NAME

docker-run - Run a command in a new container

SYNOPSIS

docker run [**-a|--attach**[=*[]*]] [**--add-host**[=*[]*]] [**--blkio-weight**[=*[BLKIO-WEIGHT]*]] [**--blkio-weight-device**[=*[]*]] [**--cpu-shares**[=*0*]] [**--cap-add**[=*[]*]] [**--cap-drop**[=*[]*]] [**--cgroupns**[=*[]*]] [**--cgroup-parent**[=*CGROUP-PATH*]] [**--cidfile**[=*CIDFILE*]] [**--cpu-count**[=*0*]] [**--cpu-percent**[=*0*]] [**--cpu-period**[=*0*]] [**--cpu-quota**[=*0*]] [**--cpu-rt-period**[=*0*]] [**--cpu-rt-runtime**[=*0*]] [**--cpus**[=*0.0*]] [**--cpuset-cpus**[=*CPUSET-CPUS*]] [**--cpuset-mems**[=*CPUSET-MEMS*]] [**-d|--detach**] [**--detach-keys**[=*[]*]] [**--device**[=*[]*]] [**--device-cgroup-rule**[=*[]*]] [**--device-read-bps**[=*[]*]] [**--device-read-iops**[=*[]*]] [**--device-write-bps**[=*[]*]] [**--device-write-iops**[=*[]*]] [**--dns**[=*[]*]] [**--dns-option**[=*[]*]] [**--dns-search**[=*[]*]] [**--domainname**[=*DOMAINNAME*]] [**-e|--env**[=*[]*]] [**--entrypoint**[=*ENTRYPOINT*]] [**--env-file**[=*[]*]] [**--expose**[=*[]*]] [**--group-add**[=*[]*]] [**-h|--hostname**[=*HOSTNAME*]] [**--help**] [**--init**] [**-i|--interactive**] [**--ip**[=*IPv4-ADDRESS*]] [**--ip6**[=*IPv6-ADDRESS*]] [**--ipc**[=*IPC*]] [**--isolation**[=*default*]] [**--kernel-memory**[=*KERNEL-MEMORY*]] [**-l|--label**[=*[]*]] [**--label-file**[=*[]*]] [**--link**[=*[]*]] [**--link-local-ip**[=*[]*]] [**--log-driver**[=*[]*]] [**--log-opt**[=*[]*]] [**-m|--memory**[=*MEMORY*]] [**--mac-address**[=*MAC-ADDRESS*]] [**--memory-reservation**[=*MEMORY-RESERVATION*]] [**--memory-swap**[=*LIMIT*]] [**--memory-swappiness**[=*MEMORY-SWAPPINESS*]] [**--mount**[=*[MOUNT]*]] [**--name**[=*NAME*]] [**--network-alias**[=*[]*]] [**--network**[=*"bridge"*]] [**--oom-kill-disable**] [**--oom-score-adj**[=*0*]] [**-P|--publish-all**] [**-p|--publish**[=*[]*]] [**--pid**[=*[PID]*]] [**--userns**[=*[]*]] [**--pids-limit**[=*PIDS_LIMIT*]] [**--privileged**] [**--read-only**] [**--restart**[=*RESTART*]] [**--rm**] [**--security-opt**[=*[]*]] [**--storage-opt**[=*[]*]] [**--stop-signal**[=*SIGNAL*]] [**--stop-timeout**[=*TIMEOUT*]] [**--shm-size**[=*[]*]] [**--sig-proxy**[=*true*]] [**--sysctl**[=*[]*]] [**-t|--tty**] [**--tmpfs**[=*[CONTAINER-DIR[:OPTIONS]]*]] [**-u|--user**[=*USER*]] [**--ulimit**[=*[]*]] [**--uts**[=*[]*]] [**-v|--volume**[=*[[HOST-DIR:]CONTAINER-DIR[:OPTIONS]]*]] [**--volume-driver**[=*DRIVER*]] [**--volumes-from**[=*[]*]] [**-w|--workdir**[=*WORKDIR*]] IMAGE [COMMAND] [ARG...]

DESCRIPTION

Run a process in a new container. **docker run** starts a process with its own file system, its own networking, and its own isolated process tree. The IMAGE which starts the process may define defaults related to the process that will be run in the container, the networking to expose, and more, but **docker run** gives final control to the operator or administrator who starts the container from the image. For that reason **docker run** has more options than any other Docker command.

If the IMAGE is not already loaded then **docker run** will pull the IMAGE, and all image dependencies, from the repository in the same way running **docker pull** IMAGE, before it starts the container from that image.

OPTIONS

-a, **--attach=**[]
Attach to STDIN, STDOUT or STDERR.

In foreground mode (the default when **-d** is not specified), **docker run** can start the process in the container and attach the console to the process's standard input, output, and standard error. It can even pretend to be a TTY (this is what most commandline executables expect) and pass along signals. The **-a** option can be set for each of stdin, stdout, and stderr.

--add-host=[]
Add a custom host-to-IP mapping (host:ip)

Add a line to /etc/hosts. The format is hostname:ip. The **--add-host** option can be set multiple times.

--blkio-weight=*0*
Block IO weight (relative weight) accepts a weight value between 10 and 1000.

--blkio-weight-device=[]
Block IO weight (relative device weight, format: `DEVICE_NAME:WEIGHT`).

