docker run

Estimated reading time: 32 minutes

Description

Run a command in a new container

Usage

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

Options

Name, shorthand	Default	Description
`--add-host`		Add a custom host-to-IP mapping (host:ip)
`--attach , -a`		Attach to STDIN, STDOUT or STDERR
`--blkio-weight`		Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)
`--blkio-weight-device`		Block IO weight (relative device weight)
`--cap-add`		Add Linux capabilities
`--cap-drop`		Drop Linux capabilities
`--cgroup-parent`		Optional parent cgroup for the container
`--cidfile`		Write the container ID to the file
`--cpu-count`		CPU count (Windows only)
`--cpu-percent`		CPU percent (Windows only)
`--cpu-period`		Limit CPU CFS (Completely Fair Scheduler) period
`--cpu-quota`		Limit CPU CFS (Completely Fair Scheduler) quota
`--cpu-rt-period`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Limit CPU real-time period in microseconds
`--cpu-rt-runtime`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Limit CPU real-time runtime in microseconds
`--cpu-shares , -c`		CPU shares (relative weight)
`--cpus`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Number of CPUs
`--cpuset-cpus`		CPUs in which to allow execution (0-3, 0,1)
`--cpuset-mems`		MEMs in which to allow execution (0-3, 0,1)

Name, shorthand	Default	Description
`--detach , -d`		Run container in background and print container ID
`--detach-keys`		Override the key sequence for detaching a container
`--device`		Add a host device to the container
`--device-cgroup-rule`		Add a rule to the cgroup allowed devices list
`--device-read-bps`		Limit read rate (bytes per second) from a device
`--device-read-iops`		Limit read rate (IO per second) from a device
`--device-write-bps`		Limit write rate (bytes per second) to a device
`--device-write-iops`		Limit write rate (IO per second) to a device
`--disable-content-trust`	`true`	Skip image verification
`--dns`		Set custom DNS servers
`--dns-opt`		Set DNS options
`--dns-option`		Set DNS options
`--dns-search`		Set custom DNS search domains
`--entrypoint`		Overwrite the default ENTRYPOINT of the image
`--env , -e`		Set environment variables
`--env-file`		Read in a file of environment variables
`--expose`		Expose a port or a range of ports
`--group-add`		Add additional groups to join
`--health-cmd`		Command to run to check health
`--health-interval`		Time between running the check (ms|s|m|h) (default 0s)
`--health-retries`		Consecutive failures needed to report unhealthy
`--health-start-period`		API 1.29+ (https://docs.docker.com/engine/api/v1.29/) Start period for the container to initialize before starting health-retries countdown (ms|s|m|h) (default 0s)
`--health-timeout`		Maximum time to allow one check to run (ms|s|m|h) (default 0s)
`--help`		Print usage
`--hostname , -h`		Container host name
`--init`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Run an init inside the container that forwards signals and reaps processes
`--interactive , -i`		Keep STDIN open even if not attached
`--io-maxbandwidth`		Maximum IO bandwidth limit for the system drive (Windows only)
`--io-maxiops`		Maximum IOps limit for the system drive (Windows only)
`--ip`		IPv4 address (e.g., 172.30.100.104)

Name, shorthand	Default	Description
`--ip6`		IPv6 address (e.g., 2001:db8::33)
`--ipc`		IPC mode to use
`--isolation`		Container isolation technology
`--kernel-memory`		Kernel memory limit
`--label , -l`		Set meta data on a container
`--label-file`		Read in a line delimited file of labels
`--link`		Add link to another container
`--link-local-ip`		Container IPv4/IPv6 link-local addresses
`--log-driver`		Logging driver for the container
`--log-opt`		Log driver options
`--mac-address`		Container MAC address (e.g., 92:d0:c6:0a:29:33)
`--memory , -m`		Memory limit
`--memory-reservation`		Memory soft limit
`--memory-swap`		Swap limit equal to memory plus swap: '-1' to enable unlimited swap
`--memory-swappiness`	`-1`	Tune container memory swappiness (0 to 100)
`--mount`		Attach a filesystem mount to the container
`--name`		Assign a name to the container
`--net`		Connect a container to a network
`--net-alias`		Add network-scoped alias for the container
`--network`		Connect a container to a network
`--network-alias`		Add network-scoped alias for the container
`--no-healthcheck`		Disable any container-specified HEALTHCHECK
`--oom-kill-disable`		Disable OOM Killer
`--oom-score-adj`		Tune host's OOM preferences (-1000 to 1000)
`--pid`		PID namespace to use
`--pids-limit`		Tune container pids limit (set -1 for unlimited)
`--platform`		experimental (daemon) (https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) API 1.32+ (https://docs.docker.com/engine/api/v1.32/) Set platform if server is multi-platform capable
`--privileged`		Give extended privileges to this container
`--publish , -p`		Publish a container's port(s) to the host
`--publish-all , -P`		Publish all exposed ports to random ports

