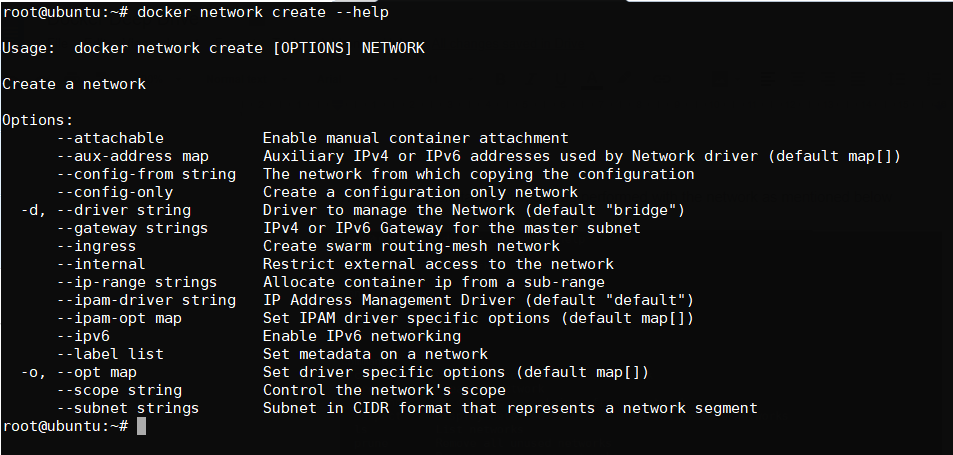
In order to display the metadata info of the network like id, subnet, containers binded to the network



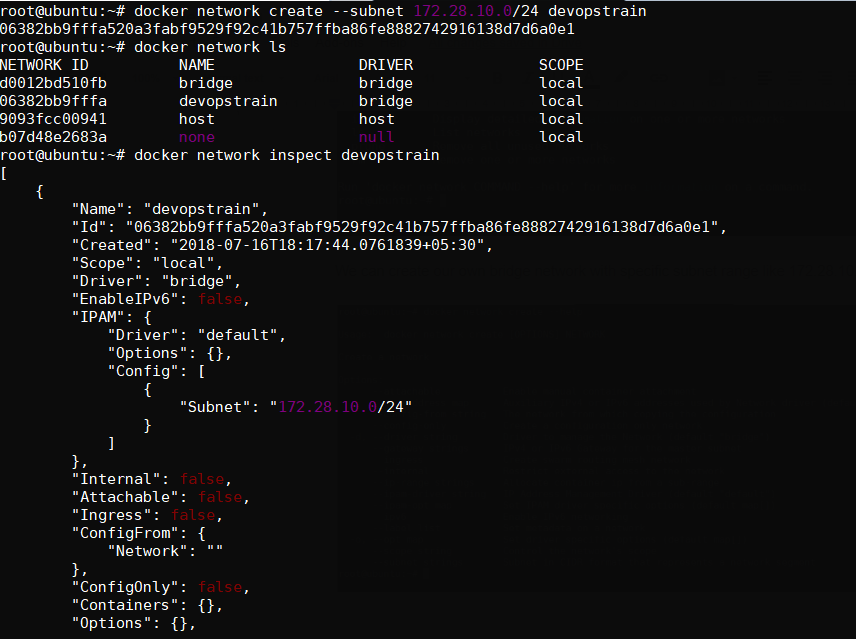
There are multiple task which can be performed with the network as mentioned below



We can create our own bridge network with specific subnet range like 172.28.10.0/24

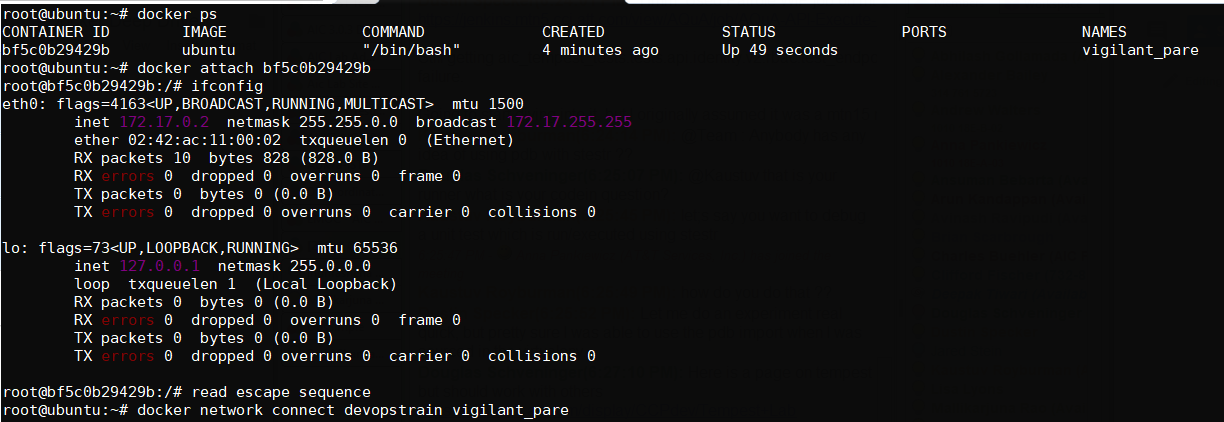


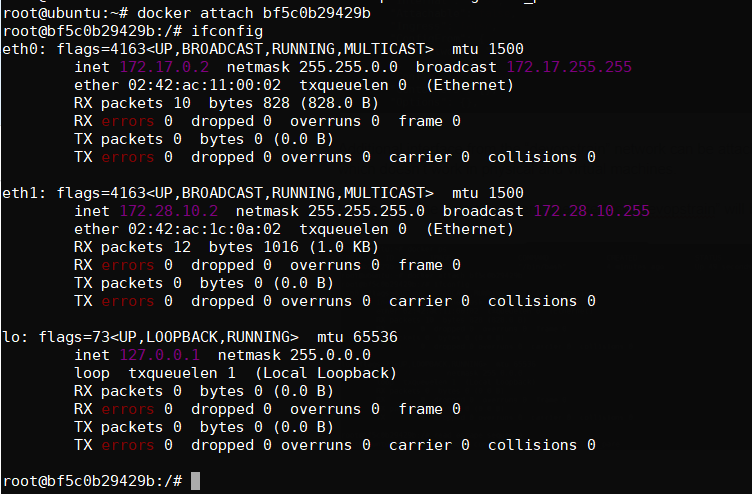
Here we are not passing the driver type bcz we are creating bridge network by default and gateway is optional which is not required in this case.



Additional interface from the “devopstrain” network can be attached to the running container, which doesn’t work in physical and virtual machines.