--cpu-shares=*0*
CPU shares (relative weight)

By default, all containers get the same proportion of CPU cycles. This proportion can be modified by changing the container's CPU share weighting relative to the weighting of all other running containers.

To modify the proportion from the default of 1024, use the **--cpu-shares** flag to set the weighting to 2 or higher.

The proportion will only apply when CPU-intensive processes are running. When tasks in one container are idle, other containers can use the left-over CPU time. The actual amount of CPU time will vary depending on the number of containers running on the system.

For example, consider three containers, one has a cpu-share of 1024 and two others have a cpu-share setting of 512. When processes in all three containers attempt to use 100% of CPU, the first container would receive 50% of the total CPU time. If you add a fourth container with a cpu-share of 1024, the first container only gets 33% of the CPU. The remaining containers receive 16.5%, 16.5% and 33% of the CPU.

On a multi-core system, the shares of CPU time are distributed over all CPU cores. Even if a container is limited to less than 100% of CPU time, it can use 100% of each individual CPU core.

For example, consider a system with more than three cores. If you start one container **{C0}** with **-c=512** running one process, and another container **{C1}** with **-c=1024** running two processes, this can result in the following division of CPU shares:

PID	container	CPU	CPU share	
100	{C0}		0	100% of CPU0
101	{C1}		1	100% of CPU1
102	{C1}		2	100% of CPU2

--cap-add=[]
Add Linux capabilities

--cap-drop=[]
Drop Linux capabilities

--cgroupns=""
Set the cgroup namespace mode for the container.
host: run the container in the host's cgroup namespace
private: run the container in its own private cgroup namespace

"": (unset) use the daemon's default configuration (**host** on cgroup v1, **private** on cgroup v2)

--cgroup-parent=""
Path to cgroups under which the cgroup for the container will be created. If the path is not absolute, the path is considered to be relative to the cgroups path of the init process. Cgroups will be created if they do not already exist.

--cidfile=""
Write the container ID to the file

--cpu-count=*0*
Limit the number of CPUs available for execution by the container.

On Windows Server containers, this is approximated as a percentage of total CPU usage.

On Windows Server containers, the processor resource controls are mutually exclusive, the order of precedence is CP

--cpu-percent=*0*
Limit the percentage of CPU available for execution by a container running on a Windows daemon.

On Windows Server containers, the processor resource controls are mutually exclusive, the order of precedence is CP

--cpu-period=*0*
Limit the CPU CFS (Completely Fair Scheduler) period

Limit the container's CPU usage. This flag tell the kernel to restrict the container's CPU usage to the period you specify.

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

--cpuset-mems=""
Memory nodes (MEMs) in which to allow execution (0-3, 0,1). Only effective on NUMA systems.

If you have four memory nodes on your system (0-3), use `--cpuset-mems=0,1` then processes in your Docker container will only use memory from the first two memory nodes.

--cpu-quota=*0*
Limit the CPU CFS (Completely Fair Scheduler) quota

Limit the container's CPU usage. By default, containers run with the full CPU resource. This flag tell the kernel to restrict the container's CPU usage to the quota you specify.

--cpu-rt-period=0
Limit the CPU real-time period in microseconds

Limit the container's Real Time CPU usage. This flag tell the kernel to restrict the container's Real Time CPU usage to the period you specify.

--cpu-rt-runtime=0
Limit the CPU real-time runtime in microseconds

Limit the containers Real Time CPU usage. This flag tells the kernel to limit the amount of time in a given CPU period Real Time tasks may consume. Ex:
Period of 1,000,000us and Runtime of 950,000us means that this container could consume 95% of available CPU and leave the remaining 5% to normal priority tasks.

The sum of all runtimes across containers cannot exceed the amount allotted to the parent cgroup.

--cpus=0.0
Number of CPUs. The default is *0.0* which means no limit.

-d, **--detach**=*true*|*false*
Detached mode: run the container in the background and print the new container ID. The default is *false*.

At any time you can run **docker ps** in the other shell to view a list of the running containers. You can reattach to a detached container with **docker attach**.

When attached in the tty mode, you can detach from the container (and leave it running) using a configurable key sequence. The default sequence is `CTRL-p CTRL-q`. You configure the key sequence using the **--detach-keys** option or a configuration file. See **config-json(5)** for documentation on using a configuration file.

--detach-keys=*key*
Override the key sequence for detaching a container; *key* is a single character from the [a-Z] range, or **ctrl**-*value*, where *value* is one of: **a-z**, **@**, **^**, **[**, **,**, or **_**.

--device=*onhost*:*incontainer*[:*mode*]
Add a host device *onhost* to the container under the *incontainer* name. Optional *mode* parameter can be used to specify device permissions, it is a combination of **r** (for read), **w** (for write), and **m** (for **mknod**(2)).

For example, **--device=/dev/sdc:/dev/xvdc:rwm** will give a container all permissions for the host device **/dev/sdc**, seen as **/dev/xvdc** inside the container.

--device-cgroup-rule="*type major*:*minor mode*"
Add a rule to the cgroup allowed devices list. The rule is expected to be in the format specified in the Linux kernel documentation (Documentation/cgroup-v1/devices.txt):
- *type*: **a** (all), **c** (char), or **b** (block);
- *major* and *minor*: either a number, or ***** for all;
- *mode*: a composition of **r** (read), **w** (write), and **m** (**mknod**(2)).