Name, shorthand	Default	Description
`--read-only`		Mount the container’s root filesystem as read only
`--restart`	`no`	Restart policy to apply when a container exits
`--rm`		Automatically remove the container when it exits
`--runtime`		Runtime to use for this container
`--security-opt`		Security Options
`--shm-size`		Size of /dev/shm
`--sig-proxy`	`true`	Proxy received signals to the process
`--stop-signal`	`SIGTERM`	Signal to stop a container
`--stop-timeout`		API 1.25+ (https://docs.docker.com/engine/api/v1.25/) Timeout (in seconds) to stop a container
`--storage-opt`		Storage driver options for the container
`--sysctl`		Sysctl options
`--tmpfs`		Mount a tmpfs directory
`--tty` , `-t`		Allocate a pseudo-TTY
`--ulimit`		Ulimit options
`--user` , `-u`		Username or UID (format: <name|uid>[:<group|gid>])
`--userns`		User namespace to use
`--uts`		UTS namespace to use
`--volume` , `-v`		Bind mount a volume
`--volume-driver`		Optional volume driver for the container
`--volumes-from`		Mount volumes from the specified container(s)
`--workdir` , `-w`		Working directory inside the container

Parent command

Command	Description
docker (https://docs.docker.com/engine/reference/commandline/docker)	The base command for the Docker CLI.

Extended description

The `docker run` command first `creates` a writeable container layer over the specified image, and then `starts` it using the specified command. That is, `docker run` is equivalent to the API `/containers/create` then `/containers/(id)/start` . A stopped container can be restarted with all its previous changes intact using `docker start` . See `docker ps -a` to view a list of all containers.

The `docker run` command can be used in combination with `docker commit` to *change the command that a container runs* (https://docs.docker.com/engine/reference/commandline/commit/). There is additional detailed information about `docker run` in the Docker run reference (https://docs.docker.com/engine/reference/run/).

For information on connecting a container to a network, see the "*Docker network overview*" (https://docs.docker.com/engine/userguide/networking/).

Examples

Assign name and allocate pseudo-TTY (--name, -it)

```
$ docker run --name test -it debian

root@d6c0fe130dba:/# exit 13
$ echo $?
13
$ docker ps -a | grep test
d6c0fe130dba        debian:7            "/bin/bash"         26 seconds ago      Exited (13) 17 se
conds ago                         test
```

This example runs a container named `test` using the `debian:latest` image. The `-it` instructs Docker to allocate a pseudo-TTY connected to the container's stdin; creating an interactive `bash` shell in the container. In the example, the `bash` shell is quit by entering `exit 13`. This exit code is passed on to the caller of `docker run`, and is recorded in the `test` container's metadata.

Capture container ID (--cidfile)

```
$ docker run --cidfile /tmp/docker_test.cid ubuntu echo "test"
```

This will create a container and print `test` to the console. The `cidfile` flag makes Docker attempt to create a new file and write the container ID to it. If the file exists already, Docker will return an error. Docker will close this file when `docker run` exits.

Full container capabilities (--privileged)

```
$ docker run -t -i --rm ubuntu bash
root@bc338942ef20:/# mount -t tmpfs none /mnt
mount: permission denied
```

This will *not* work, because by default, most potentially dangerous kernel capabilities are dropped; including `cap_sys_admin` (which is required to mount filesystems). However, the `--privileged` flag will allow it to run:

```
$ docker run -t -i --privileged ubuntu bash
root@50e3f57e16e6:/# mount -t tmpfs none /mnt
root@50e3f57e16e6:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
none            1.9G     0  1.9G   0% /mnt
```

The `--privileged` flag gives *all* capabilities to the container, and it also lifts all the limitations enforced by the `device` cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

Set working directory (-w)

```
$ docker  run -w /path/to/dir/ -i -t  ubuntu pwd
```

The `-w` lets the command being executed inside directory given, here `/path/to/dir/`. If the path does not exist it is created inside the container.