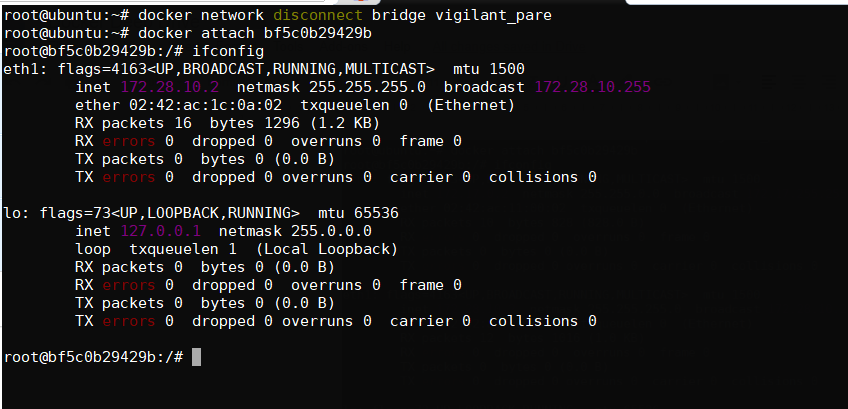
Let us take one scenario where one interface “devopstrain” will be attached to the conatiner:



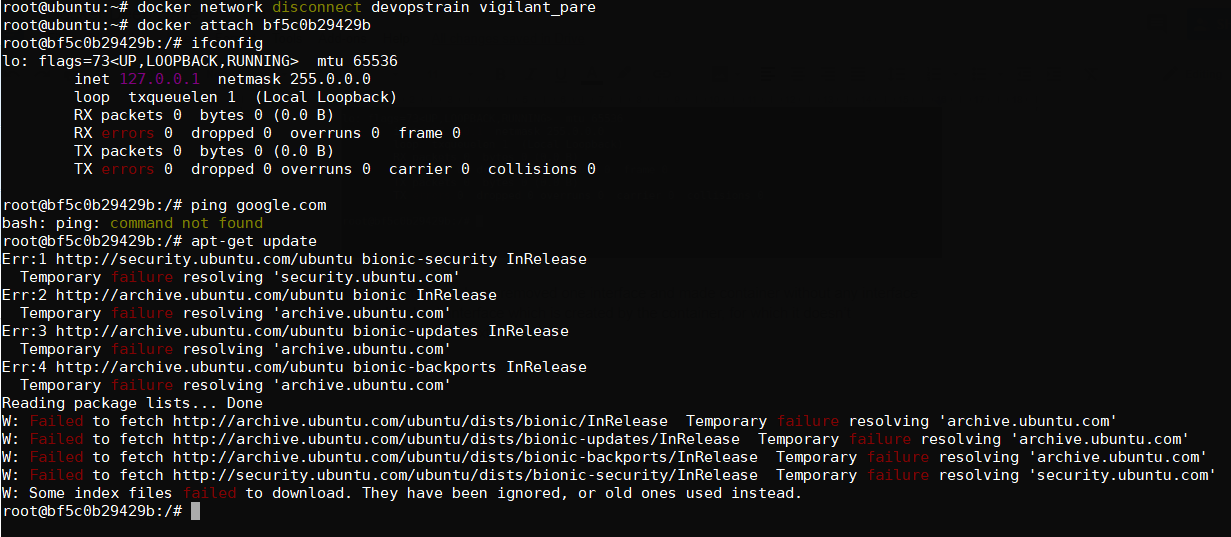


How to remove an interface from the running container?

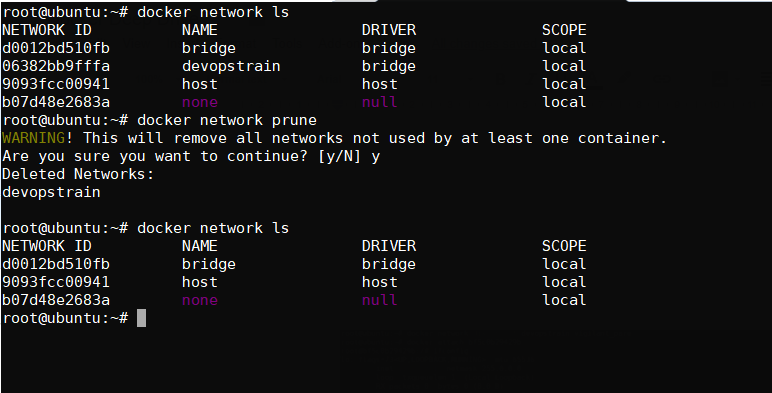
In the below example we have removed the “bridge network” from the container



Below example we have removed one interface and made container without any interface except loopback interface which is created by the container, for which it doesn’t communicate to the external world

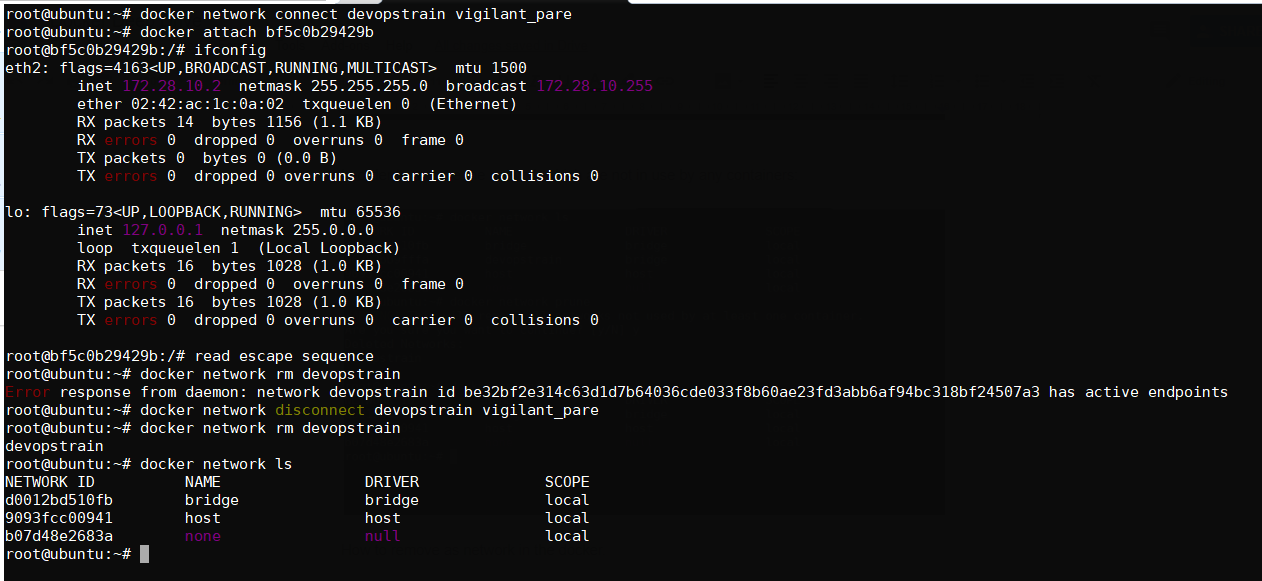


In order to remove the network which are not in use by any containers:

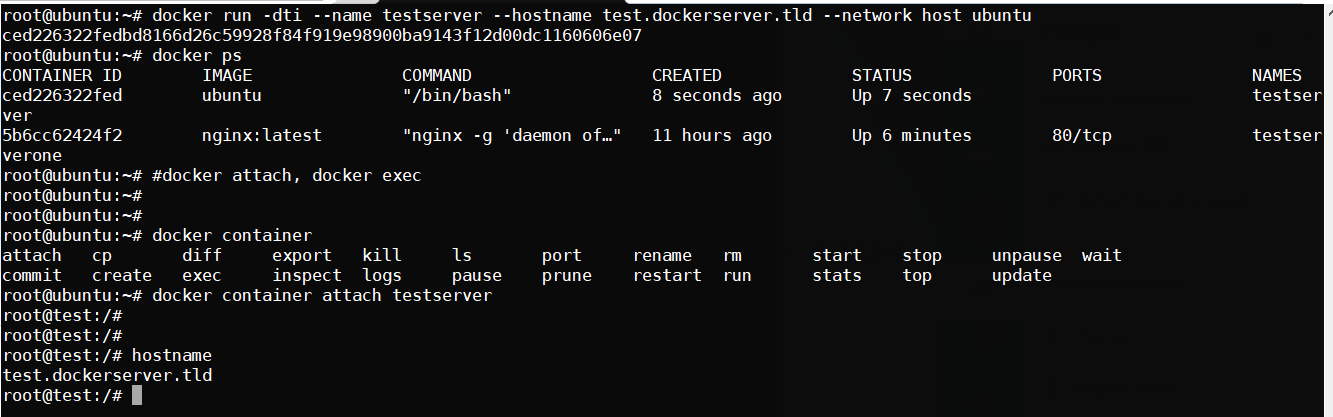


How to remove as network in the docker.

Note: Before we delete the network we need to ensure that non of the container is running on that network. (Either stop the container and disconnect the network)



Host network can be assigned to the container, which is mentioned below and we can access the application with the public ip address it self.

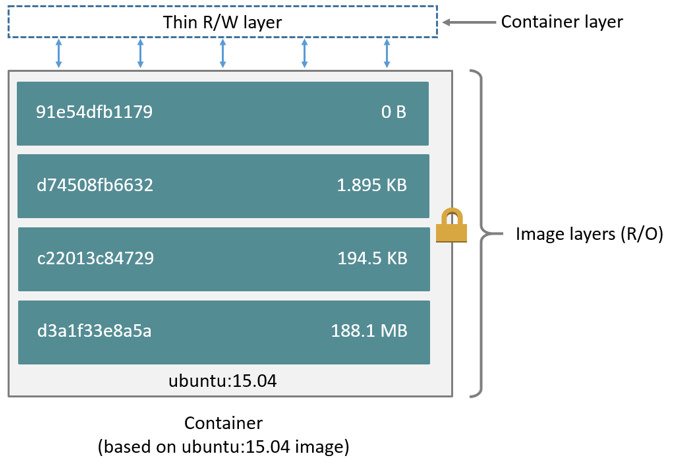


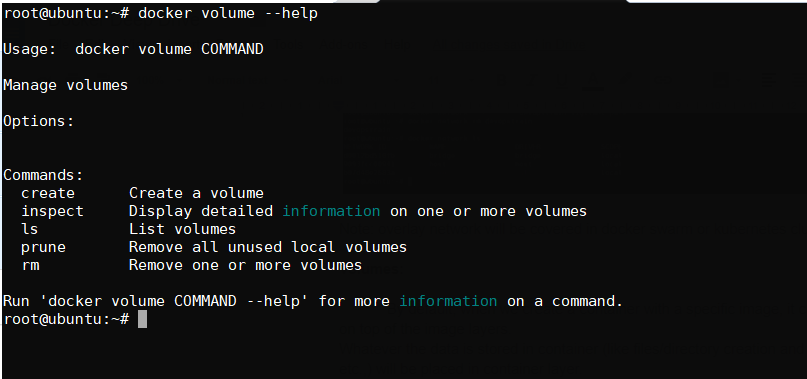
Note: overlay network will be covered in docker swarm or kubernetes cluster

**Volumes:**

By default, when we create a container with a specific image, it creates a thick layer on top of the image layers.

Whatever the data is stored in a container (like files/directory creation and installing packages etc.,) will be placed in container layer.





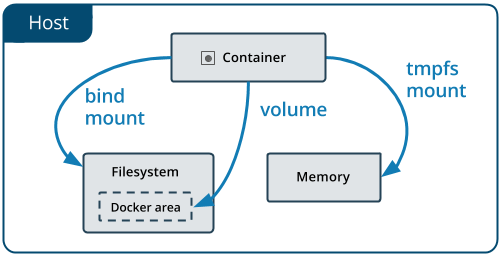
To write the data in to the container, we need to have storage driver configured to manage the file system. The storage driver provides a union file system, using the Linux kernel.

There are two ways to store the data in the host machine.

**Persistent storage and Ephemeral storage**

In persistent storage, data will remain even after the container was removed.

In ephemeral storage, data will be lost once the container is terminated.



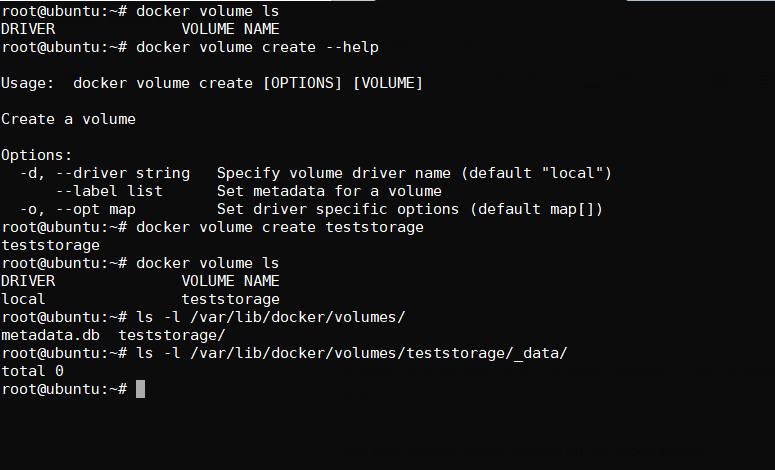
* **Volumes** are stored in a part of the host filesystem which is managed by Docker(/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

A given volume can be mounted into multiple containers simultaneously. When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using **docker volume prune.**

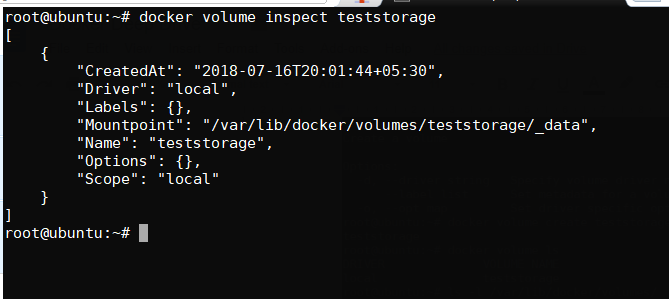
* **Bind mounts** may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
* **tmpfs mounts** are stored in the host system’s memory (RAM) only, and are never written to the host system’s filesystem.