Example: **--device-cgroup-rule "c 1:3 mr"**: allow for a character device idendified by **1:3** to be created and read.

--device-read-bps=[]

Limit read rate from a device (e.g. --device-read-bps=/dev/sda:1mb)

--device-read-iops=[]
Limit read rate from a device (e.g. --device-read-iops=/dev/sda:1000)

--device-write-bps=[]
Limit write rate to a device (e.g. --device-write-bps=/dev/sda:1mb)

--device-write-iops=[]
Limit write rate to a device (e.g. --device-write-iops=/dev/sda:1000)

--dns-search=[]
Set custom DNS search domains (Use --dns-search=. if you don't wish to set the search domain)

--dns-option=[]
Set custom DNS options

--dns=[]
Set custom DNS servers

This option can be used to override the DNS configuration passed to the container. Typically this is necessary when the host DNS configuration is invalid for the container (e.g., 127.0.0.1). When this is the case the **--dns** flags is necessary for every run.

--domainname=""
Container NIS domain name

Sets the container's NIS domain name (see also **setdomainname(2)**) that is
available inside the container.

-e, **--env**=[]
Set environment variables

This option allows you to specify arbitrary environment variables that are available for the process that will be launched inside of the container.

--entrypoint=""
Overwrite the default ENTRYPOINT of the image

This option allows you to overwrite the default entrypoint of the image that is set in the Dockerfile. The ENTRYPOINT of an image is similar to a COMMAND because it specifies what executable to run when the container starts, but it is (purposely) more difficult to override. The ENTRYPOINT gives a container its default nature or behavior, so that when you set an ENTRYPOINT you can run the container as if it were that binary, complete with default options, and you can pass in more options via the COMMAND. But, sometimes an operator may want to run something else inside the container, so you can override the default ENTRYPOINT at runtime by using a **--entrypoint** and a string to specify the new ENTRYPOINT.

--env-file=[]
Read in a line delimited file of environment variables

--expose=[]
Expose a port, or a range of ports (e.g. --expose=3300-3310) informs Docker that the container listens on the specified network ports at runtime. Docker uses this information to interconnect containers using links and to set up port redirection on the host system.

--group-add=[]
Add additional groups to run as

-h, **--hostname**=""
Container host name

Sets the container host name that is available inside the container.

--help
Print usage statement

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

-i, **--interactive**=*true*|*false*
Keep STDIN open even if not attached. The default is *false*.

When set to true, keep stdin open even if not attached.

--ip=""
Sets the container's interface IPv4 address (e.g., 172.23.0.9)

It can only be used in conjunction with **--network** for user-defined networks

--ip6=""
Sets the container's interface IPv6 address (e.g., 2001:db8::1b99)

It can only be used in conjunction with **--network** for user-defined networks

--ipc=""
Sets the IPC mode for the container. The following values are accepted:

Value	Description
(empty)	Use daemon's default.
none	Own private IPC namespace, with /dev/shm not mounted.
private	Own private IPC namespace.
shareable	Own private IPC namespace, with a possibility to share it with other containers.
container:*name-or-ID*	Join another ("shareable") container's IPC namespace.
host	Use the host system's IPC namespace.

If not specified, daemon default is used, which can either be **private** or **shareable**, depending on the daemon version and configuration.

--isolation="*default*"
Isolation specifies the type of isolation technology used by containers. Note that the default on Windows server is `process`, and the default on Windows client is `hyperv`. Linux only supports `default`.

-l, **--label** *key*=*value*
Set metadata on the container (for example, **--label com.example.key=value**).

--kernel-memory=*number*[*S*]
Kernel memory limit; *S* is an optional suffix which can be one of **b**, **k**, **m**, or **g**.

Constrains the kernel memory available to a container. If a limit of 0 is specified (not using **--kernel-memory**), the container's kernel memory is not limited. If you specify a limit, it may be rounded up to a multiple of the operating system's page size and the value can be very large, millions of trillions.

--label-file=[]
Read in a line delimited file of labels

--link=*name-or-id*[:*alias*]
Add link to another container.

If the operator uses **--link** when starting the new client container, then the client container can access the exposed port via a private networking interface. Docker will set some environment variables in the client container to help indicate which interface and port to use.

--link-local-ip=[]
Add one or more link-local IPv4/IPv6 addresses to the container's interface

--log-driver="*json-file*|*syslog*|*journald*|*gelf*|*fluentd*|*awslogs*|*splunk*|*etwlogs*|*gcplogs*|*none*"
Logging driver for the container. Default is defined by daemon **--log-driver** flag.
Warning: the `docker logs` command works only for the `json-file` and `journald` logging drivers.

--log-opt=[]
Logging driver specific options.

-m, **--memory**=*number*[*S]
Memory limit; *S* is an optional suffix which can be one of **b**, **k**, **m**, or **g**.

Allows you to constrain the memory available to a container. If the host supports swap memory, then the **-m** memory setting can be larger than physical RAM. If a limit of 0 is specified (not using **-m**), the container's memory is not limited. The actual limit may be rounded up to a multiple of the operating system's page size (the value would be very large, that's millions of trillions).

--memory-reservation=*number*[*S]
Memory soft limit; *S* is an optional suffix which can be one of **b**, **k**, **m**, or **g**.