Set 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`, `windowsfilter` and `zfs` graph drivers. For the `devicemapper`, `btrfs`, `windowsfilter` and `zfs` graph 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.

Mount tmpfs (--tmpfs)

```
$ docker run -d --tmpfs /run:rw,noexec,nosuid,size=65536k my_image
```

The `--tmpfs` flag mounts an empty tmpfs into the container with the `rw`, `noexec`, `nosuid`, `size=65536k` options.

Mount volume (-v, --read-only)

```
$ docker  run  -v `pwd`:`pwd` -w `pwd` -i -t  ubuntu pwd
```

The `-v` flag mounts the current working directory into the container. The `-w` lets the command being executed inside the current working directory, by changing into the directory to the value returned by `pwd`. So this combination executes the command using the container, but inside the current working directory.

```
$ docker run -v /doesnt/exist:/foo -w /foo -i -t ubuntu bash
```

When the host directory of a bind-mounted volume doesn't exist, Docker will automatically create this directory on the host for you. In the example above, Docker will create the `/doesnt/exist` folder before starting your container.

```
$ docker run --read-only -v /icanwrite busybox touch /icanwrite/here
```

Volumes can be used in combination with `--read-only` to control where a container writes files. The `--read-only` flag mounts the container's root filesystem as read only prohibiting writes to locations other than the specified volumes for the container.

```
$ docker run -t -i -v /var/run/docker.sock:/var/run/docker.sock -v /path/to/static-docker-binary:
/usr/bin/docker busybox sh
```

By bind-mounting the docker unix socket and statically linked docker binary (refer to get the linux binary (https://docs.docker.com/engine/installation/binaries/#/get-the-linux-binary)), you give the container the full access to create and manipulate the host's Docker daemon.

On Windows, the paths must be specified using Windows-style semantics.

```
PS C:\> docker run -v c:\foo:c:\dest microsoft/nanoserver cmd /s /c type c:\dest\somefile.txt
Contents of file

PS C:\> docker run -v c:\foo:d: microsoft/nanoserver cmd /s /c type d:\somefile.txt
Contents of file
```

The following examples will fail when using Windows-based containers, as the destination of a volume or bind mount inside the container must be one of: a non-existing or empty directory; or a drive other than C:. Further, the source of a bind mount must be a local directory, not a file.

```
net use z: \\remotemachine\share
docker run -v z:\foo:c:\dest ...
docker run -v \\uncpath\to\directory:c:\dest ...
docker run -v c:\foo\somefile.txt:c:\dest ...
docker run -v c:\foo:c: ...
docker run -v c:\foo:c:\existing-directory-with-contents ...
```

For in-depth information about volumes, refer to manage data in containers (https://docs.docker.com/engine/tutorials/dockervolumes/)

Add bind mounts or volumes using the --mount flag

The `--mount` flag allows you to mount volumes, host-directories and `tmpfs` mounts in a container.

The `--mount` flag supports most options that are supported by the `-v` or the `--volume` flag, but uses a different syntax. For in-depth information on the `--mount` flag, and a comparison between `--volume` and `--mount`, refer to the service create command reference (https://docs.docker.com/engine/reference/commandline/service_create/#add-bind-mounts-or-volumes).

Even though there is no plan to deprecate `--volume`, usage of `--mount` is recommended.

Examples:

```
$ docker run --read-only --mount type=volume,target=/icanwrite busybox touch /icanwrite/here
```

```
$ docker run -t -i --mount type=bind,src=/data,dst=/data busybox sh
```

Publish or expose port (-p, --expose)

```
$ docker run -p 127.0.0.1:80:8080/tcp ubuntu bash
```

This binds port `8080` of the container to TCP port `80` on `127.0.0.1` of the host machine. You can also specify `udp` and `sctp` ports. The Docker User Guide (https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/) explains in detail how to manipulate ports in Docker.

```
$ docker run --expose 80 ubuntu bash
```

This exposes port `80` of the container without publishing the port to the host system's interfaces.

Set environment variables (-e, --env, --env-file)

```
$ docker run -e MYVAR1 --env MYVAR2=foo --env-file ./env.list ubuntu bash
```

Use the `-e`, `--env`, and `--env-file` flags to set simple (non-array) environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.