Bind mounts and volumes can both mounted into containers using the -v or --volume flag, but the syntax for each is slightly different. For tmpfs mounts, you can use the --tmpfs flag. However, in Docker 17.06 and higher, we recommend using the --mount flag for both containers and services, for bind mounts, volumes, or tmpfs mounts, as the syntax is more clear.

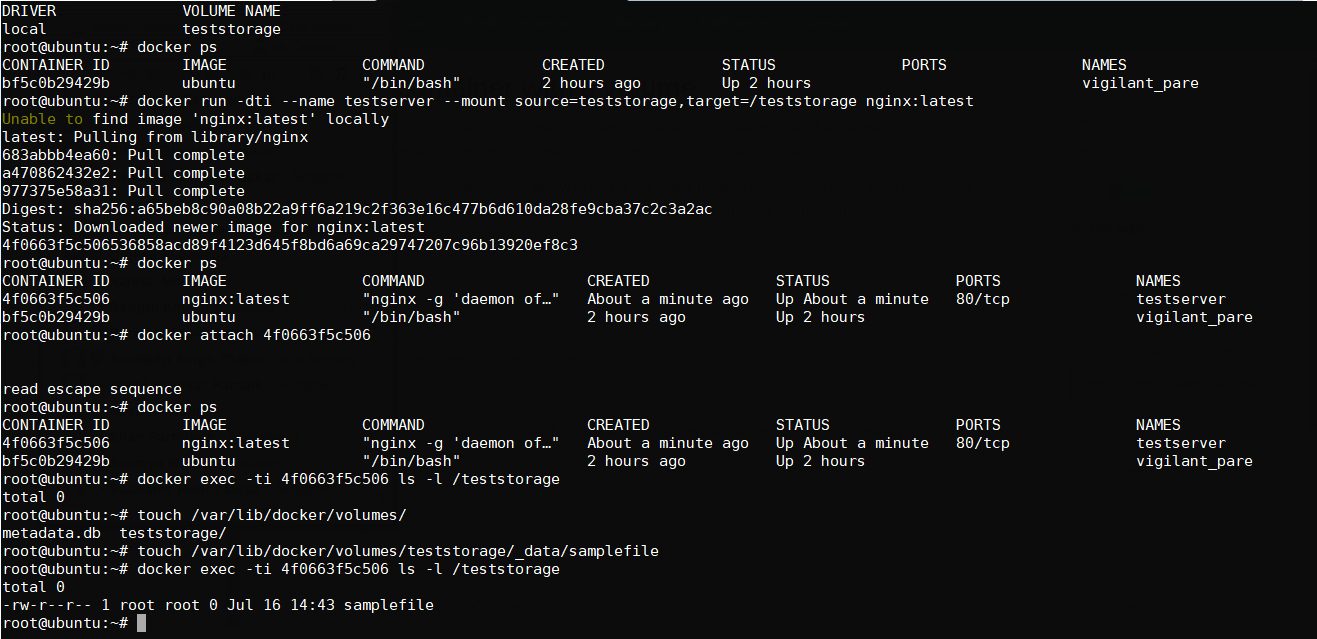
Let’s start with the volume creation on the docker engine:



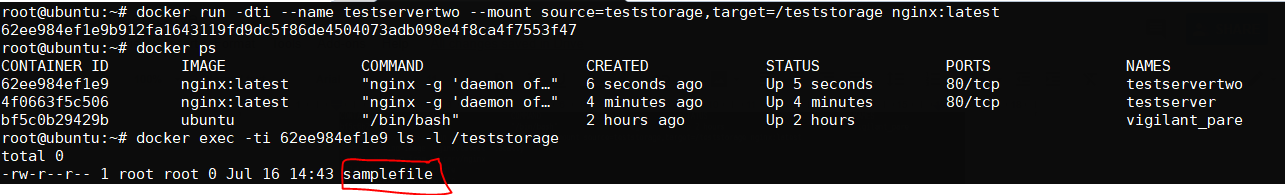
In order to view the volume details:



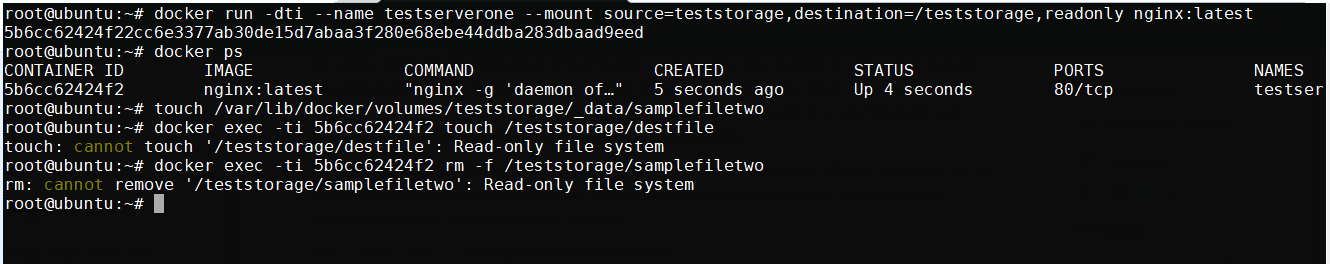
Now let us attach the volume to the container as mentioned below:



From the below example, we can see that single volume “teststorage” to second container too:



Sample example for volumes with read-only permissions:



**Dockerfile Complete:**

A Dockerfile is a text file which has some series to instructions to build own image with customized settings.

Let us look into dockerfile syntax and variable that need to passed

Key variables in Dockerfile:

FROM

MAINTAINER(optional)

RUN

CMD

ENV

ADD  
COPY

WORKDIR

ENTRYPOINT

LABEL

VOLUME

USER

EXPOSE

Example:

FROM ubuntu

MAINTAINER sudhams reddy duba ([dubareddy.383@gmail.com](mailto:dubareddy.383@gmail.com))

RUN apt-get update

RUN apt-get install nginx -y

COPY index.html /usr/share/nginx/html/

ENTRYPOINT [“/usr/sbin/nginx”, “-g”, “daemon off;”]

EXPOSE 80

The ENTRYPOINT specifies a command that will always be executed when the container starts. The CMD specifies arguments that will be fed to the ENTRYPOINT.