After setting memory reservation, when the system detects memory contention or low memory, containers are forced to restrict their consumption to their reservation. So you should always set the value below **--memory**, otherwise the hard limit will take precedence. By default, memory reservation will be the same as memory limit.

--memory-swap=*number*[*S*]
Combined memory plus swap limit; *S* is an optional suffix which can be one of **b**, **k**, **m**, or **g**.

This option can only be used together with **--memory**. The argument should always be larger than that of **--memory**. Default is double the value of **--memory**. Set to **-1** to enable unlimited swap.

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

Remember that the MAC address in an Ethernet network must be unique. The IPv6 link-local address will be based on the device's MAC address according to RFC4862.

--mount type=*TYPE*,*TYPE-SPECIFIC-OPTION*[,...]
Attach a filesystem mount to the container

Current supported mount `TYPES` are `bind`, `volume`, and `tmpfs`.

e.g.

```
type=bind,source=/path/on/host,destination=/path/in/container
```

```
type=volume,source=my-volume,destination=/path/in/container,volume-la-
bel="color=red",volume-label="shape=round"
```

```
type=tmpfs,tmpfs-size=512M,destination=/path/in/container
```

Common Options:

- `src`, `source`: mount source spec for `bind` and `volume`. Mandatory for `bind`.
- `dst`, `destination`, `target`: mount destination spec.
- `ro`, `readonly`: `true` or `false` (default).

Note: setting `readonly` for a bind mount does not make its submounts read-only on the current Linux implementation. See also `bind-nonrecursive`.

Options specific to `bind`:

- `bind-propagation`: `shared`, `slave`, `private`, `rshared`, `rslave`, or `rprivate`(default). See also `mount(2)`.
- `consistency`: `consistent`(default), `cached`, or `delegated`. Currently, only effective for Docker for Mac.
- `bind-nonrecursive`: `true` or `false` (default). If set to `true`, submounts are not recursively bind-mounted. This option is useful for `readonly` bind mount.

Options specific to `volume`:

- `volume-driver`: Name of the volume-driver plugin.
- `volume-label`: Custom metadata.
- `volume-nocopy`: `true`(default) or `false`. If set to `false`, the Engine copies existing files and directories under the mount-path into the volume, allowing the host to access them.
- `volume-opt`: specific to a given volume driver.

Options specific to `tmpfs`:

- `tmpfs-size`: Size of the tmpfs mount in bytes. Unlimited by default in Linux.
- `tmpfs-mode`: File mode of the tmpfs in octal. (e.g. `700` or `0700`.) Defaults to `1777` in Linux.

--name=""
Assign a name to the container

The operator can identify a container in three ways:

`Identifier type`	`Example value`
UUID long identifier	"f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778"
UUID short identifier	"f78375b1c487"
Name	"evil_ptolemy"

The UUID identifiers come from the Docker daemon, and if a name is not assigned to the container with **--name** then the daemon will also generate a random string name. The name is useful when defining links (see **--link**) (or any other place you need to identify a container). This works for both background and foreground Docker containers.

--network=*type*
Set the Network mode for the container. Supported values are:

Value	Description
none	No networking in the container.
bridge	Connect the container to the default Docker bridge via veth interfaces.
host	Use the host's network stack inside the container.
container:*name*|*id*	Use the network stack of another container, specified via its *name* or *id*.
network-name|*network-id*	Connects the container to a user created network (using `docker network create` command)

Default is **bridge**.

--network-alias=[]
Add network-scoped alias for the container

--oom-kill-disable=*true*|*false*
Whether to disable OOM Killer for the container or not.

--oom-score-adj=""
Tune the host's OOM preferences for containers (accepts -1000 to 1000)

-P, **--publish-all**=*true*|*false*
Publish all exposed ports to random ports on the host interfaces. The default is *false*.

When set to true publish all exposed ports to the host interfaces. The default is false. If the operator uses -P (or -p) then Docker will make the exposed port accessible on the host and the ports will be available to any client that can reach the host. When using -P, Docker will bind any exposed port to a random port on the host within an *ephemeral port range* defined by `/proc/sys/net/ipv4/ip_local_port_range`. To find the mapping between the host ports and the exposed ports, use `docker port`(1).

-p, **--publish** *ip*:[*hostPort*]:*containerPort* | [*hostPort*:]*containerPort*
Publish a container's port, or range of ports, to the host.

Both *hostPort* and *containerPort* can be specified as a range. When specifying ranges for both, the number of ports in ranges should be equal.

Examples: **-p 1234-1236:1222-1224**, **-p 127.0.0.1:$HOSTPORT:$CONTAINERPORT**.

Use `docker port`(1) to see the actual mapping, e.g. `docker port CONTAINER $CONTAINERPORT`.

--pid=""
Set the PID mode for the container
Default is to create a private PID namespace for the container
'container:': join another container's PID namespace
'host': use the host's PID namespace for the container. Note: the host mode gives the container full access to local PID and is therefore considered insecure.

--userns=""
Set the usernamespace mode for the container when `userns-remap` option is enabled.
host: use the host usernamespace and enable all privileged options (e.g., `pid=host` or `--privileged`).

--pids-limit=""
Tune the container's pids (process IDs) limit. Set to `-1` to have unlimited pids for the container.

--uts=*type*
Set the UTS mode for the container. The only possible *type* is **host**, meaning to use the host's UTS namespace inside the container.
Note: the host mode gives the container access to changing the host's hostname and is therefore considered insecure.