You can define the variable and its value when running the container:

```
$ docker run --env VAR1=value1 --env VAR2=value2 ubuntu env | grep VAR
VAR1=value1
VAR2=value2
```

You can also use variables that you've exported to your local environment:

```
export VAR1=value1
export VAR2=value2

$ docker run --env VAR1 --env VAR2 ubuntu env | grep VAR
VAR1=value1
VAR2=value2
```

When running the command, the Docker CLI client checks the value the variable has in your local environment and passes it to the container. If no `=` is provided and that variable is not exported in your local environment, the variable won't be set in the container.

You can also load the environment variables from a file. This file should use the syntax `<variable>=value` (which sets the variable to the given value) or `<variable>` (which takes the value from the local environment), and `#` for comments.

```
$ cat env.list
# This is a comment
VAR1=value1
VAR2=value2
USER

$ docker run --env-file env.list ubuntu env | grep VAR
VAR1=value1
VAR2=value2
USER=denis
```

Set metadata on container (-l, --label, --label-file)

A label is a `key=value` pair that applies metadata to a container. To label a container with two labels:

```
$ docker run -l my-label --label com.example.foo=bar ubuntu bash
```

The `my-label` key doesn't specify a value so the label defaults to an empty string (`""`). To add multiple labels, repeat the label flag (`-l` or `--label`).

The `key=value` must be unique to avoid overwriting the label value. If you specify labels with identical keys but different values, each subsequent value overwrites the previous. Docker uses the last `key=value` you supply.

Use the `--label-file` flag to load multiple labels from a file. Delimit each label in the file with an EOL mark. The example below loads labels from a labels file in the current directory:

```
$ docker run --label-file ./labels ubuntu bash
```

The label-file format is similar to the format for loading environment variables. (Unlike environment variables, labels are not visible to processes running inside a container.) The following example illustrates a label-file format:

```
com.example.label1="a label"

# this is a comment
com.example.label2=another\ label
com.example.label3
```

You can load multiple label-files by supplying multiple `--label-file` flags.

For additional information on working with labels, see *Labels - custom metadata in Docker* (https://docs.docker.com/engine/userguide/labels-custom-metadata/) in the Docker User Guide.

Connect a container to a network (--network)

When you start a container use the `--network` flag to connect it to a network. This adds the `busybox` container to the `my-net` network.

```
$ docker run -itd --network=my-net busybox
```

You can also choose the IP addresses for the container with `--ip` and `--ip6` flags when you start the container on a user-defined network.

```
$ docker run -itd --network=my-net --ip=10.10.9.75 busybox
```

If you want to add a running container to a network use the `docker network connect` subcommand.

You can connect multiple containers to the same network. Once connected, the containers can communicate easily need only another container’s IP address or name. For `overlay` networks or custom plugins that support multi-host connectivity, containers connected to the same multi-host network but launched from different Engines can also communicate in this way.

> **Note**: Service discovery is unavailable on the default bridge network. Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

You can disconnect a container from a network using the `docker network disconnect` command.

Mount volumes from container (--volumes-from)

```
$ docker run --volumes-from 777f7dc92da7 --volumes-from ba8c0c54f0f2:ro -i -t ubuntu pwd
```

The `--volumes-from` flag mounts all the defined volumes from the referenced containers. Containers can be specified by repetitions of the `--volumes-from` argument. The container ID may be optionally suffixed with `:ro` or `:rw` to mount the volumes in read-only or read-write mode, respectively. By default, the volumes are mounted in the same mode (read write or read only) as the reference container.

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 the 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.

Attach to STDIN/STDOUT/STDERR (-a)

The `-a` flag tells `docker run` to bind to the container's `STDIN` , `STDOUT` or `STDERR` . This makes it possible to manipulate the output and input as needed.

```
$ echo "test" | docker run -i -a stdin ubuntu cat -
```

This pipes data into a container and prints the container's ID by attaching only to the container's `STDIN` .

```
$ docker run -a stderr ubuntu echo test
```

This isn't going to print anything unless there's an error because we've only attached to the `STDERR` of the container. The container's logs still store what's been written to `STDERR` and `STDOUT` .

```
$ cat somefile | docker run -i -a stdin mybuilder dobuild
```

This is how piping a file into a container could be done for a build. The container's ID will be printed after the build is done and the build logs could be retrieved using `docker logs` . This is useful if you need to pipe a file or something else into a container and retrieve the container's ID once the container has finished running.