Iptables:

iptables -t nat -A DOCKER -p tcp --dport ${YOURPORT} -j DNAT --to-destination ${CONTAINERIP}:${YOURPORT}

iptables -t nat -A POSTROUTING -j MASQUERADE -p tcp --source ${CONTAINERIP} --destination ${CONTAINERIP} --dport ${YOURPORT}

iptables -A DOCKER -j ACCEPT -p tcp --destination ${CONTAINERIP} --dport ${YOURPORT}

**Docker commands Cookbook:**

**=======================**

attach:

=======

root@dockerserver:~# docker attach --help

Usage: docker attach [OPTIONS] CONTAINER

Attach local standard input, output, and error streams to a running container

Options:

--detach-keys string Override the key sequence for detaching a container

--help Print usage

--no-stdin Do not attach STDIN

--sig-proxy Proxy all received signals to the process (default true)

root@dockerserver:~#

root@dockerserver:~# docker attach --detach-keys ctrl-p,q zen\_lovelace

cp:

===

root@dockerserver:~# docker cp --help

Usage: docker cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH|-

docker cp [OPTIONS] SRC\_PATH|- CONTAINER:DEST\_PATH

Copy files/folders between a container and the local filesystem

Options:

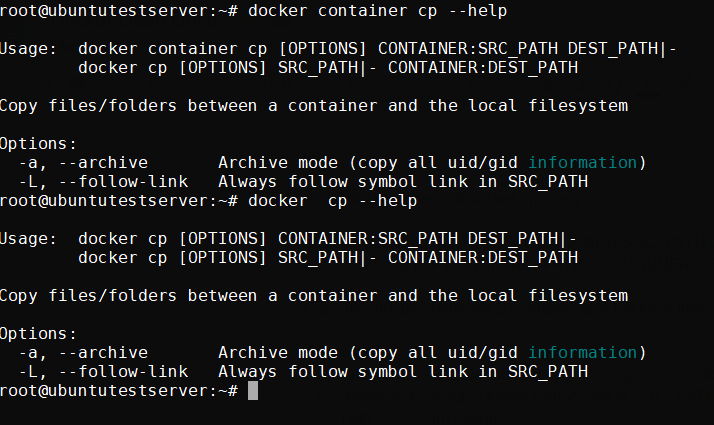
-a, --archive Archive mode (copy all uid/gid information)

-L, --follow-link Always follow symbol link in SRC\_PATH

--help Print usage

root@dockerserver:~#

Or



copy files from base machine to the container on fly.

root@dockerserver:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

eebeaefdd262 sudhams483/visualpath "/bin/bash" 4 hours ago Up 2 seconds 80/tcp zen\_lovelace

root@dockerserver:~# docker exec -ti eebeaefdd262 bash

root@eebeaefdd262:/# mkdir sudhams

root@eebeaefdd262:/# ls -l /sudhams/

total 0

root@eebeaefdd262:/# exit

exit

root@dockerserver:~# touch testfile

root@dockerserver:~# docker cp testfile eebeaefdd262:/sudhams

root@dockerserver:~# docker exec -ti eebeaefdd262 bash

root@eebeaefdd262:/# ls -l sudhams/

total 0

-rw-r--r-- 1 root root 0 Feb 28 14:52 testfile

root@eebeaefdd262:/#

pull files from the container to the base machine

root@dockerserver:~# rm -rf testfile

root@dockerserver:~# docker exec -ti eebeaefdd262 bash

root@eebeaefdd262:/# ls -l sudhams/

total 0

-rw-r--r-- 1 root root 0 Feb 28 14:52 testfile

root@eebeaefdd262:/# exit

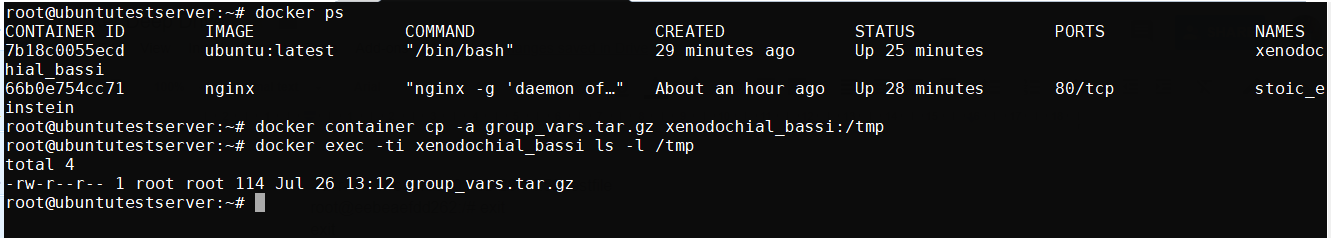
exit

root@dockerserver:~# docker cp eebeaefdd262:/sudhams/testfile .

root@dockerserver:~# ls -l testfile

-rw-r--r-- 1 root root 0 Feb 28 06:52 testfile

root@dockerserver:~#



export:

=======

root@dockerserver:~# docker export --help

Usage: docker export [OPTIONS] CONTAINER

Export a container's filesystem as a tar archive

Options:

--help Print usage

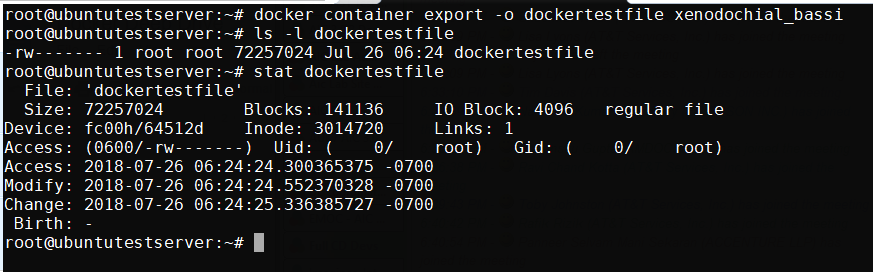
-o, --output string Write to a file, instead of STDOUT

root@dockerserver:~# docker export zen\_lovelace -o testfile

root@dockerserver:~# ls -l testfile

-rw------- 1 root root 208500736 Feb 28 07:19 testfile

root@dockerserver:~#



login & logout:

===============

root@dockerserver:~# docker login --help

Usage: docker login [OPTIONS] [SERVER]

Log in to a Docker registry

Options:

--help Print usage

-p, --password string Password

--password-stdin Take the password from stdin

-u, --username string Username

root@dockerserver:~# docker login -u sudhams483 -p \*\*\*\*\*

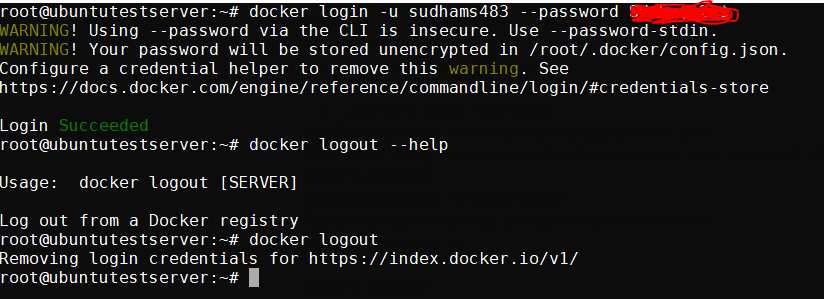
WARNING! Using --password via the CLI is insecure. Use --password-stdin.