--privileged [**true**|**false**]
Give extended privileges to this container. A "privileged" container is given access to all devices.

When the operator executes **docker run --privileged**, Docker will enable access to all devices on the host as well as set some configuration in AppArmor to allow the container nearly all the same access to the host as processes running outside of a container on the host.

--read-only=true|false
Mount the container's root filesystem as read only.

By default a container will have its root filesystem writable allowing processes to write files anywhere. By specifying the `--read-only` flag the container will have its root filesystem mounted as read only prohibiting any writes.

--restart *policy*
Restart policy to apply when a container exits. Supported values are:

`Policy`	`Result`
no	Do not automatically restart the container when it exits.
on-failure[:*max-retries*]	Restart only if the container exits with a non-zero exit status. Optionally, limit the number of restart retries the Docker daemon attempts.
always	Always restart the container regardless of the exit status. When you specify always, the Docker daemon will try to restart the container indefinitely. The container will also always start on daemon startup, regardless of the current state of the container.
unless-stopped	Always restart the container regardless of the exit status, but do not start it on daemon startup if the container has been put to a stopped state before.

Default is **no**.

--rm true|false
Automatically remove the container when it exits. The default is **false**.
`--rm` flag can work together with `-d`, and auto-removal will be done on daemon side. Note that it's incompatible with any restart policy other than `none`.

--security-opt *value*[,...]
Security Options for the container. The following options can be given:

"label=user:USER" : Set the label user for the container
"label=role:ROLE" : Set the label role for the container
"label=type:TYPE" : Set the label type for the container
"label=level:LEVEL" : Set the label level for the container
"label=disable" : Turn off label confinement for the container
"no-new-privileges" : Disable container processes from gaining additional privileges

"seccomp=unconfined" : Turn off seccomp confinement for the container
"seccomp=profile.json : White listed syscalls seccomp Json file to be used as a seccomp filter

"apparmor=unconfined" : Turn off apparmor confinement for the container
"apparmor=your-profile" : Set the apparmor confinement profile for the container

--storage-opt
Storage driver options per container

$ docker run -it --storage-opt size=120G fedora /bin/bash

This (size) will allow to set the container rootfs size to 120G at creation time.
This option is only available for the `devicemapper`, `btrfs`, `overlay2` and `zfs` graph drivers.
For the `devicemapper`, `btrfs` and `zfs` storage drivers, user cannot pass a size less than the Default BaseFS Size.
For the `overlay2` storage driver, the size option is only available if the backing fs is `xfs` and mounted with the `pquota` mount option.
Under these conditions, user can pass any size less than the backing fs size.

--stop-signal=*SIGTERM*
Signal to stop the container. Default is SIGTERM.

The `--stop-signal` flag sets the system call signal that will be sent to the
container to exit. This signal can be a signal name in the format `SIG<NAME>`,
for instance `SIGKILL`, or an unsigned number that matches a position in the
kernel's syscall table, for instance `9`.

--stop-timeout
Timeout (in seconds) to stop a container, or **-1** to disable timeout.

The `--stop-timeout` flag sets the number of seconds to wait for the container
to stop after sending the pre-defined (see `--stop-signal`) system call signal.

If the container does not exit after the timeout elapses, it is forcibly killed with a `SIGKILL` signal.

If `--stop-timeout` is set to **-1**, no timeout is applied, and the daemon will wait indefinitely for the container to exit.

The default is determined by the daemon, and 10 seconds for Linux containers, and 30 seconds for Windows containers.

--shm-size=""
Size of `/dev/shm`. The format is `<number><unit>`.
`number` must be greater than `0`. Unit is optional and can be `b` (bytes), `k` (kilobytes), `m`(megabytes), or `g` (gigabytes).
If you omit the unit, the system uses bytes. If you omit the size entirely, the system uses `64m`.

--sysctl=SYSCTL
Configure namespaced kernel parameters at runtime

IPC Namespace - current sysctls allowed:

kernel.msgmax, kernel.msgmnb, kernel.msgmni, kernel.sem, kernel.shmall, kernel.shmmax, kernel.shmmni, kernel.shm_rmid_forced
Sysctls beginning with fs.mqueue.*

If you use the `--ipc=host` option these sysctls will not be allowed.

Network Namespace - current sysctls allowed:
Sysctls beginning with net.*

If you use the `--network=host` option these sysctls will not be allowed.

--sig-proxy=*true|false*
Proxy received signals to the process (non-TTY mode only). SIGCHLD, SIGSTOP, and SIGKILL are not proxied. The default is *true*.

--memory-swappiness=""
Tune a container's memory swappiness behavior. Accepts an integer between 0 and 100.

-t, **--tty**=*true|false*
Allocate a pseudo-TTY. The default is *false*.

When set to true Docker can allocate a pseudo-tty and attach to the standard input of any container. This can be used, for example, to run a throwaway interactive shell. The default is false.

The **-t** option is incompatible with a redirection of the docker client standard input.

--tmpfs=[] Create a tmpfs mount

Mount a temporary filesystem (`tmpfs`) mount into a container, for example:

$ docker run -d --tmpfs /tmp:rw,size=787448k,mode=1777 my_image

This command mounts a `tmpfs` at `/tmp` within the container. The supported mount options are the same as the Linux default `mount` flags. If you do not specify any options, the systems uses the following options: `rw,noexec,nosuid,nodev,size=65536k`.