Add host device to container (--device)

```
$ docker run --device=/dev/sdc:/dev/xvdc \
             --device=/dev/sdd --device=/dev/zero:/dev/nulo \
             -i -t \
             ubuntu ls -l /dev/{xvdc,sdd,nulo}

brw-rw---- 1 root disk 8, 2 Feb  9 16:05 /dev/xvdc
brw-rw---- 1 root disk 8, 3 Feb  9 16:05 /dev/sdd
crw-rw-rw- 1 root root 1, 5 Feb  9 16:05 /dev/nulo
```

It is often necessary to directly expose devices to a container. The `--device` option enables that. For example, a specific block storage device or loop device or audio device can be added to an otherwise unprivileged container (without the `--privileged` flag) and have the application directly access it.

By default, the container will be able to `read` , `write` and `mknod` these devices. This can be overridden using a third `:rwm` set of options to each `--device` flag:

```
$ docker run --device=/dev/sda:/dev/xvdc --rm -it ubuntu fdisk  /dev/xvdc

Command (m for help): q
$ docker run --device=/dev/sda:/dev/xvdc:r --rm -it ubuntu fdisk  /dev/xvdc
You will not be able to write the partition table.

Command (m for help): q

$ docker run --device=/dev/sda:/dev/xvdc:rw --rm -it ubuntu fdisk  /dev/xvdc

Command (m for help): q

$ docker run --device=/dev/sda:/dev/xvdc:m --rm -it ubuntu fdisk  /dev/xvdc
fdisk: unable to open /dev/xvdc: Operation not permitted
```

> Note: `--device` cannot be safely used with ephemeral devices. Block devices that may be removed should not be added to untrusted containers with `--device` .

Restart policies (--restart)

Use Docker's `--restart` to specify a container's *restart policy*. A restart policy controls whether the Docker daemon restarts a container after exit. Docker supports the following restart policies:

Policy	Result
`no`	Do not automatically restart the container when it exits. This is the default.
`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.
`unless-stopped`	Restart the container unless it is explicitly stopped or Docker itself is stopped or restarted.
`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.

```
$ docker run --restart=always redis
```

This will run the `redis` container with a restart policy of **always** so that if the container exits, Docker will restart it.

More detailed information on restart policies can be found in the Restart Policies (--restart) (https://docs.docker.com/engine/reference/run/#restart-policies---restart) section of the Docker run reference page.

Add entries to container hosts file (--add-host)

You can add other hosts into a container's `/etc/hosts` file by using one or more `--add-host` flags. This example adds a static address for a host named `docker` :

```
$ docker run --add-host=docker:10.180.0.1 --rm -it debian

root@f38c87f2a42d:/# ping docker
PING docker (10.180.0.1): 48 data bytes
56 bytes from 10.180.0.1: icmp_seq=0 ttl=254 time=7.600 ms
56 bytes from 10.180.0.1: icmp_seq=1 ttl=254 time=30.705 ms
^C--- docker ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 7.600/19.152/30.705/11.553 ms
```

Sometimes you need to connect to the Docker host from within your container. To enable this, pass the Docker host's IP address to the container using the `--add-host` flag. To find the host's address, use the `ip addr show` command.

The flags you pass to `ip addr show` depend on whether you are using IPv4 or IPv6 networking in your containers. Use the following flags for IPv4 address retrieval for a network device named `eth0` :

```
$ HOSTIP=`ip -4 addr show scope global dev eth0 | grep inet | awk '{print \$2}' | cut -d / -f 1`
$ docker run  --add-host=docker:${HOSTIP} --rm -it debian
```

For IPv6 use the `-6` flag instead of the `-4` flag. For other network devices, replace `eth0` with the correct device name (for example `docker0` for the bridge device).

Set ulimits in container (--ulimit)

Since setting `ulimit` settings in a container requires extra privileges not available in the default container, you can set these using the `--ulimit` flag. `--ulimit` is specified with a soft and hard limit as such: `<type>=<soft limit>[:<hard limit>]` , for example:

```
$ docker run --ulimit nofile=1024:1024 --rm debian sh -c "ulimit -n"
1024
```

✓ Note: If you do not provide a **`hard limit`** , the **`soft limit`** will be used for both values. If no **`ulimits`** are set, they will be inherited from the default **`ulimits`** set on the daemon. **`as`** option is disabled now. In other words, the following script is not supported:

```
$ docker run -it --ulimit as=1024 fedora /bin/bash`
```

The values are sent to the appropriate `syscall` as they are set. Docker doesn't perform any byte conversion. Take this into account when setting the values.