Login Succeeded

root@dockerserver:~# docker logout

Removing login credentials for https://index.docker.io/v1/

root@dockerserver:~#



pause & unpause:

================

root@dockerserver:~# docker pause --help

Usage: docker pause CONTAINER [CONTAINER...]

Pause all processes within one or more containers

Options:

--help Print usage

root@dockerserver:~# docker pause zen\_lovelace

Error response from daemon: Container eebeaefdd2620271684f5118e8efadc4840cbb94828230586efd19a7ffc4233e is already paused

root@dockerserver:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

eebeaefdd262 sudhams483/visualpath "/bin/bash" 5 hours ago Up About an hour (Paused) 80/tcp zen\_lovelace

root@dockerserver:~# docker unpause zen\_lovelace

zen\_lovelace

root@dockerserver:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

eebeaefdd262 sudhams483/visualpath "/bin/bash" 5 hours ago Up About an hour 80/tcp zen\_lovelace

root@dockerserver:~# docker unpause --help

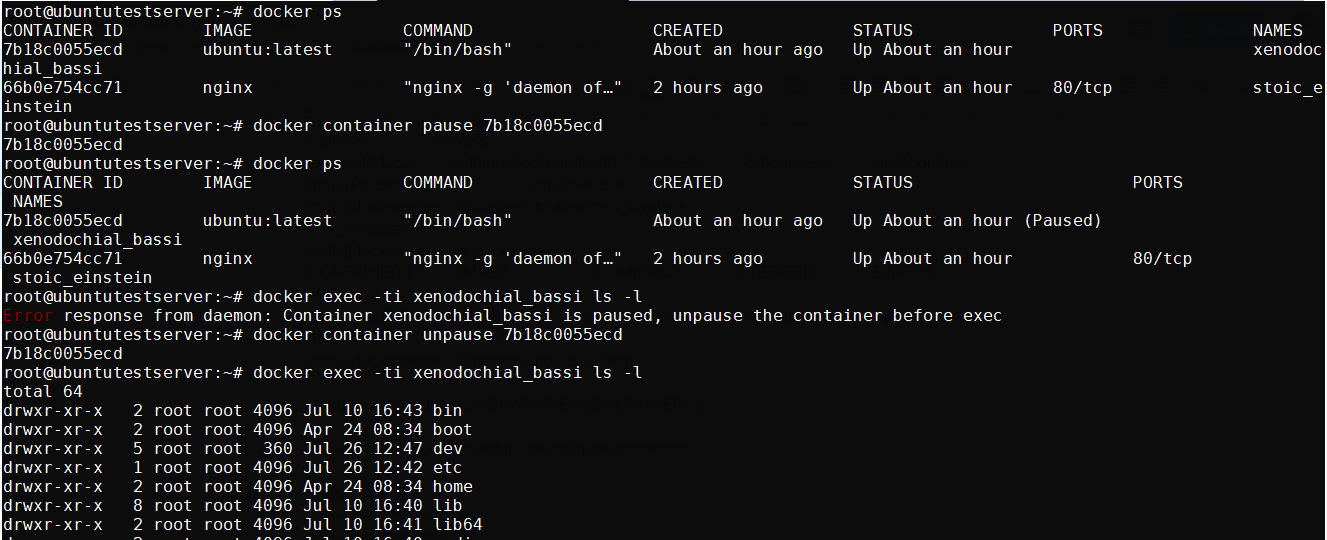
Usage: docker unpause CONTAINER [CONTAINER...]

Unpause all processes within one or more containers

Options:

--help Print usage

root@dockerserver:~#



push:

=====

root@dockerserver:~# docker push --help

Usage: docker push [OPTIONS] NAME[:TAG]

Push an image or a repository to a registry

Options:

--disable-content-trust Skip image signing (default true)

--help Print usage

root@dockerserver:~#

root@dockerserver:~# docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: sudhams483

Password:

Login Succeeded

root@dockerserver:~# docker push sudhams483/visualpath

The push refers to a repository [docker.io/sudhams483/visualpath]

31df331e1f23: Layer already exists

630730f8c75d: Layer already exists

827cd1db9e95: Layer already exists

e6e107f1da2f: Layer already exists

c41b9462ea4b: Layer already exists

latest: digest: sha256:0215754d7b43854382971f7010fba151a8a79b64cc9182bf1c111ab33f47fdf7 size: 1359

root@dockerserver:~#

Then login to the Docker Hub and verify the image which is uploaded to the repository.

run:

====

root@dockerserver:~# docker run --help

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:

--add-host list Add a custom host-to-IP mapping (host:ip)

-a, --attach list Attach to STDIN, STDOUT or STDERR

--blkio-weight uint16 Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)

--blkio-weight-device list Block IO weight (relative device weight) (default [])

--cap-add list Add Linux capabilities

--cap-drop list Drop Linux capabilities

--cgroup-parent string Optional parent cgroup for the container

--cidfile string Write the container ID to the file

--cpu-period int Limit CPU CFS (Completely Fair Scheduler) period

--cpu-quota int Limit CPU CFS (Completely Fair Scheduler) quota

--cpu-rt-period int Limit CPU real-time period in microseconds

--cpu-rt-runtime int Limit CPU real-time runtime in microseconds

-c, --cpu-shares int CPU shares (relative weight)

--cpus decimal Number of CPUs

--cpuset-cpus string CPUs in which to allow execution (0-3, 0,1)

--cpuset-mems string MEMs in which to allow execution (0-3, 0,1)

-d, --detach Run container in background and print container ID

--detach-keys string Override the key sequence for detaching a container

--device list Add a host device to the container

--device-cgroup-rule list Add a rule to the cgroup allowed devices list

--device-read-bps list Limit read rate (bytes per second) from a device (default [])

--device-read-iops list Limit read rate (IO per second) from a device (default [])

--device-write-bps list Limit write rate (bytes per second) to a device (default [])

--device-write-iops list Limit write rate (IO per second) to a device (default [])

--disable-content-trust Skip image verification (default true)