See also `--mount`, which is the successor of `--tmpfs` and `--volume`.
Even though there is no plan to deprecate `--tmpfs`, usage of `--mount` is recommended.

-u, **--user**=""
Sets the username or UID used and optionally the groupname or GID for the specified command.

The followings examples are all valid:
--user [user | user:group | uid | uid:gid | user:gid | uid:group]

Without this argument the command will be run as root in the container.

--ulimit=[]
Ulimit options

-v|**--volume**[=*[[HOST-DIR:]CONTAINER-DIR[:OPTIONS]]*]
Create a bind mount. If you specify, `-v /HOST-DIR:/CONTAINER-DIR`, Docker bind mounts `/HOST-DIR` in the host to `/CONTAINER-DIR` in the Docker container. If 'HOST-DIR' is omitted, Docker automatically creates the new volume on the host. The `OPTIONS` are a comma delimited list and can be:

- [rw|ro]
- [z|Z]
- `[[r]shared|[r]slave|[r]private]`
- `[delegated|cached|consistent]`
- [nocopy]

The `CONTAINER-DIR` must be an absolute path such as `/src/docs`. The `HOST-DIR` can be an absolute path or a `name` value. A `name` value must start with an alphanumeric character, followed by `a-z0-9`, _ (underscore), `.` (period) or - (hyphen). An absolute path starts with a `/` (forward slash).

If you supply a `HOST-DIR` that is an absolute path, Docker bind-mounts to the path you specify. If you supply a `name`, Docker creates a named volume by that `name`. For example, you can specify either `/foo` or `foo` for a `HOST-DIR` value. If you supply the `/foo` value, Docker creates a bind mount. If you supply the `foo` specification, Docker creates a named volume.

You can specify multiple **-v** options to mount one or more mounts to a container. To use these same mounts in other containers, specify the **--volumes-from** option also.

You can supply additional options for each bind mount following an additional colon. A `:ro` or `:rw` suffix

mounts a volume in read-only or read-write mode, respectively. By default, volumes are mounted in read-write mode. You can also specify the consistency requirement for the mount, either `:consistent` (the default), `:cached`, or `:delegated`. Multiple options are separated by commas, e.g. `:ro,cached`.

Labeling systems like SELinux require that proper labels are placed on volume content mounted into a container. Without a label, the security system might prevent the processes running inside the container from using the content. By default, Docker does not change the labels set by the OS.

To change a label in the container context, you can add either of two suffixes `:z` or `:Z` to the volume mount. These suffixes tell Docker to relabel file objects on the shared volumes. The `z` option tells Docker that two containers share the volume content. As a result, Docker labels the content with a shared content label. Shared volume labels allow all containers to read/write content. The `Z` option tells Docker to label the content with a private unshared label. Only the current container can use a private volume.

By default bind mounted volumes are `private`. That means any mounts done inside container will not be visible on host and vice-a-versa. One can change this behavior by specifying a volume mount propagation property. Making a volume `shared` mounts done under that volume inside container will be visible on host and vice-a-versa. Making a volume `slave` enables only one way mount propagation and that is mounts done on host under that volume will be visible inside container but not the other way around.

To control mount propagation property of volume one can use `:[r]shared`, `:[r]slave` or `:[r]private` propagation flag. Propagation property can be specified only for bind mounted volumes and not for internal volumes or named volumes. For mount propagation to work source mount point (mount point where source dir is mounted on) has to have right propagation properties. For shared volumes, source mount point has to be shared. And for slave volumes, source mount has to be either shared or slave.

Use `df <source-dir>` to figure out the source mount and then use `findmnt -o TARGET,PROPAGATION <source-mount-dir>` to figure out propagation properties of source mount. If `findmnt` utility is not available, then one can look at mount entry for source mount point in `/proc/self/mountinfo`. Look at `optional fields` and see if any propagation properties are specified. `shared:X` means mount is `shared`, `master:X` means mount is `slave` and if nothing is there that means mount is `private`.

To change propagation properties of a mount point use `mount` command. For example, if one wants to bind mount source directory `/foo` one can do `mount --bind /foo /foo` and `mount --make-private --make-shared /foo`. This will convert /foo into a `shared` mount point. Alternatively one can directly change propagation properties of source mount. Say `/` is source mount for `/foo`, then use `mount --make-shared /` to convert `/` into a `shared` mount.

> **Note**: When using systemd to manage the Docker daemon's start and stop, in the systemd unit file there is an option to control mount propagation for the Docker daemon itself, called `MountFlags`. The value of this setting may cause Docker to not see mount propagation changes made on the mount point. For example, if this value is `slave`, you may not be able to use the `shared` or `rshared` propagation on a volume.

To disable automatic copying of data from the container path to the volume, use the `nocopy` flag. The `nocopy` flag can be set on bind mounts and named volumes.

See also `--mount`, which is the successor of `--tmpfs` and `--volume`. Even though there is no plan to deprecate `--volume`, usage of `--mount` is recommended.

--volume-driver=""
Container's volume driver. This driver creates volumes specified either from a Dockerfile's `VOLUME` instruction or from the `docker run -v` flag. See **docker-volume-create(1)** for full details.