FOR `NPROC` USAGE

Be careful setting `nproc` with the `ulimit` flag as `nproc` is designed by Linux to set the maximum number of processes available to a user, not to a container. For example, start four containers with `daemon` user:

```
$ docker run -d -u daemon --ulimit nproc=3 busybox top

$ docker run -d -u daemon --ulimit nproc=3 busybox top

$ docker run -d -u daemon --ulimit nproc=3 busybox top

$ docker run -d -u daemon --ulimit nproc=3 busybox top
```

The 4th container fails and reports "[8] System error: resource temporarily unavailable" error. This fails because the caller set `nproc=3` resulting in the first three containers using up the three processes quota set for the `daemon` user.

Stop container with signal (--stop-signal)

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

Optional security options (--security-opt)

On Windows, this flag can be used to specify the `credentialspec` option. The `credentialspec` must be in the format `file://spec.txt` or `registry://keyname` .

Stop container with timeout (--stop-timeout)

The `--stop-timeout` flag sets the timeout (in seconds) that a pre-defined (see `--stop-signal`) system call signal that will be sent to the container to exit. After timeout elapses the container will be killed with SIGKILL.

Specify isolation technology for container (--isolation)

This option is useful in situations where you are running Docker containers on 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 Windows, `--isolation` can take one of these values:

Value	Description
`default`	Use the value specified by the Docker daemon's `--exec-opt` or system default (see below).
`process`	Shared-kernel namespace isolation (not supported on Windows client operating systems).
`hyperv`	Hyper-V hypervisor partition-based isolation.

The default isolation on Windows server operating systems is `process` . The default (and only supported) isolation on Windows client operating systems is `hyperv` . An attempt to start a container on a client operating system with `--isolation process` will fail.

On Windows server, assuming the default configuration, these commands are equivalent and result in `process` isolation:

```
PS C:\> docker run -d microsoft/nanoserver powershell echo process
PS C:\> docker run -d --isolation default microsoft/nanoserver powershell echo process
PS C:\> docker run -d --isolation process microsoft/nanoserver powershell echo process
```

If you have set the `--exec-opt isolation=hyperv` option on the Docker `daemon` , or are running against a Windows client-based daemon, these commands are equivalent and result in `hyperv` isolation:

```
PS C:\> docker run -d microsoft/nanoserver powershell echo hyperv
PS C:\> docker run -d --isolation default microsoft/nanoserver powershell echo hyperv
PS C:\> docker run -d --isolation hyperv microsoft/nanoserver powershell echo hyperv
```

Specify hard limits on memory available to containers (-m, --memory)

These parameters always set an upper limit on the memory available to the container. On Linux, this is set on the cgroup and applications in a container can query it at `/sys/fs/cgroup/memory/memory.limit_in_bytes` .

On Windows, this will affect containers differently depending on what type of isolation is used.

- With `process` isolation, Windows will report the full memory of the host system, not the limit to applications running inside the container

```
  PS C:\> docker run -it -m 2GB --isolation=process microsoft/nanoserver powershell Get-Com
puterInfo *memory*

  CsTotalPhysicalMemory      : 17064509440
  CsPhyicallyInstalledMemory : 16777216
  OsTotalVisibleMemorySize   : 16664560
  OsFreePhysicalMemory       : 14646720
  OsTotalVirtualMemorySize   : 19154928
  OsFreeVirtualMemory        : 17197440
  OsInUseVirtualMemory       : 1957488
  OsMaxProcessMemorySize     : 137438953344
```

- With `hyperv` isolation, Windows will create a utility VM that is big enough to hold the memory limit, plus the minimal OS needed to host the container. That size is reported as "Total Physical Memory."

```
  PS C:\> docker run -it -m 2GB --isolation=hyperv microsoft/nanoserver powershell Get-Comp
uterInfo *memory*

  CsTotalPhysicalMemory      : 2683355136
  CsPhyicallyInstalledMemory :
  OsTotalVisibleMemorySize   : 2620464
  OsFreePhysicalMemory       : 2306552
  OsTotalVirtualMemorySize   : 2620464
  OsFreeVirtualMemory        : 2356692
  OsInUseVirtualMemory       : 263772
  OsMaxProcessMemorySize     : 137438953344
```

Configure namespaced kernel parameters (sysctls) at runtime

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.

CURRENTLY SUPPORTED SYSCTLS

- `IPC Namespace` :

```
kernel.msgmax, kernel.msgmnb, kernel.msgmni, kernel.sem, kernel.shmall, kernel.shmmax, kern
el.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` :

  Sysctls beginning with net.*

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