--dns list Set custom DNS servers

--dns-option list Set DNS options

--dns-search list Set custom DNS search domains

--entrypoint string Overwrite the default ENTRYPOINT of the image

-e, --env list Set environment variables

--env-file list Read in a file of environment variables

--expose list Expose a port or a range of ports

--group-add list Add additional groups to join

--health-cmd string Command to run to check health

--health-interval duration Time between running the check (ms|s|m|h) (default 0s)

--health-retries int Consecutive failures needed to report unhealthy

--health-start-period duration Start period for the container to initialize before starting health-retries countdown

(ms|s|m|h) (default 0s)

--health-timeout duration Maximum time to allow one check to run (ms|s|m|h) (default 0s)

--help Print usage

-h, --hostname string Container host name

--init Run an init inside the container that forwards signals and reaps processes

-i, --interactive Keep STDIN open even if not attached

--ip string IPv4 address (e.g., 172.30.100.104)

--ip6 string IPv6 address (e.g., 2001:db8::33)

--ipc string IPC mode to use

--isolation string Container isolation technology

--kernel-memory bytes Kernel memory limit

-l, --label list Set meta data on a container

--label-file list Read in a line delimited file of labels

--link list Add link to another container

--link-local-ip list Container IPv4/IPv6 link-local addresses

--log-driver string Logging driver for the container

--log-opt list Log driver options

--mac-address string Container MAC address (e.g., 92:d0:c6:0a:29:33)

-m, --memory bytes Memory limit

--memory-reservation bytes Memory soft limit

--memory-swap bytes Swap limit equal to memory plus swap: '-1' to enable unlimited swap

--memory-swappiness int Tune container memory swappiness (0 to 100) (default -1)

--mount mount Attach a filesystem mount to the container

--name string Assign a name to the container

--network string Connect a container to a network (default "default")

--network-alias list Add network-scoped alias for the container

--no-healthcheck Disable any container-specified HEALTHCHECK

--oom-kill-disable Disable OOM Killer

--oom-score-adj int Tune host's OOM preferences (-1000 to 1000)

--pid string PID namespace to use

--pids-limit int Tune container pids limit (set -1 for unlimited)

--privileged Give extended privileges to this container

-p, --publish list Publish a container's port(s) to the host

-P, --publish-all Publish all exposed ports to random ports

--read-only Mount the container's root filesystem as read only

--restart string Restart policy to apply when a container exits (default "no")

--rm Automatically remove the container when it exits

--runtime string Runtime to use for this container

--security-opt list Security Options

--shm-size bytes Size of /dev/shm

--sig-proxy Proxy received signals to the process (default true)

--stop-signal string Signal to stop a container (default "SIGTERM")

--stop-timeout int Timeout (in seconds) to stop a container

--storage-opt list Storage driver options for the container

--sysctl map Sysctl options (default map[])

--tmpfs list Mount a tmpfs directory

-t, --tty Allocate a pseudo-TTY

--ulimit ulimit Ulimit options (default [])

-u, --user string Username or UID (format: <name|uid>[:<group|gid>])

--userns string User namespace to use

--uts string UTS namespace to use

-v, --volume list Bind mount a volume

--volume-driver string Optional volume driver for the container

--volumes-from list Mount volumes from the specified container(s)

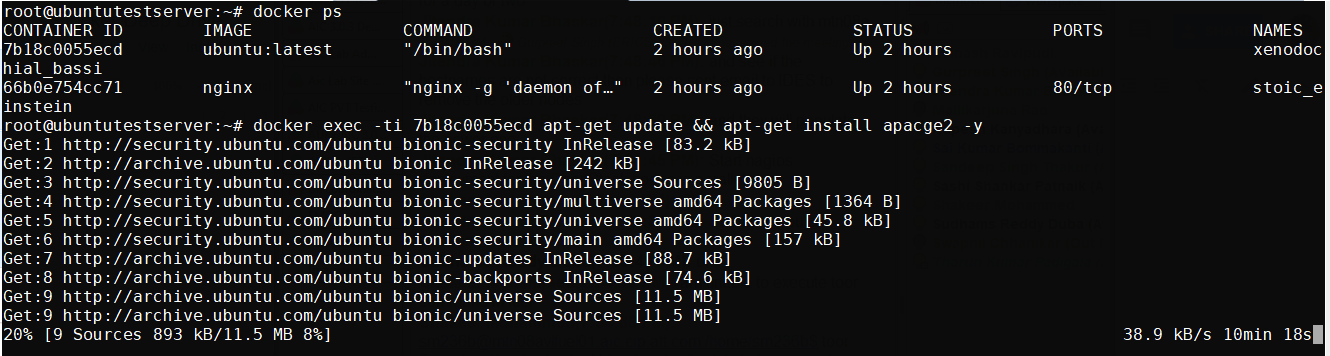
-w, --workdir string Working directory inside the container

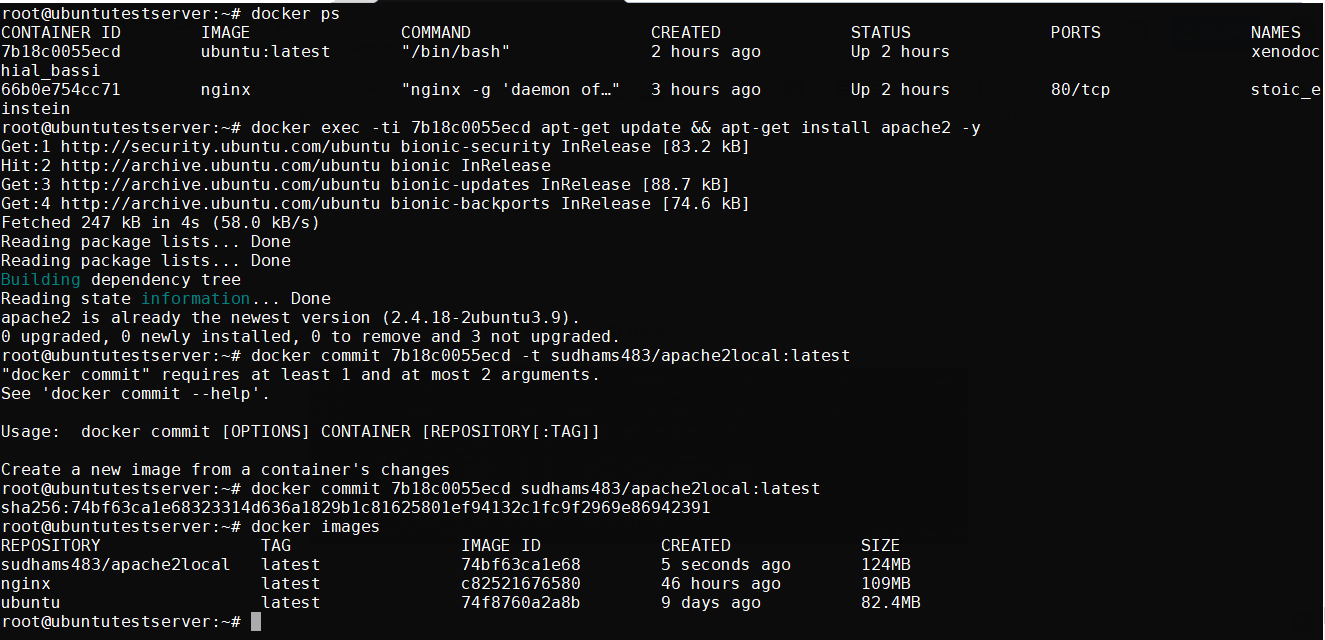
root@dockerserver:~#

Commit:

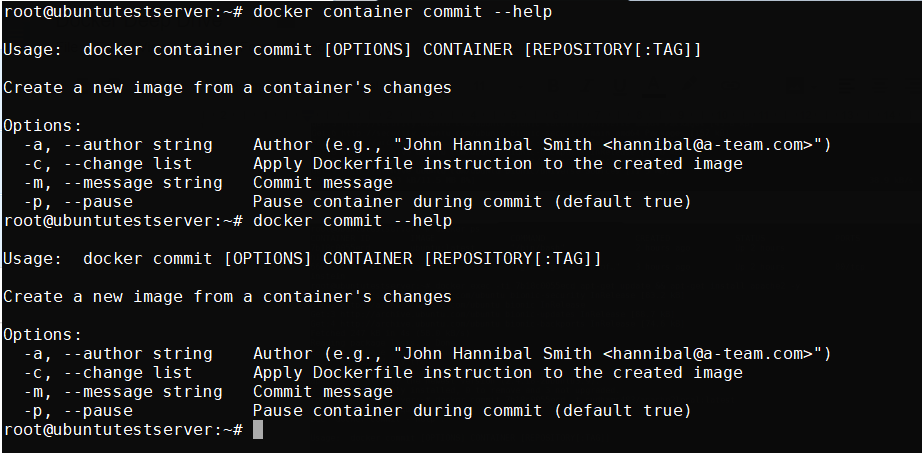
======

Commit is used to convert the container into image.





For options that need to be passed while launching the images from container:



We can also pause the container while it is creating the image from the container, but by default it is enabled.

**Note:** We can also pass the Dockerfile as an input while creating the images from container with option -c.

**Note:**

Logs need to be output to STDOUT and STDERR.

Nginx Example:

RUN ln -sf /dev/stdout /var/log/nginx/access.log \

&& ln -sf /dev/stderr /var/log/nginx/error.log

**Dockerfile Parameters:**

**FROM:** Initializes a new build stage and sets the Base Image

**RUN:** Will execute any commands in a new layer

**CMD:** Provides a default for an executing container. There can only be one CMD instruction in a Dockerfile

**LABEL:** Adds metadata to an image

**EXPOSE:** Informs Docker that the container listens on the specified network ports at runtime

**ENV:** Sets the environment variable <key> to the value <value>

**ADD:** Copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

**COPY:** Copies new files or directories from <src> and adds them to the filesystem of the container at the path <destui**ENTRYPOINT:** Allows for configuring a container that will run as an executable

**VOLUME:** Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers

**USER:** Sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD, and ENTRYPOINT instructions that follow it in the Dockerfile

**WORKDIR:** Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it in the Dockerfile

**ARG:** Defines a variable that users can pass at build-time to the builder with the docker build command, using the --build-arg <varname>=<value> flag

**ONBUILD:** Adds a trigger instruction to the image that will be executed at a later time, when the image is used as the base forced another build

**HEALTHCHECK:** Tells Docker how to test a container to check that it is still working

**SHELL:** Allows the default shell used for the shell form of commands to be overridden

**Building image by piping the Dockerfile through STDIN:**

docker image build -t <NAME>:<TAG> -<<EOF

Build instructions

EOF

Example:

docker image build -t linuxacademy/nginx:stind --rm -<<EOF

FROM nginx:latest

VOLUME ["/usr/share/nginx/html/"]

EOF

**Building an image using a URL:**

docker image build -t <NAME>:<TAG> <GIT\_URL>#<REF>

docker image build -t <NAME>:<TAG> <GIT\_URL>#:<DIRECTORY>

docker image build -t <NAME>:<TAG> <GIT\_URL>#<REF>:<DIRECTORY>

Example:

docker image build -t linuxacademy/weather-app:github https://github.com/linuxacademy/content-weather-app.git#remote-build

**Building an image from a zip file:**

docker image build -t <NAME>:<TAG> - < <FILE>.tar.gz

Example:

cd docker\_images

mkdir tar\_image

cd tar\_image

git clone https://github.com/linuxacademy/content-weather-app.

cd content-weather-app

git checkout remote-build

tar -zcvf weather-app.tar.gz Dockerfile

docker image build -t linuxacademy/weather-app:from-tar - < weather-app.tar.gz

**Docker Machine Overview**

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

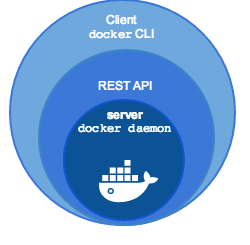
Using docker-machine commands, you can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.



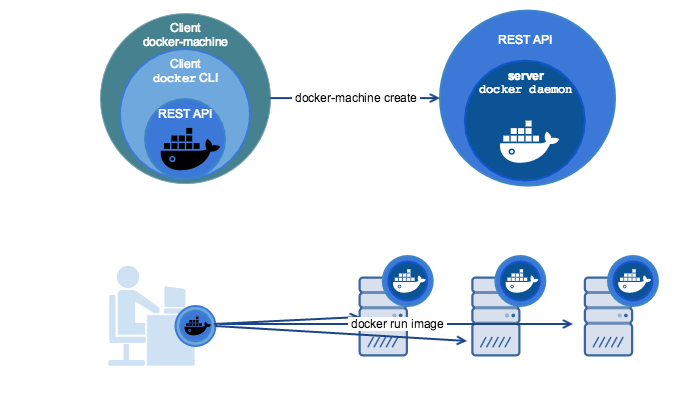
Above diag show the machine trying to run docker-engine on remote machines

Below diagram illustrates the difference between the docker engine and docker machine:

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



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



**How to install docker machine on the nodes?**

First ensure that docker is installed on the machine before we proceed further, now follow the details further mentioned

$ base=https://github.com/docker/machine/releases/download/v0.14.0 &&

curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&

sudo install /tmp/docker-machine /usr/local/bin/docker-machine

Once installation is done, execute the below command to verify the version

$ docker-machine version

docker-machine version 0.14.0, build 9371605