--volumes-from=[]
Mount volumes from the specified container(s)

Mounts already mounted volumes from a source container onto another container. You must supply the source's container-id. To share a volume, use the **--volumes-from** option when running the target container. You can share volumes even if the source container is not running.

By default, Docker mounts the volumes in the same mode (read-write or read-only) as it is mounted in the source container. Optionally, you can change this by suffixing the container-id with either the `:ro` or `:rw` keyword.

If the location of the volume from the source container overlaps with data residing on a target container, then the volume hides that data on the target.

-w, **--workdir**=""
Working directory inside the container

The default working directory for running binaries within a container is the root directory (/). The developer can set a different default with the Dockerfile WORKDIR instruction. The operator can override the working directory by using the **-w** option.

Exit Status

The exit code from `docker run` gives information about why the container failed to run or why it exited. When `docker run` exits with a non-zero code, the exit codes follow the `chroot` standard, see below:

125 if the error is with Docker daemon itself

```
$ docker run --foo busybox; echo $?
# flag provided but not defined: --foo
 See 'docker run --help'.
 125
```

126 if the contained command cannot be invoked

```
$ docker run busybox /etc; echo $?
# exec: "/etc": permission denied
 docker: Error response from daemon: Contained command could not be invoked
 126
```

127 if the contained command cannot be found

```
$ docker run busybox foo; echo $?
# exec: "foo": executable file not found in $PATH
  docker: Error response from daemon: Contained command not found or does not exist
  127
```

Exit code of contained command otherwise

```
$ docker run busybox /bin/sh -c 'exit 3'
# 3
```

EXAMPLES

Running container in read-only mode

During container image development, containers often need to write to the image content. Installing packages into /usr, for example. In production, applications seldom need to write to the image. Container applications write to volumes if they need to write to file systems at all. Applications can be made more secure by running them in read-only mode using the --read-only switch. This protects the containers image from modification. Read only containers may still need to write temporary data. The best way to handle this is to mount tmpfs directories on /run and /tmp.

```
# docker run --read-only --tmpfs /run --tmpfs /tmp -i -t fedora /bin/bash
```

Exposing log messages from the container to the host's log

If you want messages that are logged in your container to show up in the host's syslog/journal then you should bind mount the /dev/log directory as follows.

```
# docker run -v /dev/log:/dev/log -i -t fedora /bin/bash
```

From inside the container you can test this by sending a message to the log.

```
(bash)# logger "Hello from my container"
```

Then exit and check the journal.

```
# exit

# journalctl -b | grep Hello
```

This should list the message sent to logger.

Attaching to one or more from STDIN, STDOUT, STDERR

If you do not specify -a then Docker will attach everything (stdin,stdout,stderr) you'd like to connect instead, as in:

```
# docker run -a stdin -a stdout -i -t fedora /bin/bash
```

Sharing IPC between containers

Using shm_server.c available here: https://www.cs.cf.ac.uk/Dave/C/node27.html

Testing `--ipc=host` mode:

Host shows a shared memory segment with 7 pids attached, happens to be from httpd:

```
$ sudo ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01128e25 0          root       600        1000       7
```

Now run a regular container, and it correctly does NOT see the shared memory segment from the host:

```
$ docker run -it shm ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
```

Run a container with the new `--ipc=host` option, and it now sees the shared memory segment from the host httpd:

```
$ docker run -it --ipc=host shm ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01128e25 0          root       600        1000       7
```

Testing `--ipc=container:CONTAINERID` mode:

Start a container with a program to create a shared memory segment:

```
$ docker run -it shm bash
$ sudo shm/shm_server &
$ sudo ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000162e 0          root       666        27         1
```

Create a 2nd container correctly shows no shared memory segment from 1st container:

```
$ docker run shm ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
```

Create a 3rd container using the new --ipc=container:CONTAINERID option, now it shows the shared memory segment from the first:

```
$ docker run -it --ipc=container:ed735b2264ac shm ipcs -m
$ sudo ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000162e 0          root       666        27         1
```

Linking Containers

Note: This section describes linking between containers on the default (bridge) network, also known as "legacy links". Using `--link` on user-defined networks uses the DNS-based discovery, which does not add entries to `/etc/hosts`, and does not set environment variables for discovery.

The link feature allows multiple containers to communicate with each other. For example, a container whose Dockerfile has exposed port 80 can be run and named as follows:

```
# docker run --name=link-test -d -i -t fedora/httpd
```

A second container, in this case called linker, can communicate with the httpd container, named link-test, by running with the **--link=:**

```
# docker run -t -i --link=link-test:lt --name=linker fedora /bin/bash
```

Now the container linker is linked to container link-test with the alias lt. Running the **env** command in the linker container shows environment variables
with the LT (alias) context (**LT_**)

```
# env
HOSTNAME=668231cb0978
TERM=xterm
LT_PORT_80_TCP=tcp://172.17.0.3:80
LT_PORT_80_TCP_PORT=80
LT_PORT_80_TCP_PROTO=tcp
LT_PORT=tcp://172.17.0.3:80
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

```
PWD=/
LT_NAME=/linker/lt
SHLVL=1
HOME=/
LT_PORT_80_TCP_ADDR=172.17.0.3
_=/usr/bin/env
```

When linking two containers Docker will use the exposed ports of the container to create a secure tunnel for the parent to access.

If a container is connected to the default bridge network and `linked` with other containers, then the container's `/etc/hosts` file is updated with the linked container's name.

> **Note** Since Docker may live update the container's `/etc/hosts` file, there may be situations when processes inside the container can end up reading an empty or incomplete `/etc/hosts` file. In most cases, retrying the read again should fix the problem.

Mapping Ports for External Usage

The exposed port of an application can be mapped to a host port using the **-p** flag. For example, an httpd port 80 can be mapped to the host port 8080 using the following:

```
# docker run -p 8080:80 -d -i -t fedora/httpd
```

Creating and Mounting a Data Volume Container

Many applications require the sharing of persistent data across several containers. Docker allows you to create a Data Volume Container that other containers can mount from. For example, create a named container that contains directories /var/volume1 and /tmp/volume2. The image will need to contain these directories so a couple of RUN mkdir instructions might be required for you fedora-data image:

```
# docker run --name=data -v /var/volume1 -v /tmp/volume2 -i -t fedora-data true
# docker run --volumes-from=data --name=fedora-container1 -i -t fedora bash
```

Multiple --volumes-from parameters will bring together multiple data volumes from multiple containers. And it's possible to mount the volumes that came from the DATA container in yet another container via the fedora-container1 intermediary container, allowing to abstract the actual data source from users of that data:

```
# docker run --volumes-from=fedora-container1 --name=fedora-container2 -i -t fedora bash
```

Mounting External Volumes

To mount a host directory as a container volume, specify the absolute path to the directory and the absolute path for the container directory separated by a colon:

```
# docker run -v /var/db:/data1 -i -t fedora bash
```

When using SELinux, be aware that the host has no knowledge of container SELinux policy. Therefore, in the above example, if SELinux policy is enforced, the `/var/db` directory is not writable to the container. A "Permission Denied" message will occur and an avc: message in the host's syslog.

To work around this, at time of writing this man page, the following command needs to be run in order for the proper SELinux policy type label to be attached to the host directory:

```
# chcon -Rt svirt_sandbox_file_t /var/db
```

Now, writing to the /data1 volume in the container will be allowed and the changes will also be reflected on the host in /var/db.

Using alternative security labeling

You can override the default labeling scheme for each container by specifying the `--security-opt` flag. For example, you can specify the MCS/MLS level, a requirement for MLS systems. Specifying the level in the following command allows you to share the same content between containers.

```
# docker run --security-opt label=level:s0:c100,c200 -i -t fedora bash
```

An MLS example might be:

```
# docker run --security-opt label=level:TopSecret -i -t rhel7 bash
```

To disable the security labeling for this container versus running with the `--permissive` flag, use the following command:

```
# docker run --security-opt label=disable -i -t fedora bash
```

If you want a tighter security policy on the processes within a container, you can specify an alternate type for the container. You could run a container that is only allowed to listen on Apache ports by executing the following command:

```
# docker run --security-opt label=type:svirt_apache_t -i -t centos bash
```

Note:

You would have to write policy defining a `svirt_apache_t` type.

Setting device weight

If you want to set `/dev/sda` device weight to `200`, you can specify the device weight by `--blkio-weight-device` flag. Use the following command:

```
# docker run -it --blkio-weight-device "/dev/sda:200" ubuntu
```

Specify isolation technology for container (--isolation)

This option is useful in situations where you are running Docker containers on Microsoft Windows. The `--isolation <value>` option sets a container's isolation technology. On Linux, the only supported is the `default` option which uses Linux namespaces. These two commands are equivalent on Linux:

```
$ docker run -d busybox top
$ docker run -d --isolation default busybox top
```

On Microsoft Windows, can take any of these values:

- `default`: Use the value specified by the Docker daemon's `--exec-opt` . If the `daemon` does not specify an isolation technology, Microsoft Windows uses `process` as its default value.
- `process`: Namespace isolation only.
- `hyperv`: Hyper-V hypervisor partition-based isolation.

In practice, when running on Microsoft Windows without a `daemon` option set, these two commands are equivalent:

```
$ docker run -d --isolation default busybox top
$ docker run -d --isolation process busybox top
```

If you have set the `--exec-opt isolation=hyperv` option on the Docker `daemon`, any of these commands also result in `hyperv` isolation:

```
$ docker run -d --isolation default busybox top
$ docker run -d --isolation hyperv busybox top
```

Setting Namespaced Kernel Parameters (Sysctls)

The `--sysctl` sets namespaced kernel parameters (sysctls) in the container. For example, to turn on IP forwarding in the containers network namespace, run this command:

```
$ docker run --sysctl net.ipv4.ip_forward=1 someimage
```

Note:

Not all sysctls are namespaced. Docker does not support changing sysctls inside of a container that also modify the host system. As the kernel evolves we expect to see more sysctls become namespaced.

See the definition of the `--sysctl` option above for the current list of supported sysctls.

HISTORY

April 2014, Originally compiled by William Henry (whenry at redhat dot com) based on docker.com source material and internal work. June 2014, updated by Sven Dowideit SvenDowideit@home.org.au ⟨mailto:SvenDowideit@home.org.au⟩ July 2014, updated by Sven Dowideit SvenDowideit@home.org.au ⟨mailto:SvenDowideit@home.org.au⟩ November 2015, updated by Sally O'Malley somalley@redhat.com ⟨mailto:somalley@redhat.com⟩